Golden System Reference Design and Demo User Guide v2.0 for CertusPro-NX Devices

Lattice Propel 2024.1

Lattice Radiant 2024.1.1

Reference Design

FPGA-RD-02300-1.1

December 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
AHBL	Advanced High-performance Bus-Lite
AXI	Advanced eXtensible Interface
APB	Advanced Peripheral Bus
API	Application Programming Interface
AXI4-Lite	Advanced eXtensible Interface-Lite
GPIO	General Purpose Input/Output
CPU	Central Processing Unit
DDR	Double Data Rate
FIFO	First-In-First-Out
FMC	FPGA Mezzanine Card
ISR	Interrupt Service Routines
LMMI	Lattice Memory Mapped Interface
LPDDR4	*Low Power Double Data Rate Generation 4*
MPMC	Multi-Port Memory Controller
QSPI	Quad Serial Peripheral Interface
RISC-V	Reduced Instruction Set Computer-V
RTL	Register Transfer Level
SFP	Small Form-Factor Pluggable
SGDMA	Scatter-Gather Direct Memory Access
SGMII PCS and GbE	Serial Gigabit Media Independent Interface
SoC	System on Chip
TSE MAC	Tri - Speed Ethernet Media Access Controller
UART	*Universal Asynchronous Receiver-Transmitter*

1. Introduction

1.1. Overview of the System

The Lattice FPGA-based GSRD SoC design presented herein is aimed at providing a versatile and efficient platform for embedded applications requiring high-performance computing, memory access, data transfer capabilities, and network communication. By integrating various components onto a single FPGA chip, this design offers flexibility, scalability, and cost-effectiveness for a wide range of applications.

The Lattice Golden Hardware Reference Design (GHRD) is a System-on-Chip (SoC) that can be used as a baseline design to create FPGA applications as per the user requirements. It is a RISC-V based design that interacts with various Lattice Soft-IPs and peripherals such as GPIO, UART, I2C, Timer, Tri-Speed Ethernet MAC (TSE MAC), QSPI Flash Controller, Scatter-Gather DMA (SGDMA) and LPDDR4 Memory Controller. All these building blocks are connected through industry standard protocols such as AXI4-Full, AXI-Steam for data transfers and AXI4-Lite, APB for control.

The Lattice Golden Software Reference Design (GSRD) is a comprehensive embedded system which incorporates drivers and relevant firmware needed to operate various design components. Free-RTOS and First Stage Bootloader (FSBL) is built on RISC-V RX CPU Core. The primary function of FSBL is to initialize the hardware blocks in the design using the respective IP drivers and ensure the integrity of the firmware by performing CRC check.

GHRD and GSRD are integrated together to establish a complete system for which relevant binaries and executables are generated by Lattice SW tools such as Propel SDK, Propel Builder and Radiant to program the FPGA Hardware.

As a part of multi-boot demo, the executables folder comprises of compatible binaries and executable images, such as FPGA Bitstream (.bit) and Firmware Binary (.bin) for both Primary and Golden GSRD projects. The only difference is that the Primary bitstream contains the code for multi-boot enablement and Golden bitstream does not enable multi-boot.

GSRD for CertusPro™-NX device is developed and tested with the following Lattice Propel™ and Lattice Radiant™ software versions.

Notes:

1. The SGMII interface using LVDS I/O has limitations when operating across the full specified temperature range. Lattice recommends using alternative interfaces, such as SERDES or RGMII, for designs requiring Gigabit Ethernet. Refer to the Knowledge Database article for details. Contact your local Lattice sales representative for more information.
2. The Golden System Reference Design and Demonstration for CertusPro-NX version 3.0 is available in the GHRD/GSRD Reference Design and GHRD/GSRD Demonstration web pages. Refer to this reference design and demonstration for the SGMII interface that is implemented using SERDES.

1.2. Quick Facts

Table 1.1. Summary of the System

SoC Requirements	Supported FPGA Family	CertusPro-NX
	SoC Version	2.0
FPGA Device(s)	Targeted Devices	LFCPNX-100
	Supported User Interface	AXI4-Full, AXI4-Lite, AXI-Stream, APB
Design Tool Support	Lattice Implementation	Lattice Propel Software 2024.1 Lattice Radiant Software 2024.1.1
	Synthesis	Synopsys® Synplify Pro®

1.3. Features

The key features of the system include:

- FPGA device supported in this document is CertusPro-NX
- RISC-V RX CPU Core, SGDMA, TSE MAC, LPDDR4, and QSPI Flash Controller over AXI4 Interface
- Low-speed peripherals like GPIO and UART
- Bootloader, Primary and Golden FreeRTOS Application
- Application Firmware's CRC check by function implemented in RISC-V RX bootloader code
- FPGA bitstream CRC check done by FPGA Configuration Engine
- Manual and Automatic Multi-Boot capability
- FreeRTOS Application Software is run on LPDDR4 MC
- 256-bit AXI4 LPDDR4 data width to support 32-bit data to DDR at 400/533 MHz operating frequency
- AXI4/AXI4-Lite and APB Peripherals at 100 MHz
- 1 Gbps Ethernet throughput through SGMII SFP support at 125 MHz

1.4. Naming Conventions

1.4.1. Nomenclature

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

1.4.2. Signal Names

Signal names that end with:

- _n are active low (asserted when value if logic 0)
- _i are input signals
- _o are output signals
- _io are bi-directional input/output signals

2. Functional Description

The GSRD/GHRD SoC architecture comprises of a RISC-V CPU core, LPDDR4 Memory Controller, SGDMA controller, QSPI Flash Controller, and a 1 Gbps TSE MAC, interconnected through a combination of high-speed and low-speed bus fabrics such as AXI4 Interconnect and APB Interconnect. This architecture enables seamless communication and data exchange between the components, facilitating efficient operation and system performance.

2.1. System Architecture Overview

2.1.1. CertusPro-NX Architecture

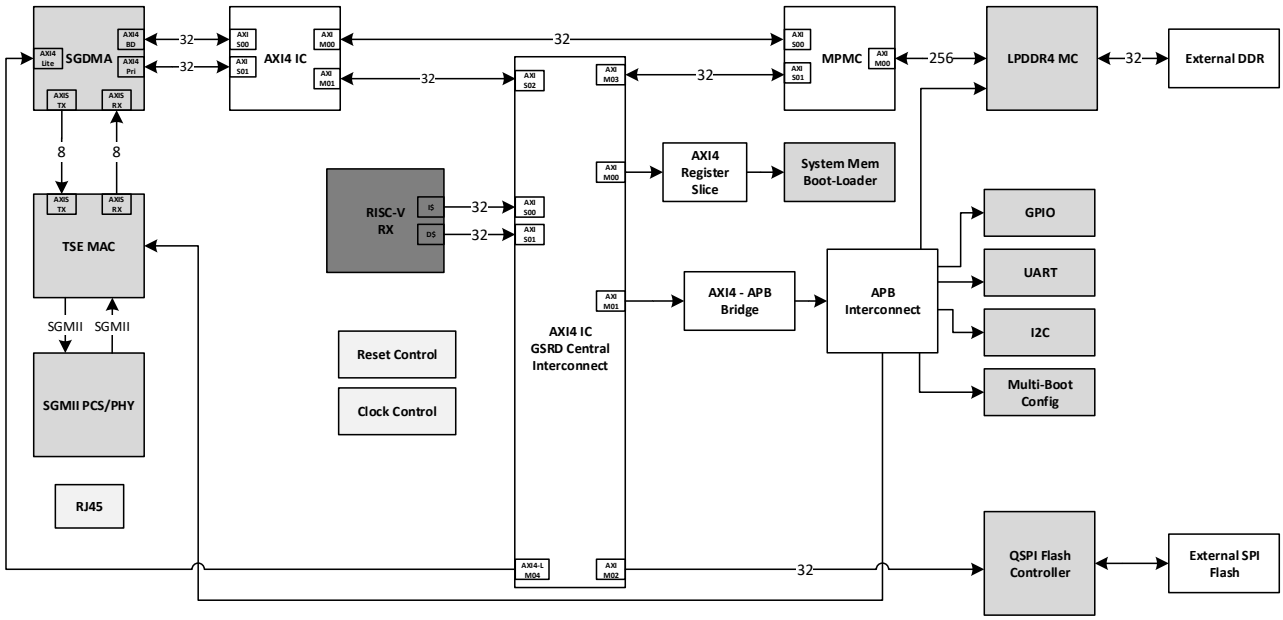

Figure 2.1. GHRD Architecture on a CertusPro-NX Device

The design includes the following components:

Table 2.1. IP Versions

Soft-IP	IP Version
RISC-V RX Processor	v2.4.0
LPDDR4 Memory Controller	v2.3.1
System Memory	v.2.2.0
QSPI Flash Controller	v1.2.0
General Purpose I/O GPIO	v1.6.2
UART	v1.3.0
I2C Controller	v2.0.1
AXI4 Interconnect	v2.0.1
APB Interconnect	v1.2.1
AXI to APB Bridge	v1.2.0
Tri-Speed Ethernet MAC	v1.6.0
SGMII PCS/PHY	V1.6.2
SGDMA	v2.2.0
AXI4 Register Slice	v1.0.0
Multi-Boot Configuration	v1.0.0

Soft-IP	IP Version
PLL	v1.9.0
OSC	v2.1.0

Each component in the block diagram is instantiated using the IP in Propel Builder. The IP features and parameters are described in the IP Configurations section.

The signals in each interface are described in the Signal Description section.

2.2. Clocking

This section describes the reference design clocking scheme. There are some minor differences between the clocking scheme and the Lattice CertusPro-NX device. Refer to the CertusPro-NX Clocking Overview section.

2.2.1. CertusPro-NX Clocking Overview

There are five clocks in GSRD CertusPro-NX Architecture. They are as follows:

- External Reference Clock for DDR4 is 100 MHz
- External Reference Clock for TSE MAC is 125 MHz
- On-Chip Oscillator Clock is 125 MHz
- RISC-V Realtime Low-Frequency Clock derived from OSC inside the FPGA Config IP
- Internal System Clock derived from FPGA PLL is 100 MHz

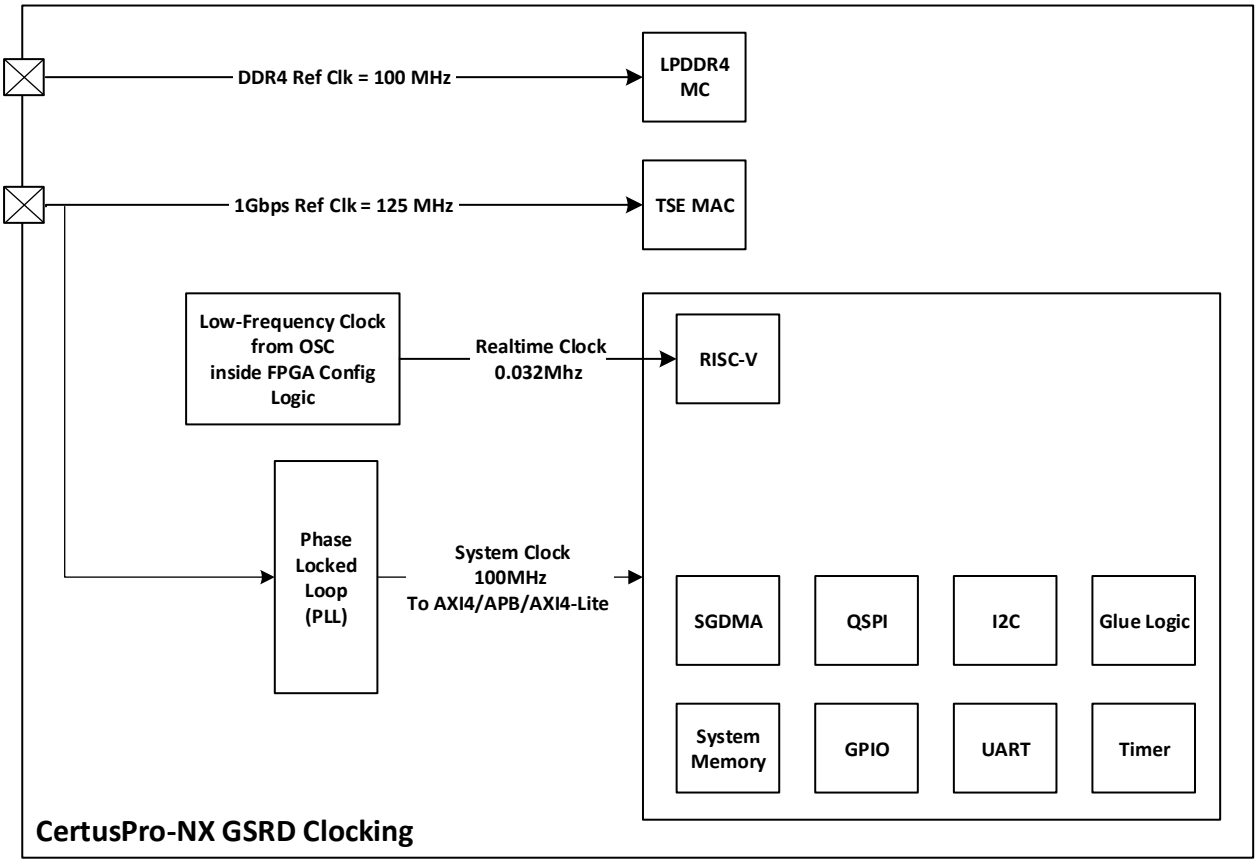

Figure 2.2. Clocking Structure for CertusPro-NX GSRD

2.3. Reset Scheme

There are two resets in the entire design:

- External Asynchronous Reset which is controlled by a push button
- Synchronous Reset for entire system is generated from RISC-V

Table 2.2. Reset Scheme

Reset Signal	Source	Destination	Description
rstn_i	Board Pushbutton	PLL reset input and Synchronizer	Reset the PLL
cpu_rstn_i	PLL Lock and synced rstn_i	RISC-V RX reset input	Release RISC-V RX from reset pin, synchronizer and PLL lock
system_rstn_o	RISC-V RX output	All components in system	RISC-V RX output reset provides reset to all components in the design. This also triggers the reset during CPU OCD debugging mode.

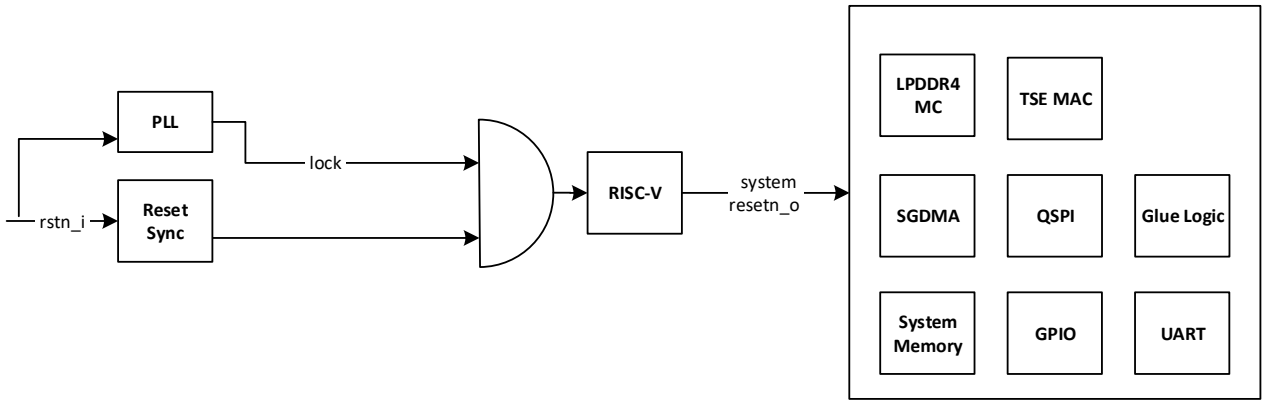

Figure 2.3. Reset Structure for CertusPro-NX GSRD

2.4. IP Configurations

The reference design is created using Lattice Propel Builder. The top-level HDL file is generated by Propel Builder and is used as the top module for the design. The design parameterization is performed by configuring the IP in Propel Builder. This section describes the following IPs and their configuration.

2.4.1. RISC-V RX CPU Core

For more information about the IP core including register map information, refer to RISC-V RX CPU IP Core User Guide (FPGA-IPUG-02241). The RISC-V RX CPU IP has AXI-based instruction and data ports. The instruction ports are connected to the memory that contains the bootloader software or the FreeRTOS application software for CPU execution. The data port is connected to the memory and peripherals for control.

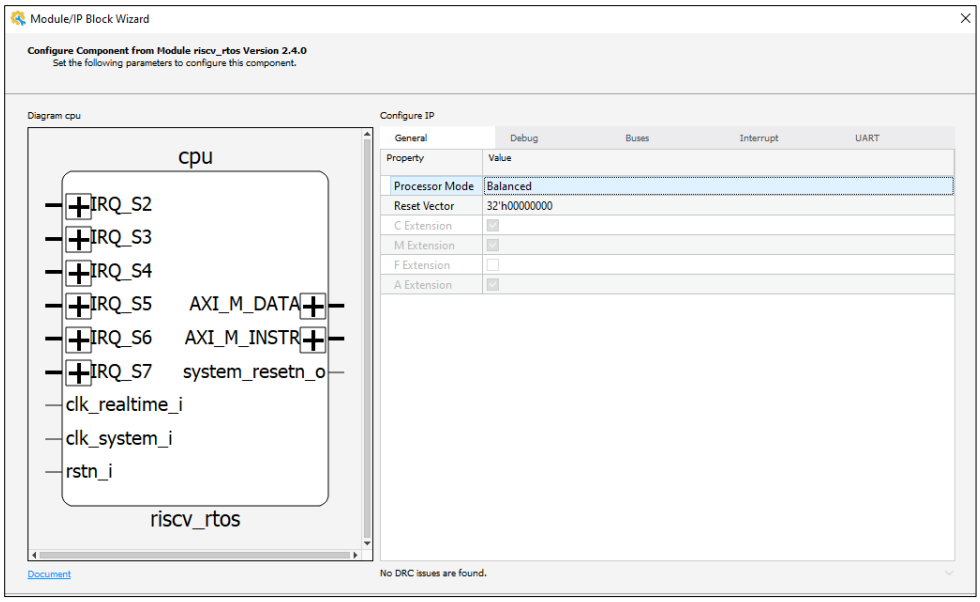

Figure 2.4. CPU Configuration – General

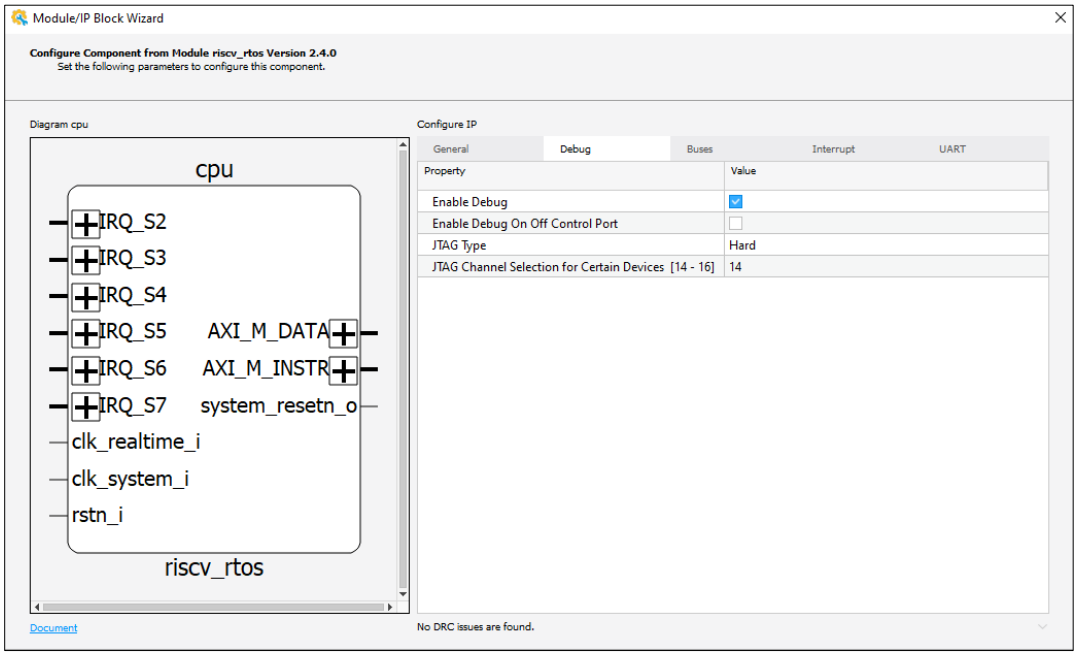

Figure 2.5. CPU Configuration – Debug

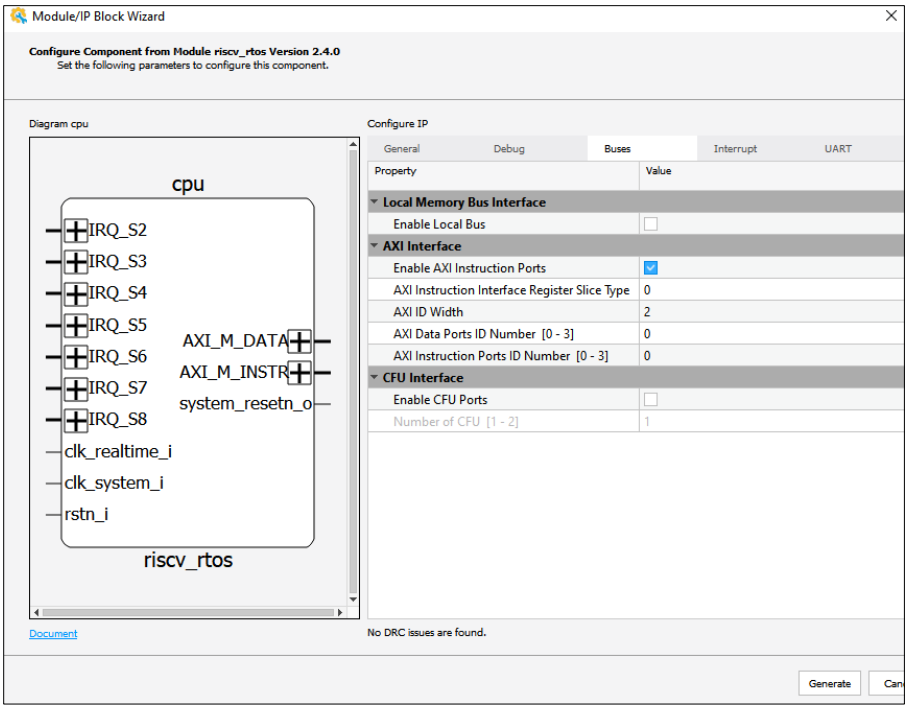

Figure 2.6. CPU Configuration – Buses

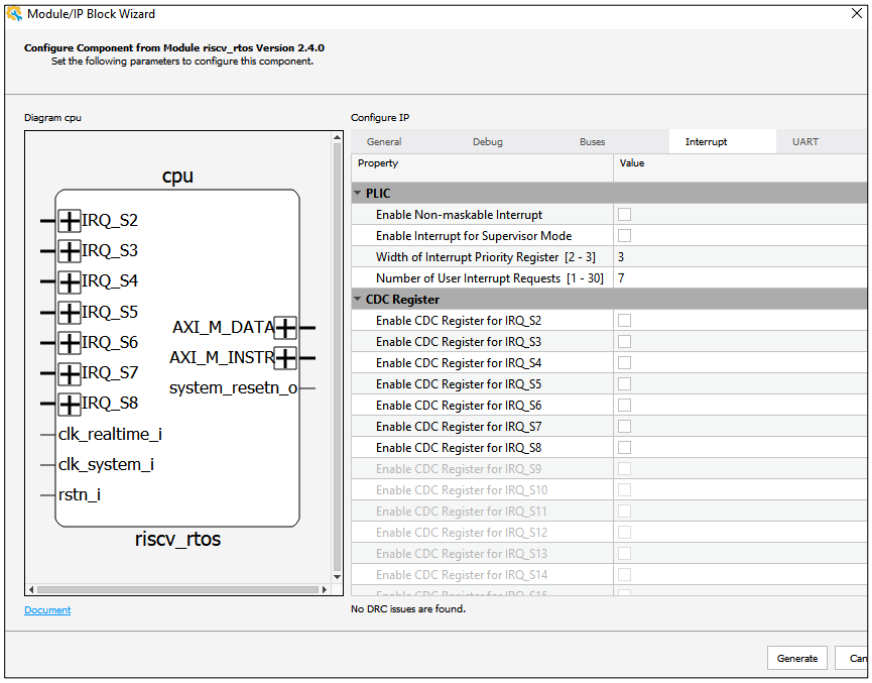

Figure 2.7. CPU Configuration – Interrupt

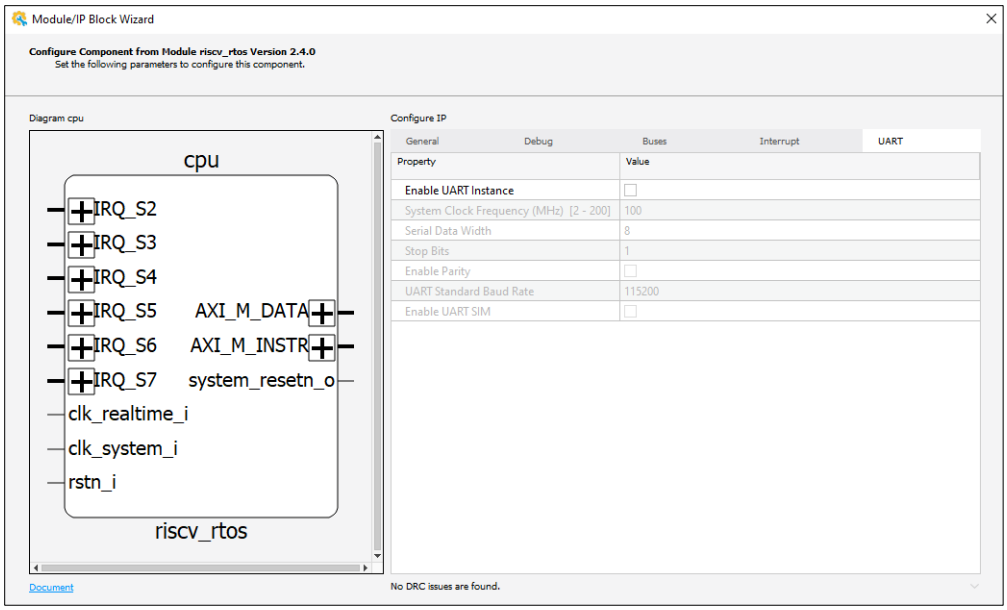

Figure 2.8. CPU Configuration – UART

2.4.2. LPDDR4 Memory Controller

For more information about the IP core including register map information, refer to Memory Controller IP Core for Nexus Devices (FPGA-IPUG-02127).

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

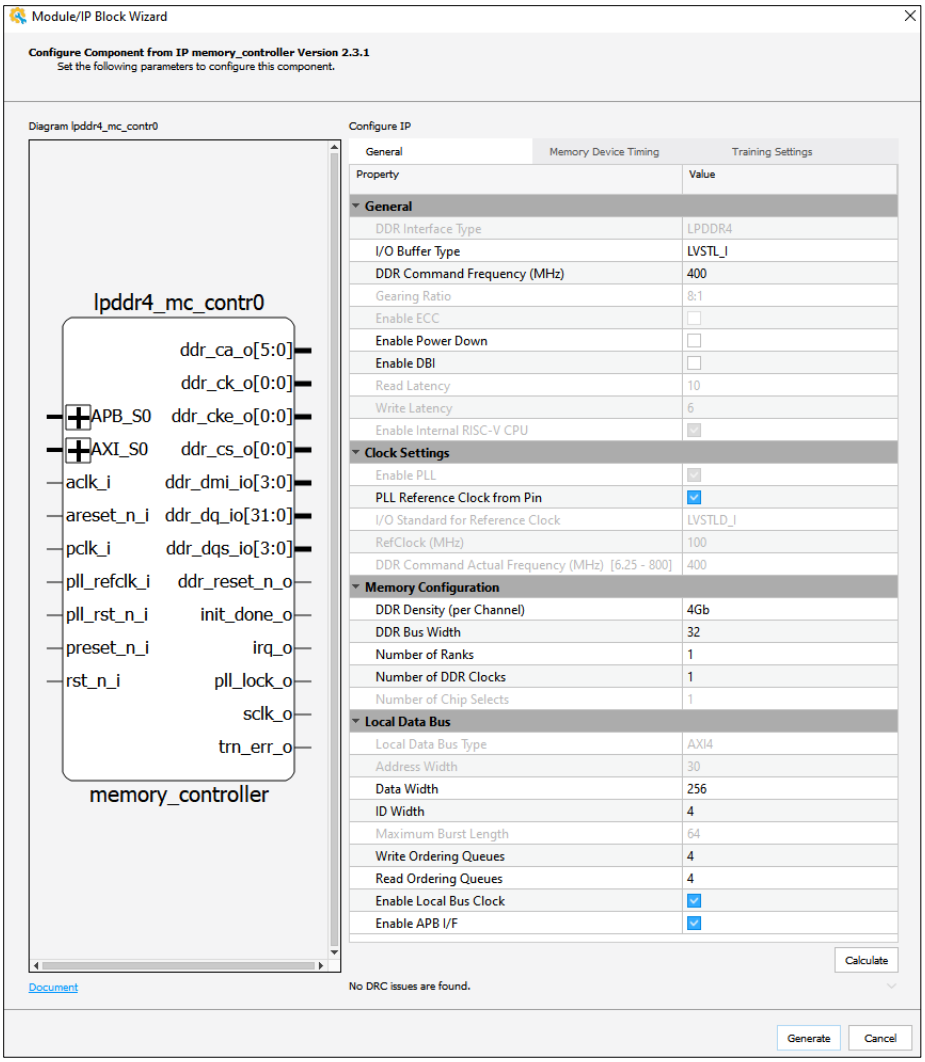

Figure 2.9. LPDDR4 MC Configuration

2.4.3. QSPI Flash Controller

For more information about the IP core including register map information, refer to QSPI Flash Controller IP User Guide (FPGA-IPUG-02248).

The QSPI Flash Controller is a four tri-state data line serial interface that is commonly used to store, program, erase and read SPI Flash memories. QSPI enhances the throughput of a standard SPI by four times since four bits are transferred every cycle. In GSRD, the QSPI is used to store the application software and bitstreams for both Primary and Golden systems.

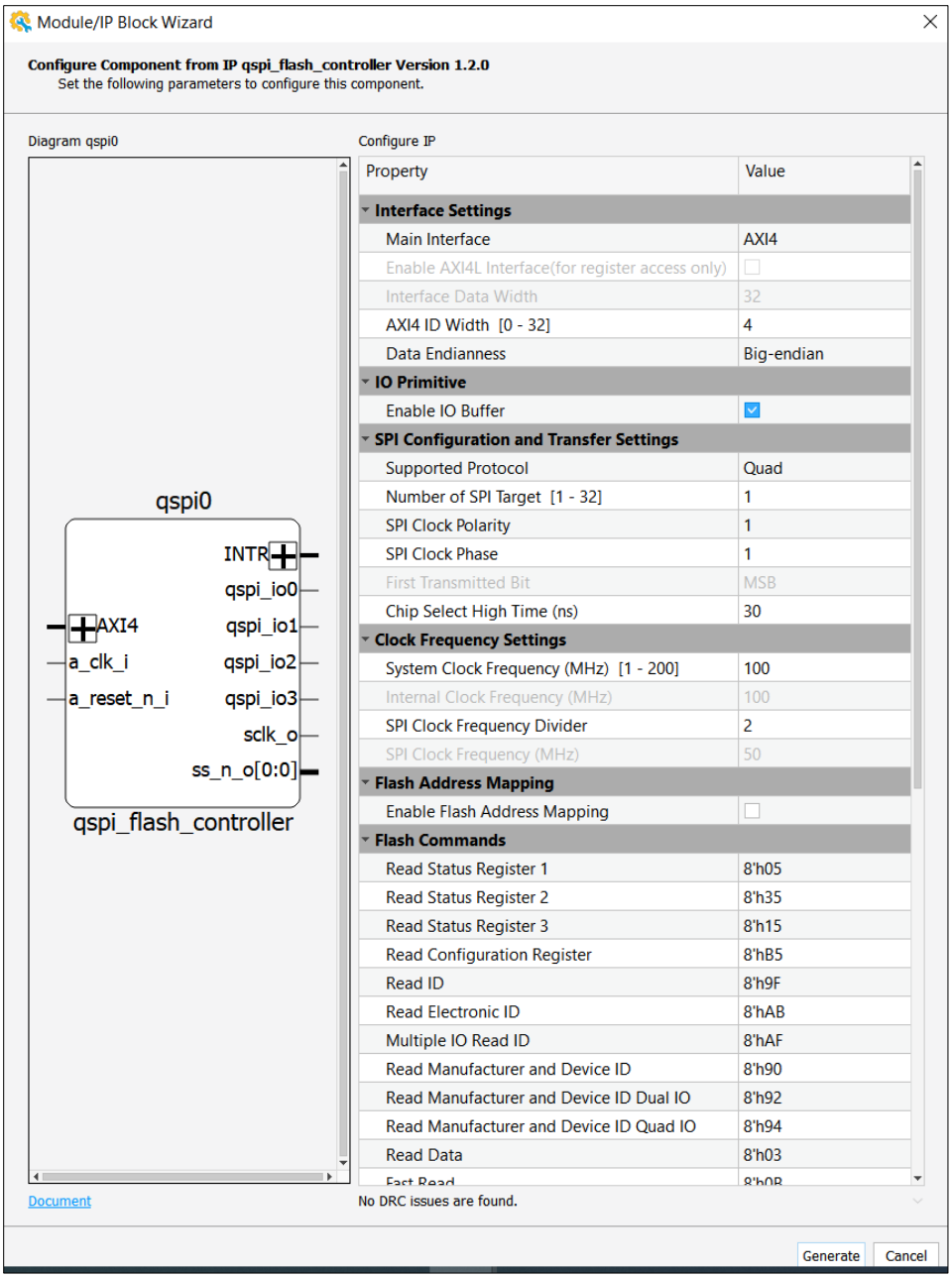

Figure 2.10. QSPI Flash Controller Configuration

2.4.4. Tri-Speed Ethernet MAC

For more information about the IP core including register map information, refer to Tri-Speed Ethernet MAC IP User Guide (FPGA-IPUG-02084).

The TSEMAC IP core is a 10/100/1000 Mbps network interface as per the IEEE 802.3 standard. It is complex core containing all the necessary logic, interfacing and clocking infrastructure to allow integrating an external industry-standard Ethernet PHY with an internal processor, with minimal overhead. The GSRD requires the SGMII Easy Connect to support 1 Gbps data-rates.

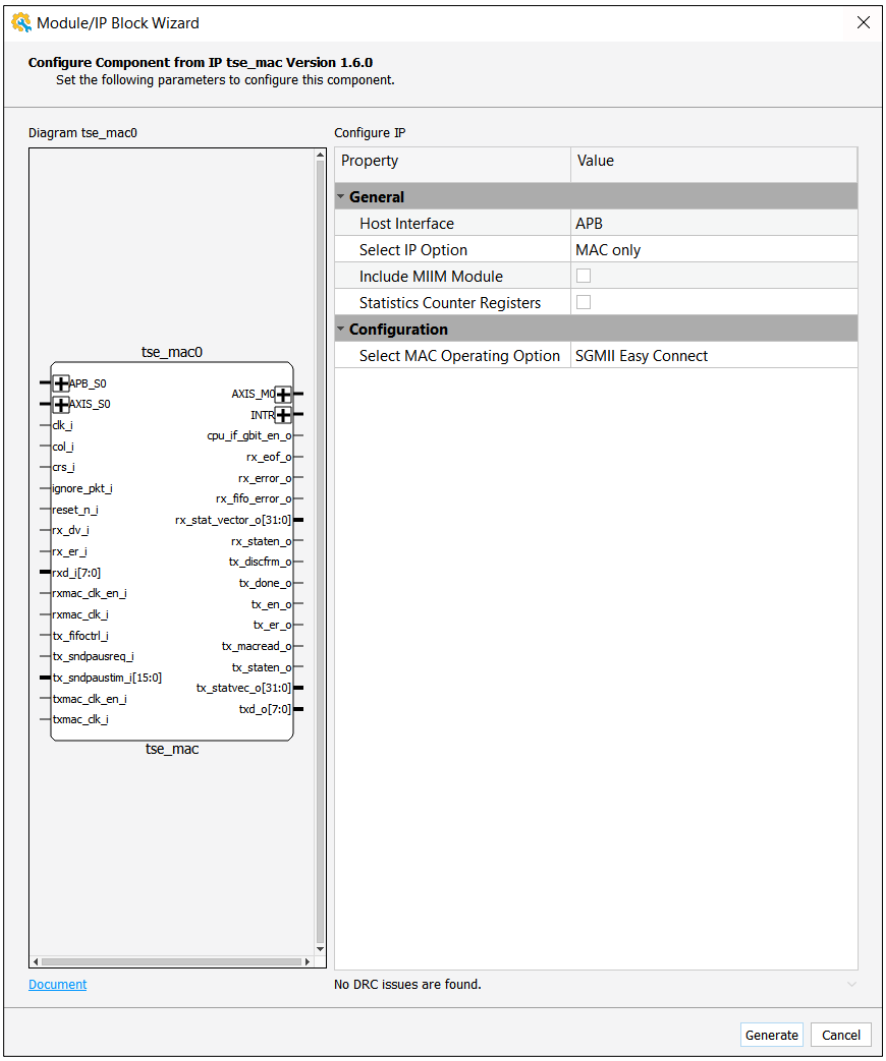

Figure 2.11. TSE MAC Configuration

2.4.5. SGMII PCS/PHY

For more information about the IP core including register map, refer to SGMII and Gb Ethernet PCS IP Core (FPGA-IPUG-02077).

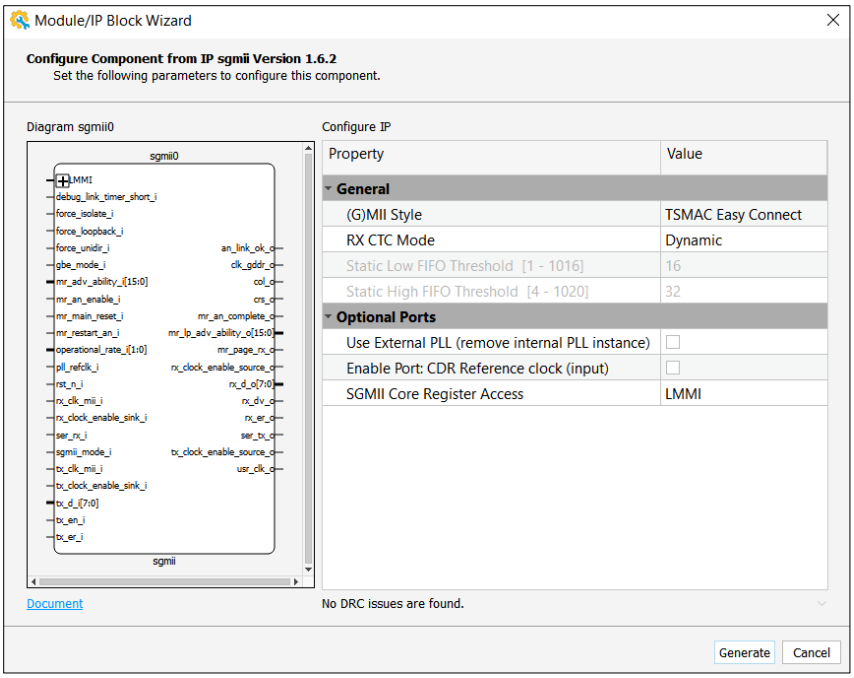

Figure 2.12. SGMII PCS/PHY Configuration

2.4.6. Scatter-Gather DMA

For more information about the IP core including register map information, refer to SGDMA Controller IP Core (FPGA-IPUG-02131).

The SGDMA Controller IP core is to access the main memory independent of the CPU processor. It offloads processor intervention. The processor initiates transfer to SGDMA Controller and receives interrupts on completion of the transfer by the DMA engine. The core implements a configurable, AXI4-compliant DMA controller with scatter-gather capability. It also implements the AXI4-Stream interface to support stream data from TSE MAC module. The AXI4-Lite CSR interface is used to configure the control and status registers by the RISC-V CPU.

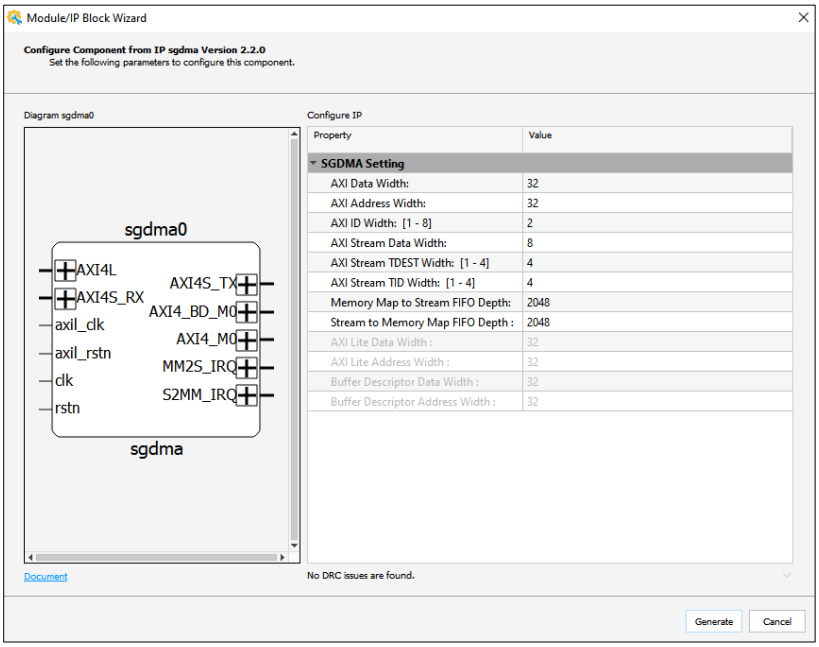

Figure 2.13. SGDMA Configuration

2.4.7. UART

For more information about the IP core including register map information, refer to UART IP User Guide (FPGA-IPUG-02105)

The Universal Asynchronous Receiver/Transmitted (UART) Transceiver IP core performs serial-to-parallel conversion of data characters received from a peripheral UART device and parallel-to-serial conversion of data characters received from the host locater insider the FPGA through an APB interface.

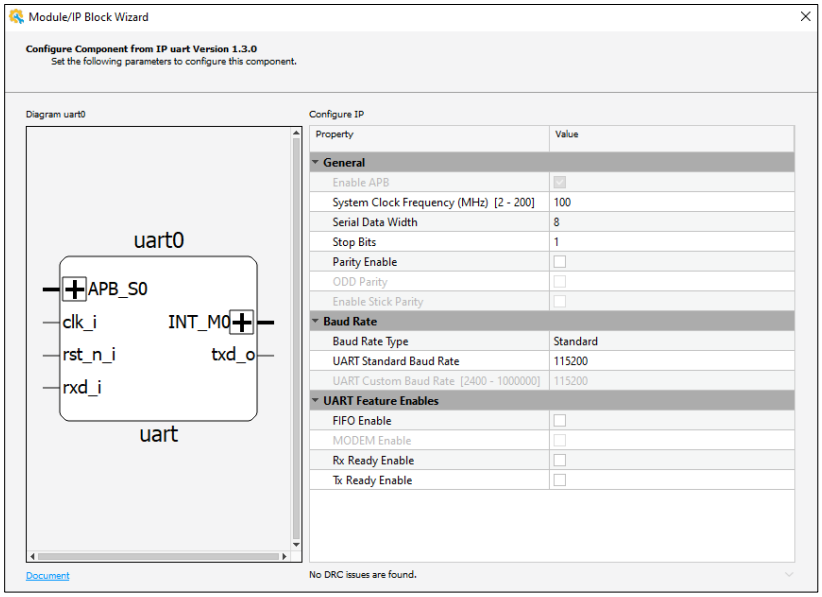

Figure 2.14. UART Configuration

2.4.8. GPIO

For more information about the IP core including register map information, refer to GPIO IP Core (FPGA-IPUG-02076).
The General Purpose Input/Output (GPIO) peripheral IP provides dedicated memory-mapped interface to configure the GPIO ports as well as the number of input and output ports.

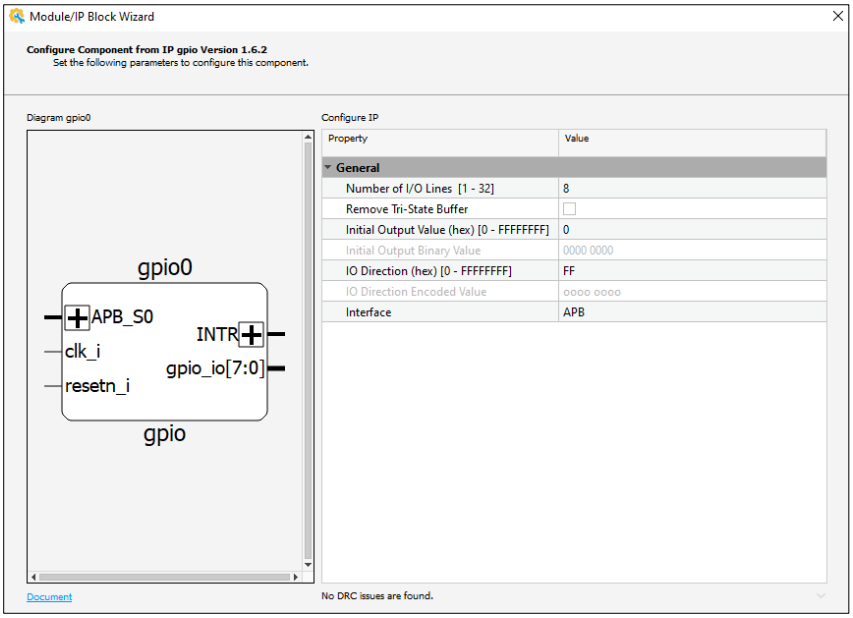

Figure 2.15. GPIO Configuration

2.4.9. I2C Controller

For more information about For more information about the IP core including register map information, refer to I2C Controller IP Core (FPGA-IPUG-02071).

The I2C Controller IP core provides a standard 2-wire external I2C bus interface. It supports out-of-band interrupt and configurable transmit/receive FIFO size to minimize intervention by the host. In GSRD, it used to control the Marvel PHY register space inside the SFP connector.

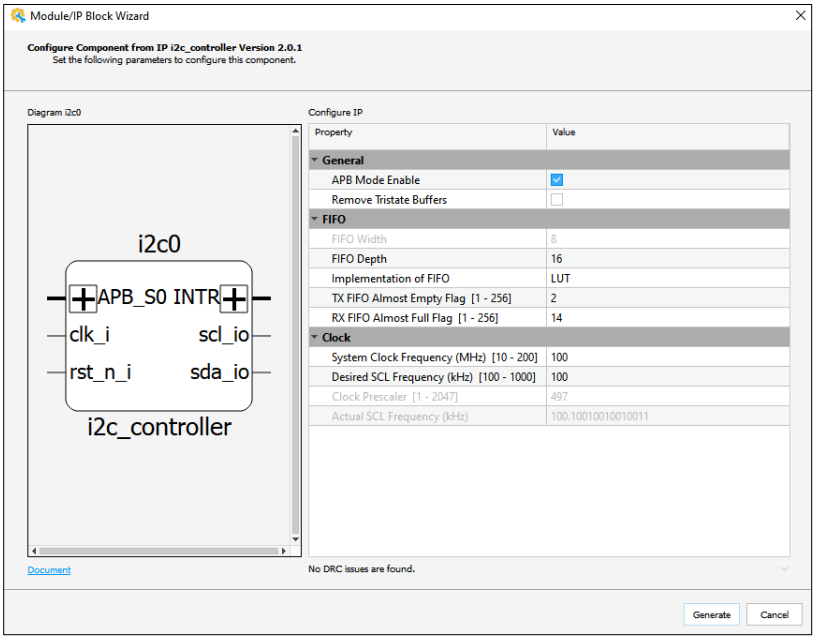

Figure 2.16. I2C Configuration for SFP PHY Control

2.4.10. Multi-Boot Configuration Module

The Multi-Boot Configuration is used to trigger an internal FPGA REFRESH/PROGRAMN command to LMMI logic. This core IP implements an APB endpoint which decodes the RISC-V CPU command data. The LMMI host FSM inside is used to execute the soft reset to load the next or alternate bitstream and application software data onto the FPGA.

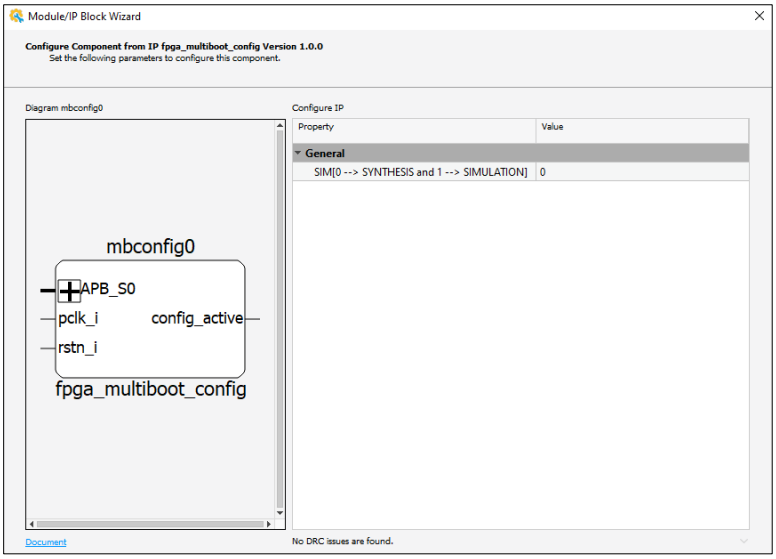

Figure 2.17. Multi-Boot Configuration

2.4.11. AXI Bridge Multi-Port Memory Controller

For more information, refer to AXI4 Multi Port Bridge for Memory Controller Module (FPGA-IPUG-02246). This is used to enable 256-bit AXI data-bus to LPDDR4 MC.

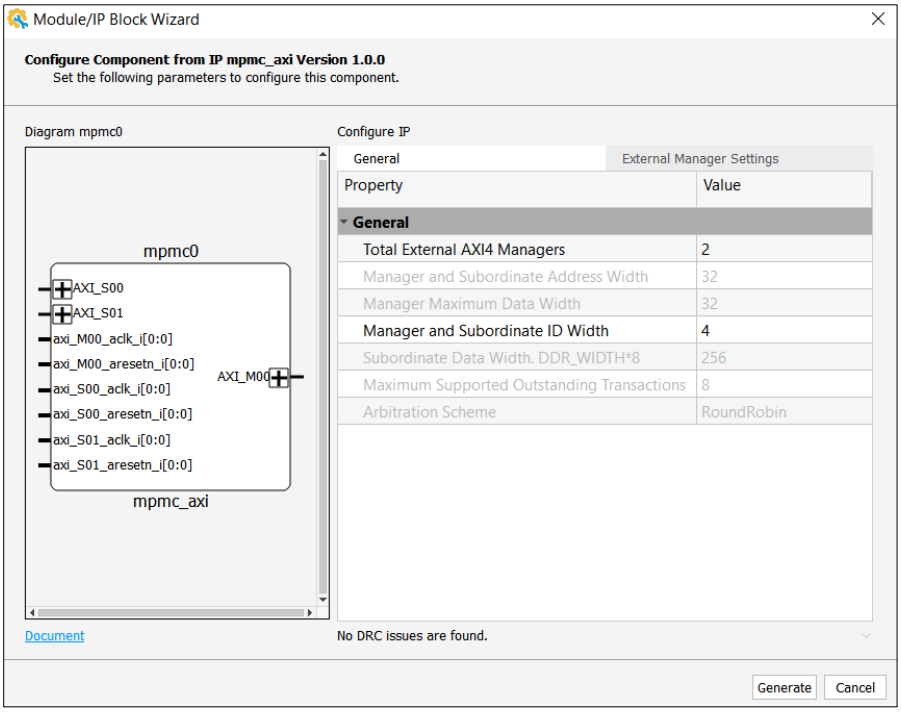

Figure 2.18. MPMC Configure IP General

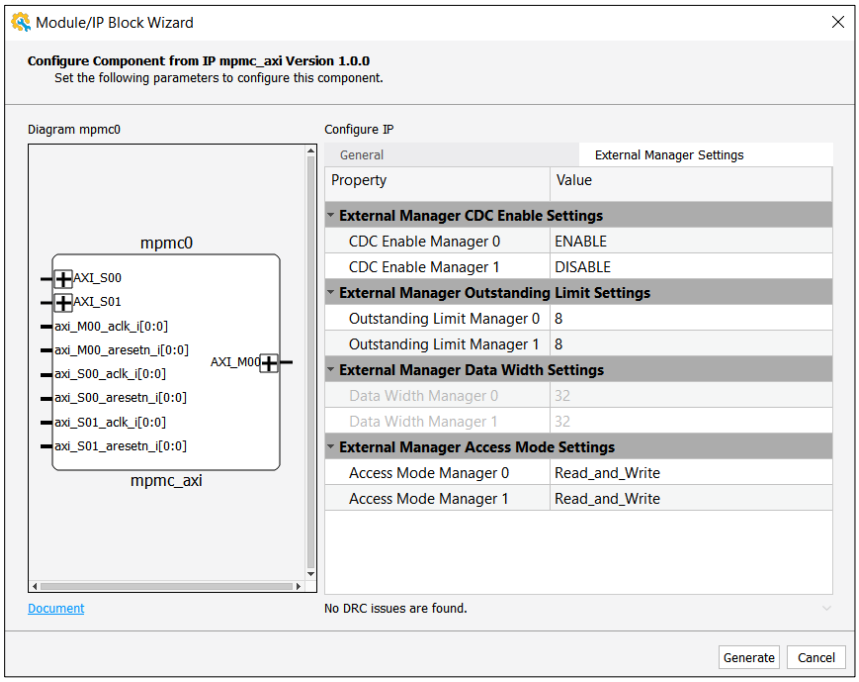

Figure 2.19. MPMC Configuration Manager Settings

2.4.12. System Memory

For more information about the IP Core, refer to System Memory Module (FPGA-IPUG-02073). The System Memory implements EBR, LRAM, or Distributed Memory in either single port or dual port AHBL or AXI4 subordinate. In GSRD, an EBR and single AXI4 port is used. The system memory in this design is used to store the bootloader.

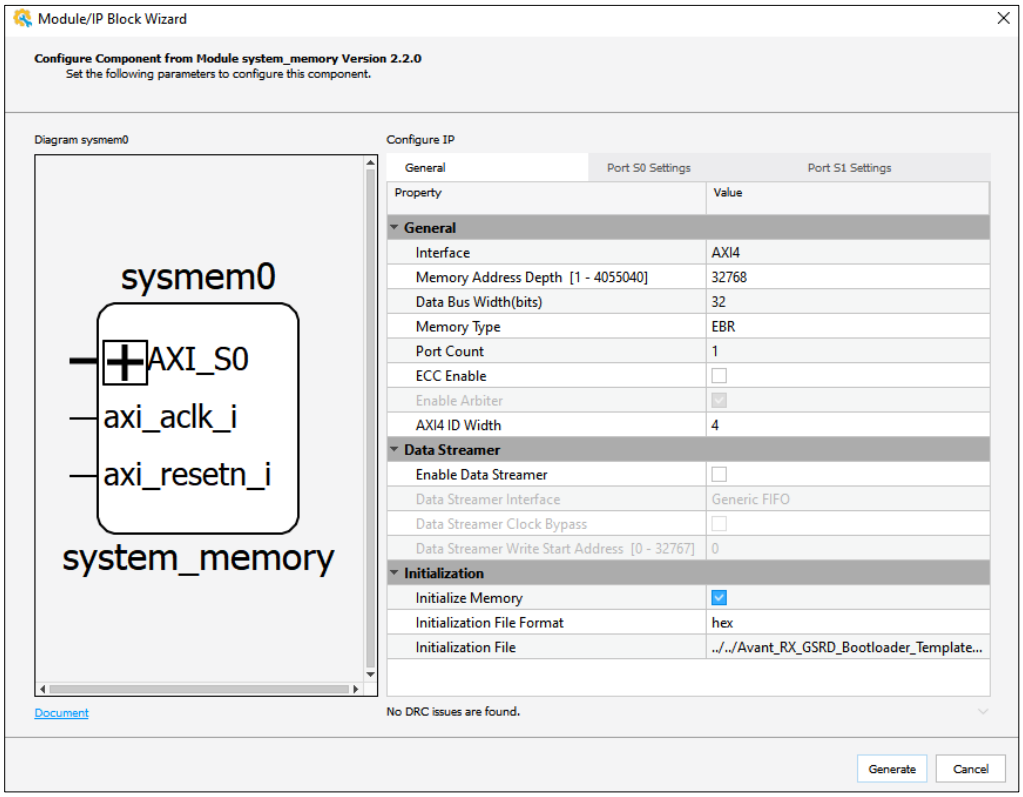

Figure 2.20. System Memory Configuration

2.5. System Level Interfaces

Table 2.3. System Level Interfaces

Top-Level Interface Name	Supported Protocol	Description
Advanced eXtensible Interface 4	AXI4	Used for Data/Control Interfaces on all IPs
Advanced eXtensible Interface 4 - Lite	AXI4-Lite	Used as Control Interface for SGDMA
Advanced Peripheral Interface	APB	Used as Control Interface for low-speed and LPDDR4 MC
Serial Peripheral Interface	QSPI	Used for communication with external SPI Flash
Dual Data Rate	LPDDR4	Used for communication with external LPDDR4
1G Ethernet	SGMII	Used for communication with external SFP

2.6. SoC Memory/Address Map

Table 2.4. Address Map of GHRD

Base Address	End Address	Size (kB/MB/1 GB)	Block
0x00000000	0x0003FFFF	256 kB	CPU Instruction/Data RAM
0xF2000000	0xF20FFFFF	1 MB	CLINT (CPU)
0xFC000000	0xFC3FFFFF	4 MB	PLIC (CPU)
0xF0000000	0xF00003FF	—	Reserved_Space1 (CPU)
0xF0000400	0xF1FFFFFF	—	Reserved_Space2 (CPU)
0xFC400000	0xFFFFFFFF	—	Reserved_Space3 (CPU)
0x40000000	0x40000FFF	4 kB	GPIO
0x40005000	0x400053FF	1 kB	I2C
0x40007000	0x4007FFFF	4 kB	LPDDR4 APB
0x80000000	0xBFFFFFFF	1 GB	LPDDR4 AXI
0x40006000	0x40006FFF	4 kB	Multi-Boot Config APB
0x40300000	0x4030FFFF	64 kB	QSPI Flash Controller
0x40093000	0x400903FF	4 kB	SGDMA
0x00000000	0x0001FFFF	128 kB	System Memory
0x40001000	0x40004FFF	16 kB	TSE MAC
0x40090000	0x40090FFF	4 kB	UART

2.7. Functional Operation

The GSRD system comprises of two SoCs (System on Chip) such as Primary and Golden. Each SoC is linked with its respective First-Stage Bootloader (FSBL) and a FreeRTOS Application Software. These SoCs are built using Lattice Propel SDK, Lattice Propel Builder, and Lattice Radiant software tools. The FPGA image comprises of the entire system and is generated by the Radiant tool in the (.bit) bitstream format. The bootloader code is stored inside the system memory and is part of the bitstream. The bitstream and the FreeRTOS application software are stored into the external SPI Flash (Macronix or Winbond).

During power-on, the Primary FPGA image is loaded into the device SRAM by the Config Engine. Before the device is completely programmed with the bitstream, the config engine checks the CRC (Cyclic Redundancy Check) for the bitstream to be loaded onto the FPGA. Once the FPGA is configured, the DONE LED on the board glows green. Immediately, the RISC-V starts executing the bootloader software stored inside system memory and initialize all the soft-IP modules and peripherals to establish a base for communication and data transfers. This process includes configuring the IPs for the desired system operation. After all the IP configuration is complete, RISC-V initiates trigger the QSPI Flash controller to copy the FreeRTOS application software from external SPI Flash into the external LPDDR4 for software execution. Once the application software is copied, the RISC-V checks the CRC on entire copied application software. If the CRC check fails, RISC-V initiates a soft reset/refresh by instructing the multi-boot configuration module to start the PROGRAMN sequence. The PROGRAMN sequence loads the next bitstream into the FPGA which in this case would be the Golden FPGA and FW images, and the same process follows. Once CRC check passes for either Primary or Golden, the RISC-V jumps to the application instruction and starts the FreeRTOS FW execution.

3. Signal Description

Table 3.1 shows the input/output interface signals for the top-level module.

Table 3.1. Top-level I/O

Signal Name	I/O Type	I/O Width	Description
clk_125MHz	Input	1	Reference clock input for internal PLL and TSE MAC/SGMII
pll_refclk_i	Input	1	Reference clock input for LPDDR4 MC internal PLL
rstn_i	Input	1	Active low reset input for reference design. Activate by pressing pushbutton on the board. CertusPro-NX devices: SW3
GPIO			
s0_gpio	Input/Output	8	General Purpose I/O signals connect to LED on board
UART			
s1_uart_txd_o	Output	1	UART transmit output. Connects to the board TXD signal.
s1_uart_rxd_i	Input	1	UART receive input. Connects to the board RXD signal.
QSPI Flash			
SPI_CLK	Output	1	Serial clock to SPI Flash
SPI_CSS	Output	1	SPI Flash chip select
SPI_MOSI	Input/Output	1	Serial data between FPGA and external SPI Flash
SPI_MISO	Input/Output	1	Serial data between FPGA and external SPI Flash
SPI_D2	Input/Output	1	Serial data between FPGA and external SPI Flash
SPI_D3	Input/Output	1	Serial data between FPGA and external SPI Flash
LPDDR4 MC			
ddr_ca_o	Output	6	LPDDR4 command/address
ddr_ck_o	Output	1	LPDDR4 clock
ddr_cke_o	Output	1	LPDDR4 clock enable
ddr_cs_o	Output	1	LPDDR4 chip select
ddr_dmi_io	Input/Output	4	LPDDR4 data mask
ddr_dq_io	Input/Output	32	LPDDR4 Data
ddr_dqs_io	Input/Output	4	LPDDR4 data strobe
ddr_reset_n_o	Output	1	External Memory chip reset signal
init_done_o	Output	1	Connects to LED for status check
irq_o	Output	1	Connects to LED for status check
pll_lock_o	Output	1	Connects to LED for status check
sclk_o	Output	1	Connects to LED for status check
trn_err_o	Output	1	Connects to LED for status check
SGMII PCS/GbE			
ser_rx_i	Input	1	SGMII PCS receive data from SFP
ser_tx_o	Output	1	SGMII PCS transmit data from SFP
an_link_ok_o	Output	1	Connects to LED to check Ethernet auto-negotiation status
Multi-Boot			
config_active_o	Output	1	Connects to LED to check Multi-Boot Configuration Block

4. Software Components

The GSRD (Golden System Reference Design) is enabled by RISC-V core based FreeRTOS and Lattice FPGA IP modules. Lattice developed BSP (Board Support Package) drivers in C language act as intermediaries, facilitating communication between the hardware elements on the FPGA and FreeRTOS software. During boot up, these drivers initialize and configure FPGA peripherals to establish effective coordination with the RISC-V processor.

4.1. Primary and Golden Bootloader

Bootloader is a bare-metal program that does the following IP configurations:

- Configures the GPIO and UART IP
- For LPDDR4 MC configuration, it first reads if the PLL Lock status is set and then initiates Memory Training and waits until training is complete. Configures the QSPI flash controller to read the application software stored into external flash while FIFO is disabled using API and copies it into LPDDR4. It then calculates the CRC value on the entire FW copied into DDR using crc16_ccit() API and compares the value with the original CRC value.
- Failure of the CRC check on the Primary application software triggers reconfiguration of the FPGA with the Golden FPGA bitstream image. This is done by triggering Multi-Boot Configuration module. This step is only used in Primary GSRD system.

4.2. Primary and Golden Application

- Executed by RISC-V RX CPU core from LPDDR4 where it initializes the BSP and Operating System
- Configures the TSE MAC handler with 1G Ethernet address, speed mode, MAC upper and lower. Passes TSE Core object to initialize the Ethernet IP and sets the MAC address.
- Creates an Ethernet frame using the API provided in ethernet_frame.h.
- For the SFP PHY initialization, it configures the PHY using i2c_phy_init() API by writing the configuration into PHY Status Register. It then performs a PHY reset before starting the TSE packet traffic. The PHY configuration checks if the Marvell 88E1111 PHY is attached before performing the configuration.
- I2c_linkup_status() API checks if auto-negotiation is established on the Ethernet port to receive or transmit Ethernet packets.
- Configure the SGDMA IP for MM2S to transfer the Ethernet frame to the TSE MAC and set up S2MM to receive packets from the TSE MAC at 2-second intervals. Runs FreeRTOS scheduler and Task Handler.

Ethernet RX packets displayed upon every incoming Ethernet packet

5. Design Constraints

The design constraints are divided into two parts, Pre-Synthesis (SDC) and Post-Synthesis Physical (PDC) constraints. They are used to ensure that the design meets the required performance, timing closure, functionality and physical placement requirements as per the FPGA device.

5.1. Clock Constraints

```
# Top level Clocks
create_clock -name {pll_refclk_i} -period 10 -waveform {0.000 5.000} [get_ports
pll_refclk_i]
create_clock -name {clk_125MHz} -period 8 -waveform {0.000 4.000} [get_ports clk_125MHz]
```

5.2. I/O Constraints

```
create_generated_clock -name {pll0_inst_clkop_o_net} -source [get_pins
pll0_inst/lscc_pll_inst/gen_no_refclk_mon.u_PLL.PLL_inst/REFCK] -divide_by 5 -
multiply_by 4 [get_pins pll0_inst/lscc_pll_inst/gen_no_refclk_mon.u_PLL.PLL_inst/CLKOP]
create_clock -name {pll_refclk_i} -period 10 -waveform {0.000 5.000} [get_ports
pll_refclk_i]
create_clock -name {clk_125MHz} -period 8 -waveform {0.000 4.000} [get_ports clk_125MHz]
create_clock -name {cdro_srclk} -period 8 [get_pins
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/genblk4.u_SGMIICDR.SGMIICDR_inst/SRCLK]
create_clock -name {fpga_config0_sys_clk} -period 35.5556 [get_pins
{fpga_config0_inst/fpga_config_inst/OSC_inst.OSC_inst/HFCLKOUT }]
create_clock -name {fpga_config0_inst_lfclock_out_net} -period 31250 [get_pins
{fpga_config0_inst/fpga_config_inst/OSC_inst.OSC_inst/LFCLKOUT }]
#GPIO
ldc_set_location -site {R9} [get_ports {s0_gpio[0]}]
ldc_set_location -site {U8} [get_ports {s0_gpio[1]}]
ldc_set_location -site {R7} [get_ports {s0_gpio[2]}]
ldc_set_location -site {R6} [get_ports {s0_gpio[3]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports {s0_gpio[*]}]
#STATUS LEDs
ldc_set_location -site {R4} [get_ports init_done_o]
ldc_set_location -site {R8} [get_ports pll_lock_o]
ldc_set_location -site {R5} [get_ports an_link_ok_o]
ldc_set_location -site {P8} [get_ports config_active_o]
#UART
ldc_set_location -site {M9} [get_ports s1_uart_txd_o]
ldc_set_location -site {L8} [get_ports s1_uart_rxd_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports s1_uart_rxd_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports s1_uart_txd_o]
#CLOCKS
ldc_set_port -iobuf {IO_TYPE=LVSTLD_I} [get_ports pll_refclk_i]
ldc_set_location -site {AB19} [get_ports pll_refclk_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports clk_125MHz]
ldc_set_location -site {P24} [get_ports clk_125MHz]
#RESET
ldc_set_location -site {N9} [get_ports rstn_i]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 PULLMODE=UP} [get_ports rstn_i]
#SGMII
ldc_set_port -iobuf {IO_TYPE=LVDS} [get_ports ser_tx_o]
```

```
ldc_set_location -site {U26} [get_ports ser_tx_o]
ldc_set_port -iobuf {IO_TYPE=LVDS} [get_ports ser_rx_i]
ldc_set_location -site {V24} [get_ports ser_rx_i]

#I2C
ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports sfp_disable_o]
ldc_set_location -site {T19} [get_ports sfp_disable_o]
ldc_set_location -site {U19} [get_ports I2C_SCL]
ldc_set_location -site {T18} [get_ports I2C_SDA]
#SPI flash
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 SLEWRATE=FAST} [get_ports SPI_CLK]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 SLEWRATE=FAST} [get_ports SPI_CSS]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 SLEWRATE=FAST} [get_ports SPI_MOSI]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 SLEWRATE=FAST} [get_ports SPI_MISO]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 SLEWRATE=FAST} [get_ports SPI_D2]
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 SLEWRATE=FAST} [get_ports SPI_D3]
ldc_set_location -site {H7} [get_ports SPI_MOSI]
ldc_set_location -site {K5} [get_ports SPI_D2]
ldc_set_location -site {H6} [get_ports SPI_MISO]
ldc_set_location -site {H4} [get_ports SPI_D3]
ldc_set_location -site {G6} [get_ports SPI_CLK]
ldc_set_location -site {G7} [get_ports SPI_CSS]
#LPDDR4 MC
#IO location assignment after back-annotation
#DQ GROUP
ldc_set_location -site {AA6} [get_ports {ddr_dq_io[0]}]
ldc_set_location -site {AA5} [get_ports {ddr_dq_io[1]}]
ldc_set_location -site {Y6} [get_ports {ddr_dq_io[2]}]
ldc_set_location -site {Y5} [get_ports {ddr_dq_io[3]}]
ldc_set_location -site {AE4} [get_ports {ddr_dq_io[4]}]
ldc_set_location -site {AD4} [get_ports {ddr_dq_io[5]}]
ldc_set_location -site {AD3} [get_ports {ddr_dq_io[6]}]
ldc_set_location -site {AC4} [get_ports {ddr_dq_io[7]}]
ldc_set_location -site {AD6} [get_ports {ddr_dq_io[8]}]
ldc_set_location -site {AE6} [get_ports {ddr_dq_io[9]}]
ldc_set_location -site {AC7} [get_ports {ddr_dq_io[10]}]
ldc_set_location -site {AB7} [get_ports {ddr_dq_io[11]}]
ldc_set_location -site {AA7} [get_ports {ddr_dq_io[12]}]
ldc_set_location -site {Y7} [get_ports {ddr_dq_io[13]}]
ldc_set_location -site {AB6} [get_ports {ddr_dq_io[14]}]
ldc_set_location -site {AC6} [get_ports {ddr_dq_io[15]}]
ldc_set_location -site {W10} [get_ports {ddr_dq_io[16]}]
ldc_set_location -site {W11} [get_ports {ddr_dq_io[17]}]
ldc_set_location -site {AD9} [get_ports {ddr_dq_io[18]}]
ldc_set_location -site {AE9} [get_ports {ddr_dq_io[19]}]
ldc_set_location -site {AD10} [get_ports {ddr_dq_io[20]}]
ldc_set_location -site {AE10} [get_ports {ddr_dq_io[21]}]
ldc_set_location -site {AA10} [get_ports {ddr_dq_io[22]}]
ldc_set_location -site {Y10} [get_ports {ddr_dq_io[23]}]
ldc_set_location -site {AC11} [get_ports {ddr_dq_io[24]}]
ldc_set_location -site {AD12} [get_ports {ddr_dq_io[25]}]
ldc_set_location -site {AD11} [get_ports {ddr_dq_io[26]}]
```

```
ldc_set_location -site {AE11} [get_ports {ddr_dq_io[27]}]
ldc_set_location -site {AB12} [get_ports {ddr_dq_io[28]}]
ldc_set_location -site {AA11} [get_ports {ddr_dq_io[29]}]
ldc_set_location -site {Y11} [get_ports {ddr_dq_io[30]}]
ldc_set_location -site {AA12} [get_ports {ddr_dq_io[31]}]
# DQS GROUP
ldc_set_location -site {AF4} [get_ports {ddr_dqs_io[0]}]
ldc_set_location -site {AF6} [get_ports {ddr_dqs_io[1]}]
ldc_set_location -site {AF10} [get_ports {ddr_dqs_io[2]}]
ldc_set_location -site {AF12} [get_ports {ddr_dqs_io[3]}]
# DMI GROUP
ldc_set_location -site {AE3} [get_ports {ddr_dmi_io[0]}]
ldc_set_location -site {AD5} [get_ports {ddr_dmi_io[1]}]
ldc_set_location -site {AB9} [get_ports {ddr_dmi_io[2]}]
ldc_set_location -site {AC12} [get_ports {ddr_dmi_io[3]}]
# CK, CKE, CS, CA
ldc_set_location -site {AA13} [get_ports {ddr_ck_o[0]}]
ldc_set_location -site {AD20} [get_ports {ddr_cke_o[0]}]
ldc_set_location -site {AC19} [get_ports {ddr_cs_o[0]}]
ldc_set_location -site {W13} [get_ports ddr_reset_n_o]
ldc_set_location -site {AE22} [get_ports {ddr_ca_o[0]}]
ldc_set_location -site {AE23} [get_ports {ddr_ca_o[1]}]
ldc_set_location -site {AD21} [get_ports {ddr_ca_o[2]}]
ldc_set_location -site {AE21} [get_ports {ddr_ca_o[3]}]
ldc_set_location -site {AF22} [get_ports {ddr_ca_o[4]}]
ldc_set_location -site {AF23} [get_ports {ddr_ca_o[5]}]
# Below are the IP constraints
# The ECLKDIV primitive already has constraint but the inferred name is not good.
# The constraint below only aims to assign a name to the generated clock.
create_generated_clock -name {eclk_w} -source [get_pins
*/lscc_lpddr4_mc_inst/u_lp4mem/u_pll/gen_no_refclk_mon.u_PLL.PLL_inst/REFCK] -divide_by
3 -multiply_by 16 [get_pins
{*/lscc_lpddr4_mc_inst/u_lp4mem/u_pll/gen_no_refclk_mon.u_PLL.PLL_inst/CLKOS }]
set_clock_uncertainty -setup 0.2 [get_clocks eclk_w]
create_generated_clock -name {clkop_o} -source [get_pins
*/lscc_lpddr4_mc_inst/u_lp4mem/u_pll/gen_no_refclk_mon.u_PLL.PLL_inst/REFCK] -divide_by
1 [get_pins {*/lscc_lpddr4_mc_inst/u_lp4mem/u_pll/gen_no_refclk_mon.u_PLL.PLL_inst/CLKOP
}]
set_clock_uncertainty -setup 0.2 [get_clocks clkop_o]
create_generated_clock -name {sclk_o} -source [get_pins
*/lscc_lpddr4_mc_inst/u_lp4mem/u1_clock_sync/u0_ECLKDIV.ECLKDIV_inst/ECLKIN] -divide_by
4 [get_pins {*/lscc_lpddr4_mc_inst/u_lp4mem/u1_clock_sync/u0_ECLKDIV.ECLKDIV_inst/DIVOUT
}]
set_clock_uncertainty -setup 0.2 [get_clocks sclk_o]
#####sgmii
create_generated_clock -name {clk_625m_pllo} -source [get_pins
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/gen_int_pll.genblk1.u_pll_125_625s/lscc_pll
_inst/u_PLL.PLL_inst/REFCK] -multiply_by 5 [get_pins
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/gen_int_pll.genblk1.u_pll_125_625s/lscc_pll
_inst/u_PLL.PLL_inst/CLKOS]
create_generated_clock -name {clk_125m_pll_o} -source [get_pins
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/gen_int_pll.genblk1.u_pll_125_625s/lscc_pll
_inst/u_PLL.PLL_inst/REFCK] -divide_by 1 [get_pins
```

```
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/gen_int_pll.genblk1.u_pll_125_625s/lscc_pll
_inst/u_PLL.PLL_inst/CLKOP]
create_generated_clock -name {usr_clk_o} -source [get_pins
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/genblk4.u_gddr/u0_ECLKDIV.ECLKDIV_inst/ECLK
IN] -divide_by 5 [get_pins
sgmii0_inst/lscc_sgmii_gbe_pcs_inst/u_serdes/genblk4.u_gddr/u0_ECLKDIV.ECLKDIV_inst/DIVO
UT]
# Constraints for Clock synchronization logic
# Group the double FF so that they will be placed in the same slice
set_false_path -from [get_clocks clk_125MHz] -to [get_clocks pll0_inst_clkop_o_net]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks clk_125MHz]
set_false_path -from [get_clocks usr_clk_o] -to [get_clocks pll0_inst_clkop_o_net]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks usr_clk_o]
set_false_path -from [get_clocks sclk_o] -to [get_clocks usr_clk_o]
set_false_path -from [get_clocks sclk_o] -to [get_clocks clk_125MHz]
set_false_path -from [get_clocks clk_125MHz] -to [get_clocks sclk_o]
set_false_path -from [get_clocks usr_clk_o] -to [get_clocks sclk_o]
set_false_path -from [get_clocks usr_clk_o] -to [get_clocks clk_125m_pll_o]
set_false_path -from [get_clocks clk_125m_pll_o] -to [get_clocks usr_clk_o]
set_false_path -from [get_clocks cdro_srclk] -to [get_clocks usr_clk_o]
set_false_path -from [get_clocks usr_clk_o] -to [get_clocks cdro_srclk]
set_false_path -from [get_clocks usr_clk_o] -to [get_clocks clk_125MHz]
set_false_path -from [get_clocks clk_125MHz] -to [get_clocks usr_clk_o]
set_false_path -from [get_clocks clk_125m_pll_o] -to [get_clocks sclk_o]
set_false_path -from [get_clocks sclk_o] -to [get_clocks clk_125m_pll_o]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks clk_125m_pll_o]
set_false_path -from [get_clocks clk_125m_pll_o] -to [get_clocks pll0_inst_clkop_o_net]
set_false_path -from [get_clocks clk_125m_pll_o] -to [get_clocks cdro_srclk]
set_false_path -from [get_clocks cdro_srclk] -to [get_clocks clk_125m_pll_o]
set_false_path -from [get_clocks cdro_srclk] -to [get_clocks sclk_o]
set_false_path -from [get_clocks sclk_o] -to [get_clocks cdro_srclk]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks cdro_srclk]
set_false_path -from [get_clocks cdro_srclk] -to [get_clocks pll0_inst_clkop_o_net]
set_false_path -from [get_clocks clk_625m_pllo] -to [get_clocks usr_clk_o]
set_false_path -from [get_clocks usr_clk_o] -to [get_clocks clk_625m_pllo]
set_false_path -from [get_clocks clk_125m_pll_o] -to [get_clocks clk_625m_pllo]
set_false_path -from [get_clocks clk_625m_pllo] -to [get_clocks clk_125m_pll_o]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks sclk_o]
set_false_path -from [get_clocks sclk_o] -to [get_clocks pll0_inst_clkop_o_net]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks
fpga_config0_sys_clk]
set_false_path -from [get_clocks fpga_config0_sys_clk] -to [get_clocks
pll0_inst_clkop_o_net]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks
fpga_config0_inst_lfclock_out_net]
set_false_path -from [get_clocks fpga_config0_inst_lfclock_out_net] -to [get_clocks
pll0_inst_clkop_o_net]
set_false_path -from [get_clocks pll0_inst_clkop_o_net] -to [get_clocks eclk_w]
set_false_path -from [get_clocks eclk_w] -to [get_clocks pll0_inst_clkop_o_net]

#set_false_path -from [get_clocks clk_125MHz] -to [get_clocks pll0_inst_clkos_o_net]
#set_false_path -from [get_clocks pll0_inst_clkos_o_net] -to [get_clocks clk_125MHz]
#set_false_path -from [get_clocks usr_clk_o] -to [get_clocks pll0_inst_clkos_o_net]
```

```
#set_false_path -from [get_clocks pll0_inst_clkos_o_net] -to [get_clocks usr_clk_o]
#set_false_path -from [get_clocks pll0_inst_clkos_o_net] -to [get_clocks clk_125m_pll_o]
#set_false_path -from [get_clocks clk_125m_pll_o] -to [get_clocks pll0_inst_clkos_o_net]
#set_false_path -from [get_clocks pll0_inst_clkos_o_net] -to [get_clocks cdro_srclk]
#set_false_path -from [get_clocks cdro_srclk] -to [get_clocks pll0_inst_clkos_o_net]
#set_false_path -from [get_clocks pll0_inst_clkos_o_net] -to [get_clocks sclk_o]
#set_false_path -from [get_clocks sclk_o] -to [get_clocks pll0_inst_clkos_o_net]
ldc_set_sysconfig {JTAG_PORT=ENABLE PROGRAMN_PORT=ENABLE CONFIG_IOSLEW=FAST
MASTER_SPI_PORT=DISABLE DONE_PORT=DISABLE INITN_PORT=DISABLE MCCLK_FREQ=28.1
BOOTMODE=DUAL CONFIGIO_VOLTAGE_BANK1=NOT_SPECIFIED CONFIGIO_VOLTAGE_BANK0=NOT_SPECIFIED}
```

6. Resource Utilization

Figure 6.1 shows the GSRD resource utilization and Table 6.1 shows the total LUT4, PFU register, I/O buffer, and EBR resource utilization for a CertusPro-NX device.

	LUT4 Logic	LUT4 Distributed RAM	LUT4 Ripple Logic	PFU Registers	IO Registers	IO Buffers	DSP MULT	EBR
soc_primary_gsrd	35512(3)	6954(0)	5992(0)	34198(4)	10(0)	81(18)	6(0)	162(0)
apb_interconnect0_inst	88(0)	0(0)	0(0)	7(0)	0(0)	0(0)	0(0)	0(0)
axi2apb0_inst	243(0)	0(0)	42(0)	197(0)	0(0)	0(0)	0(0)	0(0)
axi4_interconnect0_inst	5980(0)	3738(0)	420(0)	6201(0)	0(0)	0(0)	0(0)	0(0)
axi4_interconnect2_inst	1842(0)	1056(0)	102(0)	2743(0)	0(0)	0(0)	0(0)	0(0)
axi4_regslice0_inst	152(1)	0(0)	0(0)	284(0)	0(0)	0(0)	0(0)	0(0)
cpu_inst	5055(1)	72(0)	1304(0)	3412(0)	0(0)	0(0)	6(0)	18(0)
fpga_config0_inst	57(0)	0(0)	0(0)	55(0)	0(0)	0(0)	0(0)	0(0)
i2c0_inst	456(0)	24(0)	126(0)	515(0)	0(0)	2(0)	0(0)	0(0)
lpddr4_mc_contr0_inst	10495(0)	1488(0)	1452(0)	9077(0)	1(0)	49(0)	0(0)	33(0)
mpmc0_inst	2791(1)	576(0)	452(0)	4198(0)	0(0)	0(0)	0(0)	34(0)
pll0_inst	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
qspi0_inst	3262(0)	0(0)	408(0)	2360(0)	0(0)	4(0)	0(0)	0(0)
rst_sync0_inst	45(1)	0(0)	34(0)	36(0)	0(0)	0(0)	0(0)	0(0)
s0_apb_gpio_inst	113(0)	0(0)	0(0)	89(0)	8(0)	8(0)	0(0)	0(0)
s1_apb_uart_inst	208(0)	0(0)	46(0)	146(0)	1(0)	0(0)	0(0)	0(0)
sgdma0_inst	1522(0)	0(0)	600(0)	2014(0)	0(0)	0(0)	0(0)	8(0)
sgmii0_inst	1111(0)	0(0)	272(0)	876(0)	0(0)	0(0)	0(0)	1(0)
sysmem0_inst	524(0)	0(0)	114(0)	334(0)	0(0)	0(0)	0(0)	64(0)
tse_mac0_inst	1565(0)	0(0)	620(0)	1650(0)	0(0)	0(0)	0(0)	4(0)

Figure 6.1. CertusPro-NX GSRD Resource Utilization

Table 6.1. GSRD Total Resource Utilization

Resource	Usage	Percentage Utilization
LUT4 (Logic + Distributed RAM + Ripple Logic)	48458	60.66%
PFU Register	34198	42.81%
I/O Buffers	81	—
EBR	162	77%

7. Demo User Guide

7.1. Boot-Up Sequence

This section describes the RISC-V RX CPU boot up sequence that configures IP drivers, operating modes, bootloader and FreeRTOS application software execution.

Here is the description of the terms used throughout the document:

- Boot Up – Process of starting the RISC-V RX CPU, loading the FreeRTOS application software from external SPI Flash into the LPDDR4 memory and executing the application.
- Bootloader – Code that initializes and configures various peripherals and loads the FreeRTOS application software into LPDDR4 memory. It also checks for the CRC of the copied application and decides whether to execute the application software or load the next best bitstream hardware and corresponding software.
- FreeRTOS – Application software that is loaded into LPDDR4 Memory and executed by RISC-V CPU at the end of boot up process.
- SPI Flash – Non-volatile external memory that stores the FreeRTOS application software and multi-boot MCS bitstream.

The following is the boot up sequence shown in Figure 7.1:

- The system provides similar functionality both in Bare Metal and FreeRTOS mode. For the demo, the Golden and Primary application software binaries and their FPGA bitstream images are stored in external SPI Flash before the boot up sequence is initiated.
- The initial bootloader is a part of the internal system memory ROM embedded into the FPGA bitstream stored inside the external SPI Flash. Upon power-on boot up, the bootloader configures the peripherals and GSRD building blocks such as UART, GPIO, I^2C, SGDMA, 1G TSE MAC, SGMII PCS, LPDDR4 and QSPI Controller.
- The bootloader loads the bitstream from the SPI Flash to program the SRAM of the FPGA and fetches the respective application software (Primary) through the QSPI flash controller.
- As the application software needs to be executed from the external memory, the RISC-V module loads it into LPDDR4 Memory controller.
- This application software is stored at the beginning of the LPDDR4 Memory. After the application software is loaded, the RISC-V CPU calculates the CRC of the application code in LPDDR4.
- The calculated LPDDR4 CRC is compared with the original CRC, that is a part of SPI Flash.
  - If the condition matches, the RISC-V CPU jumps to FreeRTOS execution from LPDDR4 memory
  - If the condition mismatches, the RISC-V CPU issues a FPGA REFRESH command to load the Golden bitstream and application software from SPI Flash.
- Upon the execution of the correct Primary or Golden application software from LPDDR4, the building blocks mentioned earlier are up and running with their associated drivers. For example, if any Ethernet data is expected to arrive, the RISC-V CPU sets up the SGDMA IP accordingly with address, data length and other configuration modes to successfully route the incoming Ethernet packets.
- When the Ethernet frame is received by the TSE MAC IP, it forwards it to SGDMA to transfer the data to its destination based on parameters set. The endpoint in this case is the main memory LPDDR4.
- You can also choose to store the data into another system memory based on SGDMA configuration.
- For outgoing data, data is fetched from a location inside LPDDR4, and SGDMA transfers the data to TSE MAC for transmission outside the FPGA.
- When no data activity occurs over Ethernet, the RISC-V CPU continues running its tasks in the usual manner based on the loaded software execution.

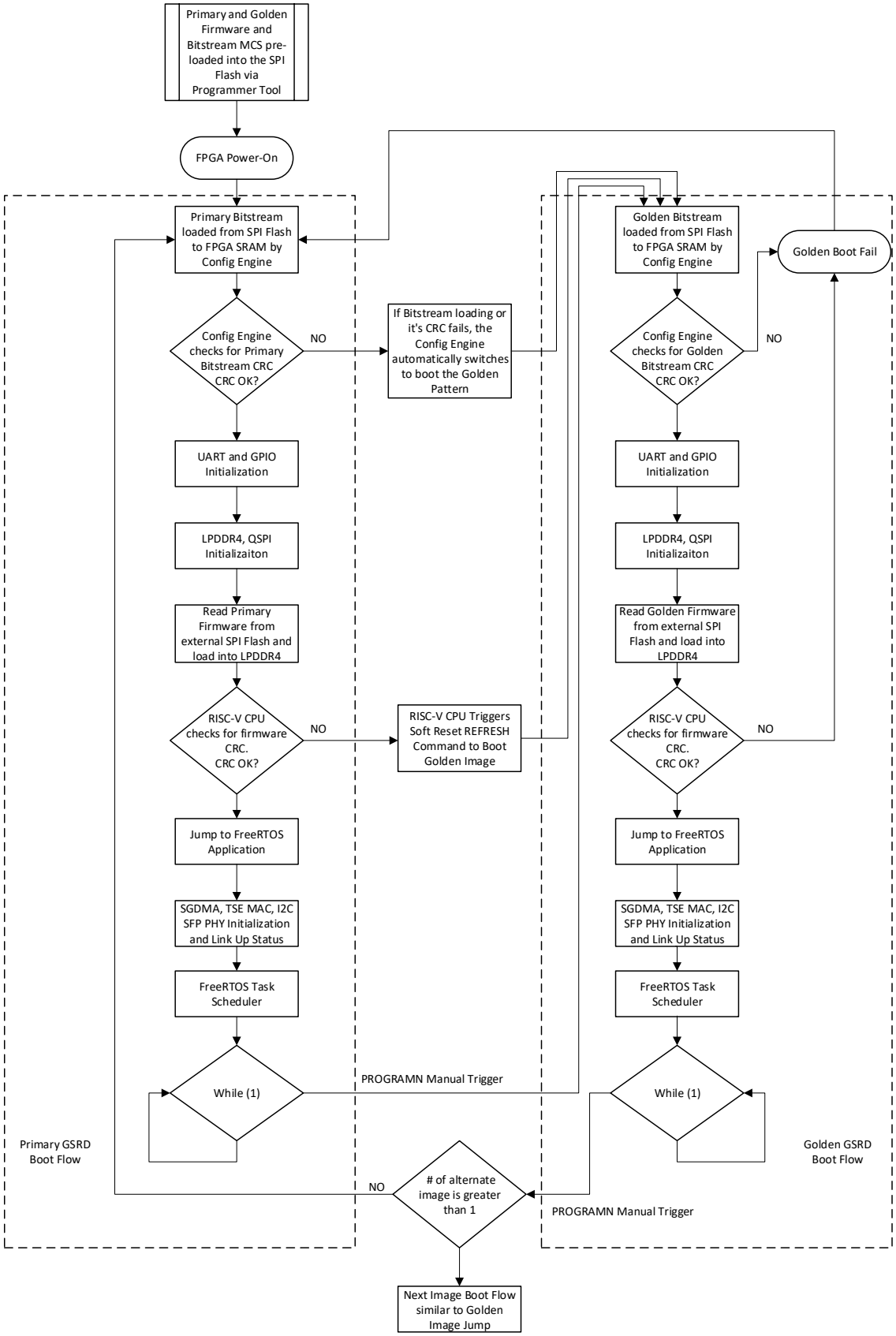

Figure 7.1 GSRD Boot-Up Sequence

7.2. Prerequisites

The following sections show the hardware and software requirements to execute the GSRD demonstration.

7.2.1. Software Requirements

- Lattice Propel 2024.1 Package – contains both Lattice Propel SDK and Lattice Propel Builder
  - Download here: Lattice Propel 2024.1
- Lattice Radiant 2024.1.1 Package – contains IP Packager, Radiant Software, QuestaSim, and Programmer
  - Download here: Lattice Radiant 2024.1.1
- Lattice Propel 2024.1 Patch for CertusPro-NX Golden System Reference Design
  - Download here: Downloadable Software tab in the GHRD/GSRD Reference Design

7.2.2. CertusPro-NX Requirements

7.2.2.1. Hardware Requirements

The section describes the hardware needed to run the GSRD demonstration.

- Lattice CertusPro-NX Versa Evaluation Board
- USB Type-A UART cable for programming the bitstream, firmware and proper terminal prints
- Electrical 1G SFP(s) Model FS SFP-GB-GE-T for Ethernet connection on the CertusPro-NX board
- Ethernet cable to connect CertusPro-NX board to the Host PC
- 12 V power adapter for board power

7.2.2.2. Hardware Setup

This section provides the procedure for setting up the CertusPro-NX Versa Evaluation board as shown in Figure 7.2.

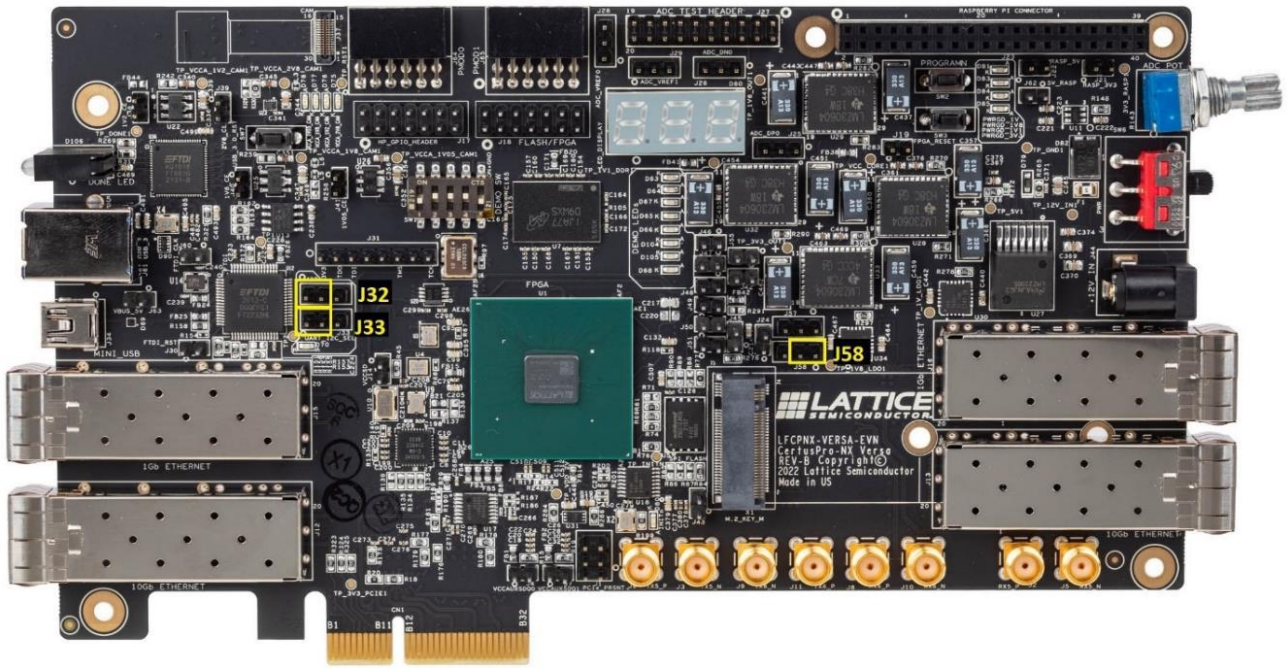

Figure 7.2. CertusPro-NX Versa Evaluation Board

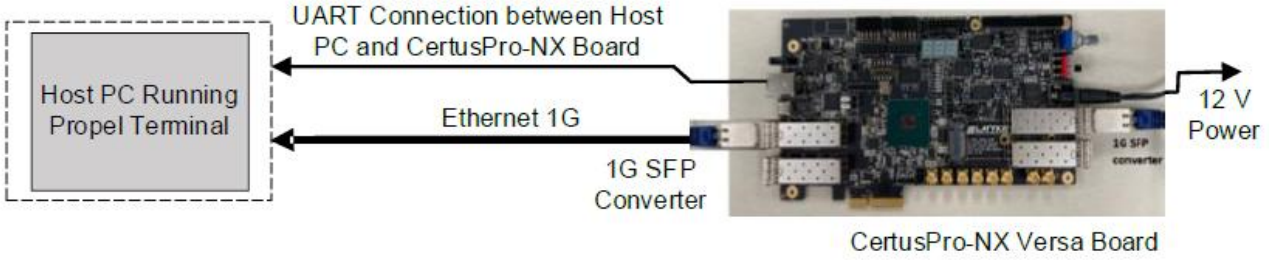

Figure 7.3. Connections and Buttons needed for Demonstration

To setup the Lattice CertusPro-NX Versa Evaluation board for GSRD demonstration:

1. Connect the 12 V power adapter.
2. Connect the Mini-USB Type-A cable from PC to J34.
3. Connect jumpers on Pin 1 and Pin 2 on both JP32 and JP33 switches to enable UART.
4. Connect/Insert the Ethernet SFP onto J15 connector.
5. Connect the Ethernet RJ45 cable from the host PC cable to SFP port.
6. Turn on switch PWR switch.

7.2.2.3. Executables

This section provides the directory structure, file names and locations of the executables (SPI Flash) required for running the GSRD demonstration.

- Download the design package from the Lattice Semiconductor website.
  - Go to *Design File* in the GHRD/GSRD Demonstration, download the CertusPro-NX Golden System Reference Design and Demo V2.0 – Bitstream file.
- Unzip the .zip file to your local directory, for example to *<C:\user_workspace>.*
- The extracted directory has the following executables listed in Table 7.1.

Below are the bitstreams and the software image binaries for programming the FPGA.

Table 7.1. Executable Files for Winbond Flash

File Description	File Name	Starting Address in SPI Flash
Primary Software with CRC	c_primary_appcrc.bin	0x028A 0000
Primary Software without CRC	c_primary_app.bin	0x028A 0000
Primary FPGA Bitstream	soc_primary_system.bit	0x0000 0000
Golden Software with CRC	c_golden_appcrc.bin	0x0280 0000
Golden Software without CRC	c_golden_app.bin	0x0280 0000
Golden FPGA Bitstream	soc_golden_system.bit	0x0000 0000
Multi-Boot MCS File (Golden + Primary Bitstream)	multiboot_system.mcs	0x0000 0000

Table 7.2. Flash Devices Supported on CertusPro-NX Boards

Board	Flash Device
CertusPro-NX Versa Board	Winbond W25Q512JV
CertusPro-NX Versa Board	Macronix MX25L51245G

7.3. Implementing the GHRD/GSRD Demo

This section describes the procedure for running the GSRD/GHRD demo using the pre-built executables and binary files in the design package.

7.3.1. Setting up the UART Terminal

The software code during the GSRD demonstration displays messages on the terminal through the UART interface.

To setup the UART terminal:

1. Connect the Lattice CertusPro-NX Versa Evaluation board to the PC/Laptop using USB Type-A UART cable.
2. Open Propel SDK 2024.1 tool.
3. Double-click on the terminal button shown in Figure 7.4.

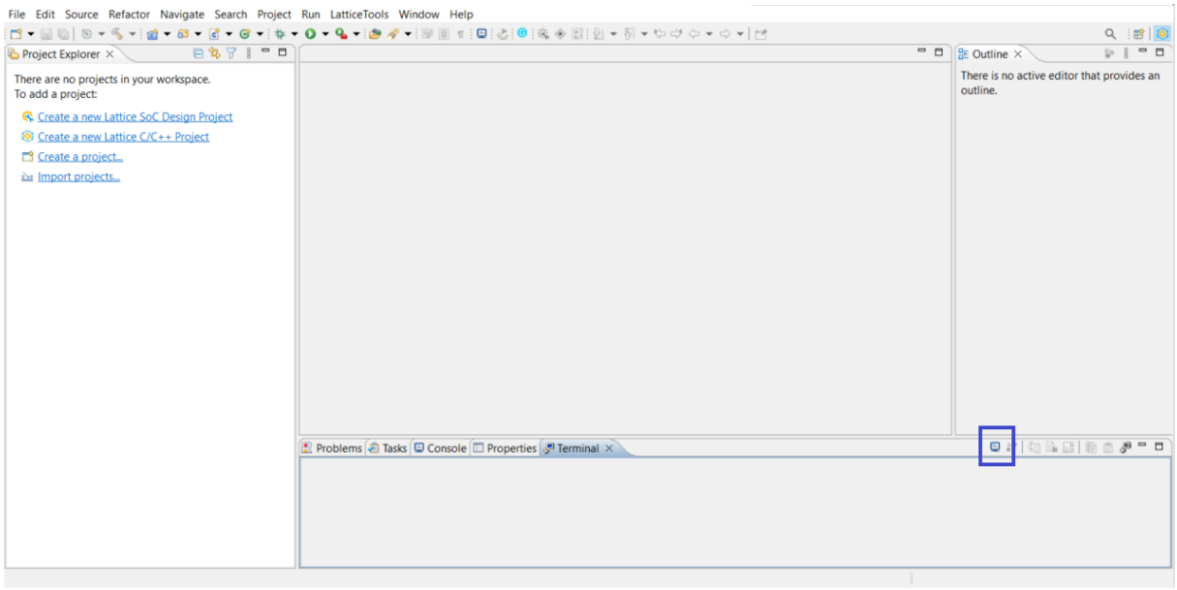

Figure 7.4. UART Terminal Icon on Propel SDK Window

4. **Choose Terminal: Serial Terminal** as shown in Figure 7.5. In Serial port dropdown list, select the last COM in the list as shown in Figure 7.6.
   Note: This detail can also be found under the Ports (COM and LPT) section in your local PC, under Device Manager. The COM port number can be different. If a USB port does not work, try a different USB port.

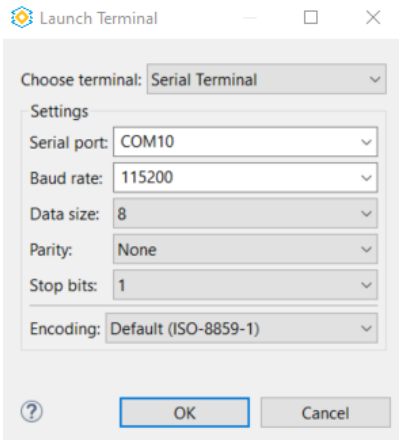

Figure 7.5. UART Launch Terminal Window

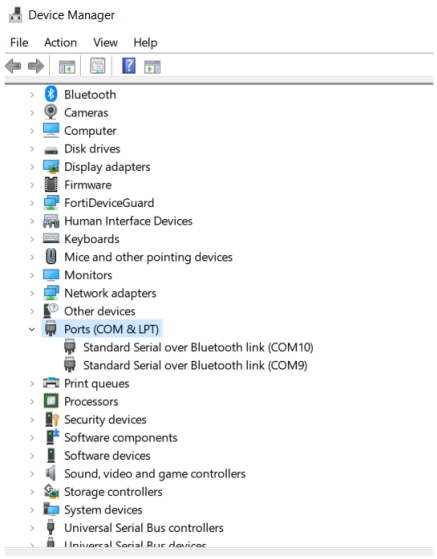

Figure 7.6. Device Manager Window on PC

5. Set Baud rate: **115200**.
6. Click **OK**.

7.3.2. Setting up the Non-Volatile Memory Register

For the GSRD design on Lattice CertusPro-NX Versa Evaluation Boards, you need to ensure the settings of the One-Time-Programmable Non-Volatile Configuration Memory. You may skip this section if you have already done this step before. Otherwise, perform a one-time step by JTAG to modify the default MSPI addressing mode from 24-bit to 32-bit. This is necessary for the multi-boot feature to function properly.

7.3.3. Programming the Standalone Golden or Primary GSRD Bitstream and Application Software

1. Connect the CertusPro-NX Versa Evaluation Board to a PC/laptop using USB cable as per the hardware setup mentioned in Hardware Setup section. Make sure the JP32 and JP33 are connected properly.
2. Power-on the board.
3. Launch Lattice Programmer tool. In the Getting Started dialog box, select **Create a new blank project**. Browse to the Project Location on your local machine. In this case, it can be the same folder as downloaded executables folder.

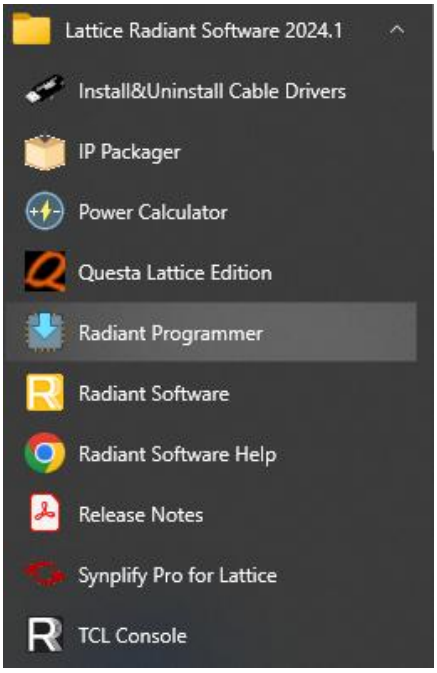

Figure 7.7. Launch Radiant Programmer from Windows Start

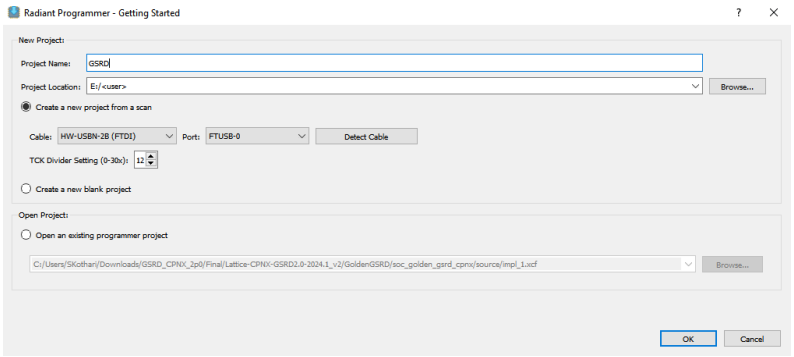

Figure 7.8. Radiant Programmer Start Window

4. Click **OK**.

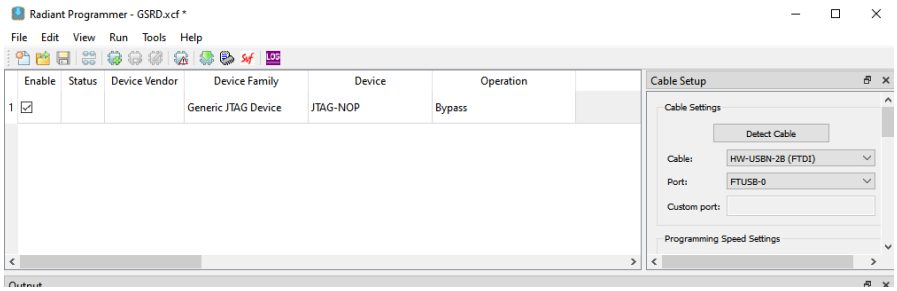

Figure 7.9. Radiant Programmer .xcf Window

5. If the Device Family shows as Generic JTAG Device, click **Scan Device** as shown in Figure 7.10 to update the Device Family information automatically.

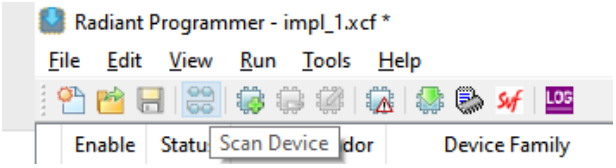

Figure 7.10. Scan Device Icon on Radiant Programmer

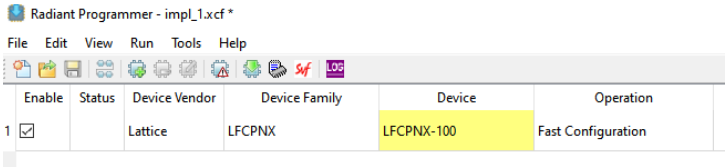

Figure 7.11. Select Device for Programming

6. Click on the highlighted item under the **Device** field to un-highlight it as shown in Figure 7.12.

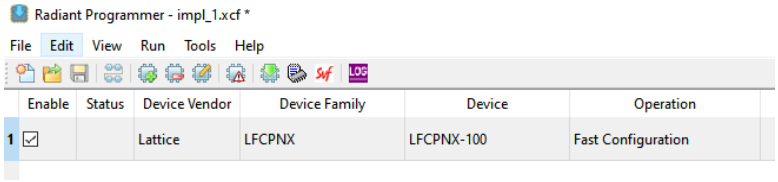

Figure 7.12. Device Selected for Programmer

7. Double-click on the **Operation** tab or right-click and select **Device Properties**.

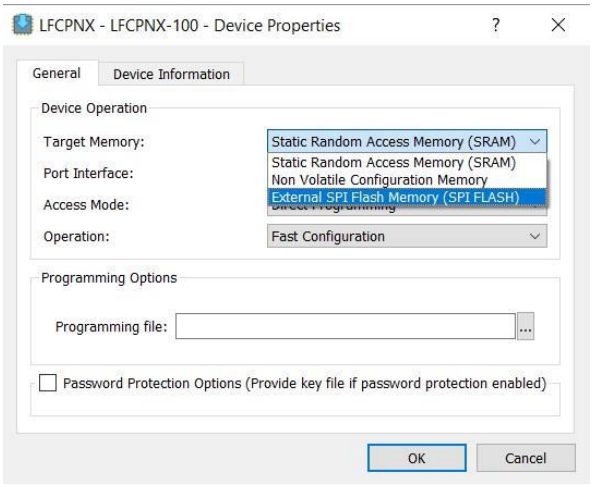

Figure 7.13. Select the Target Memory for Programming

8. Before programming, you need to erase the entire SPI Flash Memory by applying below settings.

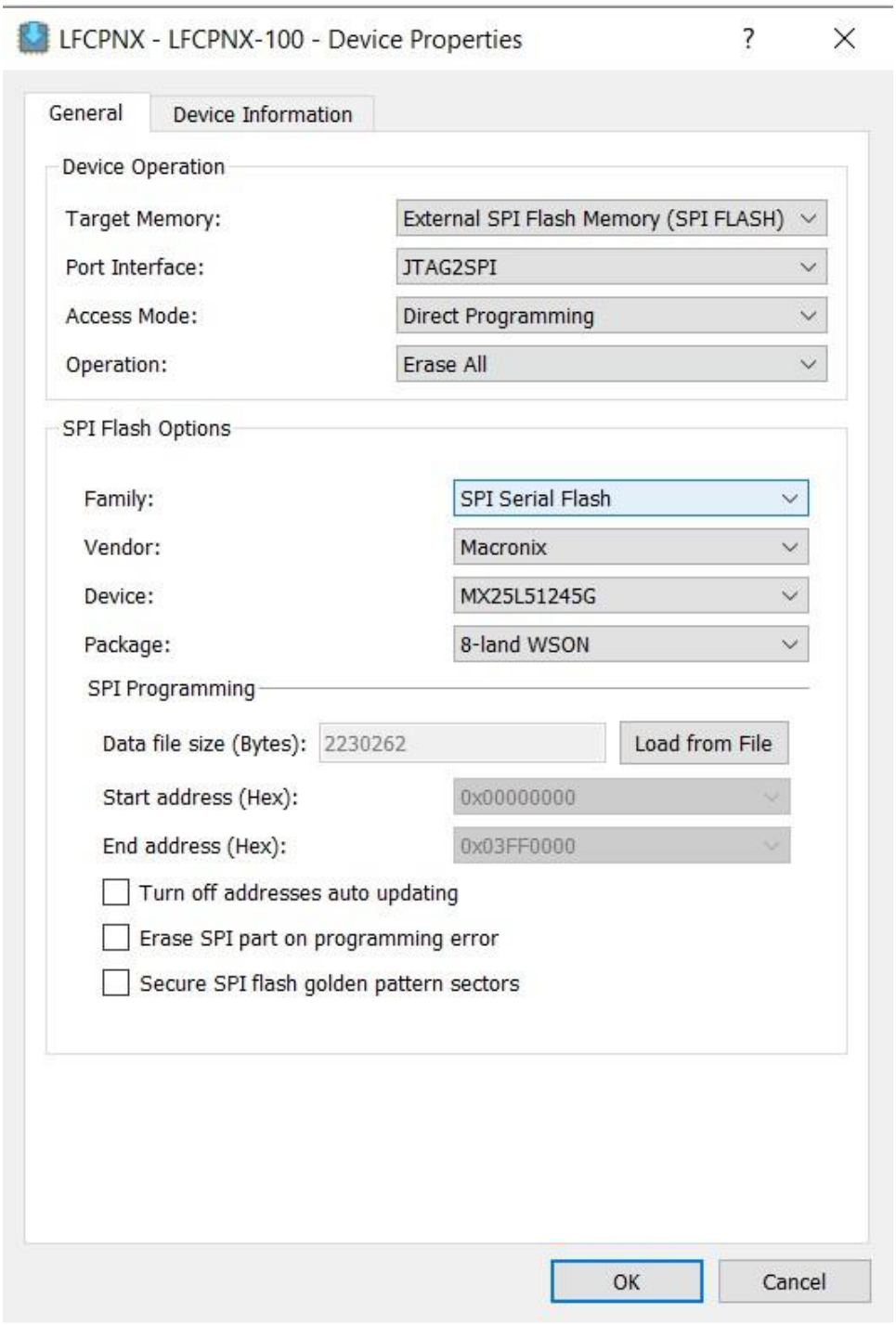

Figure 7.14. Device Properties to Erase the Macronix SPI Flash

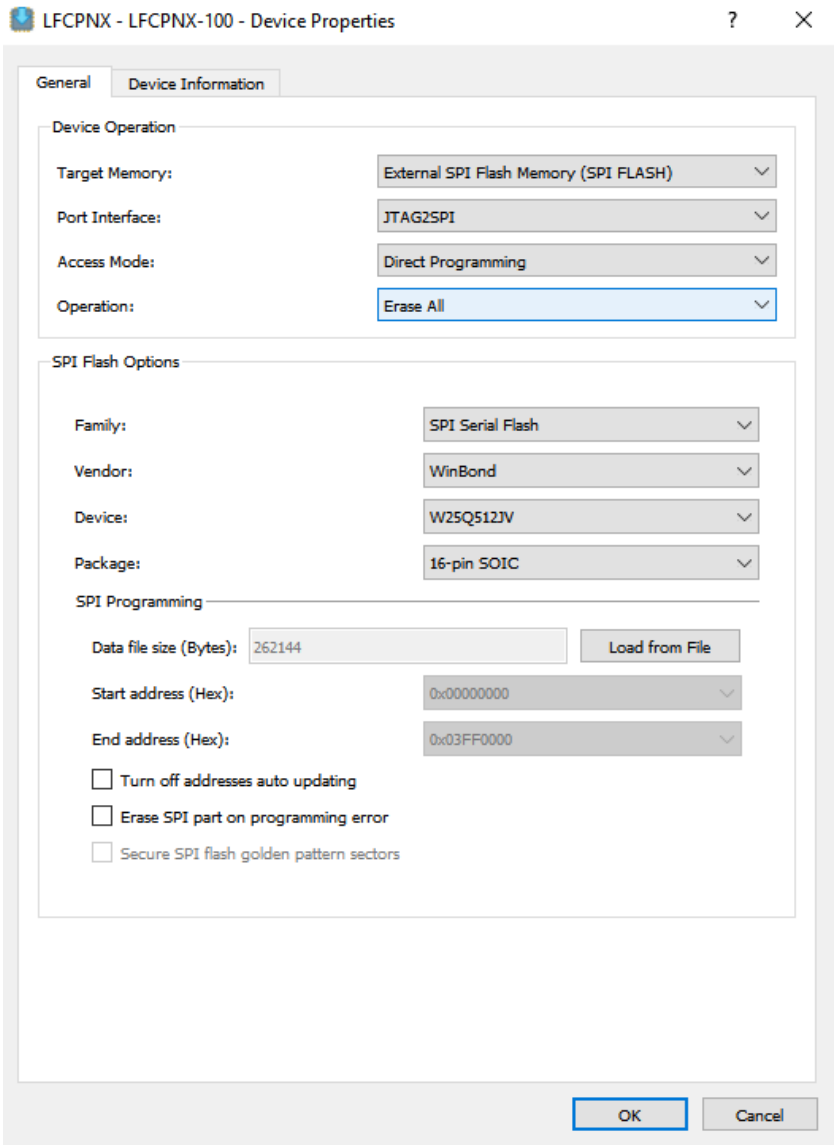

Figure 7.15. Device Properties to Erase the Winbond SPI Flash

9. Click **OK** and click on the Program Device Icon or the menu item, **Run > Program Device**. This erases the entire Flash Memory. Wait for the process to complete.

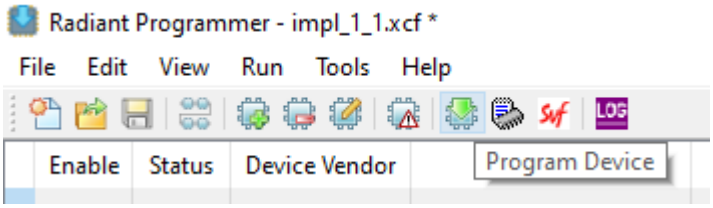

Figure 7.16. Program Button to Program the SPI Flash

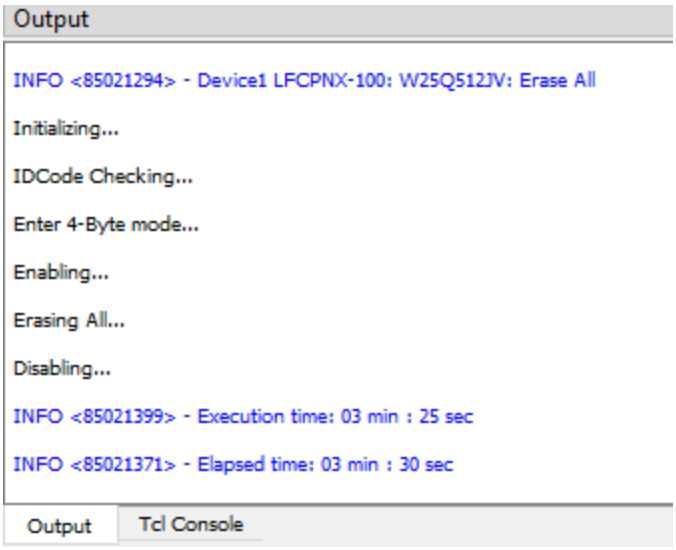

Figure 7.17. Output After Erase All

10. Power cycle the CertusPro-NX Versa Evaluation Board.
11. Erase the FPGA SRAM. Click **OK**.

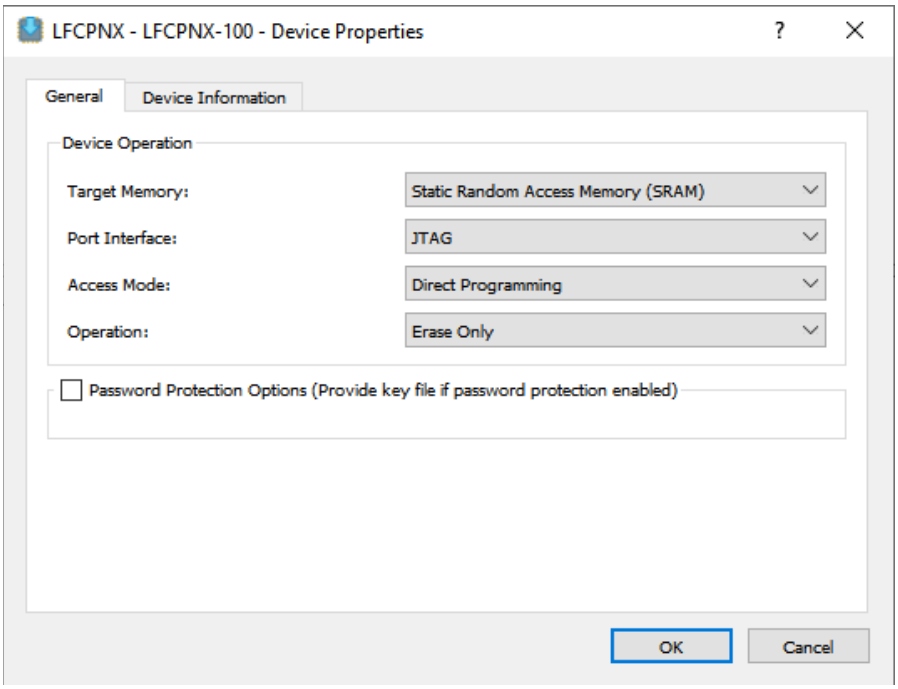

Figure 7.18. Erase Only Operation for SRAM Programming

12. Skip steps 14 to 18 if you have performed the steps before. Otherwise, perform a one-time step required by JTAG to Program NV Register 1 to modify the default SPI Addressing and Command mode from 24 bits to 32 bits. You need to this only once for your board. Subsequent programming does not need to program NV Register 1 again.

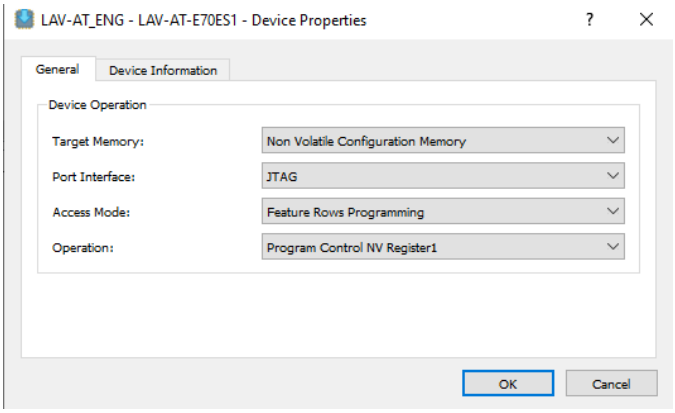

Figure 7.19. One-Time Programmable Control NV Register1

13. Click **OK** and click the **Program Device** Icon or go to the menu item, **Run > Program Device**.
14. Change bit 0 to 1 for 32-bit MSPI Address and 32-bit MSPI Commands as shown in Figure 7.20. For changing these bits, click once on the 0 values in the Chip Value row.

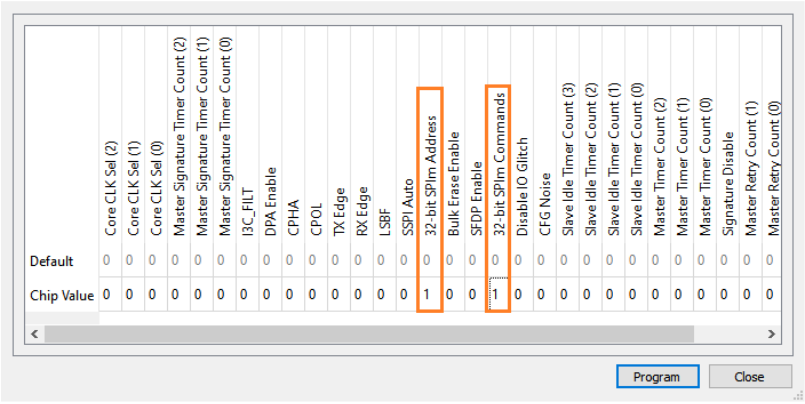

Figure 7.20. Settings to Select Chip Value

Note: Update the value of the two highlighted fields. NV Register 1 is an OTP (One-Time-Programmable) Register.

15. Click **Program**.
16. Power cycle the CertusPro-NX Versa Evaluation Board.
17. To program the **c_golden_appcrc.bin** file into the SPI Flash, Follow the settings as shown in Figure 7.21.

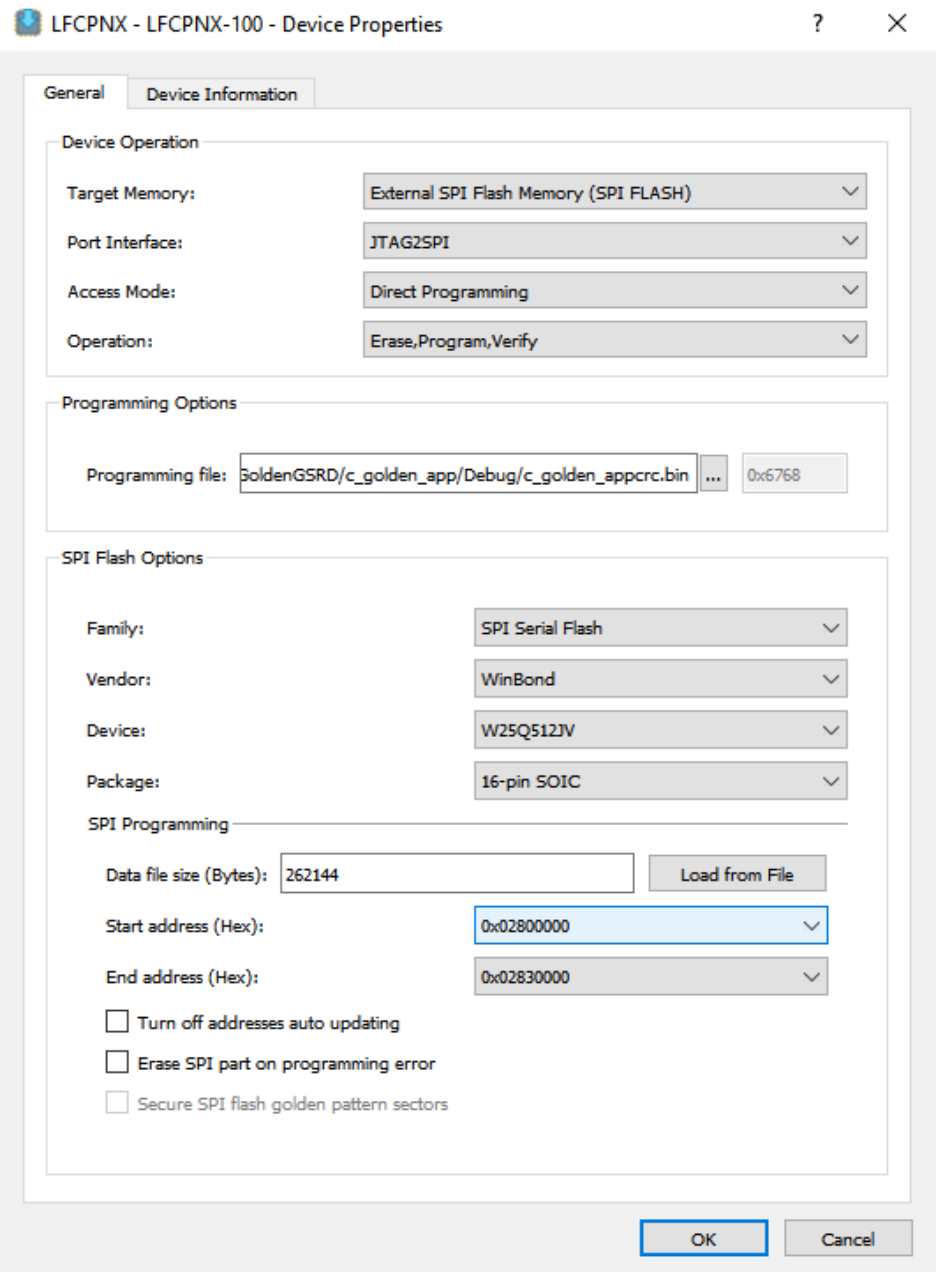

Figure 7.21. Device Properties to Program the Winbond SPI Flash

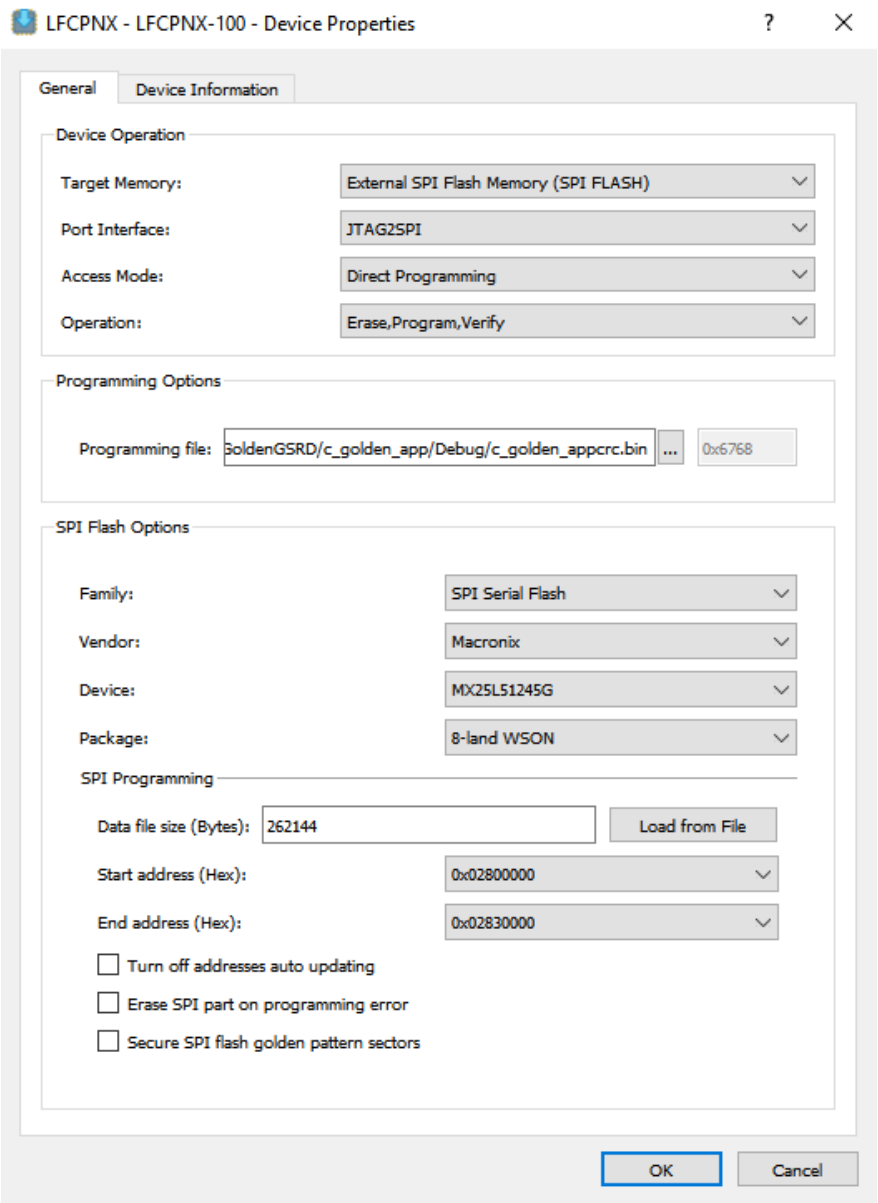

Figure 7.22. Device Properties to Program the Macronix SPI Flash

18. Click **OK**
19. Use TCK Divider Setting (0-30x) to **4**.

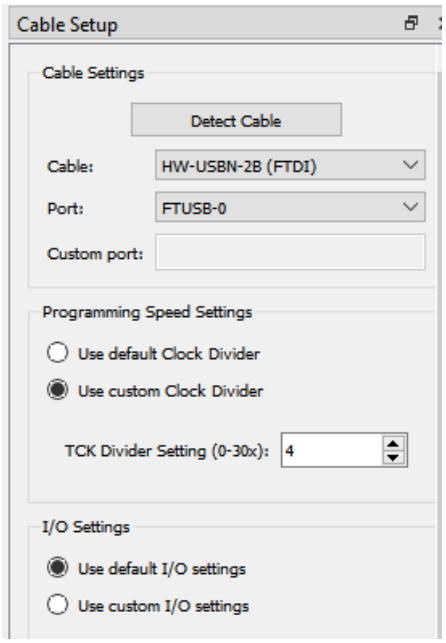

Figure 7.23. Cable Settings for Device Programming

20. Click the **Program Device** Icon or go to the menu item, **Run > Program Device**. The output console displays the Operation Successful message.

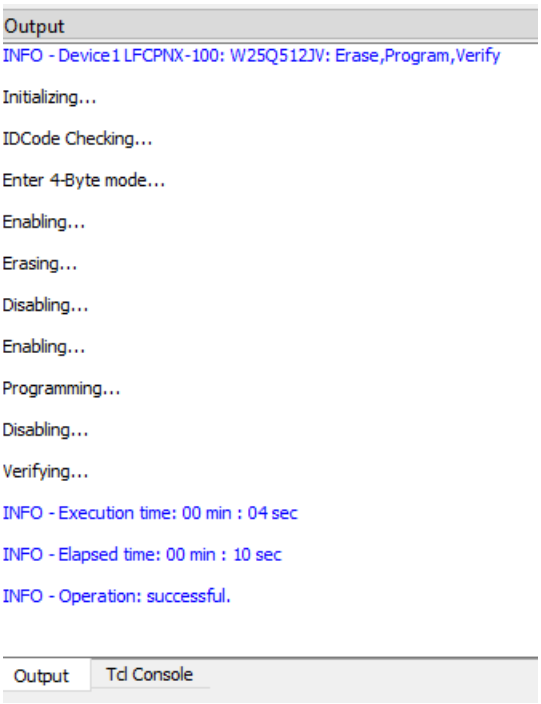

Figure 7.24. Radiant Programmer Console Output after Programming the SPI Flash

21. Power cycle the CertusPro-NX Versa Board.
22. To program the FPGA Golden GSRD bitstream, double-click on the **Operation** tab to update the selections as shown in Figure 7.25. Make sure to provide the path to the .bit file location on your local machine where you have unzipped the executables.

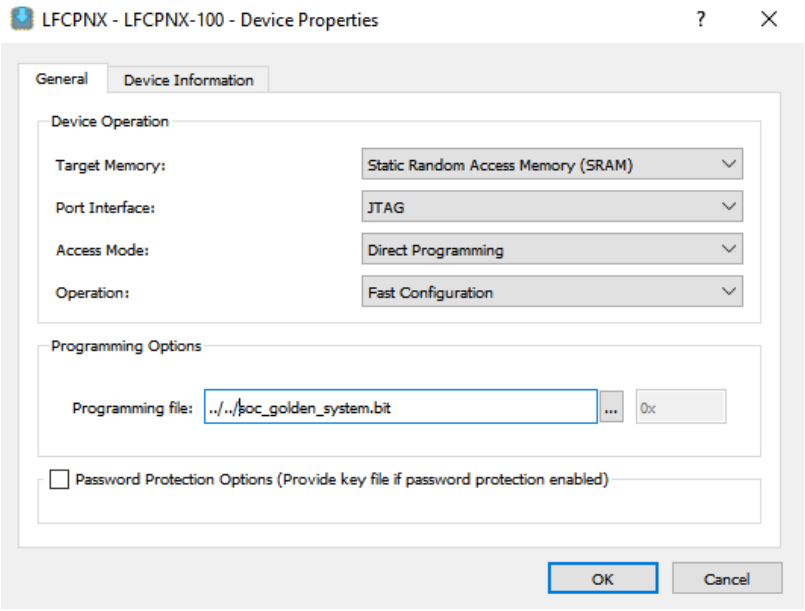

Figure 7.25. Device Properties to Program the FPGA Bitstream in SRAM

23. Click **OK** and Click the **Program Device** Icon or go to the menu item**, Run > Program Device**. Wait until the operation is successful as shown in Figure.

```
INFO <85021074> - Check configuration setup: Start.
INFO <85021076> - JTAG Chain Verification. No Errors.
INFO <85021078> - Check configuration setup: Successful.
INFO <85021278> - Device1 LFCPNX-100: Fast Configuration
INFO <85021298> - Operation Done. No errors.
INFO <85021371> - Elapsed time: 00 min : 13 sec
INFO <85021373> - Operation: successful.
```

Figure 7.26. Radiant Programmer Console Output after Bitstream is Programmed

24. Setup the UART terminal as mentioned in Setting up the UART Terminal section.
25. Press **SW3 Reset** button on the board as highlighted in Figure 7.27.

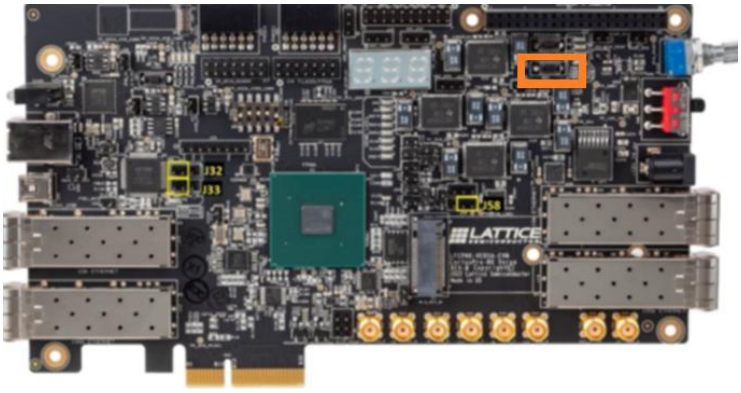

Figure 7.27. SW3 Reset Button

26. The Golden GSRD output results are as follows:
    a. Golden GSRD Bootloader
    b. LPDDR4 Configuration and Memory Training Status
    c. Ethernet PHY I2C Configuration and Status
    d. LPDDR4 and QSPI CRC matches
    e. Application jumps to Golden FreeRTOS RISC-V
    f. Waits for Ethernet Link Up.
    g. SGDMA Packet Received
    h. FreeRTOS Task Scheduler Running
    i. Hardware LED Status:
        - D63: Auto-Link Negotiation
        - D64: LPDDR4 Init Done
        - D67: LPDDR4 PLL Lock
        - D65: Running Animation
        - D66: Running Animation
        - D104: Running Animation
        - D105: Running Animation

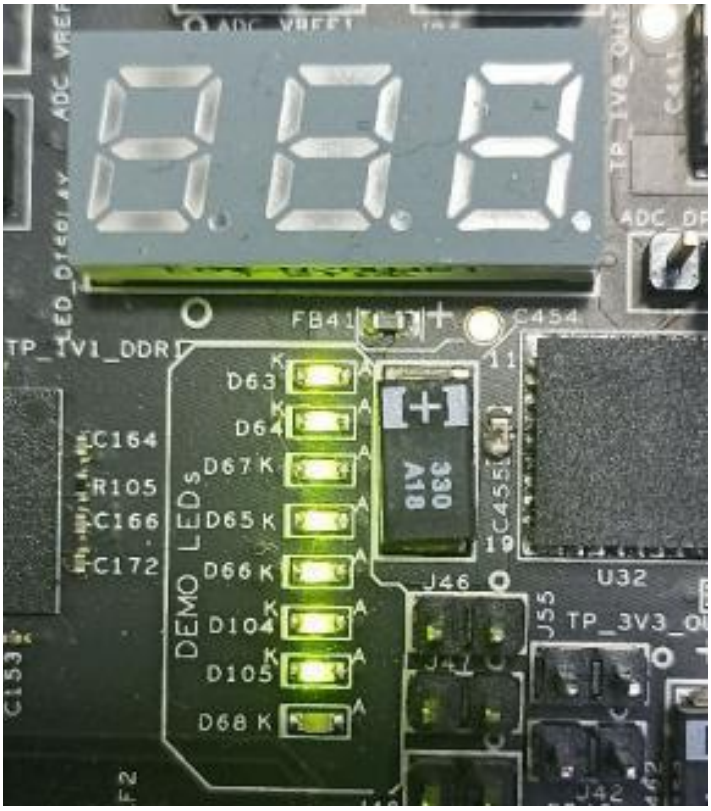

Figure 7.28. LED Status

```
**************************************************
***         GSRD Golden Bootloader LFCPNX        ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: dd8a
Calculated Firmware CRC value: dd8a

CRC matches successfully !!

Jumping to FreeRTOS application ...
```

Figure 7.29. Golden GSRD - Output on UART Terminal for Bootloader and FreeRTOS Start

```
**************************************************
***     GSRD Golden FreeRTOS on RISC-V CPNX     ***
**************************************************

PHY Initialization:

INFO: PHY Model: 0x01410cc2

INFO: Marvell Alaska 88E1111 PHY detected, proceed to configure to SGMII mode

INFO: Configure SFP to SGMII mode

INFO: After configuration: I2C Extended PHY Specific Status Register: 0x80

INFO: After configuration: I2C Extended PHY Specific Status Register: 0x84

PHY Initialization Complete.

The granularity of pmp is 4.
################################################################################

pmp entry0: mode=0x01, perm=0x07, addr=0x2000355c(*4)=0x8000d570, locked=0

pmp entry1: mode=0x01, perm=0x00, addr=0x2000355d(*4)=0x8000d574, locked=1

pmp entry2: mode=0x01, perm=0x00, addr=0x20003560(*4)=0x8000d580, locked=0

pmp entry3: mode=0x01, perm=0x07, addr=0x3fffffff(*4)=0xfffffffc, locked=0

################################################################################

---------------Waiting for link-up packet  ----------

INFO: PHY Model: 0x01410cc2

INFO: I2C Copper Status Register 1: 79 6d

task rx tx id:2147520672 is running, 0

task 2147516288 is running, 0

task print rx data id:2147522864 is running, 0

rx reg received data:
ff ff ff ff ff ff f8 ce 72 1b 79 72  8  0 45  0
 1 48 58 a4  0  0 80 11 e1  1  0  0  0  0 ff ff
ff ff  0 44  0 43  1 34 41 12  1  1  6  0 55 20
ee 7e  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0 f8 ce 72 1b 79 72  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 7.30. Golden GSRD - Output on UART Terminal for FreeRTOS Running

27. Follow the same steps 5 to Step 25 to load the Primary GSRD Software and Bitstream. For Primary App CRC software, refer to Executables section for the folder and file names.
28. The Primary GSRD output results are as shown in Figure 7.31 and Figure 7.32.

```
**************************************************
***         GSRD Primary Bootloader LFCPNX         ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 331e
Calculated Firmware CRC value: 331e

CRC matches successfully !!

Jumping to FreeRTOS application ...
```

Figure 7.31. Primary GSRD Bootloader– Output on UART Terminal

```
**************************************************
***     GSRD Primary FreeRTOS on RISC-V CPNX     ***
**************************************************

PHY Initialization:

INFO: PHY Model: 0x01410cc2

INFO: Marvell Alaska 88E1111 PHY detected, proceed to configure to SGMII mode

INFO: Configure SFP to SGMII mode

INFO: After configuration: I2C Extended PHY Specific Status Register: 0x80

INFO: After configuration: I2C Extended PHY Specific Status Register: 0x84

PHY Initialization Complete.

The granularity of pmp is 4.
################################################################################

pmp entry0: mode=0x01, perm=0x07, addr=0x2000355c(*4)=0x8000d570, locked=0

pmp entry1: mode=0x01, perm=0x00, addr=0x2000355d(*4)=0x8000d574, locked=1

pmp entry2: mode=0x01, perm=0x00, addr=0x20003560(*4)=0x8000d580, locked=0

pmp entry3: mode=0x01, perm=0x07, addr=0x3fffffff(*4)=0xfffffffc, locked=0

################################################################################

---------------Waiting for link-up packet  ----------

INFO: PHY Model: 0x01410cc2

INFO: I2C Copper Status Register 1: 79 6d

task rx tx id:2147520672 is running, 0

task 2147516288 is running, 0

task print rx data id:2147522864 is running, 0

rx reg received data:
ff ff ff ff ff ff f8 ce 72 1b 79 72  8  0 45  0
 1 48 7b 85  0  0 80 11 be 20  0  0  0  0 ff ff
ff ff  0 44  0 43  1 34 4f 66  1  1  6  0 f6 ad
3e 9d  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0 f8 ce 72 1b 79 72  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 7.32. Primary GSRD FreeRTOS– Output on UART Terminal

7.3.4. Programming the Golden, Primary Software and MCS file

This section describes the three scenarios of GSRD boot-up.

Note: The intent of the below three scenarios is to showcase how manual and automatic multi-boot results would look like. Scenario 1 should be enough for you to test out the multi-boot. Scenario 2 and 3 are merely to showcase the different multi-boot scenarios in GSRD.

7.3.4.1. Scenario 1: Manual Booting when both Primary and Golden FW are with CRC with MCS File Programming

1. Follow Steps 1 to 9 from the Programming the Golden, Primary Software and MCS file section. Do not power-cycle the board.
2. Keep the folder/file of executables handy.

3. To program the **multiboot_system.mcs** file, locate the file in the executables folder that you downloaded and add the file in the Programming file area. The Start Address and End Address are allocated automatically. Confirm the settings as shown in Figure 7.33.

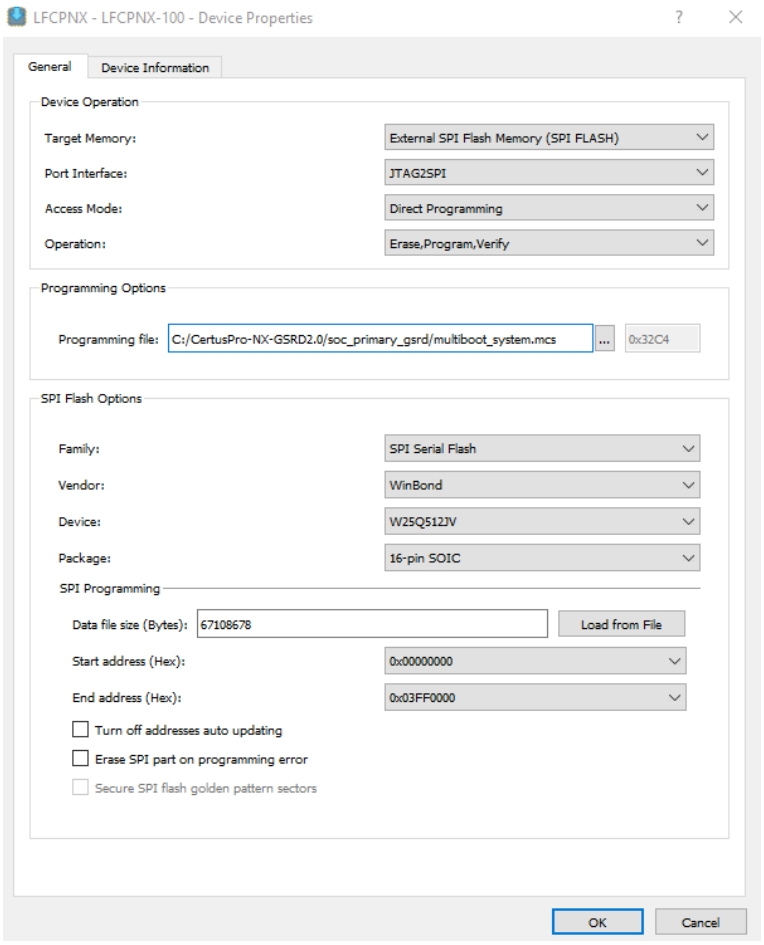

Figure 7.33. Device Properties Window to Setup MCS Programming File

4. Click **OK** and the **Program Device** icon or go to the menu item **Run > Program Device**. Wait until the operation is successful, which takes about 30 to 40 minutes.
5. Do not power-cycle the board yet.
6. Program both the **primary_appcrc.bin** and **golden_appcrc.bin** file as mentioned in the Programming the Golden, Primary Software and MCS file section, Step 17. Make sure to confirm that Start Address (Hex) for both the binaries as per Executables section.
7. Once both binaries are programmed, switch off the board.
8. Switch on the board.
9. Setup the UART terminal as mentioned in Setting up the UART Terminal section.
10. Wait for a few seconds for the FPGA to load the bitstream from the flash and press **SW3** Reset button.
11. The results in the UART Terminal are displayed as shown in Figure 7.34.
    - It loads the Primary GSRD project.
    - If you press the SW3 button again, it loads the same Primary GSRD project.

```
**************************************************
***         GSRD Primary Bootloader LFCPNX         ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 331e
Calculated Firmware CRC value: 331e

CRC matches successfully !!

Jumping to FreeRTOS application ...
```

Figure 7.34. UART Terminal Output after Power-Cycling Board with MCS and Binaries Programmed

12. For the manual multi-boot, press **SW2 (PROGRAMN)** yellow button mentioned in the Hardware Setup section. on the board just above the **SW3 Reset** button.
13. The results on UART terminal are displayed as shown in Figure 7.35. It switches/jumps to Golden GSRD project and stays there unless the **SW2 PROGRAMN** button is pressed.

```
**************************************************
***         GSRD Golden Bootloader LFCPNX        ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: dd8a
Calculated Firmware CRC value: dd8a

CRC matches successfully !!

Jumping to FreeRTOS application ...

**************************************************
***     GSRD Golden FreeRTOS on RISC-V CPNX     ***
**************************************************

PHY Initialization:

INFO: PHY Model: 0x01410cc2
```

Figure 7.35. Switches to Golden GSRD upon SW2 PROGRAMN Button

7.3.4.2. Scenario 2: Manual Booting when Primary FW is with CRC, and Golden FW is without CRC

1. Power-off the board.
2. Power-on the board.
3. Program the MCS file as shown in the Scenario 1: Manual Booting when both Primary and Golden FW are with CRC with MCS File Programming section, Step 2 to 5. If you have already run the scenario 1 entirely, skip this step. Otherwise, follow steps 2 to 5.
4. To program the **c_golden_app.bin** software file into the SPI Flash, Follow the settings as shown in Figure 7.36.

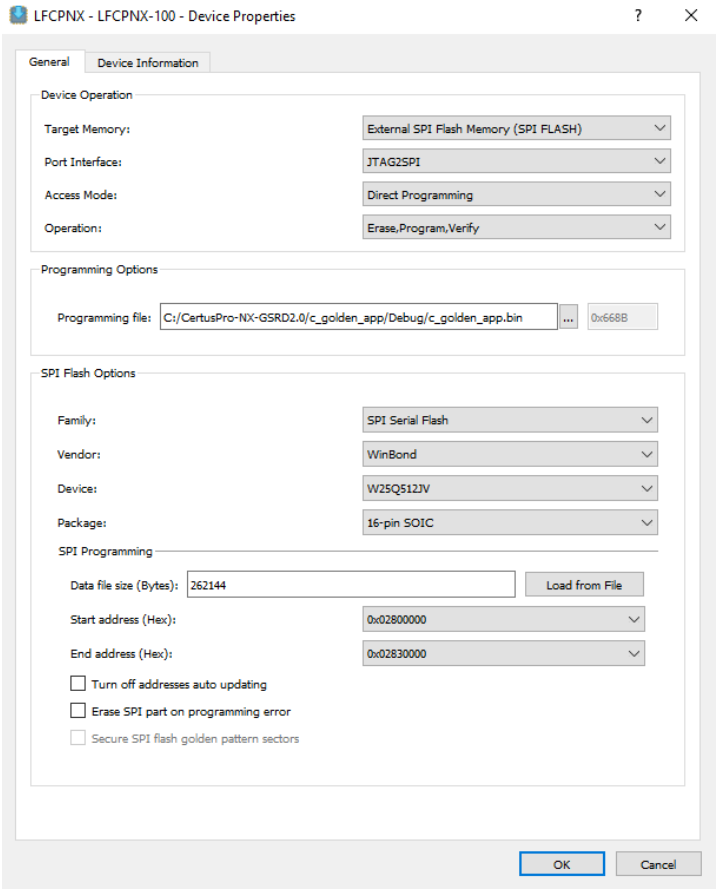

Figure 7.36. Device Properties settings to program the Golden Application

5. Click **OK** and the **Program Device** Icon or go to **Run > Program Device**. Wait until the operation is successful.

Figure 7.37. Radiant Programmer Console Output after Programming

6. Switch off the board.
7. Switch on the board.
8. Setup UART terminal as mentioned in the Setting up the UART Terminal section.
9. Wait for a few seconds for the FPGA to load the bitstream from the flash and press the **SW3 Reset** button.
10. The results on the UART terminal are displayed as shown in Figure 7.38 and loads the Primary GSRD project.

```
**************************************************
***         GSRD Primary Bootloader LFCPNX       ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 331e
Calculated Firmware CRC value: 331e

CRC matches successfully !!

Jumping to FreeRTOS application ...

**************************************************
***     GSRD Primary FreeRTOS on RISC-V CPNX     ***
**************************************************

PHY Initialization:
```

Figure 7.38. Primary GSRD - UART Output after SW3 Reset where CRC Matched

11. Manually press the **SW2 PROGRAMN** button.
12. Results on the UART terminal are displayed as shown in Figure 7.39. It jumps to Golden GSRD project and shows as CRC mismatch and stays there.

```
---------------Waiting for link-up packet  ----------

INFO: PHY Model: 0x01410cc2

************************************************
***         GSRD Golden Bootloader LFCPNX        ***
************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: dd8a
Calculated Firmware CRC value: ffff

 ERROR: CRC Mis-matched!!!
```

Figure 7.39. UART Output Switching to Golden GSRD where CRC Mis-Matched Intentionally

13. Press **SW2 PROGRAMN** button, and it loads back the Primary GSRD project.

7.3.4.3. Scenario 3: Automatic Booting when Primary FW is without CRC and Golden FW has CRC

1. Power-off the board.
2. Power-on the board.
3. Program the MCS file as shown in Scenario 1: Manual Booting when both Primary and Golden FW are with CRC with MCS File Programming section, step 2 to 5. If you have already run scenario 1 entirely, skip this step. Otherwise, follow steps 2 to 5.
4. To program the **c_golden_appcrc.bin** software file into the SPI Flash, Follow the settings as shown in Figure 7.40.

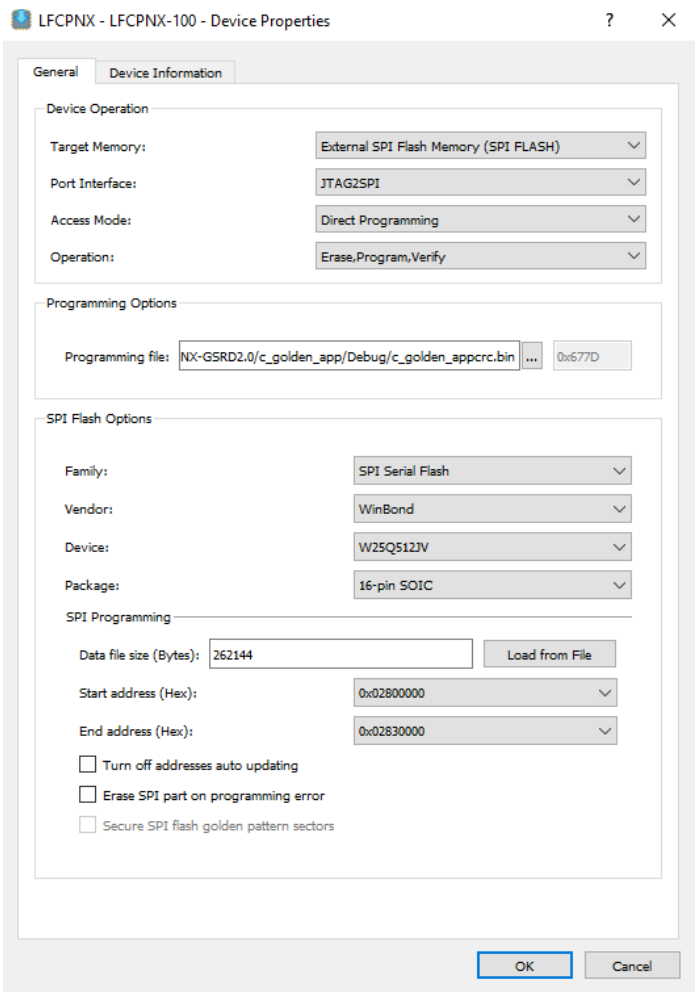

Figure 7.40. Device Properties for Programming Golden App Binary with CRC

5. Click **OK** and the **Program Device** icon or go to **Run > Program Device**. Wait until the operation is successful.

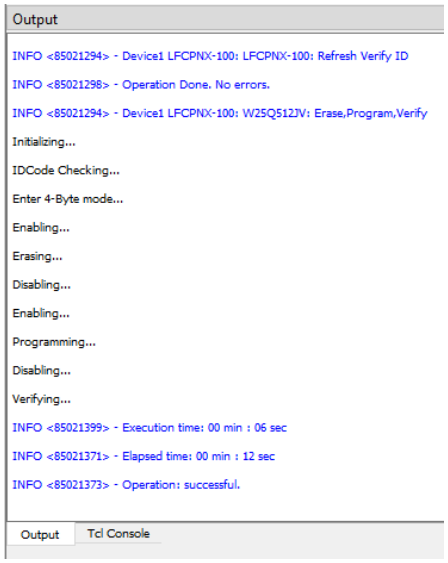

Figure 7.41. Radiant Programmer Console Output after Successful Programming

6. To program the **c_primary_app.bin** software file into the SPI Flash, Follow the settings as shown in Figure 7.42.

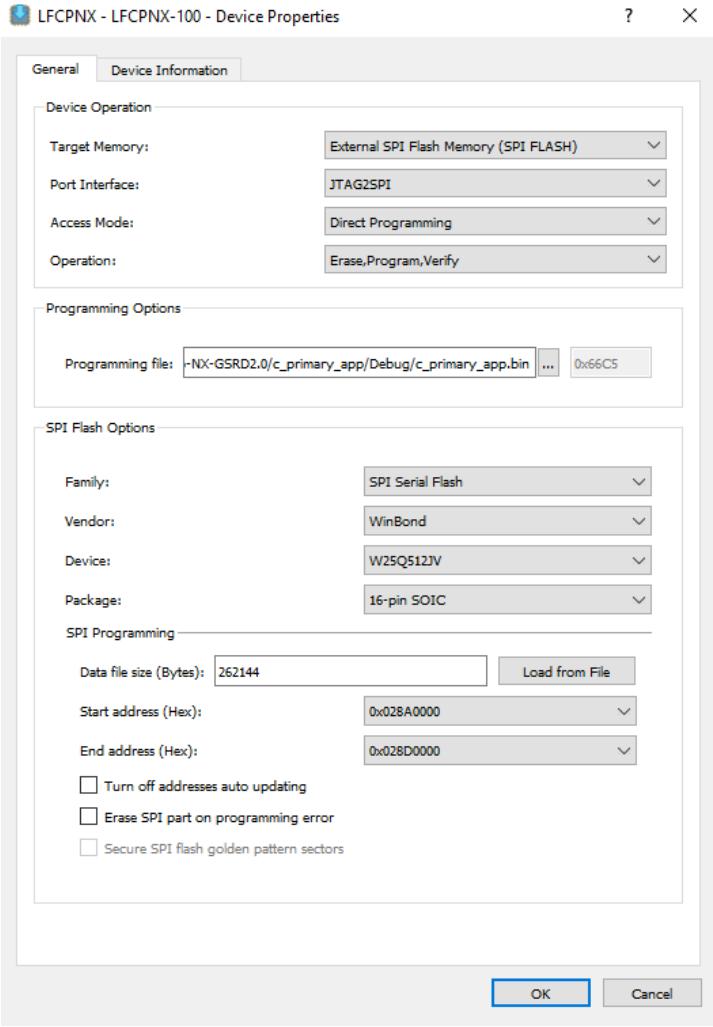

Figure 7.42. Device Properties Window for Primary App Binary without CRC

7. Click **OK** and the **Program Device** icon or go to **Run > Program Device**. Wait until the operation is successful.

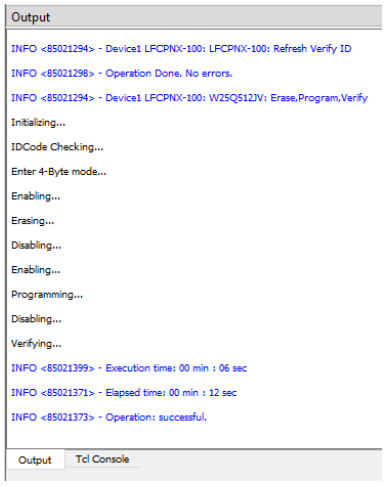

Figure 7.43. Radiant Programmer Console Output after successful programming

8. Switch off the board.
9. Switch on the board.
10. Setup UART terminal as mentioned in the Setting up the UART Terminal section.
11. Wait a few seconds for the FPGA to load the bitstream from the flash. Press **SW3 Reset** button.
12. The results on the UART terminal are displayed as shown in Figure 7.44.
    - It loads the Primary GSRD project first.
    - As soon as the FW CRC for Primary GSRD mismatches, it triggers a soft reset or automatic PROGRAMN.
    - It automatically loads the Golden GSRD bitstream and software.

```
************************************************
***         GSRD Primary Bootloader LFCPNX       ***
************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 331e
Calculated Firmware CRC value: ffff

 ERROR: CRC Mis-matched!!ÿ
------------------------------------------------
```

Figure 7.44. Primary GSRD without CRC Fails – UART Terminal

```
**************************************************
***         GSRD Golden Bootloader LFCPNX        ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
```

Figure 7.45. Automatically Jumps to Boot-up Golden GSRD

13. Manually press **SW2 PROGRAMN** button.
14. The results on the UART terminal are displayed as shown in Figure 7.46. It jumps back to Primary GSRD where CRC fails again and switches to working the Golden GSRD project and stays there.

```
**************************************************
***         GSRD Primary Bootloader LFCPNX       ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

QSPI Fast Read -- Cycle: 0
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8000f000

QSPI Fast Read -- Cycle: 1
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8001e000

QSPI Fast Read -- Cycle: 2
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8002d000

QSPI Fast Read -- Cycle: 3
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8003c000

QSPI Fast Read -- Cycle: 4
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8004b000

QSPI Fast Read -- Cycle: 5
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x8005a000

QSPI Fast Read -- Cycle: 6
INFO: Executing Quad I/O SPI Fast Read Command ...
Copying SPI content to LPDDR start addr: 0x80069000

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 331e
Calculated Firmware CRC value: ffff

 ERROR: CRC Mis-matched!!ÿ
**************************************************
***         GSRD Golden Bootloader LFCPNX        ***
**************************************************

Memory Controller Initialization:

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

Memory Controller Initialization Complete.
```

Figure 7.46. UART Terminal Output - After Pressing SW2 PROGRAMN Button

15. Connect the Ethernet cable to see the traffic.
16. Re-plug the ethernet cable in the case of FreeRTOS stop at ethernet link-up message.

8. Compiling the Reference Design

This section describes the process of compiling the GSRD/GHRD Reference Design. You can always start with the Golden GHRD/GSRD project, which is a part of the Propel Template. Compilation is required to generate the necessary binary files and bitstreams from the source files. The compilation process involves the following software tools for generating the FPGA bitstream and software executable files.

- Lattice Propel Builder 2024.1 Patch
- Lattice Propel Software Development Kit (SDK) 2024.1
- Lattice Radiant Software 2024.1.1

These sections show the typical design and compilation follow for GSRD design.

- Building Bootloader and FreeRTOS binary files using Lattice Propel SDK
- Validating and generating the GSRD design using Lattice Propel Builder
- Synthesizing the RTL files and generating the bitstream using Lattice Radiant Software
- Generating the Multi-Boot MCS File

8.1. Building the Bootloader and FreeRTOS Binary Files using Lattice Propel SDK

To build the Bootloader and FreeRTOS binary files:

1. Create a folder for your project on your PC. For example, CertusPro-NX-GSRD2.0.
2. Launch Propel SDK 2024.1 application.

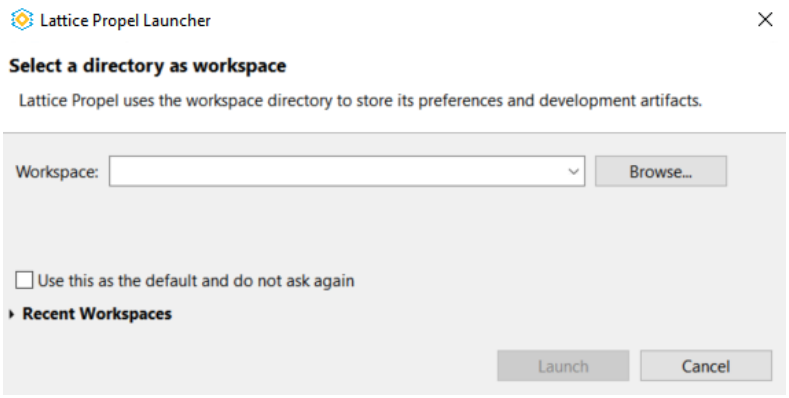

Figure 8.1. Propel Builder Launcher

3. To select the workspace, browse to the created folder by clicking on the **Browse** button as shown in Figure 8.2 and click on **Launch** to launch the workspace.

   Note: Name given below is Primary GSRD as you may be using this template to create your own project titles and make modifications on top of Golden SoC/C Templates.

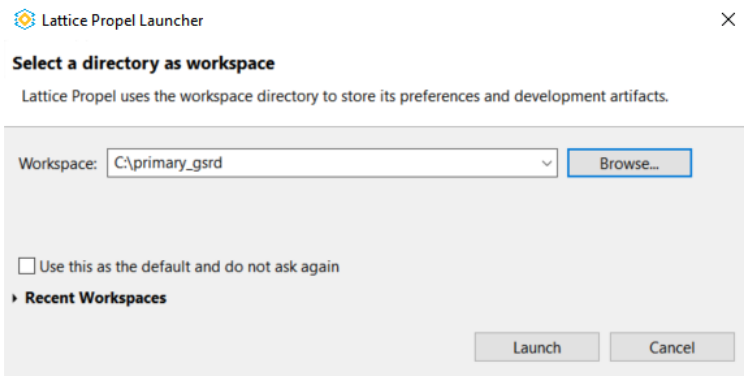

Figure 8.2.Provide Name for the Workspace Directory

4. Click **File > New > Lattice SoC Design Project**.

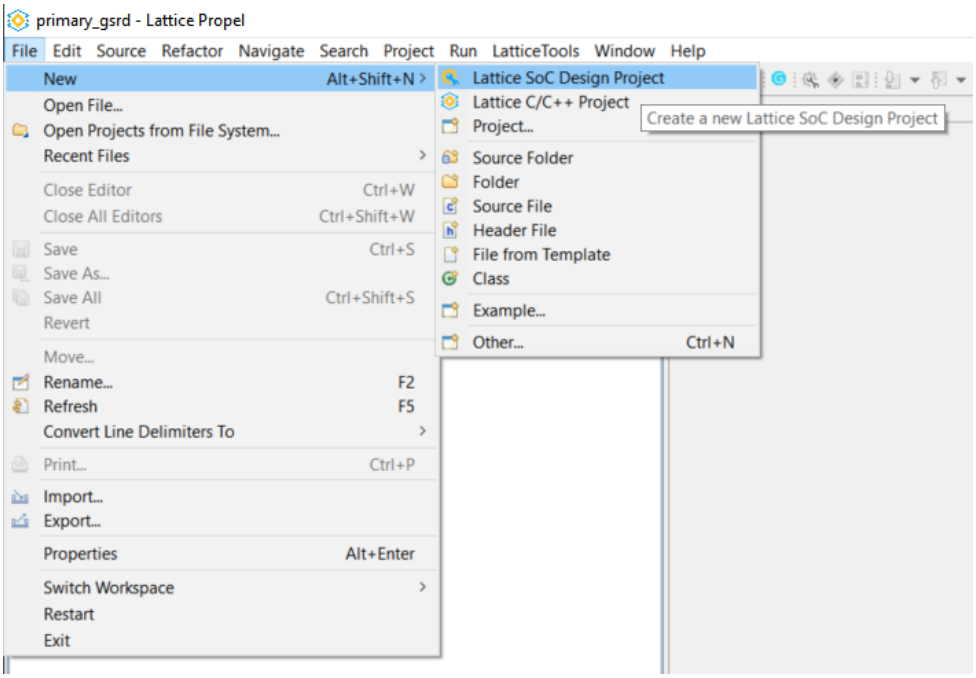

Figure 8.3. Creating Lattice SoC Design Project

5. On the SoC Project window, provide name for your SoC project. Choose Processor as **RISC-V RX**, Family as **LFCPNX**, Device as **LFCPNX-100**, Package as **LFG672**, and choose the GHRD SoC Project LFCPNX from Template Design as shown in Figure 8.4. Since you can make your own desired modifications to this project, the project name is called as **soc_primary_gsrd**.

   Note: For GSRD, do not use the *board* checkbox as you can see some errors while migrating between the Propel SDK and Propel Builder.

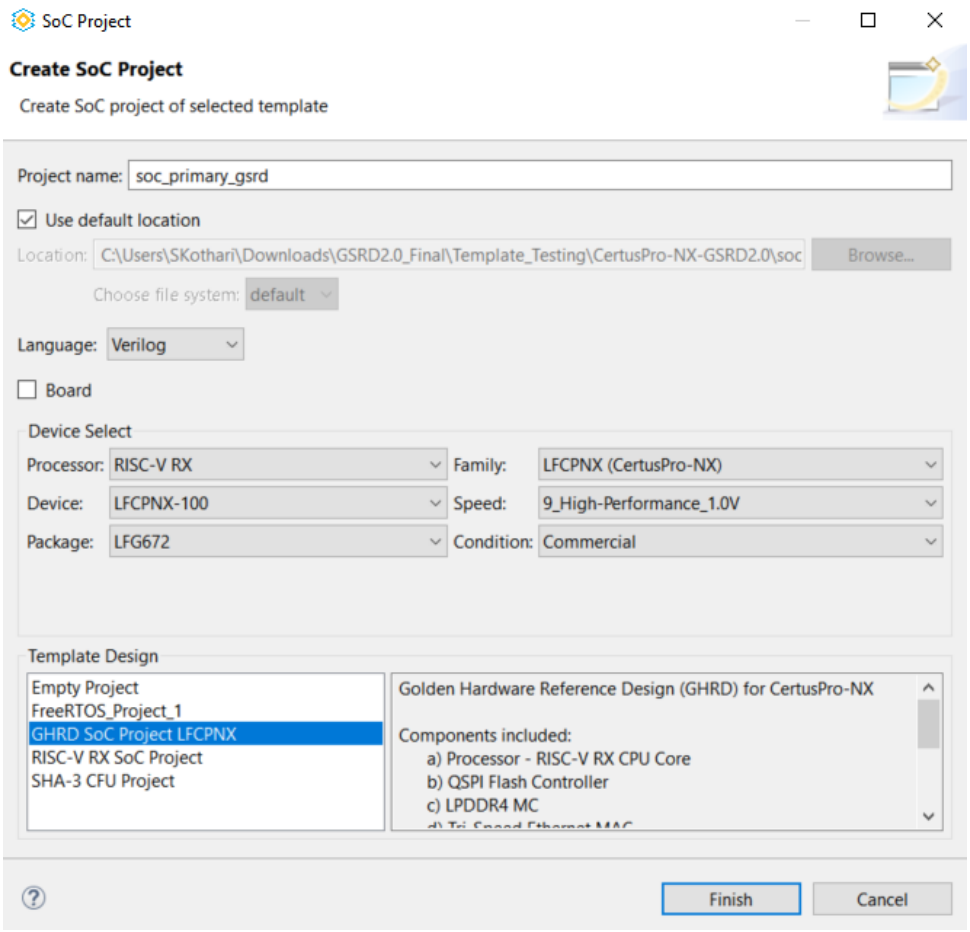

Figure 8.4. SoC Project Window

6. Click **Finish** and it launches the SoC in Propel Builder. This generates the HDL files for IPs instantiated in the project.

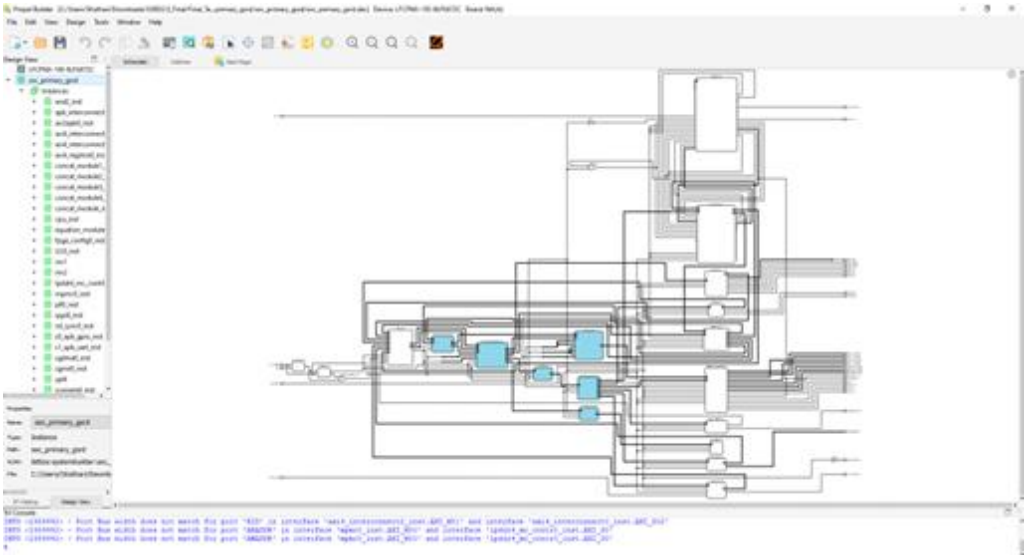

Figure 8.5. Launched SoC in Propel Builder

7. Click on **Design > Validate Design**.

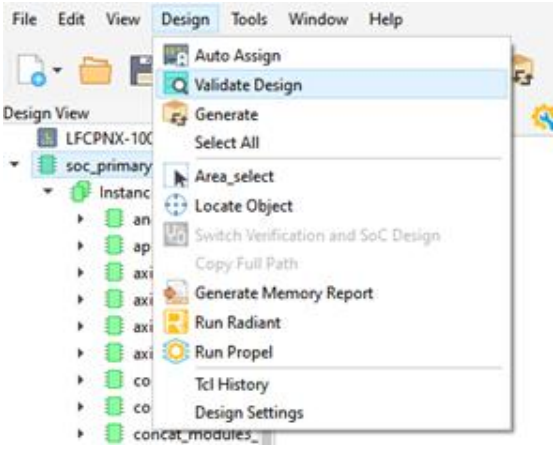

Figure 8.6. Validate Design in Propel Builder

8. The TCL console must be as follows.

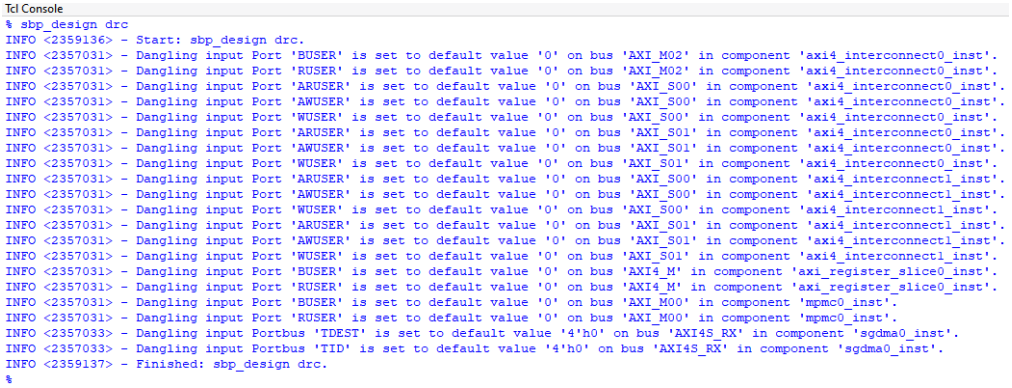

Figure 8.7. TCL Console Output after Validating Design

9. Click on **Design > Generate.**

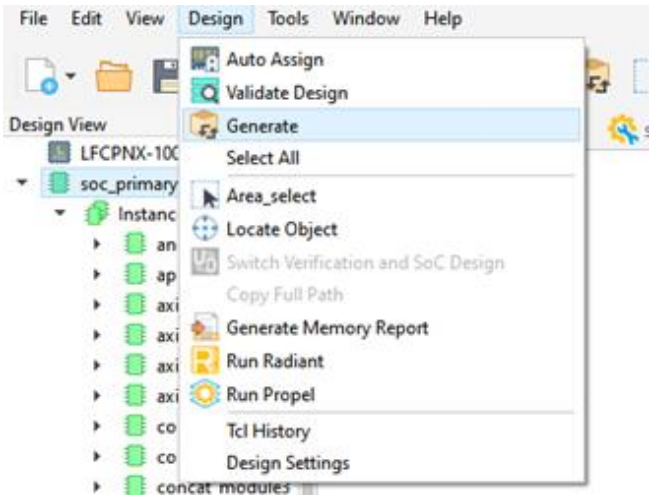

Figure 8.8. Generate in Propel Builder

10. The TCL console must be as follows. Note that the **WARNING** and **INFO** messages are expected.

```
WARNING <2359991> - (PGE SoCDesign) sgmii0_inst_LMMI remains unconnected

INFO <2359992> - (PGE FileExistence) No available driver for fpga_config
%
```

Figure 8.9. TCL Console Output after Generating Design

11. Notice that after Generating the Design, it creates the *sys_env.xml* file in the project directory as shown in Figure 8.10.

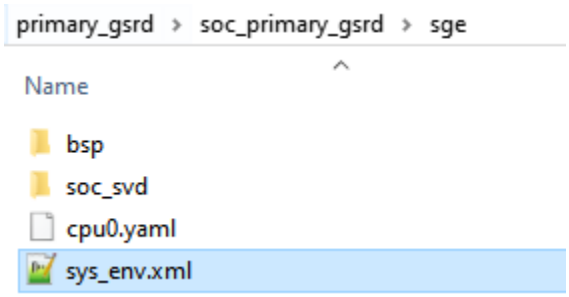

Figure 8.10. sys_env.xml File Created

12. Proceed to the Propel SDK window. Click on **File > New > Lattice C/C++ Project**.

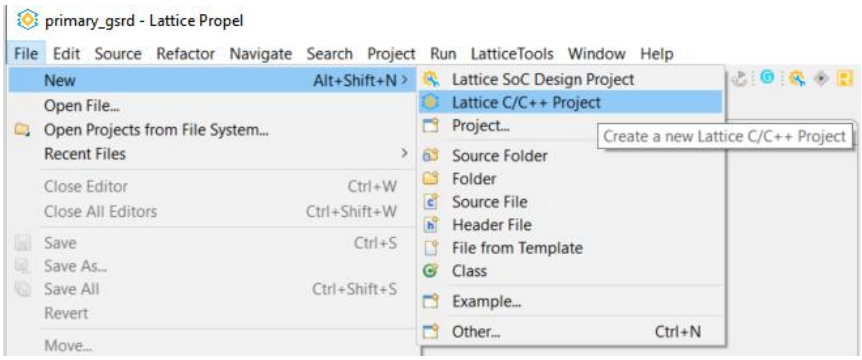

Figure 8.11. Creating Lattice C/C++ Project for Bootloader

13. In C/C++ Project window, it automatically selects the generated sys_env.xml file. In the *Select Example Application* section, select the **GSRD CPNX Bootloader** and provide a name for the bootloader as shown in Figure 8.12.

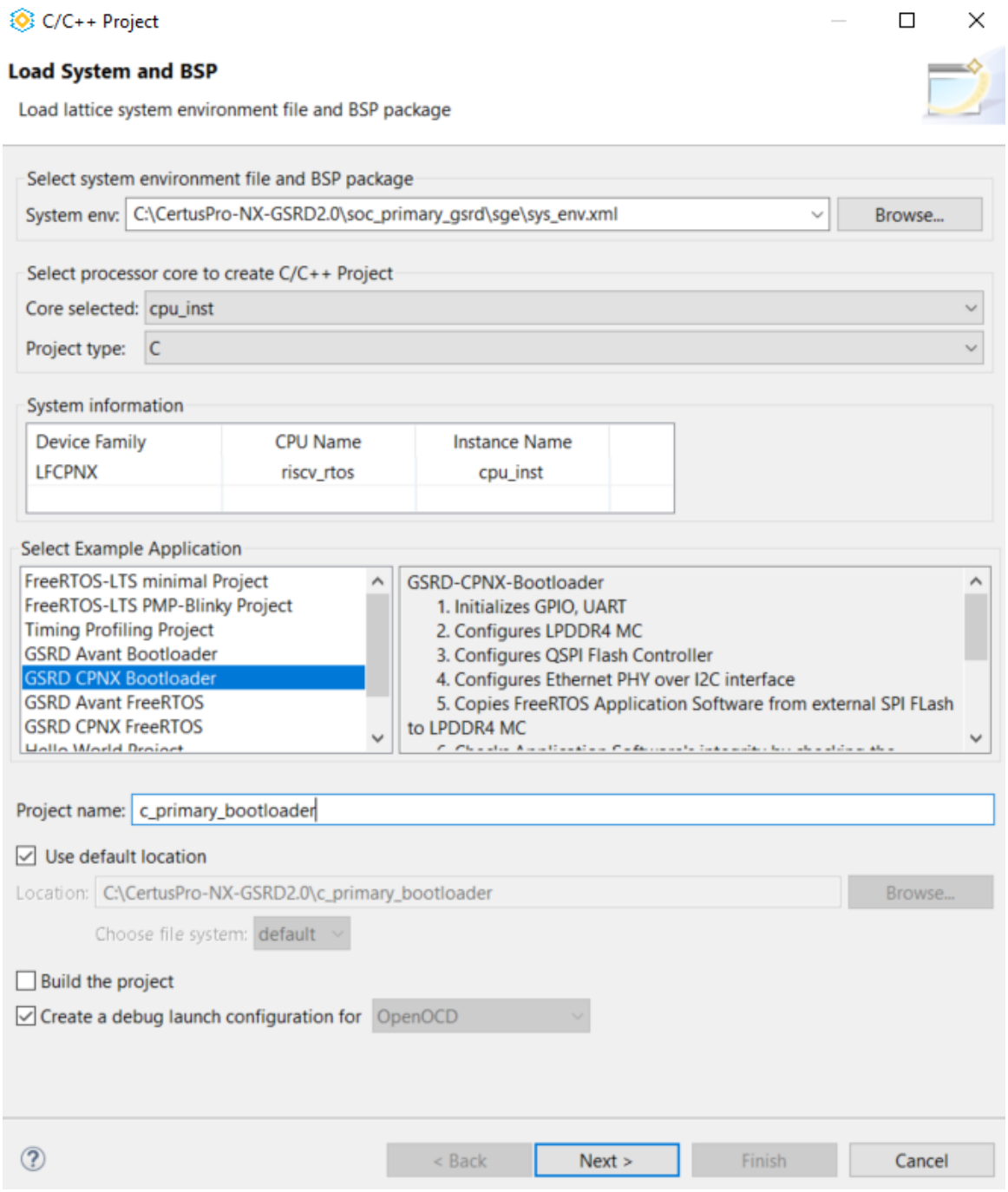

Figure 8.12. Bootloader C/C++ Selection

14. Click **Next** and **Finish**.

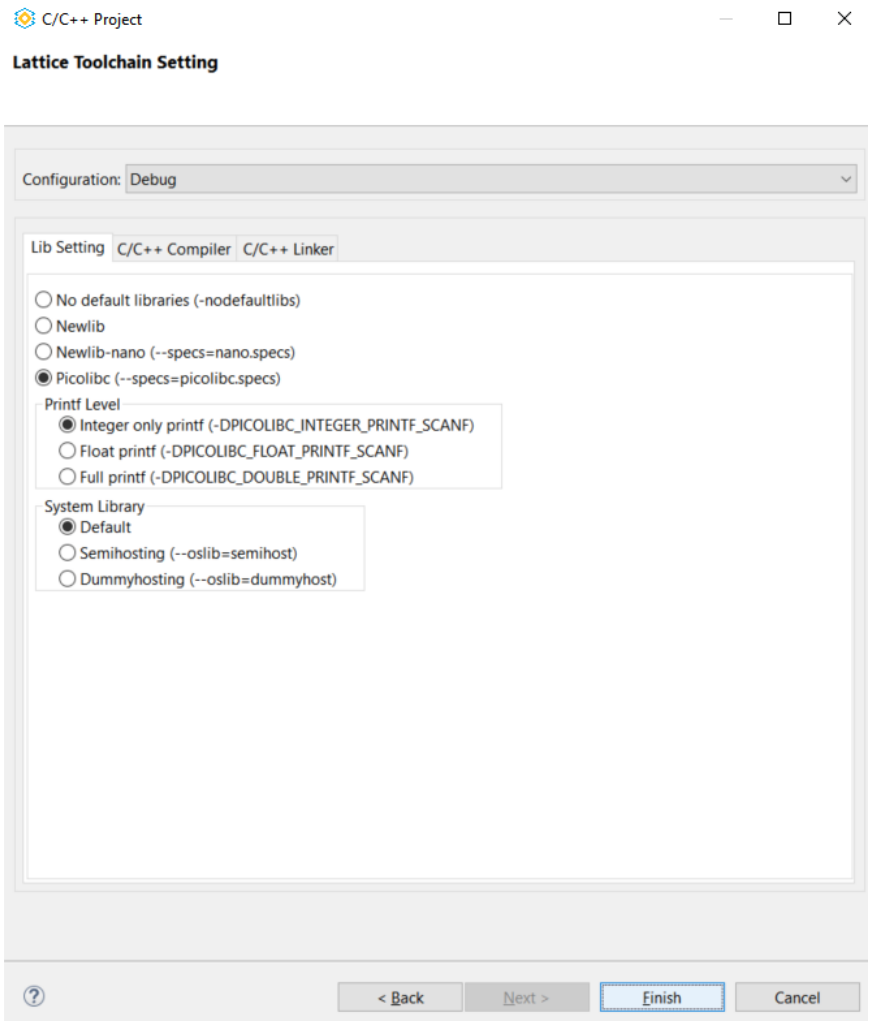

Figure 8.13. C/C++ Lattice Toolchain Setting

15. This loads the bootloader project in the workspace as shown in Figure 8.14.

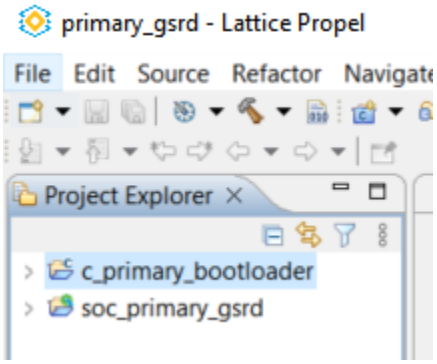

Figure 8.14. Bootloader C Project Created

16. Similarly, create the FreeRTOS C Project. Go to **File > New > Lattice C/C++ Project**.

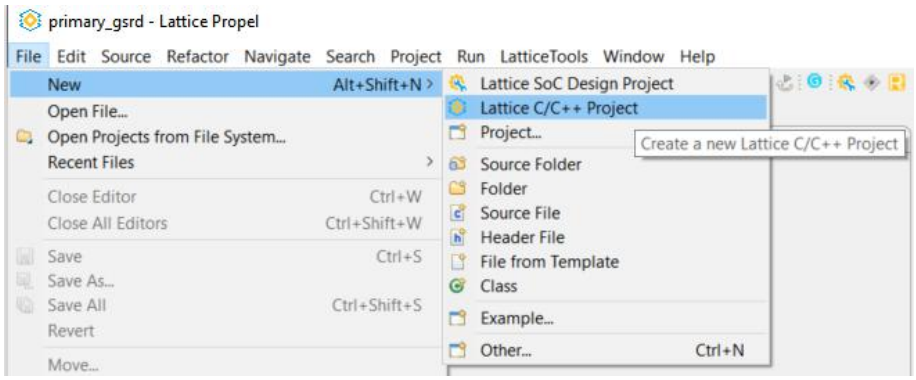

Figure 8.15. Creating C/C++ Project for FreeRTOS

17. In the *Select Example Application*, select the **GSRD CPNX FreeRTOS** project. Provide the project name as shown in Figure 8.16.

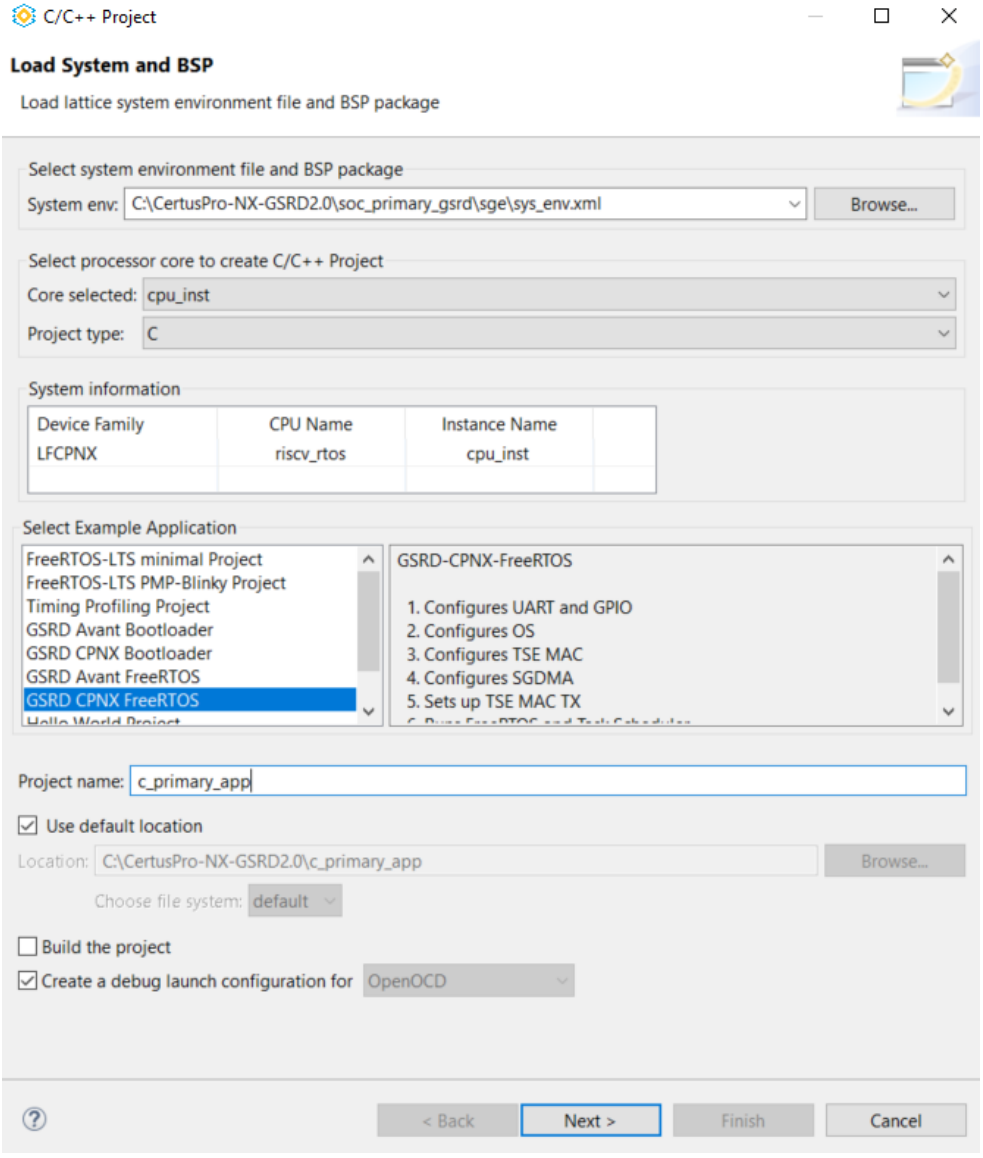

Figure 8.16. FreeRTOS C/C++ Selection

18. Click **Next** and **Finish**.

    Note: This is a manual step you need to perform. After the application project is created, it includes the crc_add_debug.txt,nocrc_add_debug.txt, crc_add_release.txt,nocrc_add_release.txt as shown in Figure 8.17.

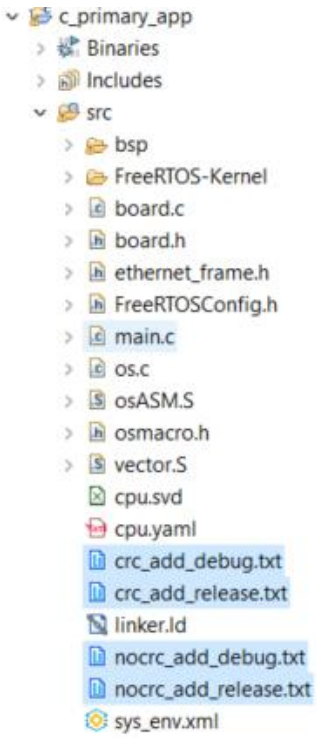

Figure 8.17. FreeRTOS Project Created

19. Copy these four files.

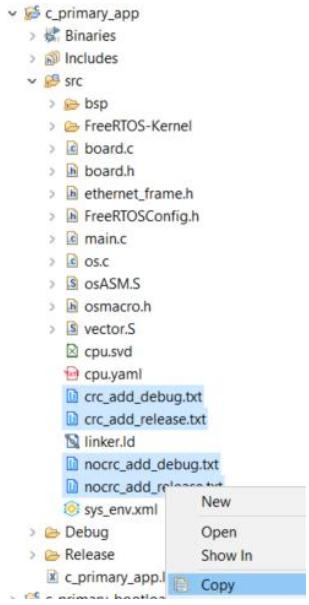

Figure 8.18. Copy the CRC and NoCRC Add files

20. Paste it in your main c_primary_app project as shown in Figure 8.19.

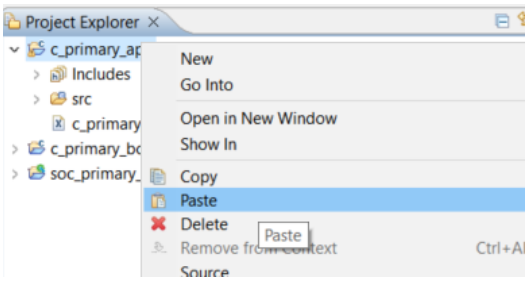

Figure 8.19. Paste in FreeRTOS C project

21. It now adds the txt files under the FreeRTOS App C project as shown in Figure 8.20.

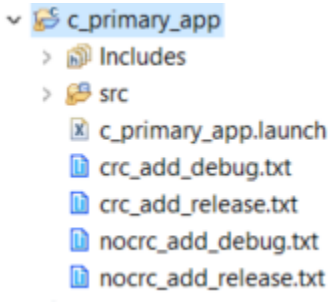

Figure 8.20. Copied Text Files

22. In the c_primary_bootloader main.c file,

    Edit the *new_addr* from *GOLDEN_COPY_MEMORY_START_ADDR to PRIMARY_COPY_MEMORY_START_ADDR*. Note: You must update the *new_addr* as per the design requirements. If you are testing the Golden template, skip this update.

    For Multi-Boot, Edit the main.c file to add the fw_softreset() function as shown in the figure below. For the Golden GSRD, this update is not needed. It is only needed if you are working on the Primary GSRD where you want to enable the **Multi-Boot** feature.

```
int main(int argc, char **argv) {
    static uint8_t idx = 0;
    static uint8_t pin_state = 0xFF;
    unsigned int seq =0;
    unsigned int new_addr = PRIMARY_COPY_MEMORY_START_ADDR;
    unsigned int lppdr_addr=LPDDR_APPLICATION_MEMORY_START_ADDR;
    unsigned int crc_calculated = 0;
    unsigned int crc_fw = 0;
    uint32_t app_base = LPDDR_APPLICATION_MEMORY_START_ADDR;
    bsp_init();
```

Figure 8.21. Updated Primary Address

```
if(crc_calculated == crc_fw)
{
    printf("\r\nCRC matches successfully !!\r\n");
    printf("\r\nJumping to FreeRTOS application ...\r\n");
    asm volatile ("lui  x5, %[appStart]" : : [appStart] "i" (LPDDR_APPLICATION_MEMORY_START_ADDR >> 12));
    asm volatile ("addi x5, x5, %[appStart]" : : [appStart] "i" (LPDDR_APPLICATION_MEMORY_START_ADDR & 0xFFF));
    asm volatile ("jalr x0, x5, 0");
}
else
{
    printf("\r\n ERROR: CRC Mis-matched!!!\n");
    fw_softreset();
}
```

Figure 8.22. Add the fw_softreset() Function to Enable Multi-Boot

23. In the c_primary_app folder, update the *crc_add_debug.txt* or *crc_add_release.txt* file as shown in Figure 8.23. Line 5 and line 12 must be replaced with the app project name that you provided; in this case the name is c_primary_app. Hence, the name of the file is replaced with *c_primary_app.bin*. Line 16 must contain the name as c_primary_appcrc.bin. This output file has the CRC for application software appended at the end of binary file.

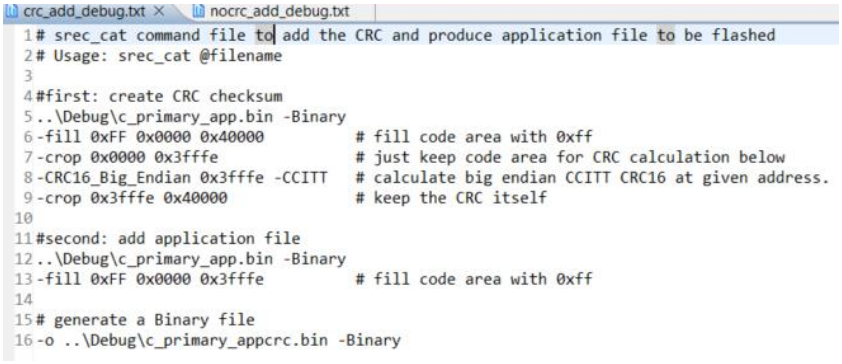

Figure 8.23. Update crc_add_debug.txt

24. Similarly, in the noncrc_add_debug.txt or nocrc_add_release.txt, update line 5 and line 8 with the file name without CRC as shown in Figure 8.24.

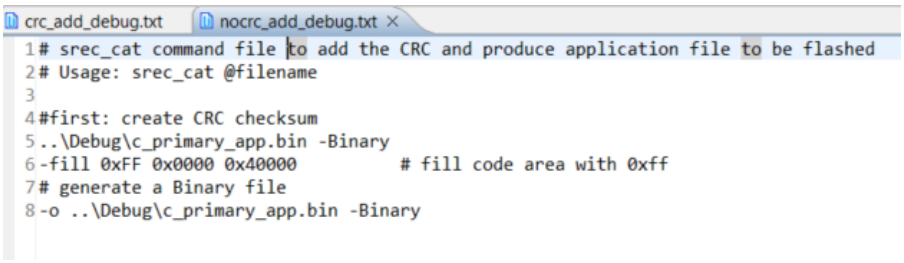

Figure 8.24. Update noncrc_add.txt

25. Open the *Linker.ld* file from *c_primary_app*.

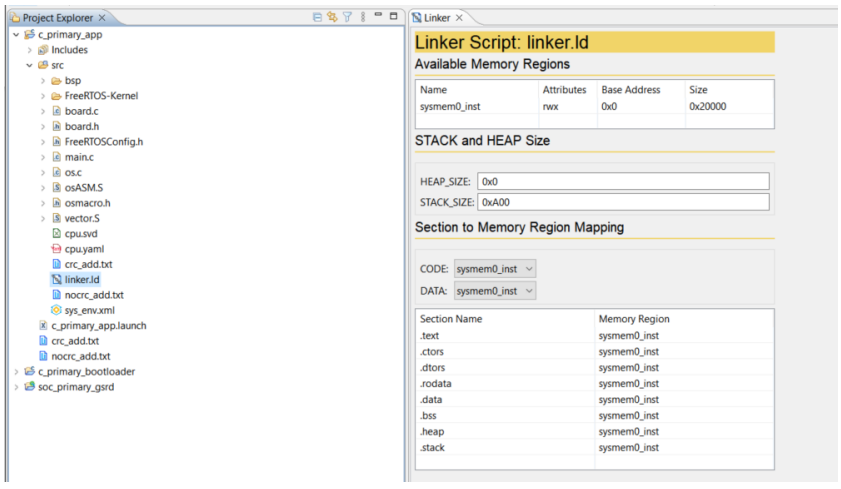

Figure 8.25. Open Linker.Ld File

26. Update the MEMORY org address to **0x80000000** for FreeRTOS to run from LPDDR4.

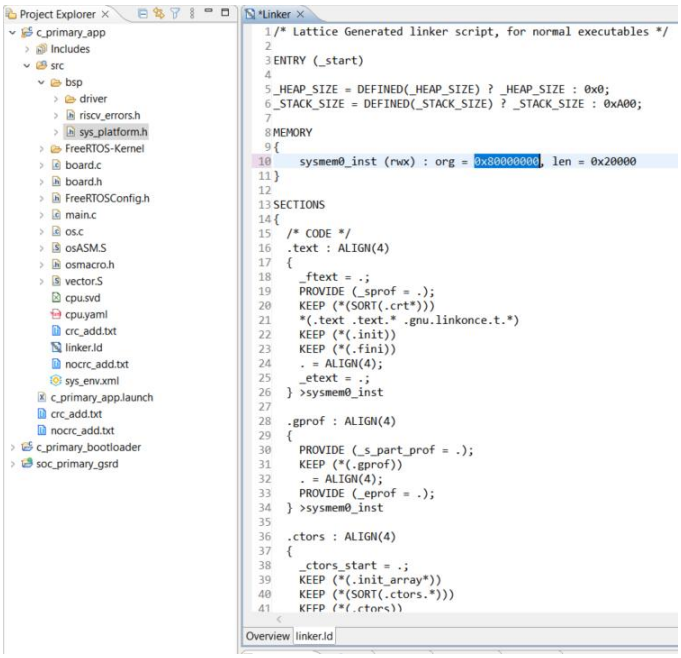

Figure 8.26. Update Linker.ld File

27. Press **Ctrl + s** to save the change as shown in Figure 8.27.

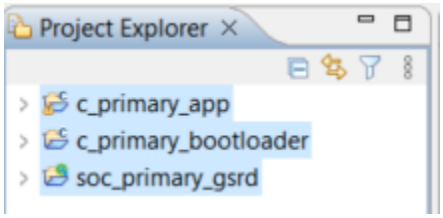

Figure 8.27. Workspace

28. To create the bootloader binary file, right-click on c_primary_bootloader and select **Build Project**.

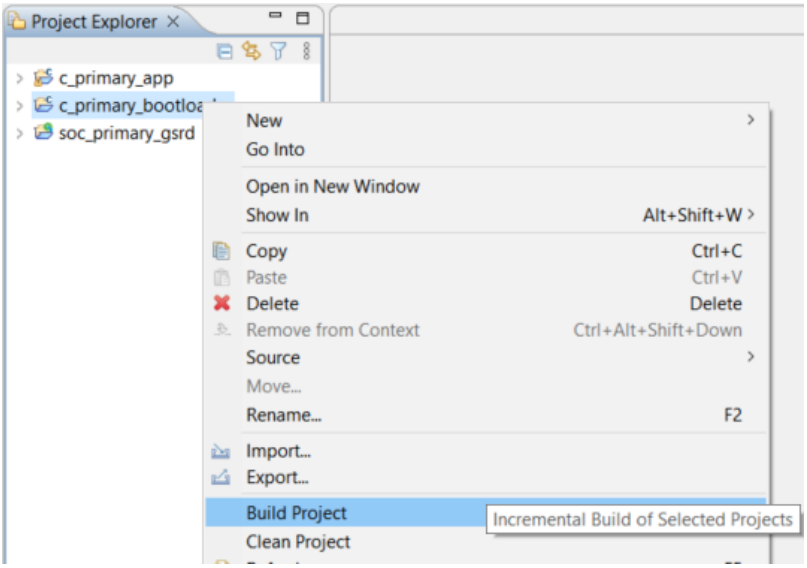

Figure 8.28. Build Bootloader Project

29. The console output is displayed as shown in Figure 8.29.

```
Building target: c_primary_bootloader.elf
Invoking: GNU RISC-V Cross C Linker
riscv-none-embed-gcc -march=rv32imac -mabi=ilp32 -msmall-data-limit=8 -mno-save-restore -O0 -fmessage-length=0 -fsigned-char -ffunction-sections -fdata-sections  -g3 -T "C:/CertusPro-NX-GSRD2.0/c_
Finished building target: c_primary_bootloader.elf

Invoking: GNU RISC-V Cross Create Listing
riscv-none-embed-objdump --source --all-headers --demangle --line-numbers --wide "c_primary_bootloader.elf" > "c_primary_bootloader.lst"
Finished building: c_primary_bootloader.lst

Invoking: GNU RISC-V Cross Print Size
riscv-none-embed-size --format=berkeley "c_primary_bootloader.elf"
   text    data     bss     dec     hex filename
   9668      20    3500   13188    3384 c_primary_bootloader.elf
Finished building: c_primary_bootloader.siz

Invoking: Lattice Create Memory Deployment
riscv-none-embed-objcopy -O binary --gap-fill 0 "c_primary_bootloader.elf" "c_primary_bootloader.bin"; srec_cat "c_primary_bootloader.bin" -Binary -byte-swap 4 -DISable Header -Output "c_primary_b
Finished building: c_primary_bootloader.mem


01:09:51 Build Finished. 0 errors, 4 warnings. (took 4s.731ms)
```

Figure 8.29. Bootloader Build Project Console Output

30. This creates a debug folder as shown in Figure 8.30. The binary created is named as c_primary_bootloader.mem. This goes as a part of system memory in Propel Builder design.

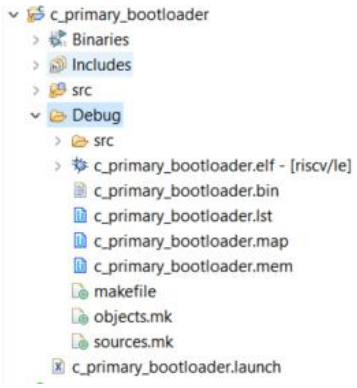

Figure 8.30. Bootloader Binary Created

31. Before creating the binary for FreeRTOS project, right-click on **c_primary_app** project and click **Properties**.

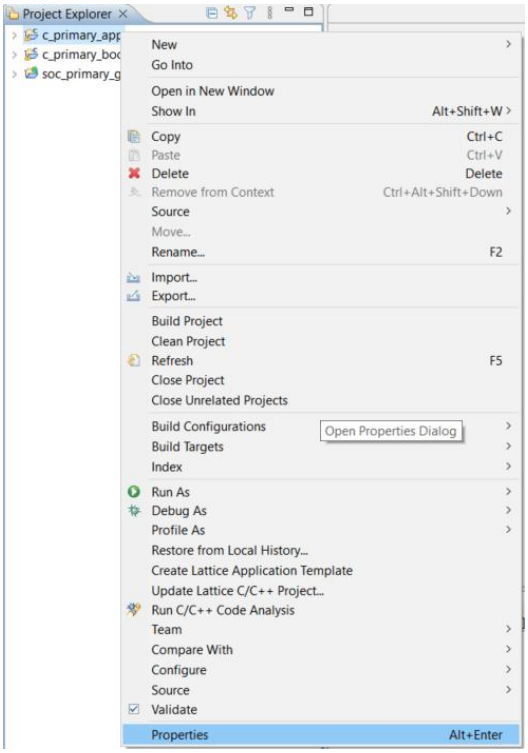

Figure 8.31. Properties

32. In case you wish to create a release folder on the C project build, go to **C/C++ Build > Settings**. In the **Configuration** tab, select **Release**. Otherwise, skip to step 47.

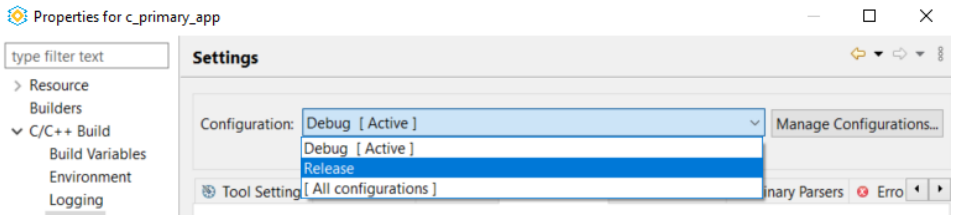

Figure 8.32. Select Release as Configuration

33. Click on **Manage Configurations**, select **Release** and click **Set Active**.

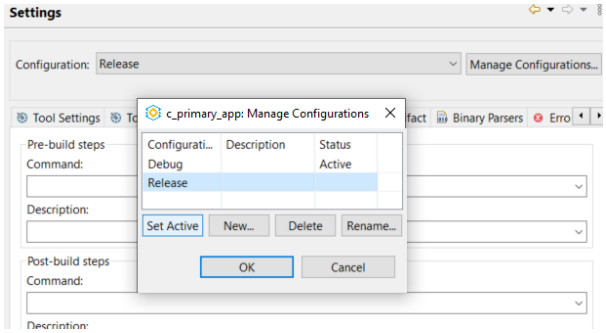

Figure 8.33. Set Release as Active Configuration

34. Go to **C/C++ Build > Settings > Build Steps** and under **Post-Build Steps > Command**, add these commands as shown below for your respective builds i.e. for Debug or Release.

```
srec_cat.exe "@..\crc_add_debug.txt" && srec_cat.exe "@..\nocrc_add_debug.txt"

srec_cat.exe "@..\crc_add_release.txt" && srec_cat.exe "@..\nocrc_add_release.txt"
```

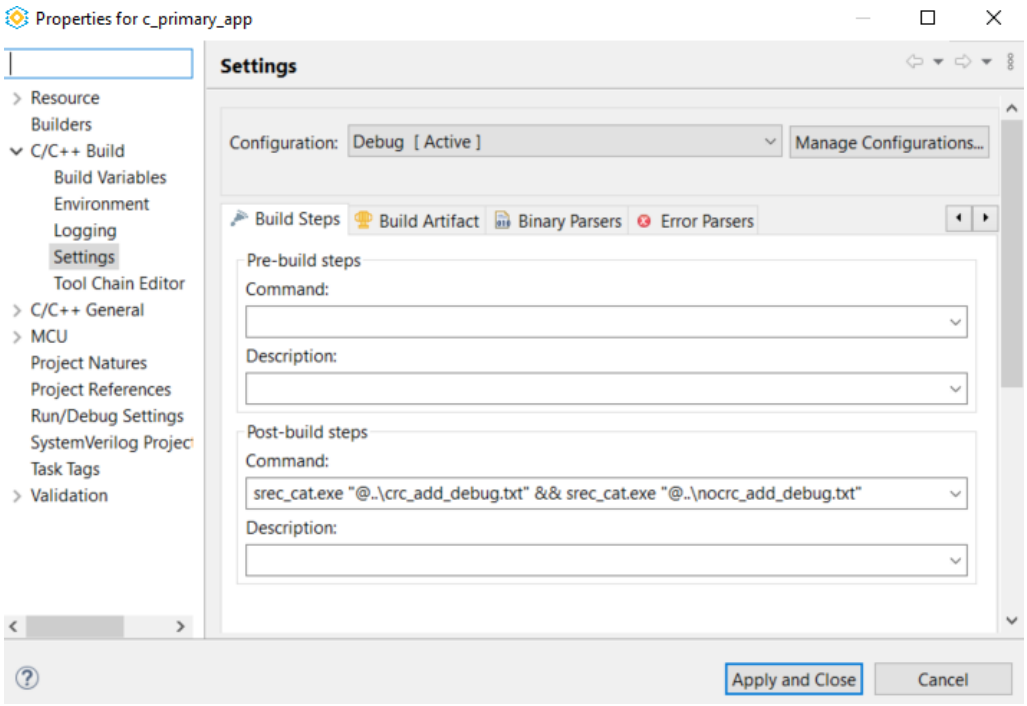

Figure 8.34. Adding Post Build Step for FreeRTOS Application CRC Binary Append

35. Click **Apply** and **Close**.
36. Right-click on **c_primary_app** and click on **Build Project**.

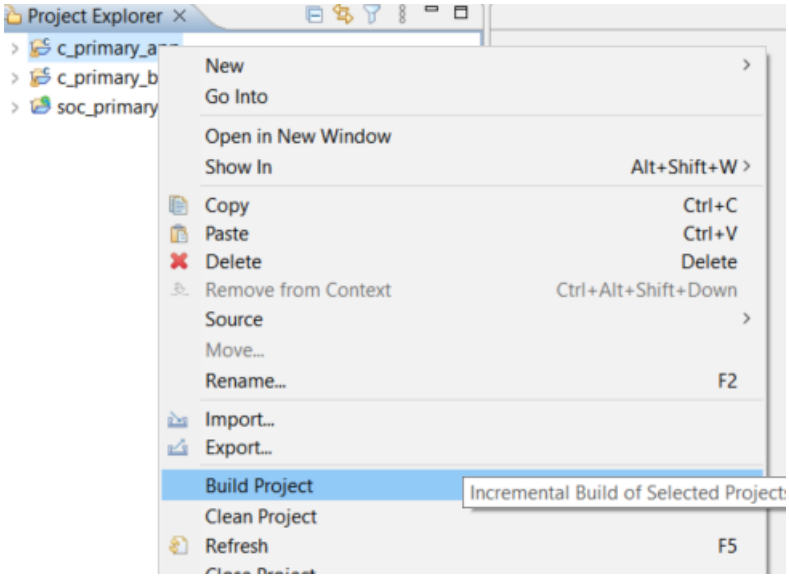

Figure 8.35. Build c_primary_app C/C++ Project

37. The console output is displayed as shown in Figure 8.36.

```
Invoking: GNU RISC-V Cross C Linker
riscv-none-embed-gcc -march=rv32imac -mabi=ilp32 -msmall-data-limit=8 -mno-save-restore -O0 -fmessage-length=0 -fsigned-char -ffunction-sections -fdata-sections  -g3 -T "C:/CertusPro-NX-GSRD2.0/
Finished building target: c_primary_app.elf

Invoking: GNU RISC-V Cross Create Listing
riscv-none-embed-objdump --source --all-headers --demangle --line-numbers --wide "c_primary_app.elf" > "c_primary_app.lst"
Finished building: c_primary_app.lst

Invoking: GNU RISC-V Cross Print Size
riscv-none-embed-size --format=berkeley "c_primary_app.elf"
   text    data     bss     dec     hex filename
  29556     140   27648   57344    e000 c_primary_app.elf
Finished building: c_primary_app.siz

Invoking: Lattice Create Memory Deployment
riscv-none-embed-objcopy -O binary --gap-fill 0 "c_primary_app.elf" "c_primary_app.bin"; srec_cat "c_primary_app.bin" -Binary -byte-swap 4 -DISable Header -Output "c_primary_app.mem" -MEM 32
Finished building: c_primary_app.mem

srec_cat.exe "@..\crc_add_debug.txt" && srec_cat.exe "@..\nocrc_add_debug.txt"


01:09:22 Build Finished. 0 errors, 3 warnings. (took 8s.261ms)
```

Figure 8.36. FreeRTOS App Build Project Console Output

38. This creates the debug folder and the following files with **c_primary_app.bin** and **c_primary_appcrc.bin** binaries with CRC and without CRC as shown in Figure 8.37.

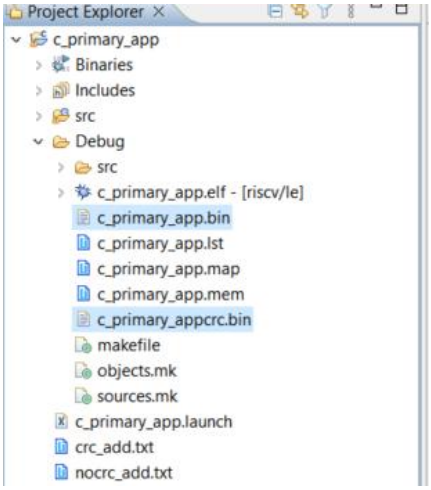

Figure 8.37. FreeRTOS App Binaries Created with CRC and without CRC

8.2. Using Different SPI Flash Manufacturer

There are few SPI flash manufacturers supported by QSPI Flash Controller IP and drivers such as Winbond, Macronix, and Micron.

To build the GSRD bootloader for specific SPI flash model, perform the following steps:

1. Change the 6th parameter of the *qspi_flash_cntl_read_fifo_dis()* function in main.c #line 282 to the respective SPI flash model (see below)

```
for(seq=0;seq<QSPI_FETCH_SEQ;seq++)
{
    printf("\r\nQSPI Fast Read -- Cycle: %d\r\n", seq);
    qspi_flash_cntl_read_fifo_dis(new_addr, FLASH_ADDR_WIDTH, FLASH_CNTL_QSPI_MODE,
                                  (unsigned int *)lppdr_addr, QSPI_FW_FETCH_SIZE,
                                  JEDEC_MANUFACTURER_ID_WINBOND);
    new_addr = new_addr+QSPI_FW_FETCH_SIZE;
    lppdr_addr = lppdr_addr+QSPI_FW_FETCH_SIZE;
    printf("Copying SPI content to LPDDR start addr: 0x%x \r\n", lppdr_addr);
}
```

Figure 8.38. SPI Flash Manufacturer Parameters Changes in QSPI Read Function

- JEDEC_MANUFACTURER_ID_MICRON = Micron MT25QU128
- JEDEC_MANUFACTURER_ID_MACRONIX = Macronix MX25L12833F

- JEDEC_MANUFACTURER_ID_WINBOND = Winbond W25Q512JV

2. Repeat the steps 28 mentioned in the Building the Bootloader and FreeRTOS Binary Files using Lattice Propel SDK section.

8.3. Validating and Generating the GSRD Design using Propel Builder

To validate and generate the GSRD design, perform the following steps:

1. Select the soc project and Launch Propel Builder from Propel SDK as shown below, if it is not already opened.

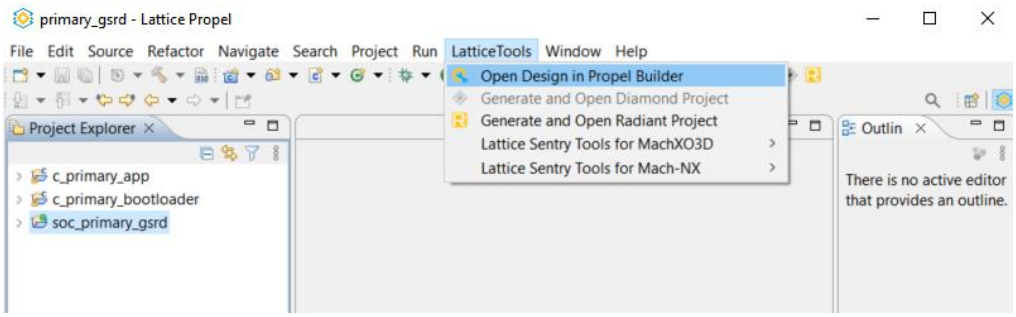

Figure 8.39. Open Propel Builder

2. The Primary GHRD SoC opens as shown in Figure 8.40.

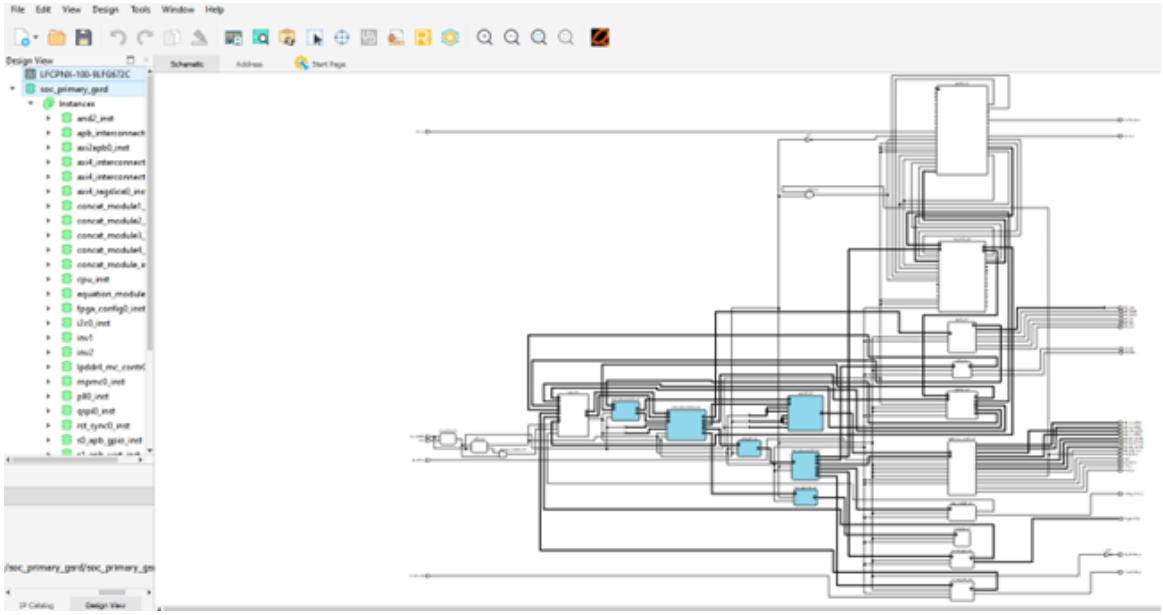

Figure 8.40. Propel Builder SoC Project

3. Before validating and generating the system, you need to pre-initialize the system memory.
4. Select the **System Memory** Instance from the Design View in the left column and it highlights the *sysmem0_inst* as shown in Figure 8.41.

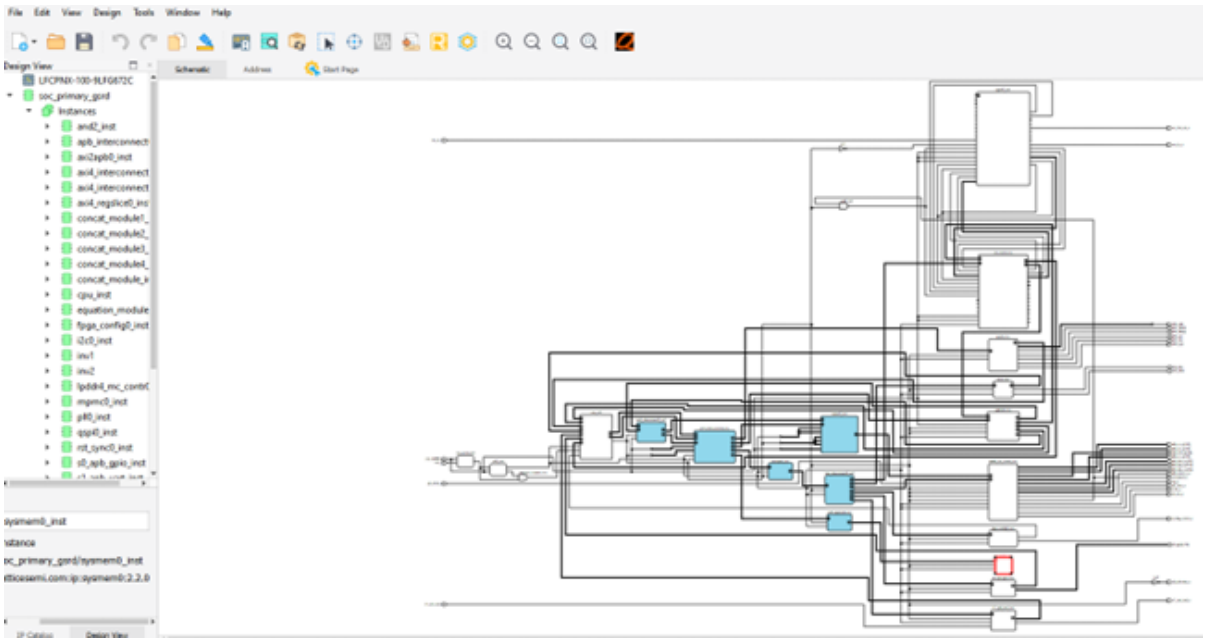

Figure 8.41. System Memory Highlighted

5. Double-click on the highlighted component shown in the schematic and it opens the **Module/IP Block Wizard** as shown in Figure 8.42.

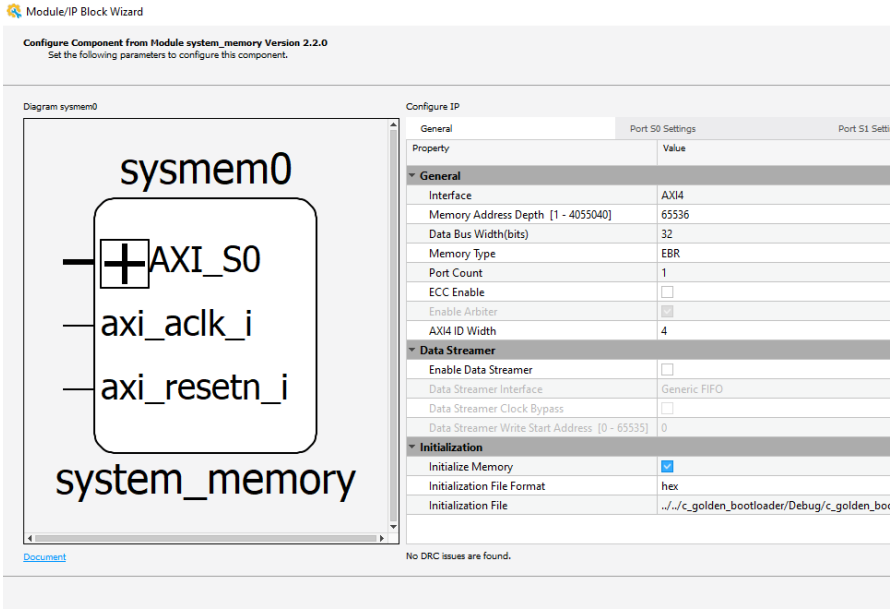

Figure 8.42. System Memory IP Configuration

6. Click on **Initialization File's Value** area with three dots to locate the generated **c_primary_bootloader.mem** Bootloader file as shown in Figure 8.43.

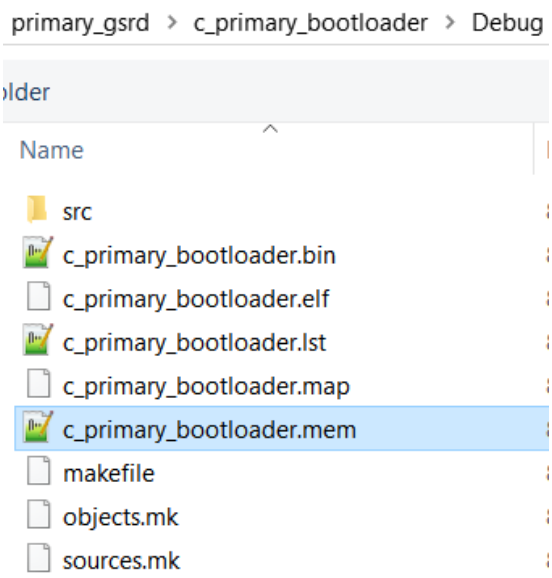

Figure 8.43. Initialize System Memory with c_primary_bootloader.mem

7. Select the **c_primary_bootloader.mem** file created as shown in Figure 8.43.
8. Click **Generate** on the bottom-right corner of the IP Block wizard as shown in Figure 8.44.

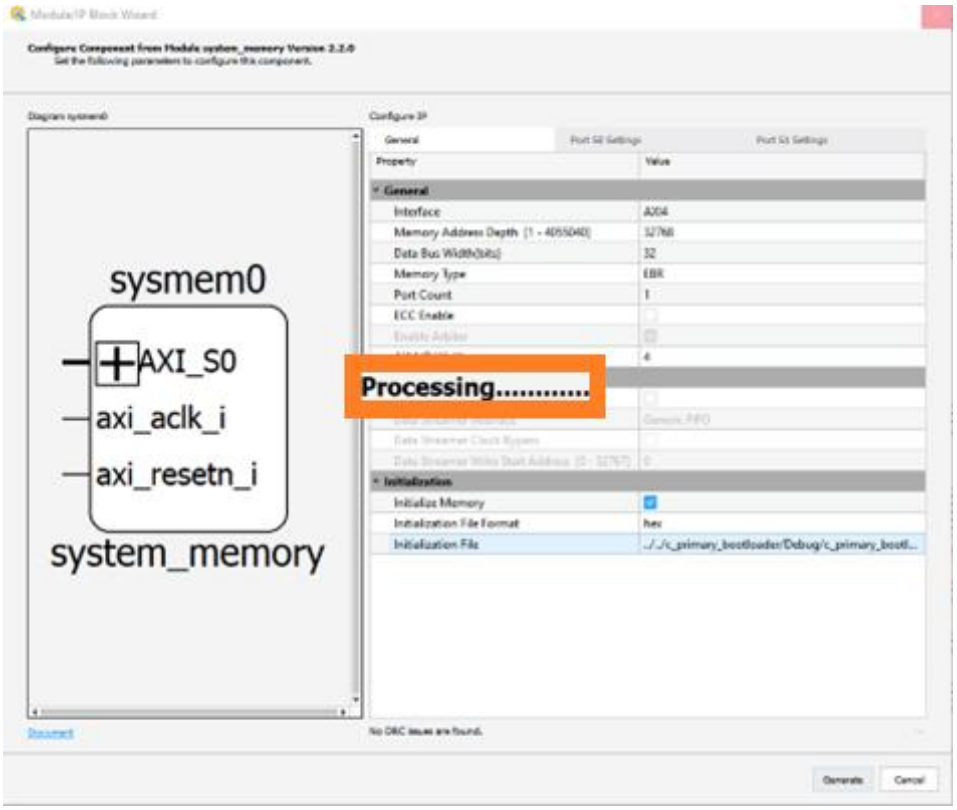

Figure 8.44. Click Generate

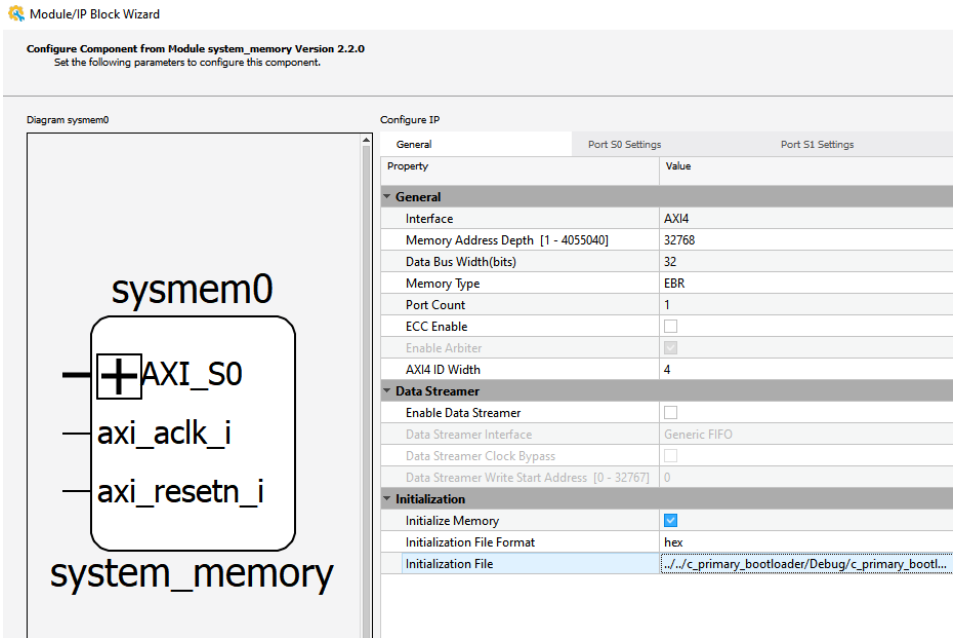

Figure 8.45.Bootloader File Updated in System Memory

9. Wait until the processing is complete and click **Finish** as shown in Figure 8.46.

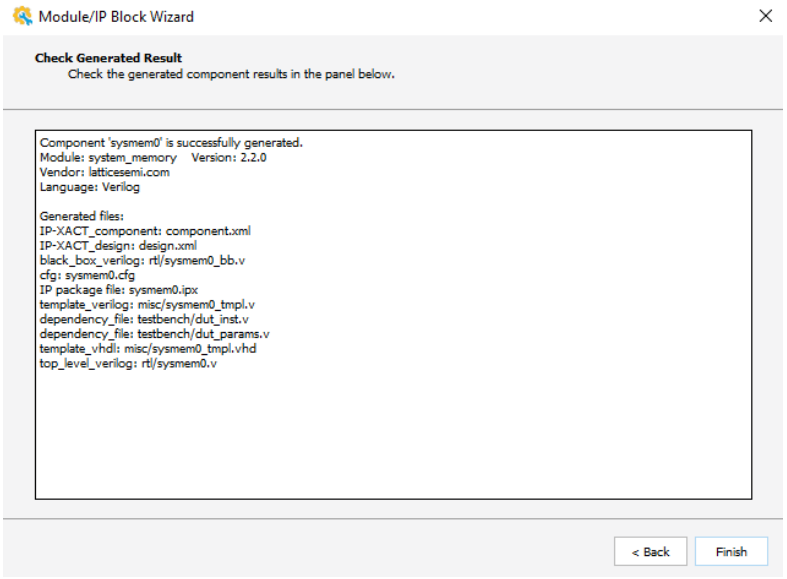

Figure 8.46. Click Finish

10. Go to **Design > Validate** and **Design and Design > Generate Icon**. Output shown is similar to Figure 8.7 and Figure 8.9. Make sure there are no DRC errors.
11. Make sure there are no errors. Note that the *INFO <235999*>* messages on the TCL console are expected since these modules or IPs do not contain any software IP drivers.

8.4. Synthesizing the RTL Files and Generating the Bitstream using Lattice Radiant

To synthesize the RTL files and generate the bitstream, perform the following steps:

1. Click the Radiant icon from Propel Builder to launch the Radiant software as shown in Figure 8.47.

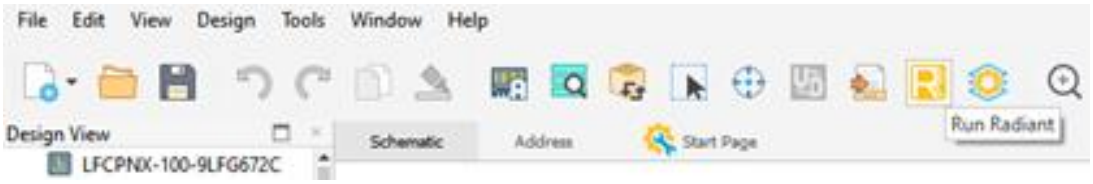

Figure 8.47.Run Radiant Icon

2. The Radiant window of your project is launched as shown in Figure 8.48.

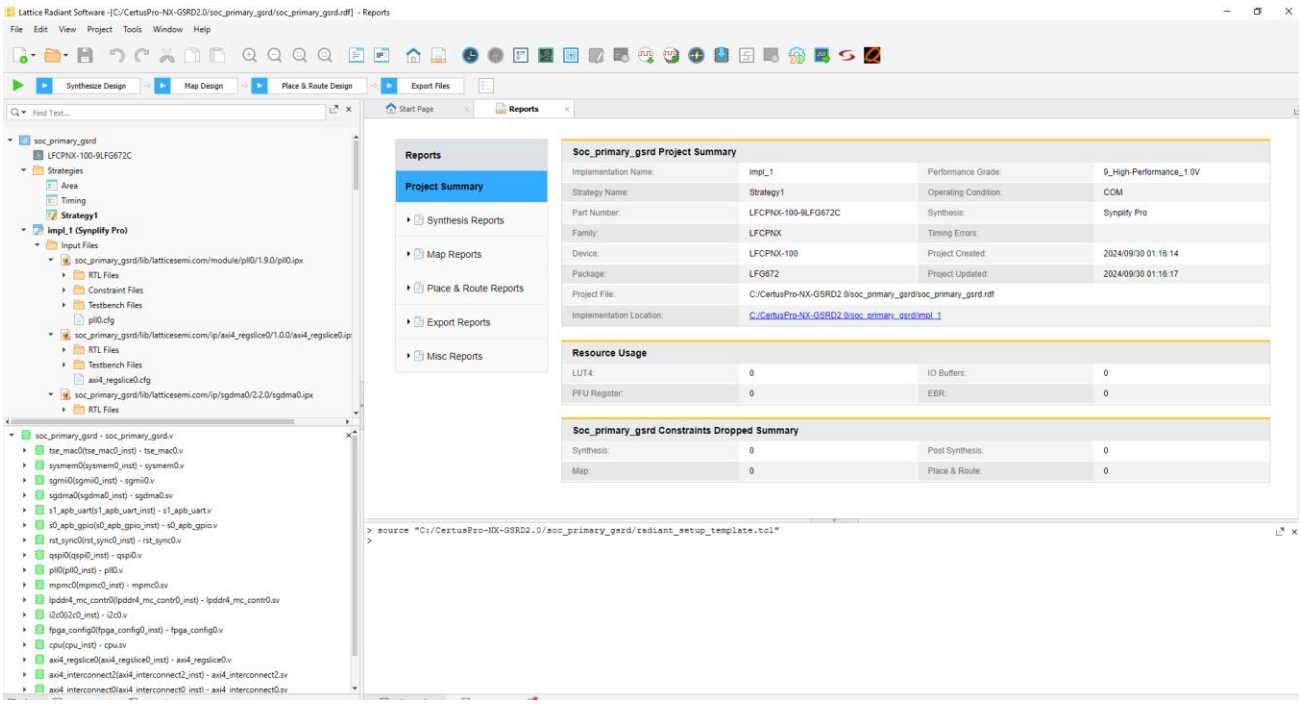

Figure 8.48. Lattice Radiant 2024.1 Window

3. Click on **Close** on the pop-up window.
4. Locate the following clock_constraint.sdc and soc_board_constraint.pdc files in your Propel Installation area.

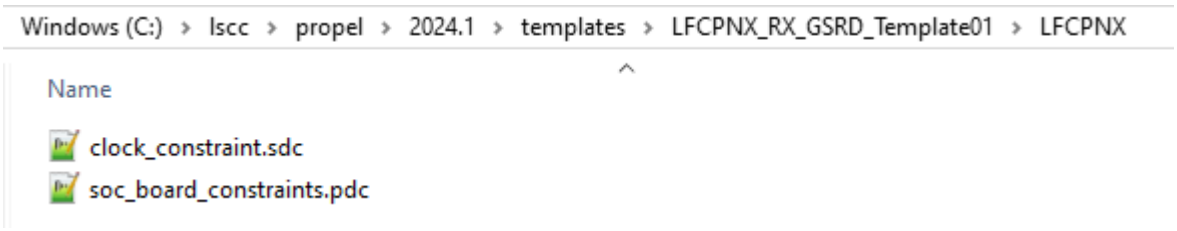

Figure 8.49. Constraint Files Folder

5. Right-click on **Pre-Synthesis Constraint Files > Add > Existing File..** as shown in Figure 8.50.

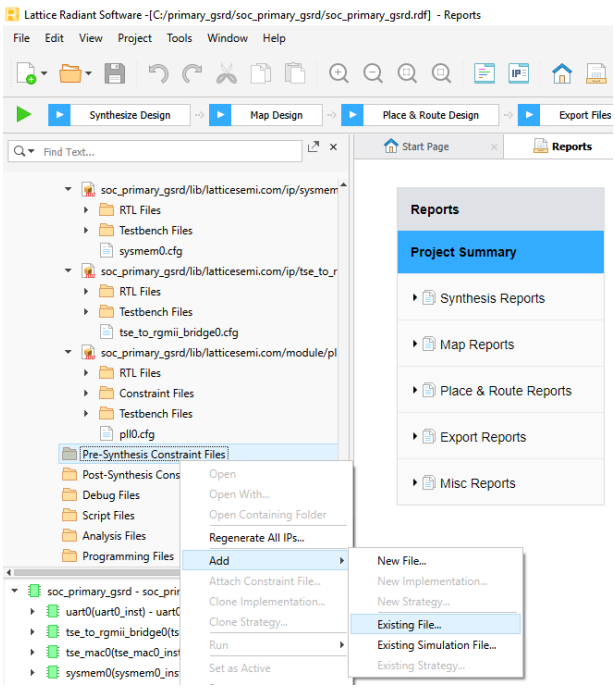

Figure 8.50. Add Existing Pre-Synthesis Constraint

6. Select the clock_constraint.sdc and Ensure that the Copy File to Directory is enabled as shown below. Click **Add**.

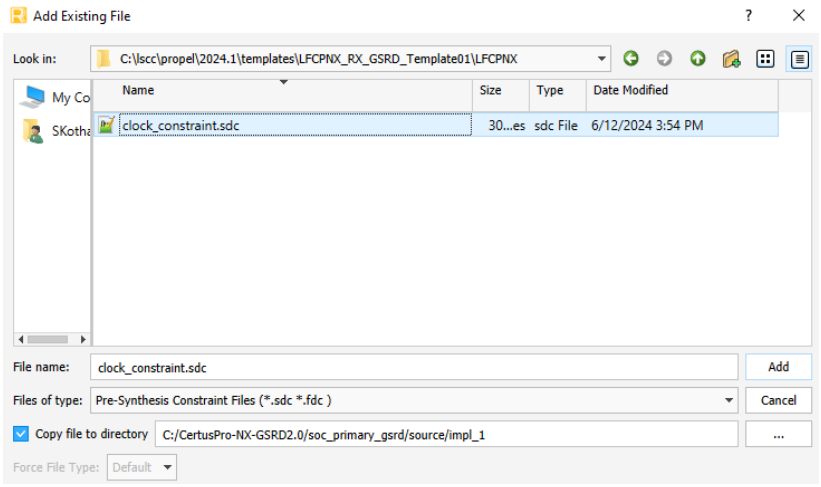

Figure 8.51. Select clock_constraint.sdc file

7. Similarly, add the soc_board_constraints.pdc file to the Post-Synthesis Constraint Files as shown in Figure 8.52. Click **Add**.

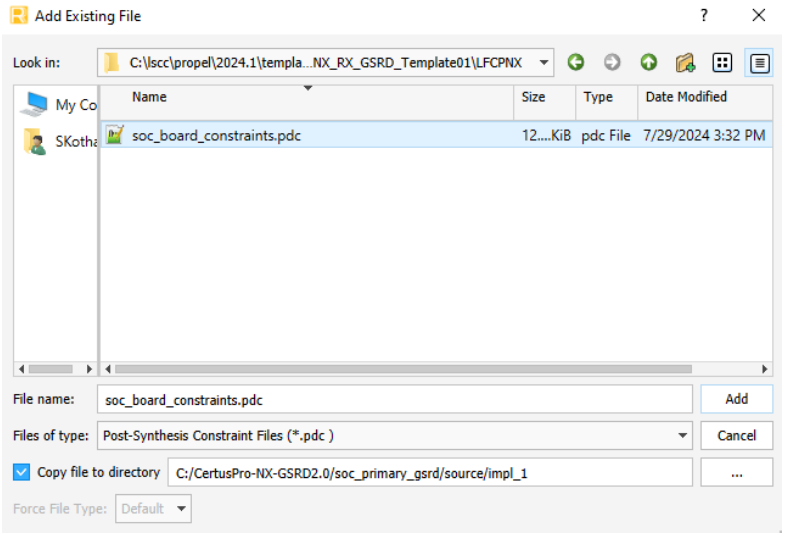

Figure 8.52. Select soc_board_constraint.pdc File

8. After you add these constraint files, the Radiant file list is updated as shown in Figure 8.53.

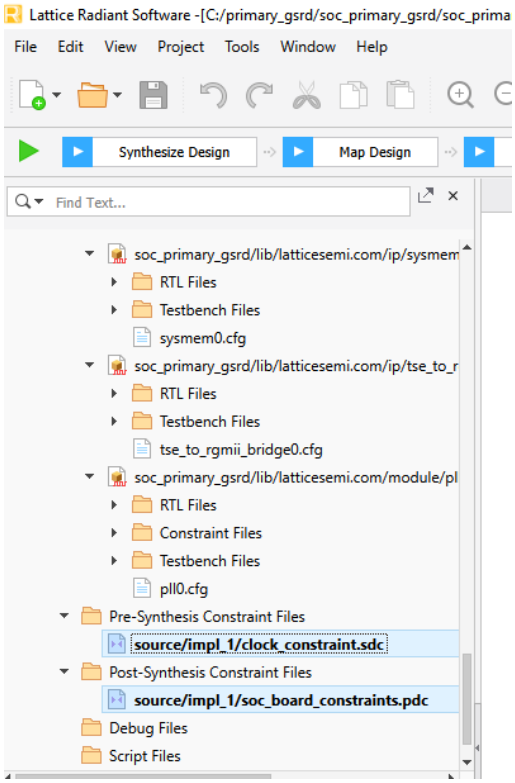

Figure 8.53. Pre and Post-Synthesis Constraint Files Added

9. Double-click on Strategy1.

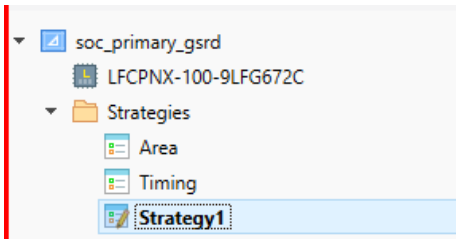

Figure 8.54. Update the Strategy

10. The Place & Route strategy is used for the GSRD testing.

    Note: You can update these fields as per their machine and run-time requirements. However, due to this change, the Radiant tool might update the warnings and place & route details.

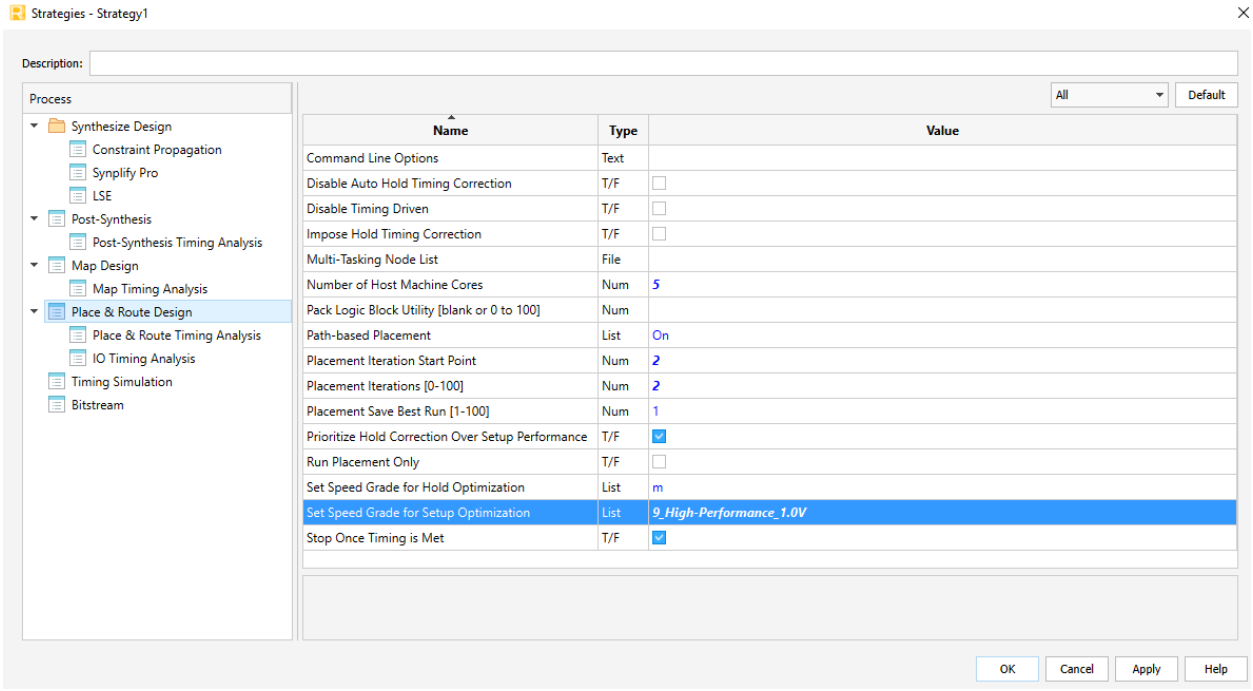

Figure 8.55. Strategy Used for CertusPro-NX GSRD Testing

11. Click on the **Green Run All** icon to generate a bit file. Wait for the bitstream generation and check the logs.

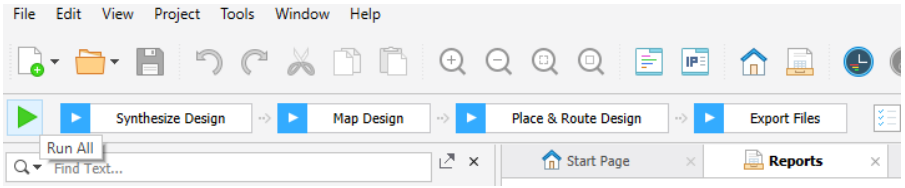

Figure 8.56. Generating the Bit File

12. The compilation flow takes some time to complete. The .bit file is generated/updated in the project path (<root_directory>/soc_primary_gsrd/ impl_1) after compilation is completed successfully as shown in Figure 8.57.

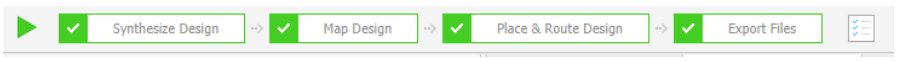

Figure 8.57. Successful Radiant Flow and Bitstream Generation

Note: If you observe timing violations during Lattice Radiant compilation, this may be due to the Placement Iteration Point number for Place and Route not working optimally on your machine. In this case, perform the following steps.

a. In Lattice Radiant software, go to **Project > Active Strategy > Place & Route Design Settings**.
b. Change Placement Iterations from 1 to 5.
c. Change Placement Save Best Run from 1 to 5.
d. Click **OK**.
e. Click on **Export Files**.

13. This reruns the Place & Route Design with five different incremental start point values. The Place & Route Reports shows all the five placement results and selects the best timing result.

8.5. Generating the Multi-Boot MCS File

To generate the multi-boot MCS file, perform the following steps:

Note: Follow these steps only when you have or re-created both Golden and Primary bitstreams.

1. Launch the Lattice Programmer from Lattice Radiant as shown in Figure 8.58.

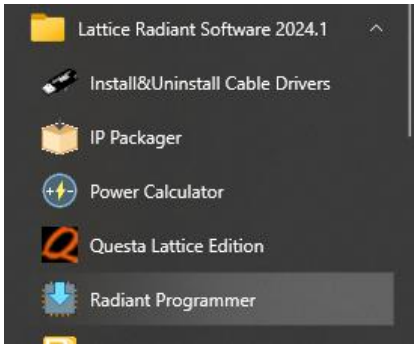

Figure 8.58. Launch Radiant Programmer from Windows Start

2. Provide the location of where you like to store the programmer .xcf file. Click **OK**.

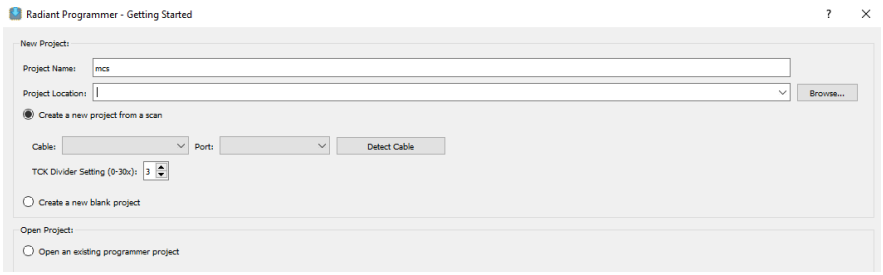

Figure 8.59.Radiant Programmer Getting Started Window

Note: This displays the error message shown in Figure 8.60 since there is no HW board connected. This error message can be ignored.

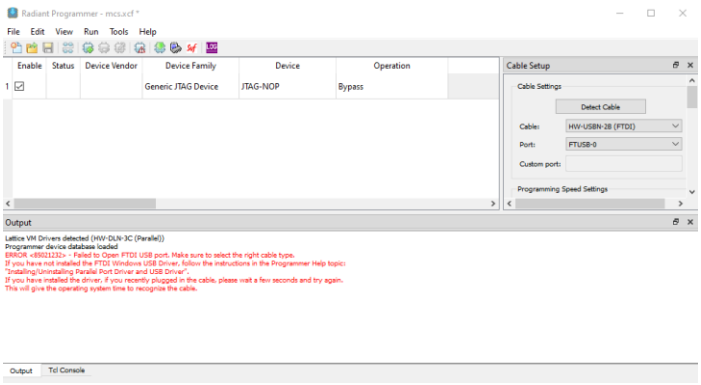

Figure 8.60.Error if No HW Board is Connected

3. Click **Tools > Deployment Tool**.

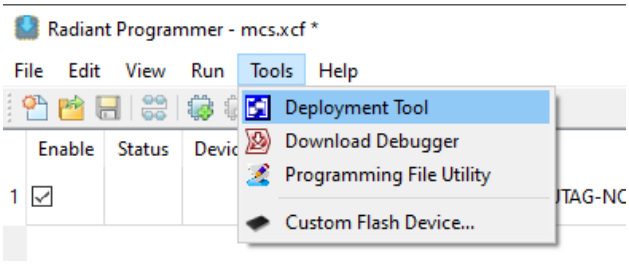

Figure 8.61.Open Deployment Tool from Radiant Programmer

4. This opens the Deployment Tool window as shown in Figure 8.62.

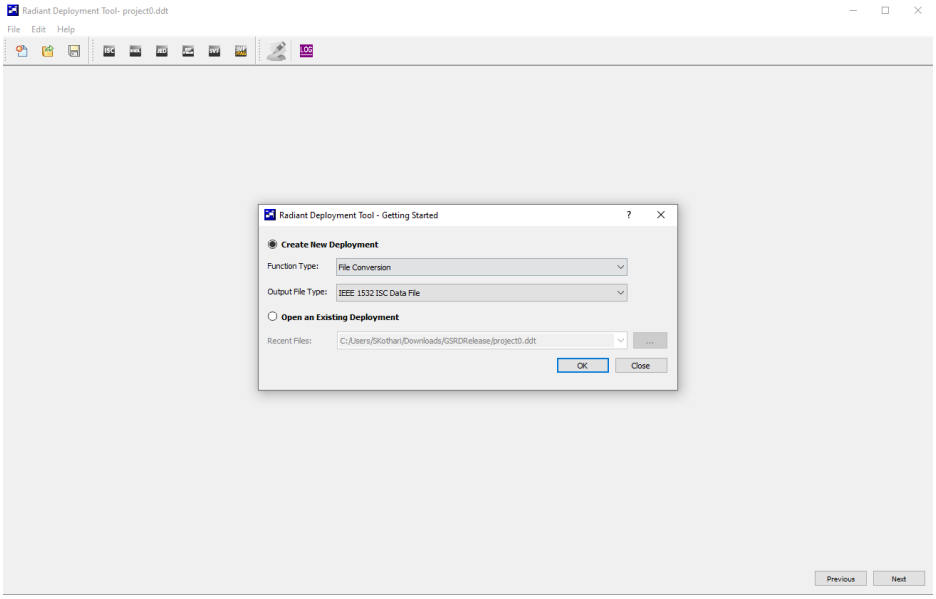

Figure 8.62.Deployment Tool Start window

5. Apply the settings shown in Figure 8.63 and click **OK**.

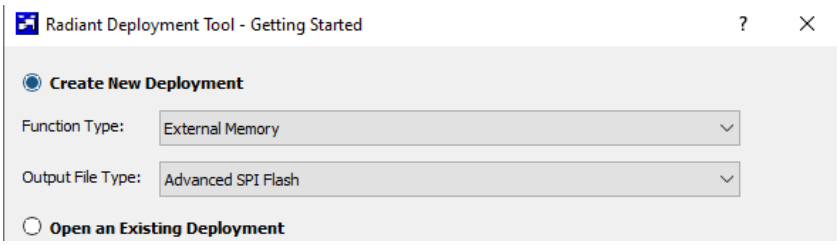

Figure 8.63. Options for Creating New Deployment

6. Step 1 of 4: Select the Input File(s) window, as shown in Figure 8.64:
   a. Click the **File Name field** to browse and select the **primary gsrd's .bit** from your primary soc project.
   b. The Device Family and Device fields auto populates based on the bitstream .bit file selected.
   c. Click **Next**.

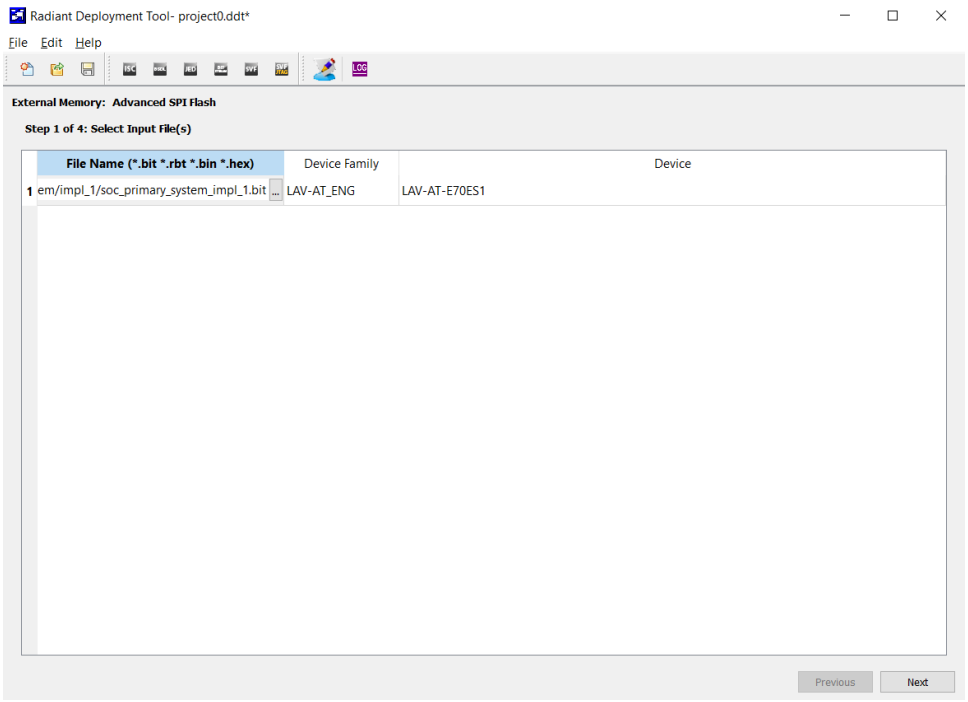

Figure 8.64. External Memory Step 1 of 4: Select Input Files

7. In Step 2 of 4: Select the fields as shown in Figure 8.65.
   a. Under Options tab: Choose Output Format as **Intel Hex** and **SPI Flash Size (Mb) as 512**.

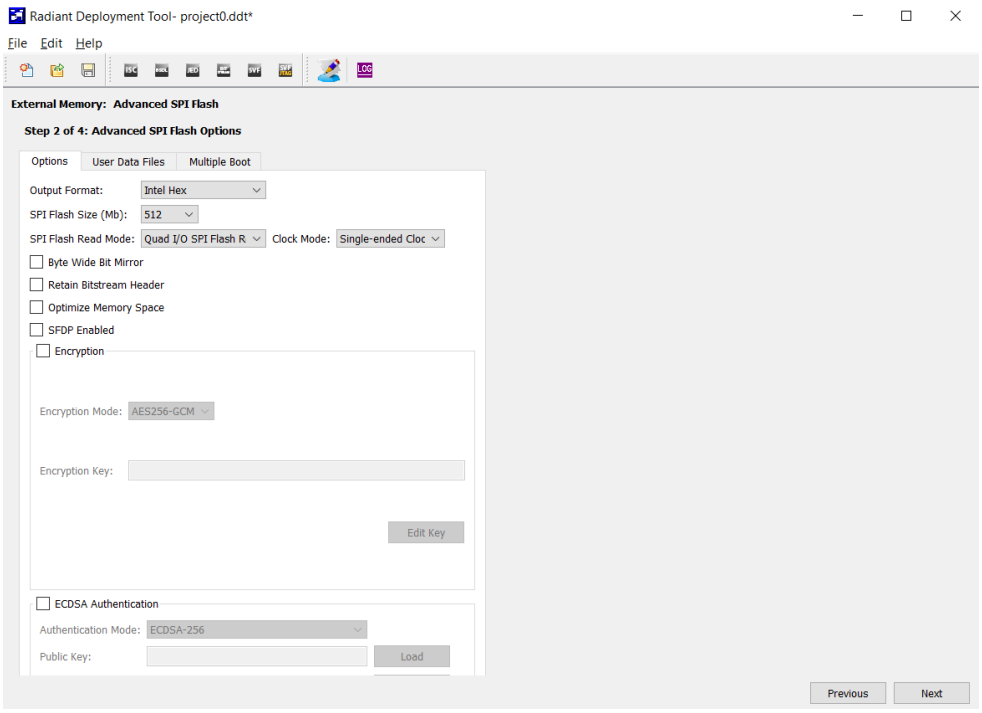

Figure 8.65. External Memory Step 2 of 4: Select Options

b. No changes needed in User Data Files section.

c. Under Multiple Boot tab.

   i. Select the Multiple Boot Option.

   ii. Select Number of Alternate Patterns as 1.

   iii. Under the Golden Pattern, browse and select the **Golden pattern bitstream** file. The Starting Address of Golden Pattern is automatically assigned. You can change it by clicking on the drop-down menu.

   iv. Under Alternate Pattern 1 field, click on the browse button and select the **golden soc gsrd's bitstream**. The Starting Address of this pattern is automatically assigned. You can change it by clicking on the drop-down menu. This is the pattern loaded during an event of next PROGRAMN/REFRESH or soft reset.

   v. Click **Next**.

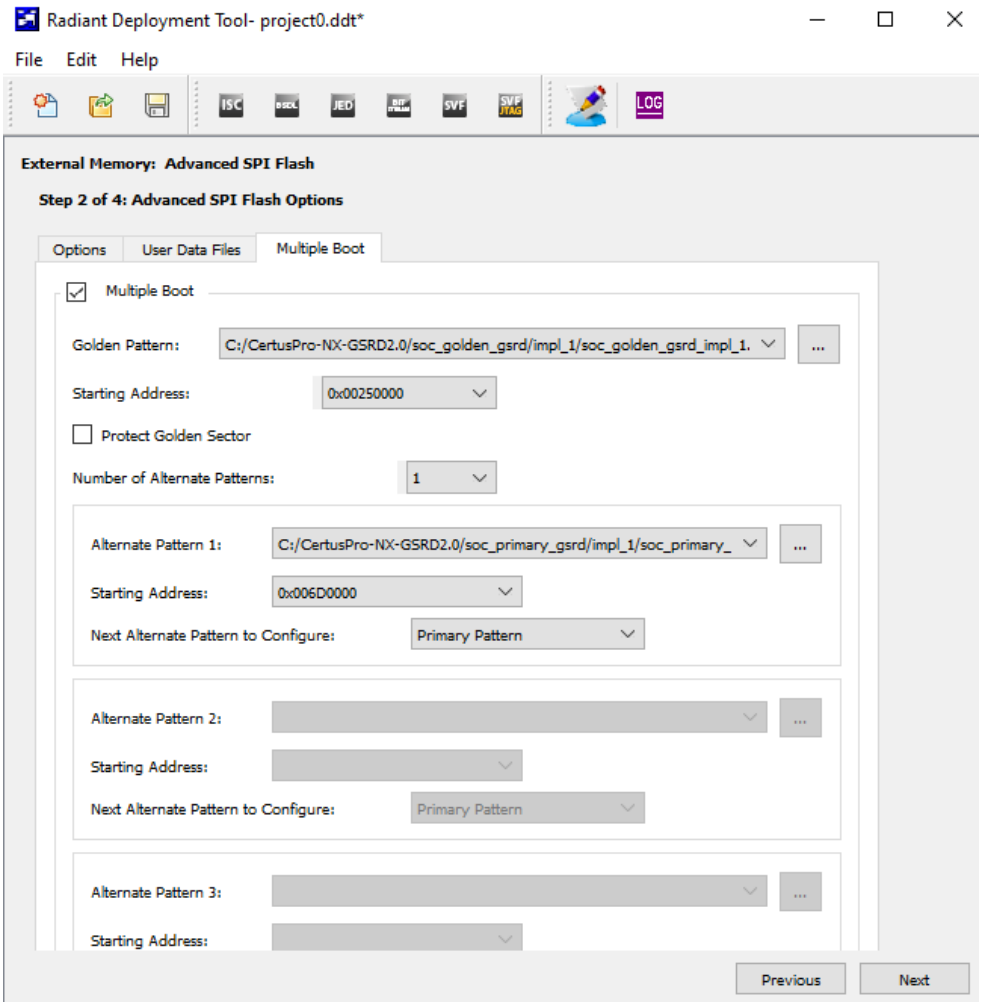

Figure 8.66. External Memory Step 2 of 4: Multi-Boot

8. In Step 3 of 4: Select the Output File(s) as shown in Figure. Choose the location on your machine to generate an .mcs file.

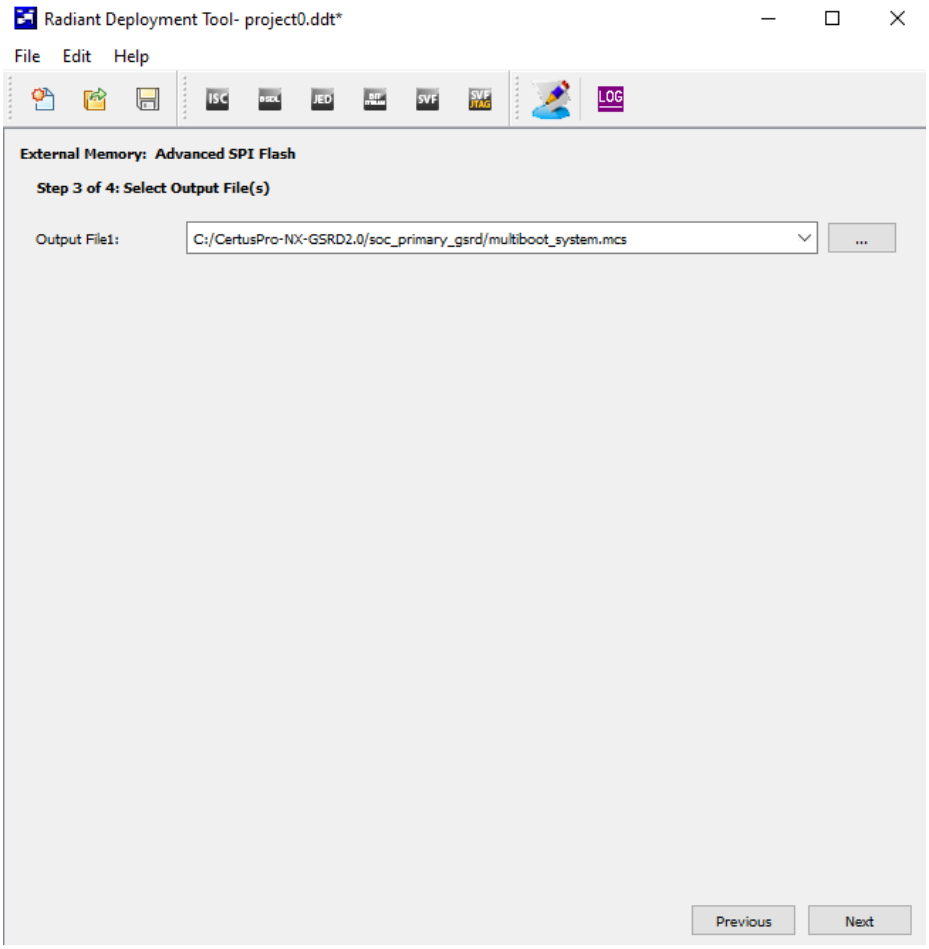

Figure 8.67. External Memory Step 3 of 4: Select Output File(s)

9. Click **Next**.
10. In Step 4 of 4: Generate Deployment, click **Generate** at the bottom right corner as shown in Figure 8.68.

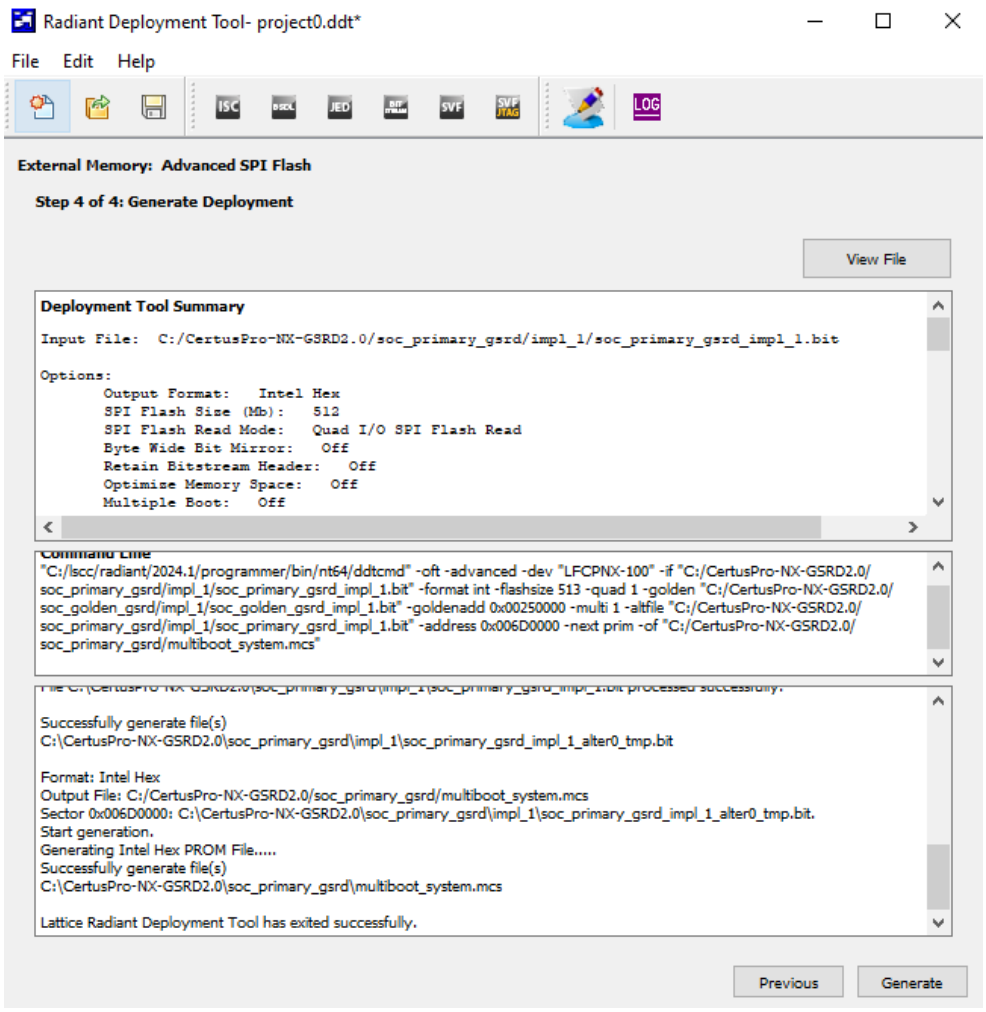

Figure 8.68. External Memory Step 4 of 4: General Development

11. Check for the following output as shown in Figure 8.69.

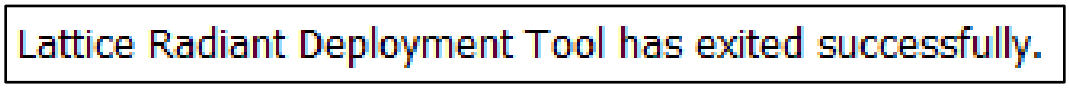

Figure 8.69. MCS File Generated Successfully

12. The generated final .mcs file is now ready to be programmed into the external flash using the Radiant Programmer.
13. Close the Deployment Tool window.

9. Customizing the IP in the GSRD Design

This section describes the customization that can be applied to IP in the reference design. The hardware related modifications are made using Propel Builder, since it is the main design entry tool. The software related modifications are made using Propel SDK.

The following demonstrates this using an example of QSPI Flash Controller IP.

9.1. Changing QSPI Flash Controller User Interface Parameters

The QSPI Flash Controller IP parameters are set according to the SPI Flash chips available on CertusPro-NX board. You can change the IP parameters to suit the board and Flash chip in your project.

To modify the IP parameters:

1. In Propel Builder, double-click on the **qspi0_inst** block.

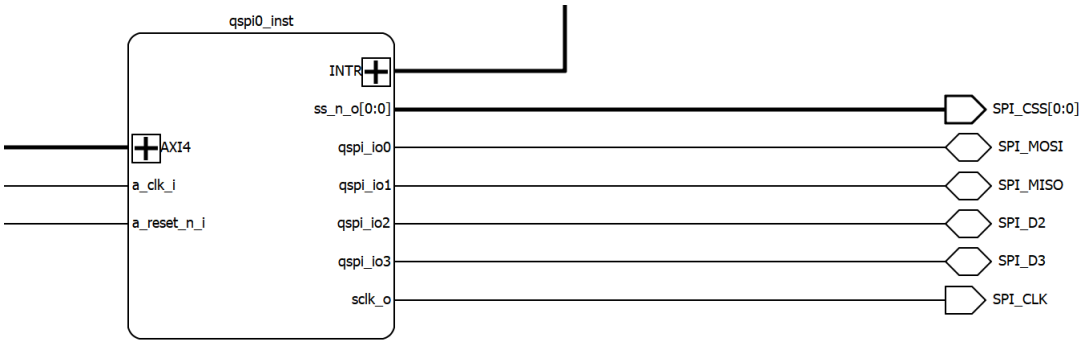

Figure 9.1. IP Block

2. In the Module/IP Block Wizard, make the desired changes to the parameters that suits your board and Flash chip. For example, you'd like to change any or all.
   - Data Endianness from Big Endian to Little Endian and
   - System Clock Frequency from 100 to 200

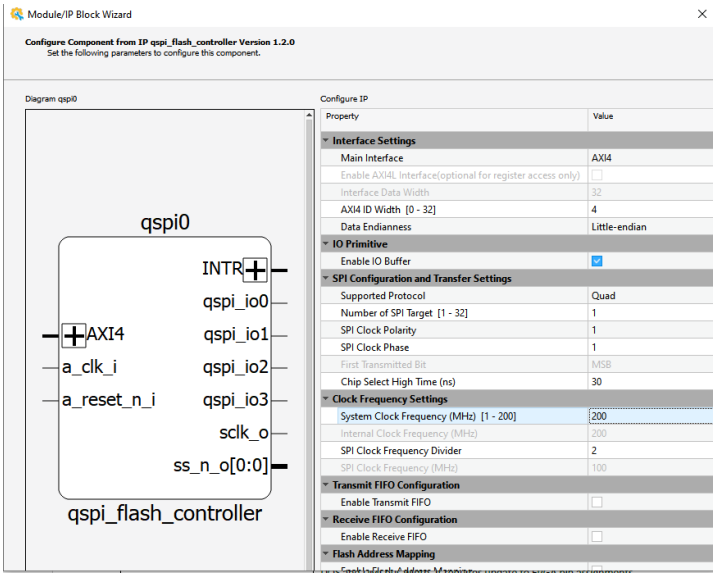

Figure 9.2. Customize IP

3. Click **Generate** to make sure RTL is updated as per the customization and must generate without an error.

4. You can see the below message in the TCL Console as shown in Figure 9.3. Ensure that there are no errors.

```
% sbp_config_ip -vlnv {latticesemi.com:ip:qspi0:1.2.0} -meta_vlnv {latticesemi.com:ip:qspi_flash_controller:1.2.0} -cfg_value {SPI_CLOCK_PHASE:1,SPI_CLOCK_POLARITY:1,SYSTEM_CLOCK_FREQUENCY:
200,data_endianness:Little-endian,enable_io_primitive:true,interface:AXI4,supported_protocol:Quad}
% sbp_replace -vlnv {latticesemi.com:ip:qspi0:1.2.0} -name {qspi0_inst} -component {soc_golden_system/qspi0_inst}
%
```

Figure 9.3. TCL Console Output After Desired Customization

5. Click the **Validate** button as shown in Figure 9.4. Ensure that there are no errors.

Figure 9.4. Validate Design Again if Any IP is Updated

6. Click the **Generate** button as shown in Figure 9.5. Ensure that there are no errors.

Figure 9.5. Generate Again if any IP is Updated

7. Once validated and generated, it updates the **sys_env.xml** file in the **sge** folder under the project directory as shown in Figure 9.6.

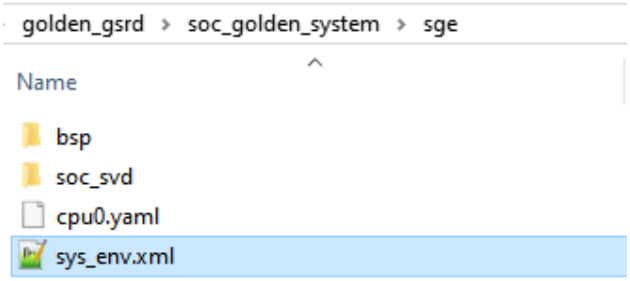

Figure 9.6. New sys_env.xml File Generated

8. If your IP is software driver dependent, the C projects must also be updated.
9. In your project under Propel SDK window, go to **Project > Update Lattice C/C++ Project**.

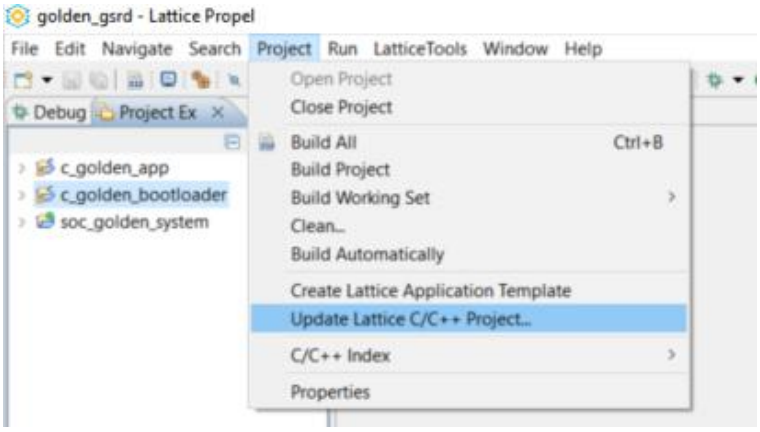

Figure 9.7. Update Lattice C/C++ Project

10. It opens the dialog box as shown in Figure 9.8.

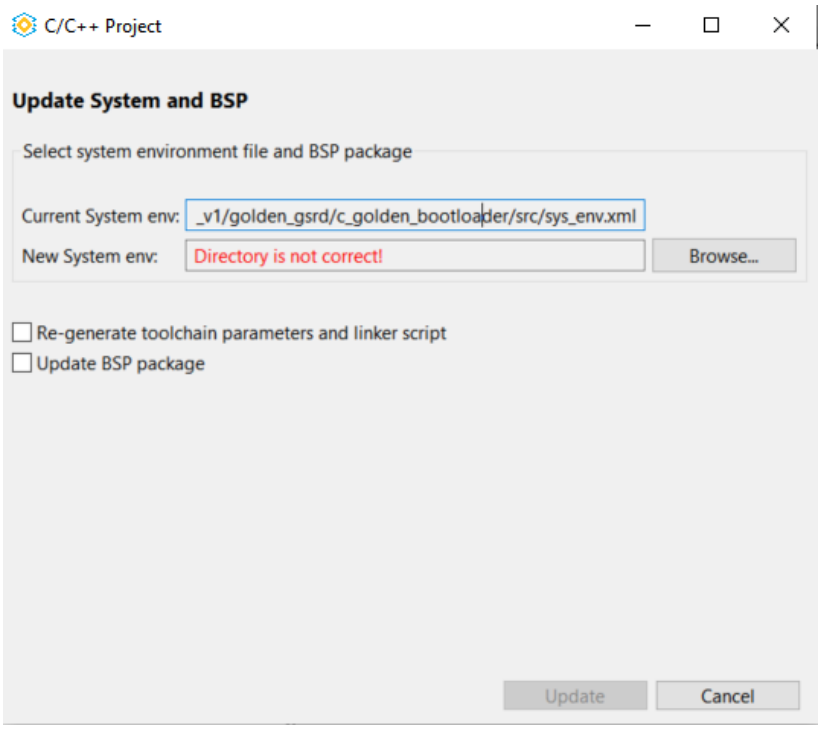

Figure 9.8. Update System Dialog Window

11. Click **Browse** and select the **sys_env.xml** file generated as shown in Figure 9.8.

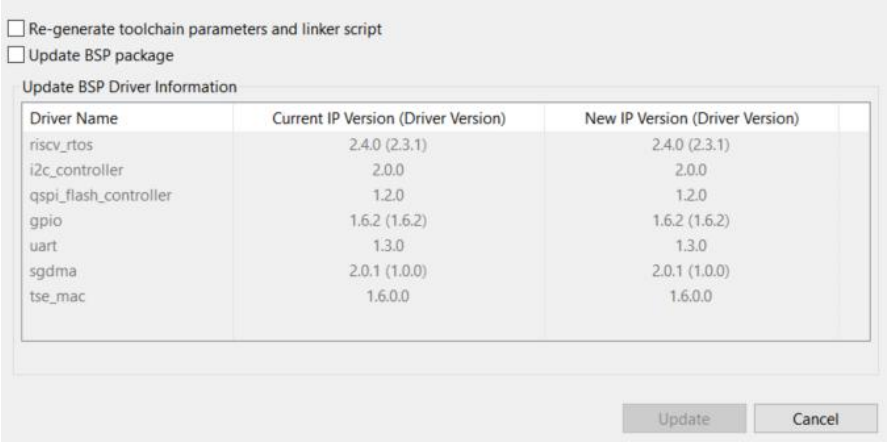

Figure 9.9. New IP version Details

12. Select *Re-generate toolchain parameters and linker script* and *Update BSP Package* and click **Update** to reflect the latest configurations.

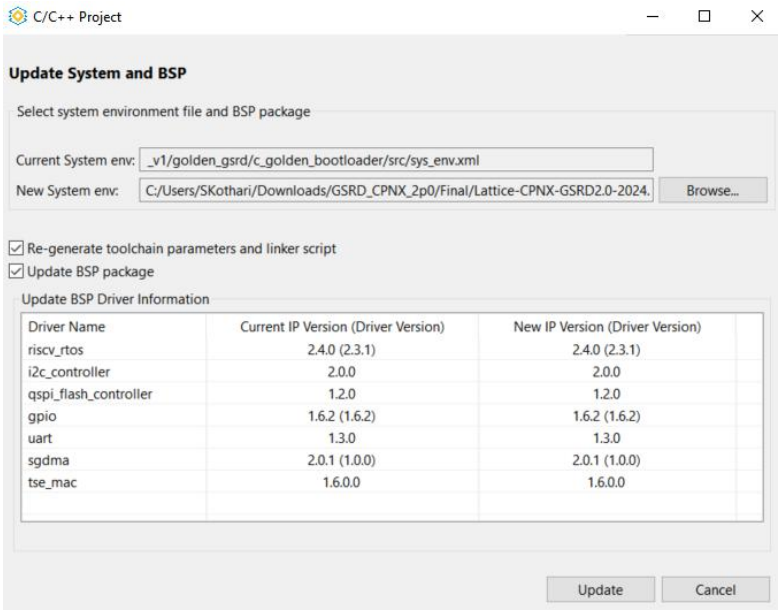

Figure 9.10. Generate BSP and Toolchain Parameters

13. Click **Yes** as shown in Figure 9.11 and the update window closes.

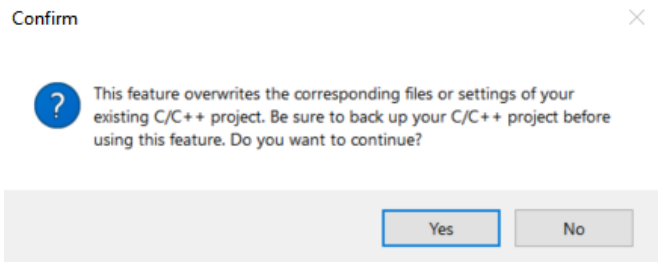

Figure 9.11. Confirm the Changes

14. Confirm that the BSP package was updated successfully. You can check it by reviewing the following:
    - **sys_platform.h** file to review the updated parameter change.

```
#define QSPI0_INST_SYSTEM_CLOCK_FREQUENCY 200
#define QSPI0_INST_DATA_ENDIANNESS Little-endian
```

Figure 9.12. Confirm the Configuration Changes in sys_platform.h

15. Clean and Build the Project.
16. If you hit any errors, it could be one of the following reasons:
    - Driver files has been updated.
    - Addresses in your C project might have changed.
17. Do the same XML updates for **c_golden_app** FreeRTOS project too.
18. Clean and Re-Build the Project.
19. Once new .mem file and binaries are created, follow the steps in the Validating and Generating the GSRD Design using Propel Builder section. Validating and Generating the GSRD design using Propel Builder.

10. Compiling U-Boot Bootloader

The U-Boot Bootloader is an open-source bootloader that provides support for hardware initialization before booting to OSes.

This initializes CPU, LPDDR, and peripherals and can be used in replace of the custom bootloader.

This section describes the process of compiling the U-Boot bootloader for GSRD/GHRD Reference Design instead of using custom bootloader. Compilation is required to generate the necessary binary files from the source files. The compilation process involves the following setup to generate First-Stage Bootloader (SPL) and U-Boot Proper (SSBL) software executable files.

- Lattice Propel for Linux
- Lattice Radiant
- Linux Machine with Ubuntu setup (Ubuntu 22.04.4 LTS)
- Windows Machine for OpenOCD and GDB
- Git Access to the Lattice's U-Boot repository

10.1. Setup Ubuntu Machine and Git Access

To set up the Ubuntu machine and Git access, perform the following:

1. Install Ubuntu OS from the official website with the Long-Term support.
   - Tested version is Ubuntu 22.04.4 LTS.
2. To setup git, please follow the steps below:
   a. Generate ssh key pairs and attach to the github ssh-keygen. This creates a new SSH key, using the provided email as a label.
   ```
   $ ssh-keygen -t rsa -b 4096 -C "<email_addr>"
   ```
3. Generating public/private ALGORITHM key pair.
   a. Enter a file in which to save the key (/home/YOU/.ssh/id_ALGORITHM):[Press enter].
   b. Enter passphrase (empty for no passphrase): **[Type a passphrase]**.
   c. Enter same passphrase again: **[Type passphrase again]**.
4. Run the ssh-agent:
   ```
   $ exec ssh-agent bash
   ```
5. Add your SSH private key to the ssh-agent. If the key is created with a different name or adding an existing key that has a different name, replace *id_rsa* in the command with the name of the private key file.
   ```
   $ ssh-add ~/.ssh/id_rsa
   ```
6. Add the SSH public key to your account on GitHub. For more information, go to Adding a new SSH key to your GitHub account in the GitHub website.

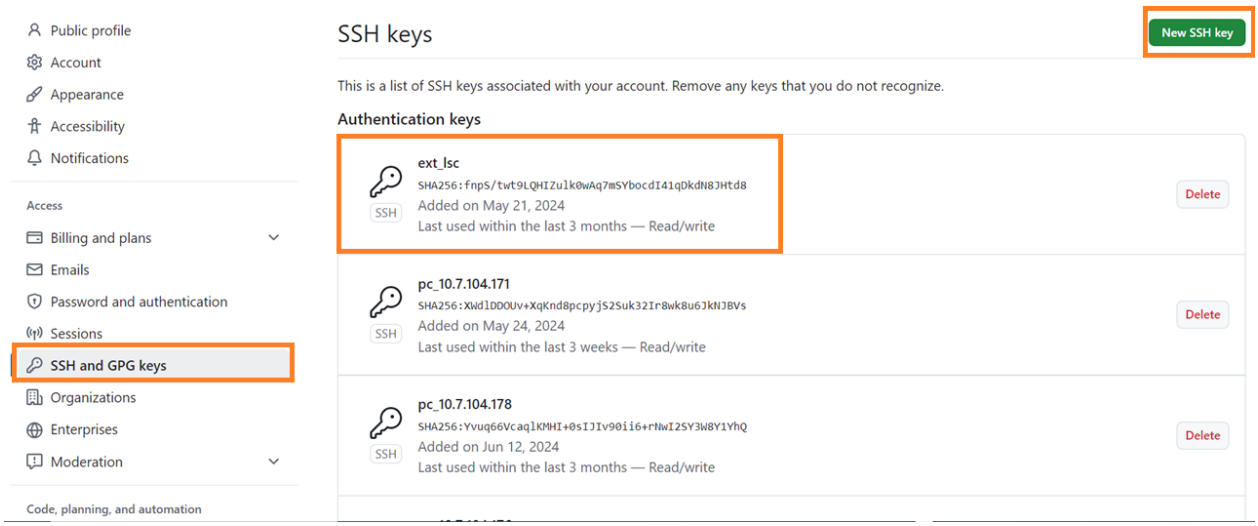

Figure 10.1. Add SSH keys into Github

7. Proceed to git clone the repository by running the script below.

```
$ git clone <github_link> -b lattice_v2024.04
```

The branch name is the current released branch in the release note: lattice_v2024.04.

The github link can be obtained in the Lattice U-Boot's repository page.

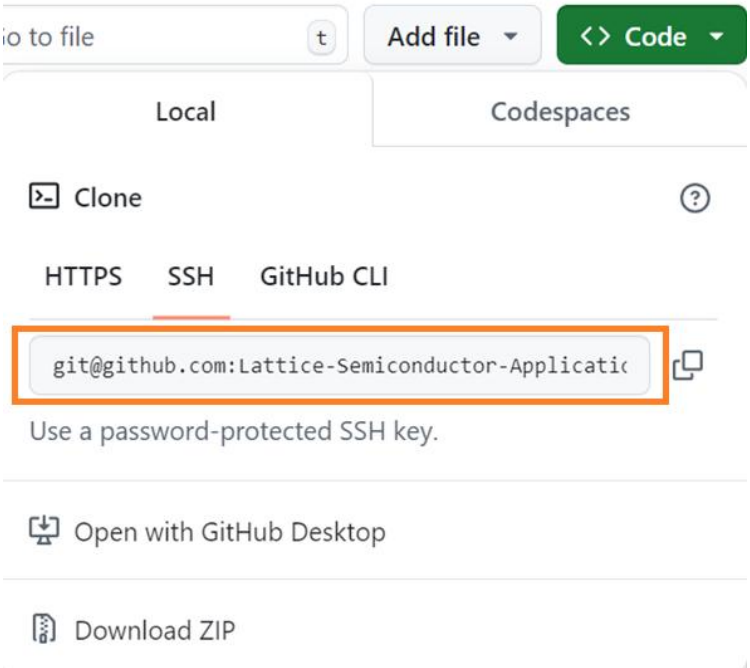

Figure 10.2. Copy U-Boot Link for Cloning

For example:

The authenticity of host 'github.com (140.82.116.3)' can't be established.

ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

8. Check the U-Boot repository is cloned to the local directory.

10.2. Building the U-Boot Bootloader

To build the U-Boot Bootloader, perform the following:

1. For first-time setup, install the following on the Ubuntu Machine (required sudo access)
```
$ sudo apt install bison flex python3-setuptools python3-pip swig
$ sudo apt-get install libncurses-dev
```
2. Install libssl-dev if there is openssl error:
```
$ sudo apt-get install libssl-dev
```
3. Install Lattice Propel for Linux.
4. Set the environment, refer to the propel installation path:
```
$ export PATH=$PATH:<path_to_propel>/lscc/propel/2024.1/sdk/riscv-none-embed-gcc/bin
$ export CROSS_COMPILE="riscv-none-embed-"
$ export ARCH=riscv
```
5. Clean build U-Boot:
```
$ make mrproper && make lattice_riscv_defconfig
```
6. Build u-boot:
```
$ make -j8
```
7. Once successfully build, spl/u-boot-spl-dtb.bin and u-boot.bin are generated.

```
pc@pc-desktop:~/kj/u-boot-rebase/u-boot$ ls u-boot u-boot.bin spl/u-boot-spl spl/u-boot-spl.bin
spl/u-boot-spl  spl/u-boot-spl.bin  u-boot  u-boot.bin
```

Figure 10.3. U-Boot Binaries and Symbols Generated

8. Check the u-boot symbol file to cross-check if it is cross-compiled correctly using RISC-V compiler:

```
pc@pc-desktop:~/kj/u-boot-rebase/u-boot$ readelf -h u-boot
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x80100000
  Start of program headers:          52 (bytes into file)
  Start of section headers:          4424324 (bytes into file)
  Flags:                             0x1, RVC, soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         27
  Section header string table index: 26
pc@pc-desktop:~/kj/u-boot-rebase/u-boot$ readelf -h u-boot.bin
```

Figure 10.4. Check if U-Boot Binary Cross-Compiled Correctly

10.3. QSPI Configuration for U-Boot

To configure QSPI for U-Boot, perform the following:

1. For QSPI Configuration, select Winbond or Macronix using menuconfig (Default set to Winbond):
   ```
   $ make menuconfig
   ```
2. Press **'/'** to search for CONFIG.

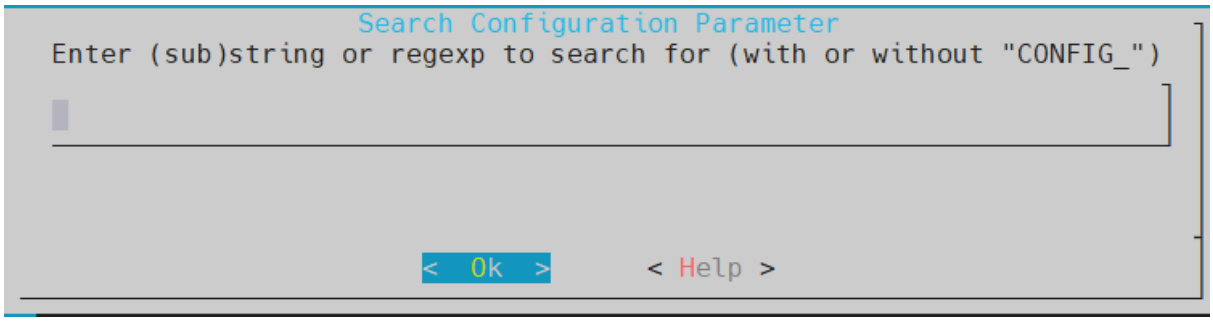

Figure 10.5. Menuconfig – Search for QSPI CONFIG

3. Type in the CONFIG name, CONFIG_SPI_FLASH_MACRONIX or CONFIG_SPI_FLASH_WINBOND > OK
4. Press space and select **Winbond QSPI (Default)**.

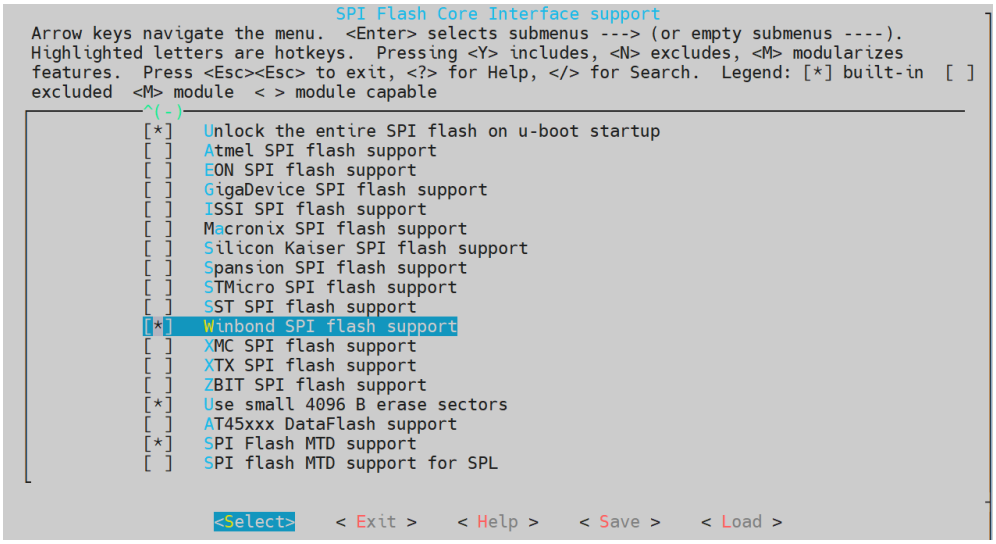

Figure 10.6. Menuconfig – Press Space to select Winbond QSPI

You can also press space and select Macronix QSPI > Press Space > Unselect Winbond.

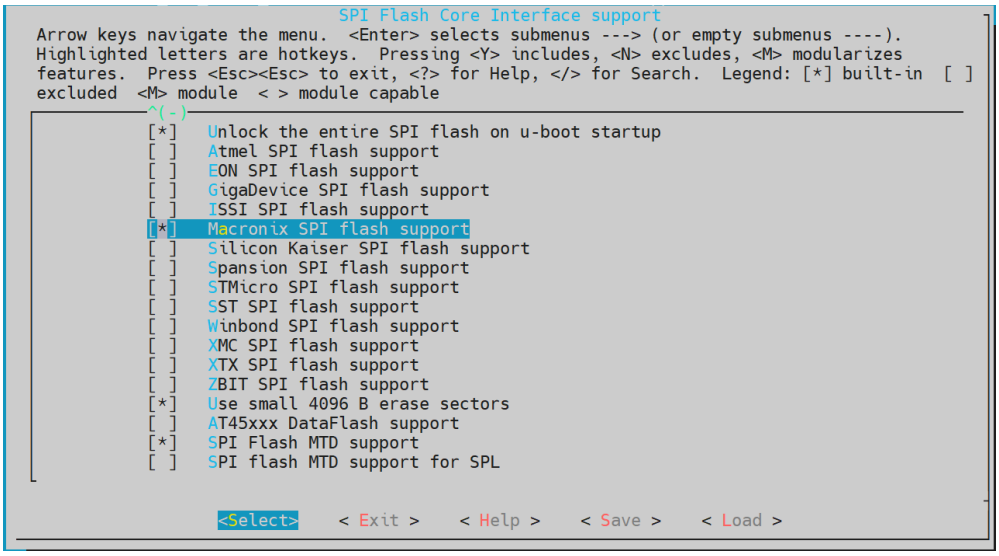

Figure 10.7. Menuconfig - Select Macronix QSPI

5. Select **Save > Ok > Exit** to save the .config file:

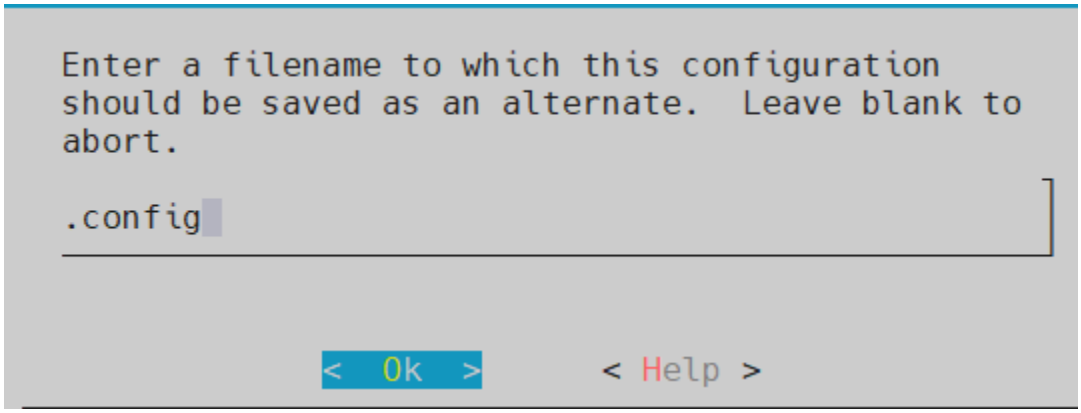

Figure 10.8. Menuconfig - Save > Ok > Exit to save the .config file

6. Once you have exited from the Menuconfig, re-build the U-Boot:

```
$ make -j8
```

10.4. Basic Boot-Up Flow with U-Boot Bootloader

Figure 10.9 shows the boot-up flow for U-Boot.

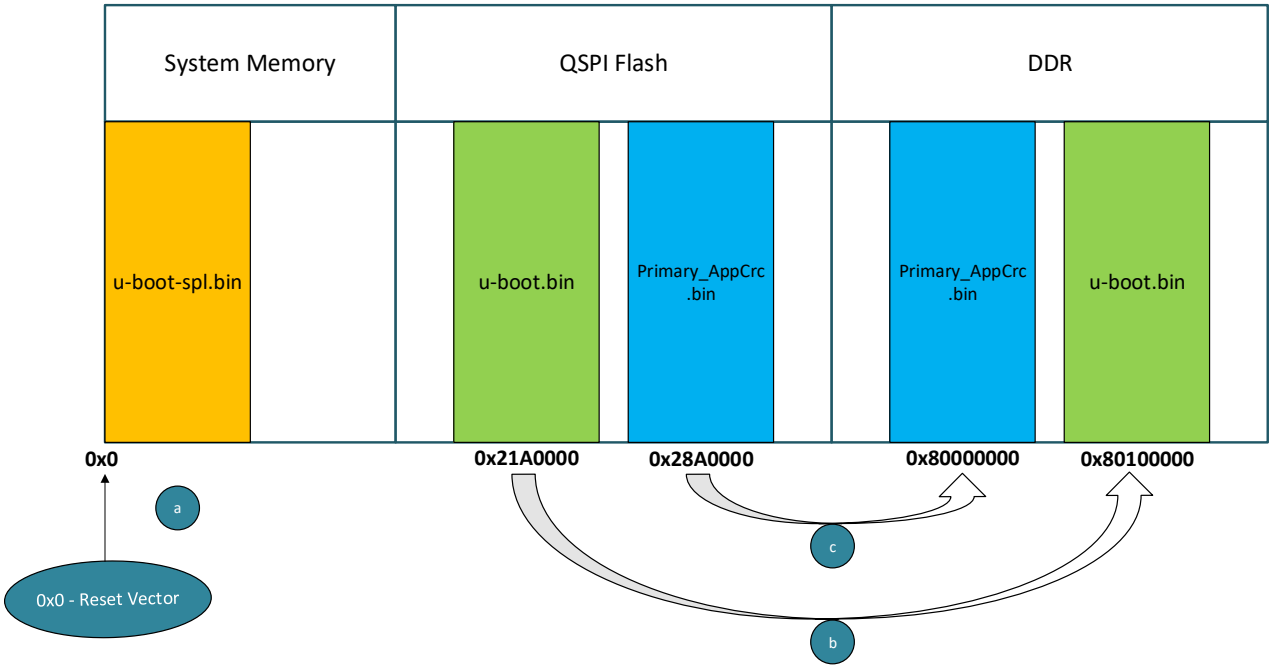

Figure 10.9. Boot-up Flow for U-Boot

- U-Boot SPL is run upon power up/loaded through OpenOCD.
- SPL copy U-Boot Proper from flash address 0x21A0000 to DDR address 0x80100000 and jump to 0x80100000.
- U-Boot Proper copy FreeRTOS application from flash address 0x28A0000 to DDR address 0x80000000 and jump to 0x80000000.

There are two methods to boot-up with U-Boot bootloader:

- Build SPL with the bitstream (refer to Running U-Boot Bootloader with SPL Built into Bitstream (First Method) section)
- Load SPL using OpenOCD (refer to Running U-Boot Bootloader with OpenOCD (Second Method) section)

10.5. Programming U-Boot Bootloader

To program the U-Boot Bootloader, perform the following:

1. Before booting with U-Boot bootloader, you need to first program the U-Boot second-stage bootloader and FreeRTOS to the QSPI flash.
2. Configure the TCK Divider settings to 4.

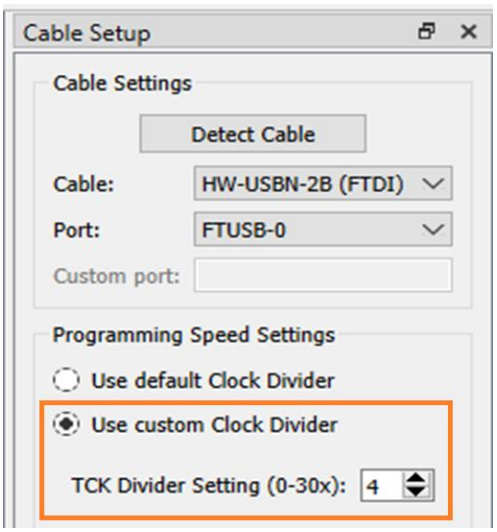

Figure 10.10. Set TCK Divider Settings to 4 for Flash Programming

3. Program *u-boot.bin* to the address 0x21a0000.

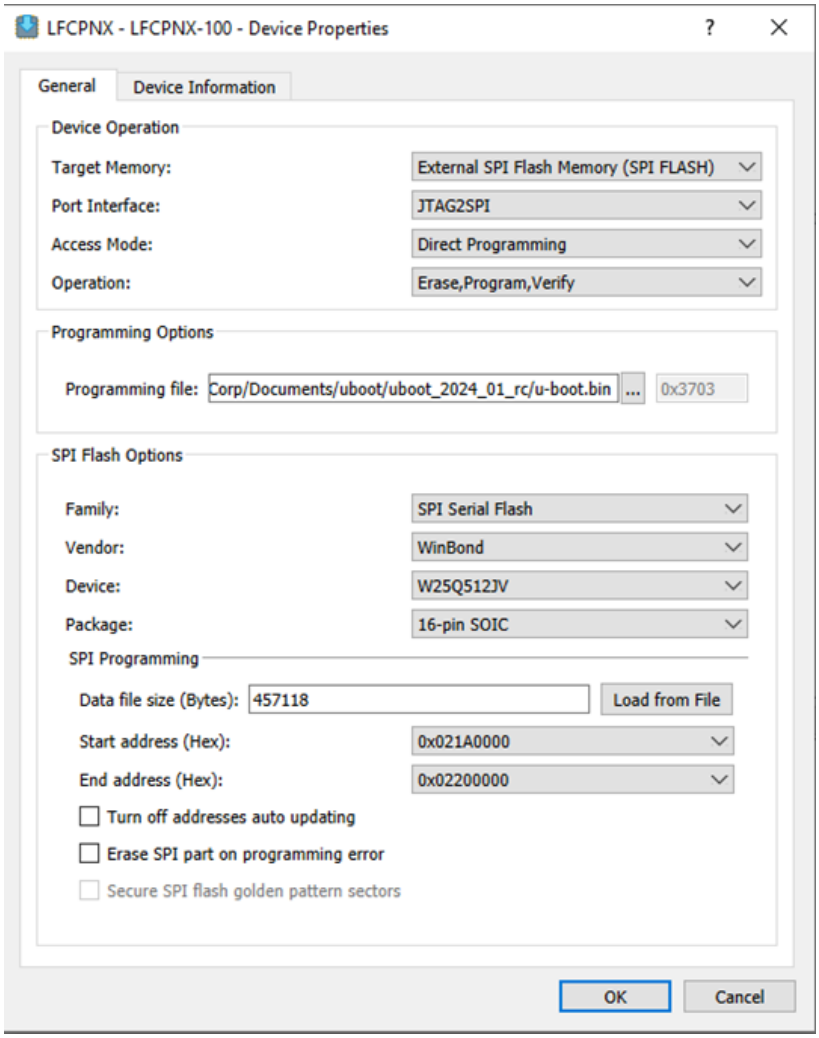

Figure 10.11. U-Boot SSBL – Winbond QSPI Programming

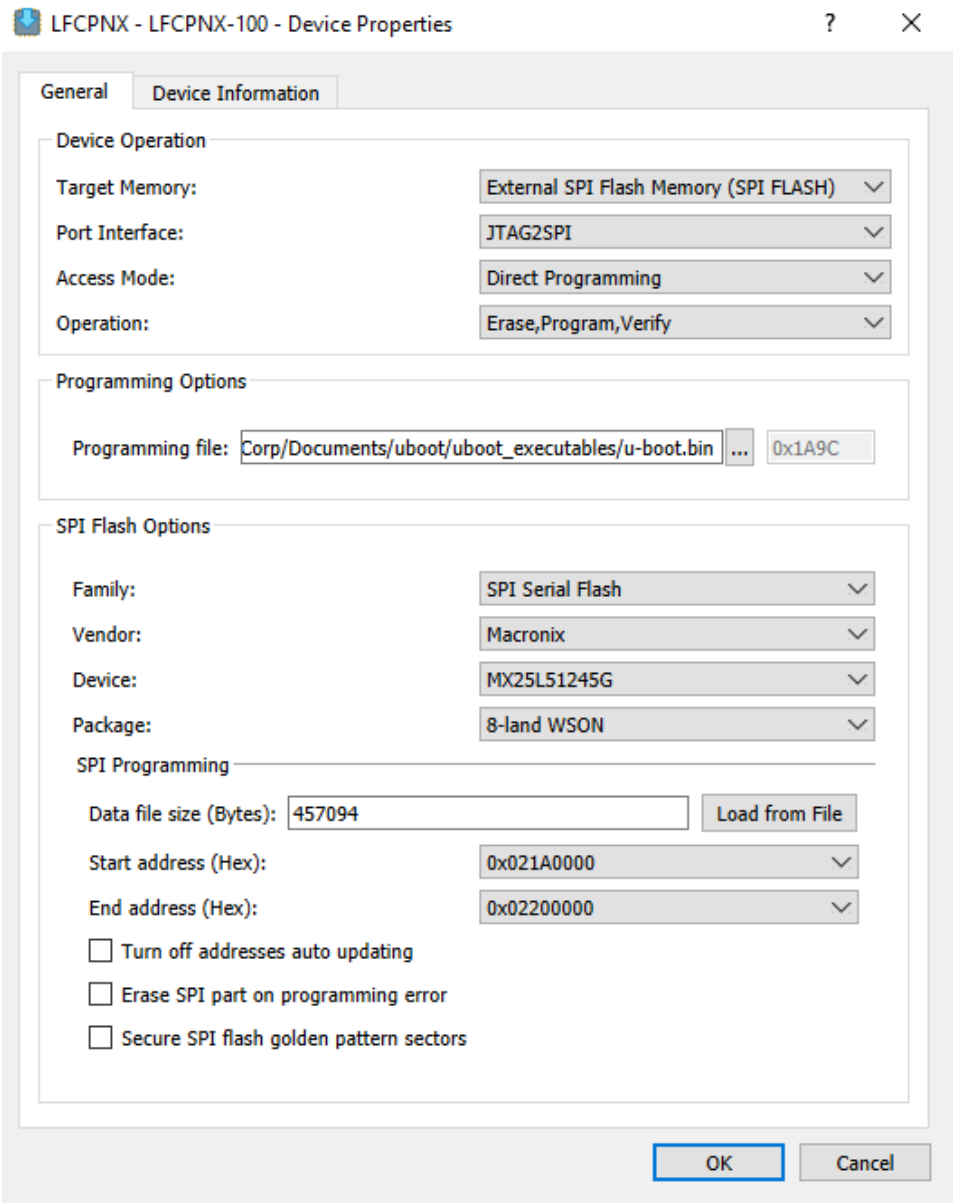

Figure 10.12. U-Boot SSBL - Macronix QSPI Programming

4. Program *c_primary_appcrc.bin* (FreeRTOS image) to the **0x28a0000** address.

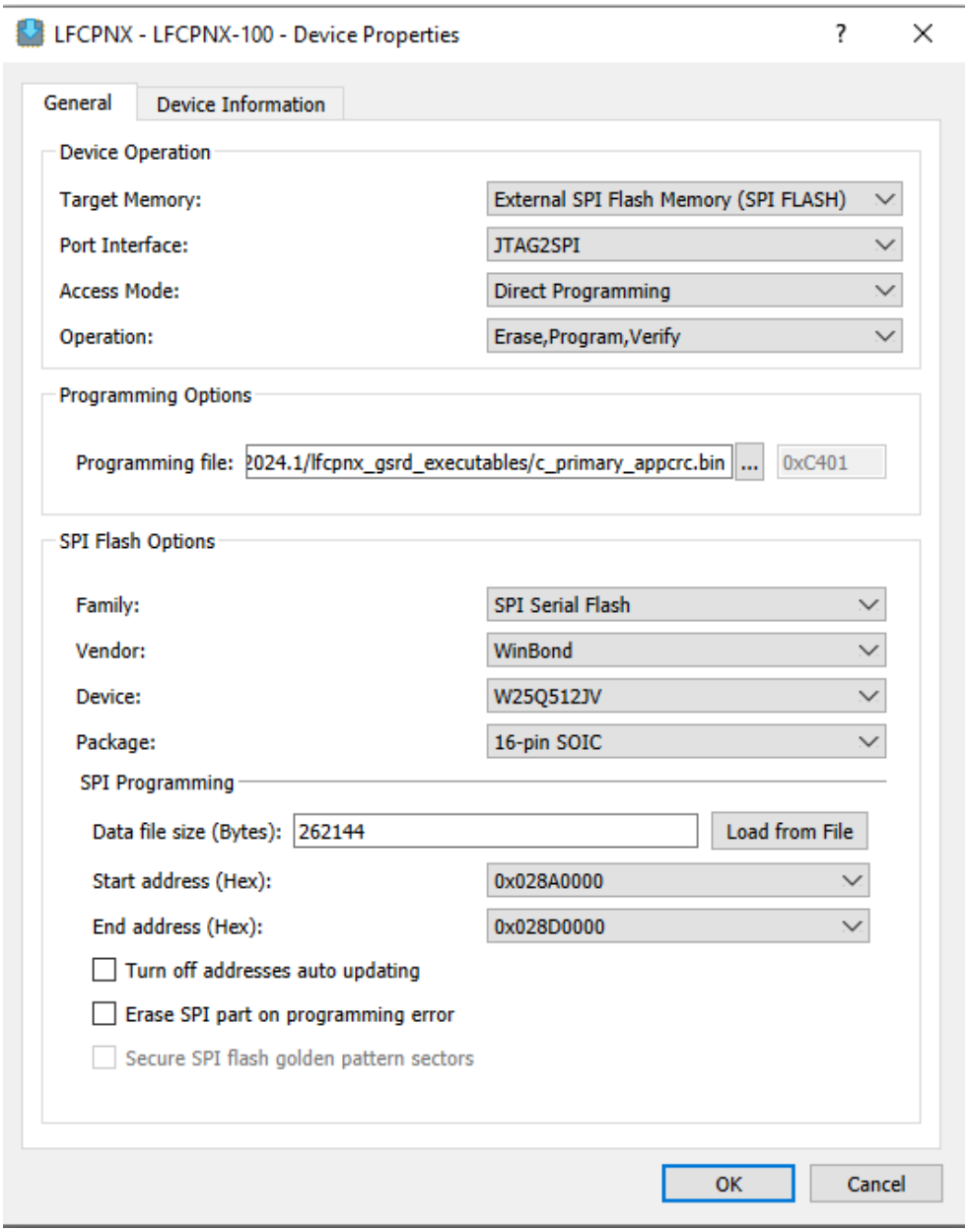

Figure 10.13. Primary_AppCrc – Winbond QSPI Programming

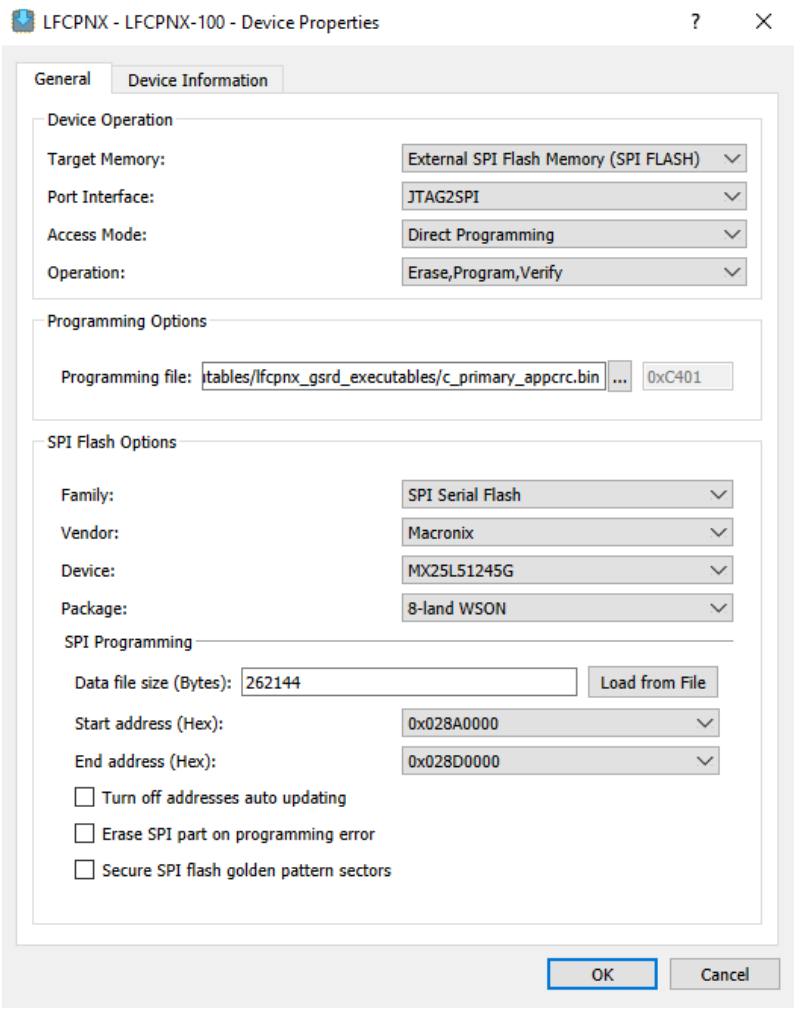

Figure 10.14. Primary_AppCrc – Macronix QSPI Programming

10.6. Running U-Boot Bootloader with SPL Built into Bitstream (First Method)

To run the U-Boot Bootloader with SPL, perform the following:

1. Build SPL with the bitstream.
2. Create u-boot-spl.mem in Ubuntu Machine:
   ```
   $ srec_cat "u-boot-spl.bin" -Binary -byte-swap 4 -DISable Header -Output "u-boot-spl.mem" -MEM 32
   ```
3. Go to **Propel Builder > Generate > Radiant Build** to replace hex in the sysmem.

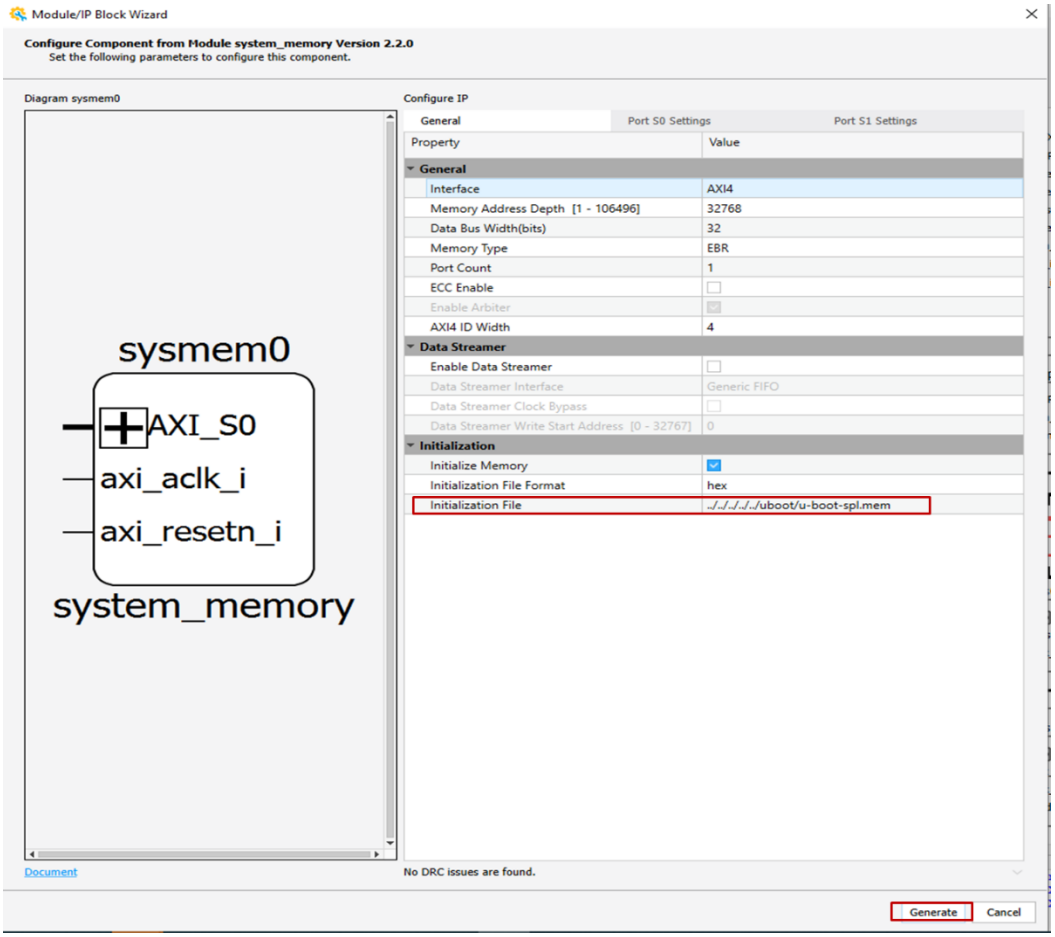

Figure 10.15. Configure sysmem with u-boot-spl.mem

4. Select **Design > Validate and Design > Generate**.

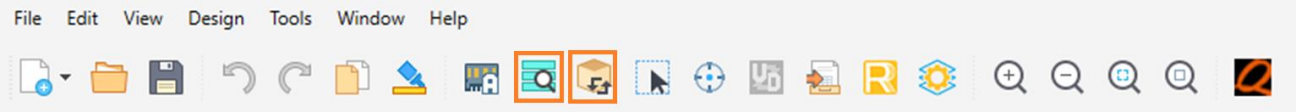

Figure 10.16. Validate and Generate the Design

```
Tcl Console
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi4_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi4_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi4_interconnect0_inst'.
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi4_interconnect2_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi4_interconnect2_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S00' in component 'axi4_interconnect2_inst'.
INFO <2357031> - Dangling input Port 'ARUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi4_interconnect2_inst'.
INFO <2357031> - Dangling input Port 'AWUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi4_interconnect2_inst'.
INFO <2357031> - Dangling input Port 'WUSER' is set to default value '0' on bus 'AXI_S01' in component 'axi4_interconnect2_inst'.
INFO <2357031> - Dangling input Port 'BUSER' is set to default value '0' on bus 'AXI4_M' in component 'axi4_regslice0_inst'.
INFO <2357031> - Dangling input Port 'RUSER' is set to default value '0' on bus 'AXI4_M' in component 'axi4_regslice0_inst'.
INFO <2357031> - Dangling input Port 'BUSER' is set to default value '0' on bus 'AXI_M00' in component 'mpmc0_inst'.
INFO <2357031> - Dangling input Port 'RUSER' is set to default value '0' on bus 'AXI_M00' in component 'mpmc0_inst'.
INFO <2357033> - Dangling input Portbus 'TDEST' is set to default value '4'h0' on bus 'AXI4S_RX' in component 'sgdma0_inst'.
INFO <2357033> - Dangling input Portbus 'TID' is set to default value '4'h0' on bus 'AXI4S_RX' in component 'sgdma0_inst'.
INFO <2359137> - Finished: sbp_design drc.
%
```

Figure 10.17. Design > Validate the Design

```
% sbp_design generate
INFO <2359189> - Start: sbp_design generate.
INFO <2359190> - Finished: sbp_design generate.
% C:/Users/kjlee/OneDrive - Lattice Semiconductor Corp/Documents/GSRD/Lattice-LCPNX-100-GSRD2.0-Beta-2024.1/standalone_c_soc_projects/soc_primary_gsrd_cpnx/sge/soc_svd'
% sbp_design save
% sbp_design pge sge  -i (C:/Users/kjlee/OneDrive - Lattice Semiconductor Corp/Documents/GSRD/Lattice-LCPNX-100-GSRD2.0-Beta-2024.1/standalone_c_soc_projects/soc_primary_gsrd_cpnx/soc_primary_gsrd_cpnx/soc_primary_gsrd_cpnx.sbx) -o (C:/
Users/kjlee/OneDrive - Lattice Semiconductor Corp/Documents/GSRD/Lattice-LCPNX-100-GSRD2.0-Beta-2024.1/standalone_c_soc_projects/soc_primary_gsrd_cpnx/soc_primary_gsrd_cpnx/../) -nc
WARNING <2359991> - (PGE SocDesign) sgmii0_inst_LDDI remains unconnected
```

Figure 10.18. Design > Generate the Design

5. Open **Radiant > Build**.

Figure 10.19. Open Radiant and Trigger build

6. Once Radiant build is done, program the bitstream to SRAM.

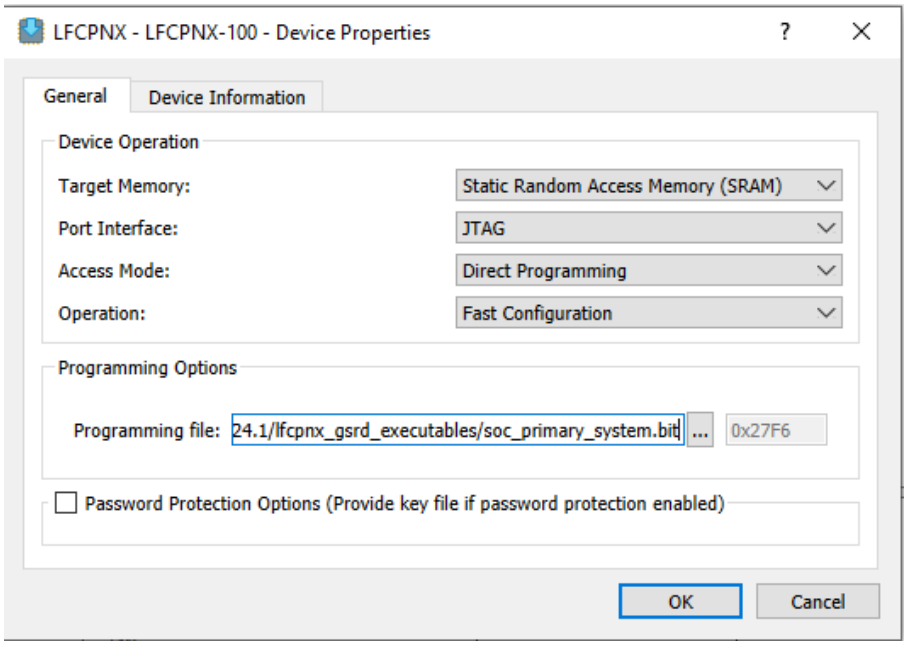

Figure 10.20. Program the Bitstream with SPL Built-In

7. Open the terminal and U-Boot is booted on the UART terminal.

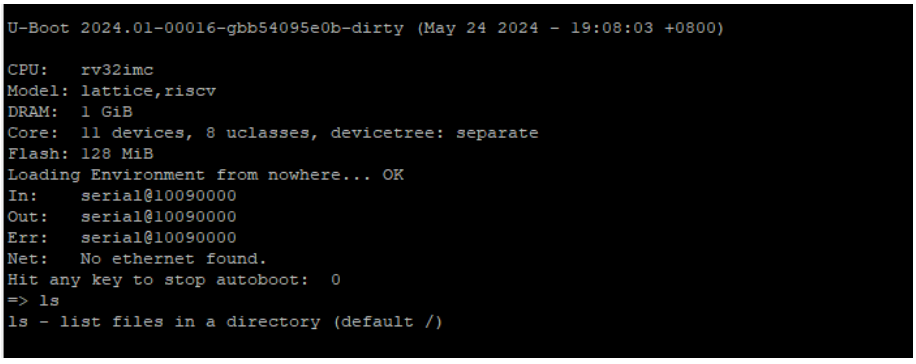

Figure 10.21. U-Boot Booted-up on UART Terminal

8. Once U-Boot is up, load primary_appcrc.bin from QSPI and jump to FreeRTOS:

```
$ mtd list
$ mtd read lattice_qspi 0x80000000 0x28a0000 0x40000
$ go 0x80000000 0x40000
```

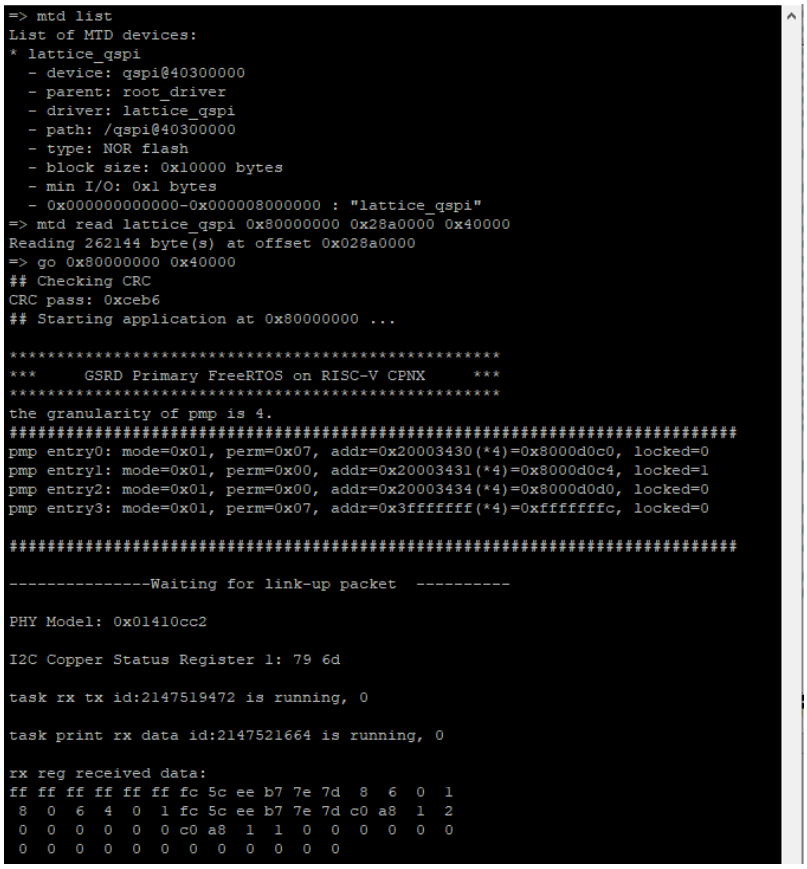

Figure 10.22. FreeRTOS is Booted-Up on UART Terminal

10.7. Running U-Boot Bootloader with OpenOCD (Second Method)

To run the U-Boot Bootloader with OpenOCD, perform the following:

1. Open the command prompt in Windows and run **cd <openOCD directory>**.
   For example:

```
cd C:\lscc\propel\2024.1\sdk\openocd\bin.
```

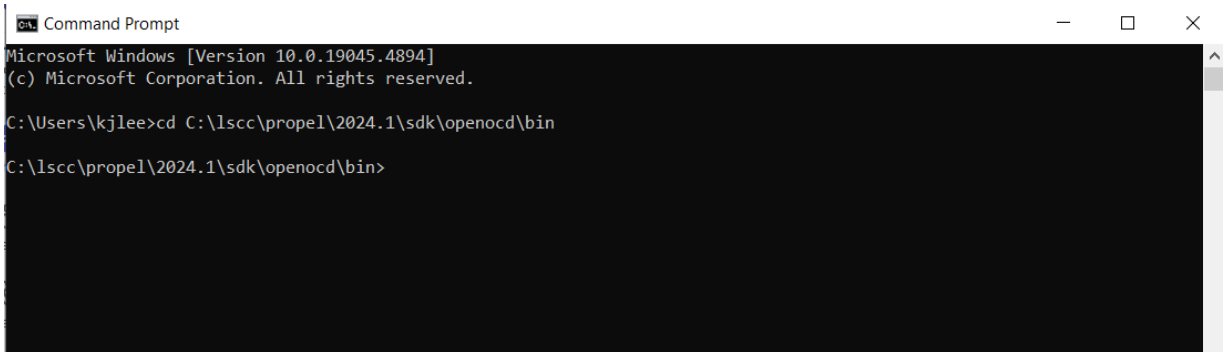

Figure 10.23. OpenOCD Directory

2. Create *cpnx.cfg* inside the directory:

```
reset init
# uncomment below for uboot ssbl
# mww 0x40007004 0x3
sleep 8
load_image u-boot-spl.bin 0x0 bin
# uncomment below for uboot ssbl
# load_image u-boot.bin 0x80100000 bin
```

3. Copy the u-boot binary to the local user directory, you need the following uboot binaries:

```
- u-boot
- u-boot.bin
- u-boot-spl
- u-boot-spl.bin
```

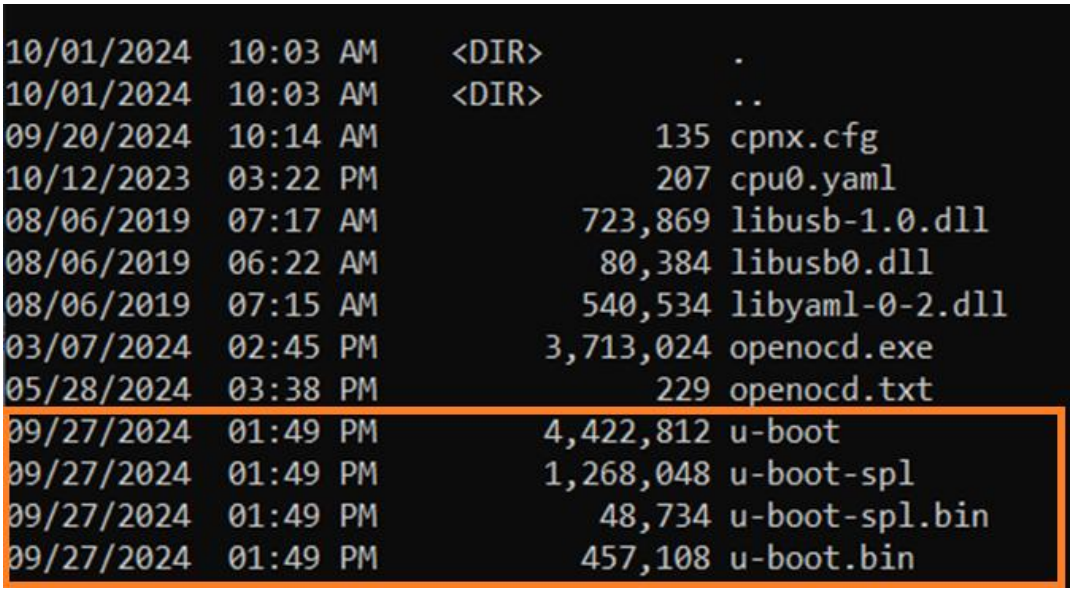

Figure 10.24. Copy the U-Boot Binaries to the OpenOCD Directory

4. Flash the primary bitstream (either bitstream with SPL or custom bootloader) using Radiant programmer to the board first, to boot up with GSRD designs.
5. Once board is booted, run this in cmd prompt:

```
$ openocd.exe -c "set target 0" -c "set tck 5" -c "set port FTUSB-0" -c "set channel
14" -c "set cmdlength 0" -c "set loc 0" -f interface/lattice-cable.cfg -c "set
RISCV_SMALL_YAML {cpu0.yaml}" -f target/riscv-small.cfg -f <local_path>/cpnx.cfg
```

6. Wait until the TCL and telnet connection is up.

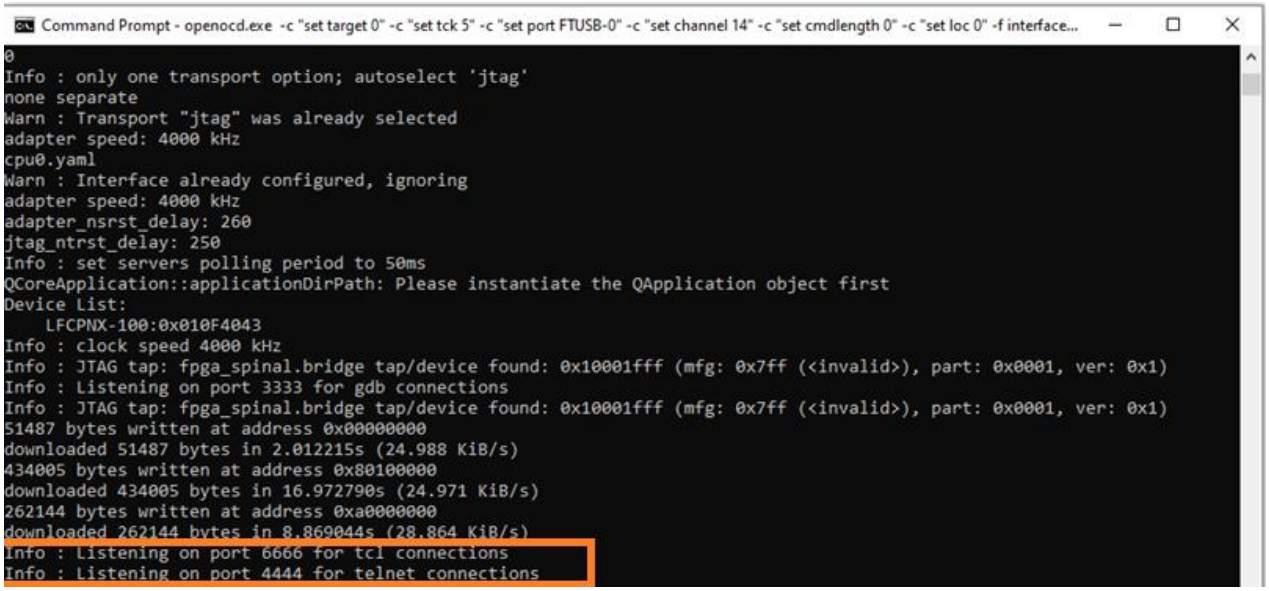

Figure 10.25. OpenOCD is Connected

7. Open another cmd prompt, go to gdb path (for example, C:\lscc\propel\2024.1\sdk\riscv-none-embed-gcc\bin). Create the spl.gdb and uboot.gdb scripts.

   spl.gdb Script

   ```
   set remotetimeout 100
   set arch riscv:rv32
   set mem inaccessible-by-default off
   set can-use-hw-watchpoints 0
   tar rem :3333
   symbol-file u-boot-spl
   set $pc=0x0
   ```

   uboot.gdb Script

   ```
   set remotetimeout 100
   set arch riscv:rv32
   set mem inaccessible-by-default off
   set can-use-hw-watchpoints 0
   tar rem :3333
   symbol-file u-boot
   set $pc=0x80100000
   ```

8. Once TCL and telnet is connected, go to *gdb path* (for example, C:\lscc\propel\2024.1\sdk\riscv-none-embed-gcc\bin), run the script below:

   ```
   $ riscv-none-embed-gdb.exe
   # To load spl first, run source spl.gdb
   # To load U-Boot, run source uboot.gdb. Change the cpnx.cfg accordingly in step 2
   $ source uboot.gdb <- wait for _start() func to show, run 2 times
   $ continue
   ```

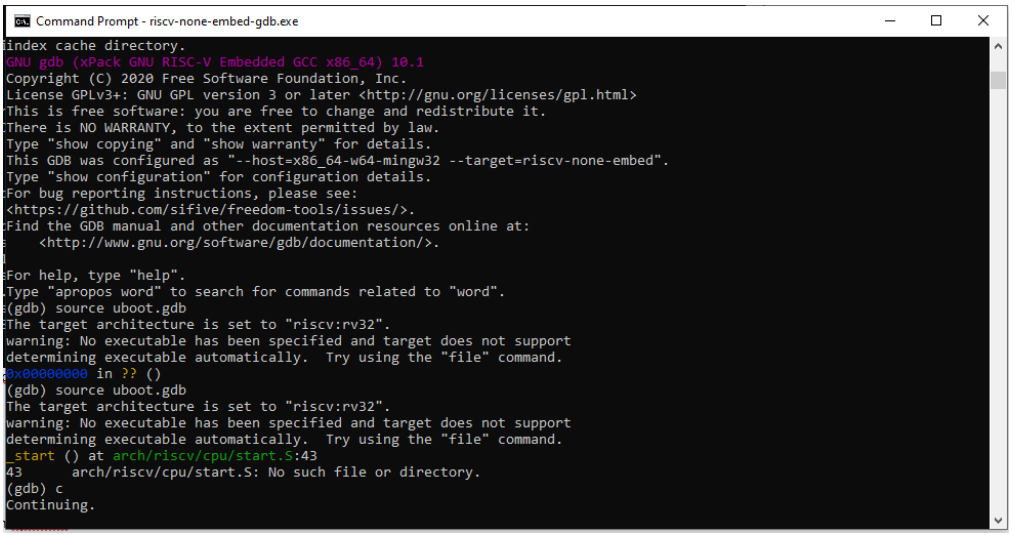

Figure 10.26. GDB is Started

9. Once this is working, you should be able to see below:

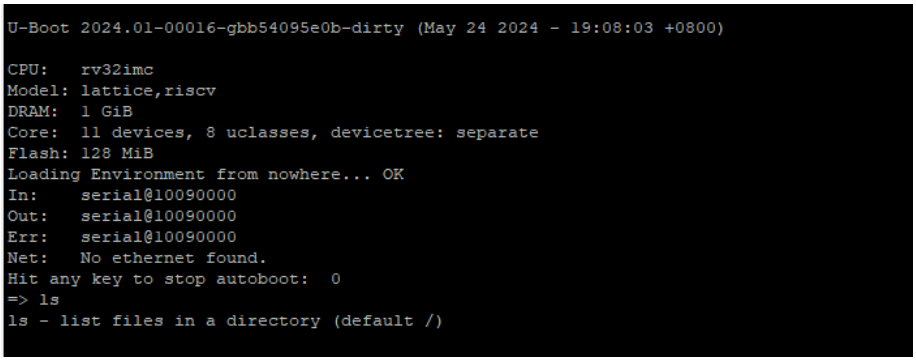

Figure 10.27. U-Boot Booted-Up on UART Terminal

10. Once U-Boot is up, load primary_appcrc.bin from QSPI and jump to FreeRTOS:

```
$ mtd list
$ mtd read lattice_qspi 0x80000000 0x28a0000 0x40000
$ go 0x80000000 0x40000
```

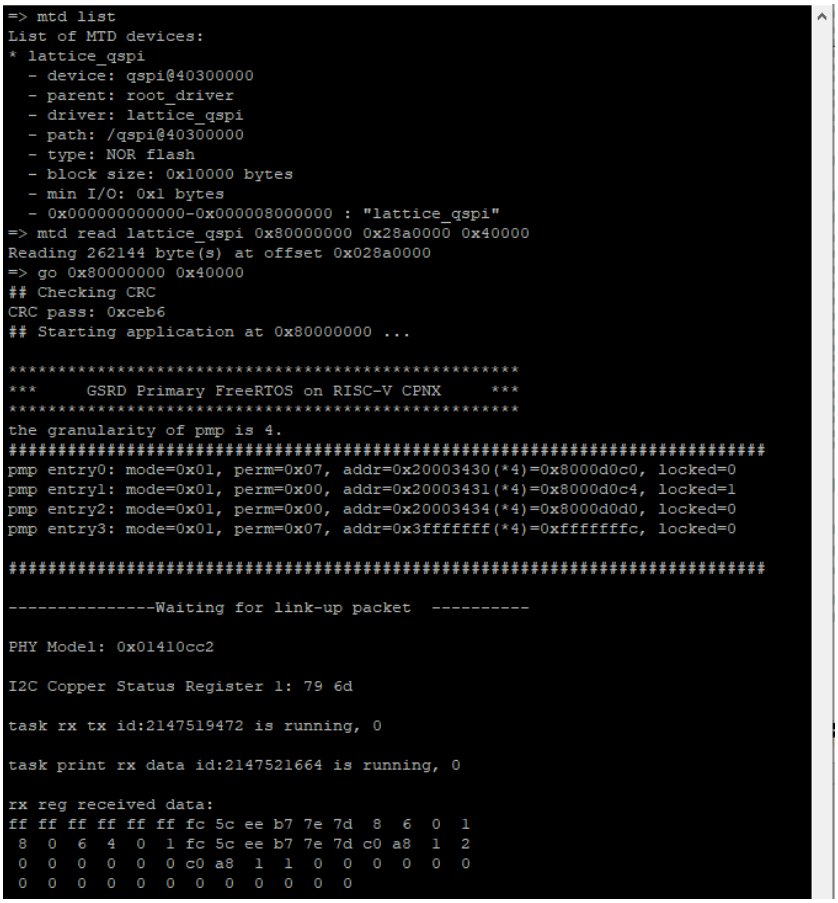

Figure 10.28. FreeRTOS is Booted-Up on UART Terminal

References

- RISC-V RX CPU IP Core (FPGA-IPUG-02254)
- System Memory IP (FPGA-IPUG-02073)
- Tri-Speed Ethernet MAC IP Core (FPGA-IPUG-02084)
- LPDDR4 Memory Controller IP Core for Nexus Devices (FPGA-IPUG-02127)
- QSPI Flash Controller IP Core (FPGA-IPUG-02248)
- AXI Multi Port Bridge for Memory Controller Module (FPGA-IPUG-02246)
- SGDMA Controller IP Core (FPGA-IPUG-02131)
- UART IP Core (FPGA-IPUG-02105)
- GPIO IP Core (FPGA-IPUG-02076)
- AXI4 Interconnect IP (FPGA-IPUG-02196)
- AXI to APB Bridge IP (FPGA-IPUG-02198)
- Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059)
- Lattice Radiant FPGA design software
- Lattice Solutions IP Cores web page
- Lattice Propel Design Environment web page
- Lattice Radiant Software User Guide
- 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 1.1, December 2025

Section	Change Summary
All	Changed document title from Golden System Reference Design and Demo User Guide for CertusPro-NX Devices to *Golden System Reference Design and Demo User Guide v2.0 for CertusPro-NX Devices*.

Introduction	• Added the following notes in the Overview of the System section.
	  • *The SGMII interface using LVDS I/O has limitations when operating across the full specified temperature range. Lattice recommends using alternative interfaces, such as SERDES or RGMII, for designs requiring Gigabit Ethernet. Refer to the Knowledge Database article for details. Contact your local Lattice sales representative for more information.*
	  • *The Golden System Reference Design and Demonstration for CertusPro-NX version 3.0 is available in the GHRD/GSRD Reference Design and GHRD/GSRD Demonstration web pages. Refer to this reference design and demonstration for the SGMII interface that is implemented using SERDES.*
	• Removed Licensing and Ordering Information section.

Revision 1.0, October 2024

Section	Change Summary
All	Initial preliminary release.

www.latticesemi.com