



Golden System Reference Design and Demo User Guide v3.0 for CertusPro-NX Devices

Lattice Propel 2025.1.1

Lattice Radiant 2025.1.1

Reference Design

FPGA-RD-02322-1.0

October 2025

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language [FAQ 6878](#) for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Contents	3
Abbreviations in This Document.....	10
1. Introduction	11
1.1. Quick Facts	11
1.2. Features	12
1.3. Comparison to SoC Version 2.0.....	12
1.4. Naming Conventions.....	12
1.4.1. Nomenclature.....	12
1.4.2. Signal Names	12
1.5. Prerequisites	13
1.5.1. Lattice Software Tools Requirements	13
1.5.2. Hardware Requirements	13
2. Functional Description.....	14
2.1. GHRD System Architecture Overview	14
2.2. GHRD Clocking Overview	15
2.3. GHRD Reset Overview	16
2.4. GHRD Interrupts Overview.....	17
2.5. IP Configurations	17
2.5.1. RISC-V RX CPU	17
2.5.2. AXI Interconnect.....	20
2.5.3. APB Interconnect.....	26
2.5.4. AXI4 to APB Bridge	27
2.5.5. LPDDR4 Memory Controller	27
2.5.6. Octal SPI Controller	28
2.5.7. Tri-Speed Ethernet MAC + SGMII (SERDES).....	29
2.5.8. Scatter-Gather DMA Controller.....	30
2.5.9. UART.....	31
2.5.10. GPIO	32
2.5.11. I2C Controller	32
2.5.12. Multi-Boot Configuration Module.....	33
2.5.13. System Memory	34
2.5.14. PLL	35
2.5.15. Reset Modules.....	37
2.5.16. AXIS FIFO Module.....	38
2.6. System Level Interfaces.....	39
2.7. SoC Memory/Address Map	39
2.8. Design Constraints	40
2.9. Resource Utilization	40
3. Signal Description	41
4. Software Components.....	43
4.1. Primary and Golden Bootloader	43
4.2. Primary and Golden Application	43
5. Theory of Operation	44
5.1. Boot-Up Sequence	44
5.2. Data Movement	46
6. Running the GSRD Demonstration	47
6.1. Executables	47
6.2. Setting Up the Hardware.....	47
6.3. Setting Up the UART Terminal	49
6.4. Setting Up the Non-Volatile Memory Register	50
6.5. Programming Standalone Golden or Primary GSRD Bitstream and Application Software	51
6.6. Programming the Golden, Primary Software, and MCS file.....	64

7.	Compiling and Running the Reference Design	67
7.1.	Building the GHRD SoC project Using Lattice Propel SDK and Propel Builder	67
7.2.	Synthesizing the RTL Files and Generating the Bitstream using Lattice Radiant	71
7.3.	Building the Hello World Program Using Lattice Propel SDK (Optional)	74
7.4.	Building the Bare-metal Bootloader Using the Lattice Propel SDK (Primary and Golden).....	79
7.5.	Building the FreeRTOS Application Software using Lattice Propel SDK (Primary and Golden).....	84
7.6.	Generating the Multi-Boot MCS File	93
7.7.	Setup Host Machine for Ping Test (Windows)	101
8.	Customizing the GSRD	106
8.1.	Adding Component to the GSRD	106
8.1.1.	Hardware Flow	106
8.1.2.	Software Flow.....	107
8.2.	Changing LPDDR4 Memory Controller Memory Density	108
8.3.	Using ECO Editor	112
9.	Debugging the GSRD	116
9.1.	Debugging Using Reveal Signal	116
9.2.	Debugging Using the OpenOCD Debugger.....	116
9.3.	Debugging with Verbosity Level.....	116
	Appendix A. Changing the SPIM Settings in the NV Register.....	117
	Appendix B. Enabling Verbosity Level in Software	118
	Appendix C. Using Different SPI Flash Manufacturer in GSRD Bare-metal Bootloader.....	120
	Appendix D. Using Octal SPI Controller for Read, Write, and Erase Self-Diagnostic Check (FreeRTOS Application Software)	121
	Appendix E. Configuring SPI Clock (SCK) Pulse Width	122
	References.....	124
	Technical Support Assistance	125
	Revision History.....	126

Figures

Figure 2.1. GHRD System Architecture	14
Figure 2.2. GHRD Clock Overview	15
Figure 2.3. GHRD Reset Overview	16
Figure 2.4. RISC-V RX CPU IP Configuration – General	18
Figure 2.5. RISC-V RX CPU IP Configuration – Debug.....	18
Figure 2.6. RISC-V RX CPU IP Configuration – Buses.....	19
Figure 2.7. RISC-V RX CPU IP Configuration – Interrupt	19
Figure 2.8. RISC-V RX CPU IP Configuration – UART	20
Figure 2.9. AXI Interconnect IP Configuration – General.....	20
Figure 2.10. AXI Interconnect IP Configuration – External Manager Settings	21
Figure 2.11. AXI Interconnect IP Configuration – External Manager Settings	21
Figure 2.12. AXI Interconnect IP Configuration – External Manager Settings	22
Figure 2.13. AXI Interconnect IP Configuration – External Manager Settings	22
Figure 2.14. AXI Interconnect IP Configuration – External Manager Settings	23
Figure 2.15. AXI Interconnect IP Configuration – External Subordinate Settings.....	23
Figure 2.16. AXI Interconnect IP Configuration – External Subordinate Settings.....	24
Figure 2.17. AXI Interconnect IP Configuration – External Subordinate Settings.....	24
Figure 2.18. AXI Interconnect IP Configuration – External Subordinate Settings.....	25
Figure 2.19. AXI Interconnect IP Configuration – External Subordinate Settings.....	25
Figure 2.20. APB Interconnect IP Configuration – General.....	26
Figure 2.21. APB Interconnect IP Configuration – Requestor Priority Settings	26
Figure 2.22. AXI4 to APB Bridge IP Configuration – General	27
Figure 2.23. LPDDR4 Memory Controller IP Configuration – General	27
Figure 2.24. LPDDR4 Memory Controller IP Configuration – General	28
Figure 2.25. Octal SPI Controller IP Configuration – General	28
Figure 2.26. Octal SPI Controller IP Configuration – General	29
Figure 2.27. Octal SPI Controller IP Configuration – General	29
Figure 2.28. TSE IP Configuration	30
Figure 2.29. SGDMA Controller IP Configuration.....	31
Figure 2.30. UART IP Configuration	31
Figure 2.31. GPIO IP Configuration	32
Figure 2.32. I2C IP Configuration	33
Figure 2.33. Multi-Boot Module Configuration	34
Figure 2.34. System Memory IP Configuration -General	34
Figure 2.35. System Memory IP Configuration -Port S0 Settings	35
Figure 2.36. PLL IP Configuration -General	35
Figure 2.37. PLL IP Configuration – General	36
Figure 2.38. PLL IP Configuration – General	36
Figure 2.39. PLL IP Configuration – Optional Ports.....	37
Figure 2.40. Reset Module Configuration – General	37
Figure 2.41. AXIS RX FIFO Module Configuration – General.....	38
Figure 2.42. AXIS TX FIFO Module Configuration – General.....	38
Figure 2.43. LFCPNX-100 Device GHRD Approximate Resource Utilization	40
Figure 5.1. GSRD Boot-Up Sequence	45
Figure 5.2. Ethernet Data RX Flow	46
Figure 5.3. Ethernet Data TX Flow	46
Figure 6.1. CertusPro-NX Versa Evaluation Board	48
Figure 6.2. Connections, Jumpers and Switches Needed for Demonstration	48
Figure 6.3. UART Terminal Icon on Propel SDK Window	49
Figure 6.4. UART Launch Terminal Window	50
Figure 6.5. Device Manager Window on PC	50
Figure 6.6. Launch Radiant Programmer from Windows Start.....	51

Figure 6.7. Radiant Programmer Start Window	51
Figure 6.8. Radiant Programmer. xcf Window	52
Figure 6.9. Scan Device Icon on Radiant Programmer.....	52
Figure 6.10. Select Device for Programming	52
Figure 6.11. Device Selected for Programmer.....	52
Figure 6.12. Select the Target Memory for Programming – SPI Flash	53
Figure 6.13. Device Properties to Erase the Macronix SPI Flash	53
Figure 6.14. Device Properties to Erase the Winbond SPI Flash.....	54
Figure 6.15. Program Button to Program the SPI Flash	54
Figure 6.16. Output After Erase All.....	55
Figure 6.17. Erase Only Operation for SRAM Programming.....	55
Figure 6.18. Device Properties to Program the Winbond SPI Flash.....	56
Figure 6.19. Device Properties to Program the Macronix SPI Flash	57
Figure 6.20. Cable Settings for Device Programming	58
Figure 6.21. Radiant Programmer Console Output after Programming the SPI Flash.....	58
Figure 6.22. Device Properties to Program the FPGA Bitstream in SRAM.....	59
Figure 6.23. Radiant Programmer Console Output after Bitstream is Programmed.....	59
Figure 6.24. SW3 Reset Button and SW2 PROGRAMN Button	60
Figure 6.25. LED Status with SGMII Link Speed at 1000Mbps	61
Figure 6.26. Golden GSRD - Output on UART Terminal for Bootloader and FreeRTOS Start	61
Figure 6.27. Golden GSRD – Output on UART Terminal for FreeRTOS Running	62
Figure 6.28. Primary GSRD Bootloader– Output on UART Terminal	63
Figure 6.29. Primary GSRD FreeRTOS – Output on UART Terminal.....	63
Figure 6.30. Device Properties Window to Setup MCS Programming File	64
Figure 6.31. UART Terminal Output after Power-Cycling Board with MCS and Binaries Programmed	65
Figure 6.32. Switches to Golden GSRD upon SW2 PROGRAMN Button	66
Figure 7.1. Propel SDK Launcher.....	67
Figure 7.2. Provide Name for the Workspace Directory.....	68
Figure 7.3. Creating Lattice SoC Design Project.....	68
Figure 7.4. SoC Project Window	69
Figure 7.5. Launched SoC in Propel Builder.....	70
Figure 7.6. Validate Design in Propel Builder	70
Figure 7.7. TCL Console Output after Validating Design.....	70
Figure 7.8. Generate in Propel Builder	70
Figure 7.9. TCL Console Printout after Generating Design	71
Figure 7.10. sys_env.xml File Created	71
Figure 7.11. Run Radiant Icon.....	71
Figure 7.12. Lattice Radiant Window.....	72
Figure 7.13. Lattice Radiant Window.....	72
Figure 7.14. Strategies Used for GSRD Testing	73
Figure 7.15. Generating the Bit File.....	74
Figure 7.16. Successful Radiant Flow and Bitstream Generation	74
Figure 7.17. Creating Lattice C/C++ Project for Hello World	75
Figure 7.18. Hello World C/C++ Selection.....	76
Figure 7.19. C/C++ Lattice Toolchain Setting.....	77
Figure 7.20. Hello World C Project Created.....	77
Figure 7.21. Build Hello World Project	78
Figure 7.22. Hello World Project Build Console Output	78
Figure 7.23. Hello World C Program	79
Figure 7.24. Creating Lattice C/C++ Project for Bootloader	79
Figure 7.25. Bootloader C/C++ Selection.....	80
Figure 7.26. C/C++ Lattice Toolchain Setting.....	81
Figure 7.27. Bootloader C Project Created	81
Figure 7.28. Primary Build Define – Set _PRIMARY_BUILD_ for Primary Build in main.c.....	82

Figure 7.29. Build Bootloader Project.....	82
Figure 7.30. Bootloader Build Project Console Output.....	82
Figure 7.31. Bootloader Binary Created	83
Figure 7.32. Bootloader boots up without FreeRTOS application	83
Figure 7.33. Golden Build Define – Set <code>_GOLDEN_BUILD_</code> for Golden Build in <code>main.c</code>	83
Figure 7.34. Creating C/C++ Project for FreeRTOS	84
Figure 7.35. FreeRTOS C/C++ Selection	84
Figure 7.36. FreeRTOS Project Created	85
Figure 7.37. Copy the CRC Add files.....	86
Figure 7.38. Paste in FreeRTOS Application C Project	86
Figure 7.39. Copied Text Files.....	86
Figure 7.40. Primary Build Define – Set <code>_PRIMARY_BUILD_</code> for Primary Build in <code>main.c</code>	87
Figure 7.41. Update <code>crc_add_debug.txt</code>	87
Figure 7.42. Open linker.ld File.....	87
Figure 7.43. Update linker.ld File.....	88
Figure 7.44. Workspace	88
Figure 7.45. Properties	89
Figure 7.46. Set Release as Active Configuration	89
Figure 7.47. Adding Post Build Step for FreeRTOS Application CRC Binary Append (Windows)	90
Figure 7.48. Adding Post Build Step for FreeRTOS Application CRC Binary Append (Linux).....	91
Figure 7.49. Update the CRC Script to Match Linux Path Format.....	91
Figure 7.50. Build <code>c_primary_app</code> C/C++ Project.....	92
Figure 7.51. FreeRTOS App Build Project Console Output	92
Figure 7.52. FreeRTOS App Binaries Created with CRC	93
Figure 7.53. Golden Build Define – Set <code>_GOLDEN_BUILD_</code> for Golden Build in <code>main.c</code>	93
Figure 7.54. Launch Radiant Programmer from Windows Start.....	94
Figure 7.55. Radiant Programmer Getting Started Window	94
Figure 7.56. Error if No Board is Connected	95
Figure 7.57. Open Deployment Tool from Radiant Programmer	95
Figure 7.58. Deployment Tool Start Window	95
Figure 7.59. Options for Creating New Deployment	96
Figure 7.60. External Memory Step 1 of 4: Select Input Files.....	96
Figure 7.61. External Memory Step 2 of 4: Select Options.....	97
Figure 7.62. External Memory Step 2 of 4: Multiple Boot	98
Figure 7.63. External Memory Step 3 of 4: Select Output File(s)	99
Figure 7.64. External Memory Step 4 of 4: General Development.....	100
Figure 7.65. MCS File Generated Successfully	100
Figure 7.66. Configure Ethernet Settings.....	101
Figure 7.67. Select Ethernet Port.....	102
Figure 7.68. Edit IP settings	103
Figure 7.69. Set the IP Settings to Manual, IP address, Subnet Prefix Length, and Gateway.....	104
Figure 7.70. Check Ethernet Connection	104
Figure 7.71. Ping the Device	105
Figure 7.72. UART Terminal Showing the Printout when Ping Packet Received	105
Figure 8.1. Bootloader File Updated in System Memory	107
Figure 8.2. New System ENV Error	108
Figure 8.3. Identifying LPDDR4 Memory Device through BRI Number.....	109
Figure 8.4. IP Block	109
Figure 8.5. Customize IP	110
Figure 8.6. TCL Console Output After Desired Customization.....	111
Figure 8.7. Updated Address Map	111
Figure 8.8. Updated <code>constraints.sdc</code>	111
Figure 8.9. Validate Design Again if Any IP is Updated.....	111
Figure 8.10. Generate Again if any IP is Updated	111

Figure 8.11. Update the LPDDR length in linker.ld	112
Figure 8.12. ECO Editor Icon in Radiant Software	112
Figure 8.13. ECO Editor sysIO Settings Tab.....	113
Figure 8.14. ECO Editor Memory Initialization Tab	113
Figure 8.15. Select bootloader in Memory Initialization Settings	114
Figure 8.16. Change file format in Memory Initialization Settings	114
Figure 8.17. Updated System Memory content	114
Figure 8.18. Save Icon.....	114
Figure 8.19. Save ECO Changes.....	115
Figure 8.20. Re-run partial Radiant Flow	115
Figure 8.21. Bitstream Generation Flow Completed Successfully	115
Figure 8.22. Bitstream is Re-Generated.....	115
Figure A.1. One-Time Programmable Control NV Register1	117
Figure A.2. Settings to Select Chip Value	117
Figure B.1. Set Release Build Configurations	118
Figure C.1. SPI Flash Manufacturer Changes in octal_spi_controller.h	120
Figure D.1. Octal SPI Controller Read, Write and Erase check in FreeRTOS Application Software	121
Figure D.2. Self-diagnostic Check Failed	121
Figure D.3. Self-diagnostic Check Passed.....	121
Figure E.1. Programmable SCK Divider Option in the IP User Interface	122
Figure E.2. Octal SPI Controller sck_rate Configuration	123

Tables

Table 1.1. Summary of the System	11
Table 1.2. List of Hardware Required by GSRD.....	13
Table 2.1. IP Versions.....	14
Table 2.2. GHRD Clocking Overview	16
Table 2.3. GHRD Reset Overview	17
Table 2.4. GHRD Interrupt Overview	17
Table 2.5. System Level Interfaces	39
Table 2.6. Address Map of GHRD	39
Table 2.7. Design Constraints	40
Table 2.8. GSRD Total Approximate Resource Utilization	40
Table 3.1. Top-level I/O	41
Table 6.1. Supported Flash Devices	47
Table 6.2. Executable Files for Winbond Flash	47
Table 6.3. Executable Files for Macronix Flash	47
Table 6.4. SGMII Link Speed Definition.....	60
Table 7.1. List of Actions and Expected Outputs	67
Table 8.1. Types of LPDDR4 SDRAM in CertusPro-NX Versa Evaluation Board	108
Table B.1. Debuglib Verbose Levels.....	118

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
CRC	Cyclic Redundancy Check
DDR	Double Data Rate
FIFO	First-In-First-Out
GHRD	Golden Hardware Reference Design
GSRD	Golden System Reference Design
ICMP	Internet Control Message Protocol
ISR	Interrupt Service Routines
LMMI	Lattice Memory Mapped Interface
LPDDR4	Low Power Double Data Rate Generation 4
IwIP	lightweight Internet Protocol
OPN	Ordering Part Number
PLL	Phase-Locked Loop
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

The Lattice FPGA-based Golden System Reference Design (GSRD) 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 Controller, Tri-Speed Ethernet (TSE), Octal SPI Controller, Scatter-Gather DMA (SGDMA) Controller and LPDDR4 Memory Controller. All these building blocks are connected through industry standard protocols such as AXI-MM, AXI-Stream for data transfers and APB for control.

The GSRD is a comprehensive embedded system which incorporates drivers and relevant firmware needed to operate various design components. FreeRTOS and bootloader are built on RISC-V RX CPU. The primary function of bootloader is to initialize the hardware blocks in the design using the respective IP drivers and ensure the integrity of the application software by performing CRC check.

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

As a part of multi-boot demonstration, the executables folder comprises compatible binaries and executable images, that is FPGA bitstream (.bit) and software 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 Lattice Propel™ and Lattice Radiant™ software versions as indicated in [Table 1.1](#).

1.1. Quick Facts

Table 1.1. Summary of the System

SoC Requirements	Supported FPGA Family	CertusPro-NX
	SoC Version	3.0
FPGA Device(s)	Targeted Board	CertusPro-NX Versa Board Rev -B OPN: LFCPNX-VERSA-EVN
	Targeted Device	LFCPNX-100-9LFG672I
	Supported User Interface	AXI-MM, AXI-Stream, APB
Design Tool Support	Lattice Implementation	Lattice Propel Software 2025.1.1 Lattice Radiant Software 2025.1.1
	Synthesis	Synopsys® Synplify Pro®

1.2. Features

The key features of the system include:

- The FPGA device supported in this document is LFCPNX-100
- RISC-V RX CPU, SGDMA Controller, TSE IP, LPDDR4 Memory Controller, and Octal SPI Controller over AXI4 Interface
- 4 Gbit of LPDDR4 cacheable memory with 32-bit DDR data width at 1066 Mbps data rate
- 10/100/1000 Mbps Ethernet throughput through SGMII SFP support at 125 MHz
- Low-speed peripherals like GPIO, UART, and I2C
- Primary and Golden bootloader and FreeRTOS application software
- Application software CRC check by function implemented in RISC-V RX bootloader code
- FPGA bitstream CRC check done by FPGA Configuration Engine
- FreeRTOS application software is run on external LPDDR4 Memory Device
- Manual and Automatic Multi-Boot capability

1.3. Comparison to SoC Version 2.0

The following are the key changes from SoC version 2.0:

- Enhanced architecture for performance and area.
- Centralized interconnect for flow control.
- Multiple clock domains and IP specific clocking scheme for performance.
- Replaced LVDS I/O based SGMII PCS with Serdes based SGMII PCS.
- Enabled LPDDR4 device in cacheable region and data width converter in LPDDR4 Memory Controller.
- Replaced QSPI Flash Controller with Octal SPI Controller.
- Enabled RISC-V watchdog module.
- Updated Radiant, Propel, and soft IPs for 2025.1.1
- Added support for Lightweight IP (lwIP) software stack
- Removed support for U-boot.

1.4. Naming Conventions

1.4.1. Nomenclature

The following are the nomenclature used in this 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 hardware bitstream and the corresponding software.
- FreeRTOS application software – 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 bitstreams, FreeRTOS application software and multi-boot MCS bitstream.

1.4.2. Signal Names

Signal names that end with:

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

1.5. Prerequisites

The following sections show the software and hardware requirements to run the demonstration and compiling the reference design.

1.5.1. Lattice Software Tools Requirements

- Lattice Propel 2025.1.1 Package – contains both Lattice Propel SDK and Lattice Propel Builder
 - Download here: [Lattice Propel Design Environment](#)
- Lattice Radiant 2025.1.1 Package – contains IP Packager, Radiant Software, QuestaSim, and Programmer
 - Download here: [Lattice Radiant Software](#)
- Lattice Propel 2025.1.1 Patch for CertusPro-NX Golden System Reference Design
 - Download here: [Downloadable Software tab in the GHRD/GSRD Reference Design](#)

1.5.2. Hardware Requirements

[Table 1.2](#) describes the hardware needed to run the GSRD.

Table 1.2. List of Hardware Required by GSRD

Sr No	Hardware Requirements	Quantity	Comment
1	Lattice CertusPro-NX Versa Evaluation Board OPN: LFCPNX-VERSA-EVN	1	CertusPro-NX Versa Board web page The Evaluation board is available in two configurations of LPDDR4 that are 4 Gb and 16 Gb and two vendors of SPI flash, that are Winbond and Macronix. Boards with 4 Gb LPDDR4 SDRAM are marked with <i>BRI No: LSC-2309-001</i> all others with 16 Gb LPDDR4 SDRAM.
2	Mini USB Type-A UART cable for programming bitstream, firmware and proper terminal prints	1	Included in CertusPro-NX Versa Board Evaluation Kit
3	10/100/1000BASE-T SFP SGMII Copper RJ-45 Transceiver Module for Ethernet connection on the CertusPro-NX board	1	Not included in CertusPro-NX Versa Board Evaluation Kit. GSRD is validated with SFP-GEB-T from FS , FCLF8522P2BTL from Coherent and NXT-SFP-T from techNXT .
4	Cat6 RJ45 Ethernet cable to connect CertusPro-NX board to the Host PC	1	Not included in CertusPro-NX Versa Board Evaluation Kit.
5	12 V power adapter for board power	1	Included in CertusPro-NX Versa Board Evaluation Kit

2. Functional Description

The GSRD/GHRD SoC architecture comprises of a RISC-V RX CPU, LPDDR4 Memory Controller, SGDMA Controller, Octal SPI Controller, and 10/100/1000 Mbps TSE IP, 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. GHRD System Architecture Overview

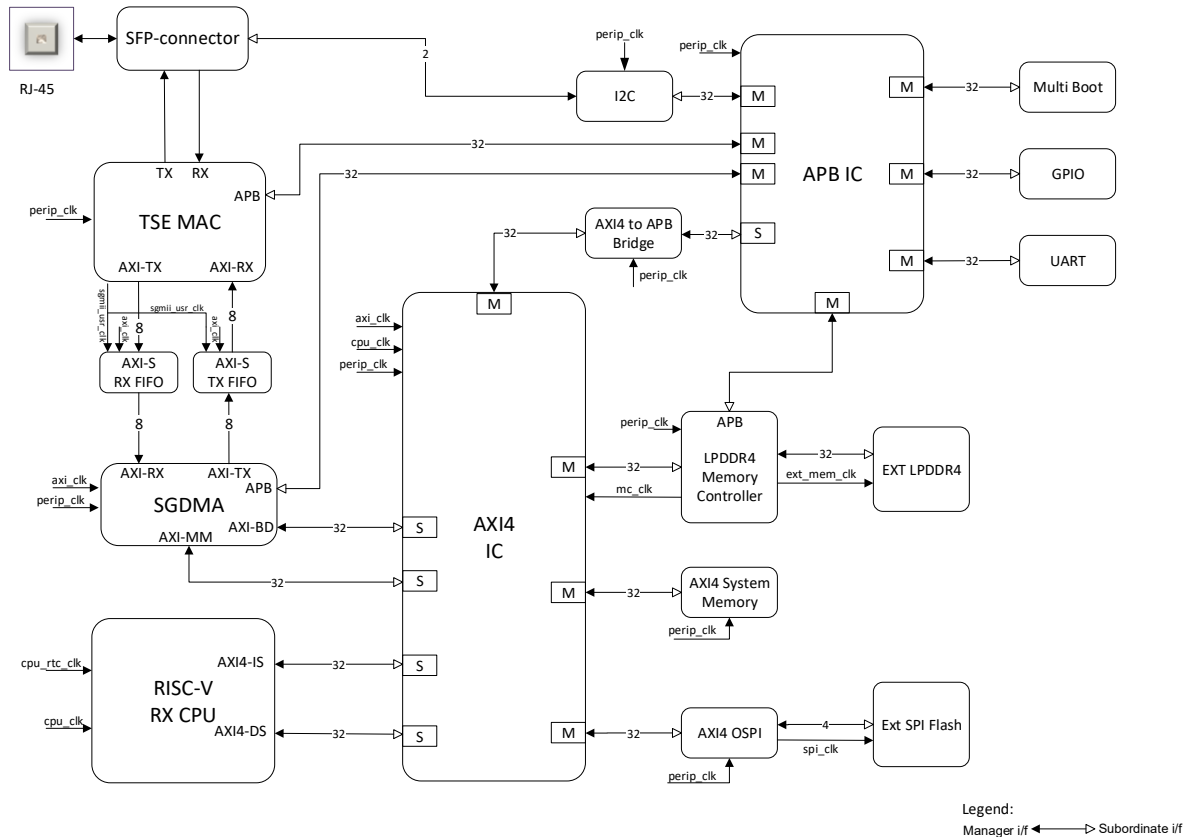


Figure 2.1. GHRD System Architecture

The design includes the following components listed in [Table 2.1](#).

Table 2.1. IP Versions

Soft IP	IP Version
RISC-V RX CPU	2.7.0
LPDDR4 Memory Controller	2.6.2
System Memory	2.4.0
Octal SPI Controller	1.3.0
General Purpose I/O GPIO	1.8.0
UART	1.4.0
I2C Controller	2.3.0
AXI4 Interconnect	2.2.0
APB Interconnect	1.3.0
AXI4 to APB Bridge	1.4.0

Soft IP	IP Version
Tri-Speed Ethernet	2.1.0
SGDMA Controller	2.5.0
Multi-Boot Configuration	1.0.0
PLL	1.9.1
Reset Modules	1.0.0
AXIS FIFO IP	1.0.0

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 on each interface are described in the [Signal Description](#) section.

2.2. GHRD Clocking Overview

[Figure 2.2](#) describes the reference design clock scheme.

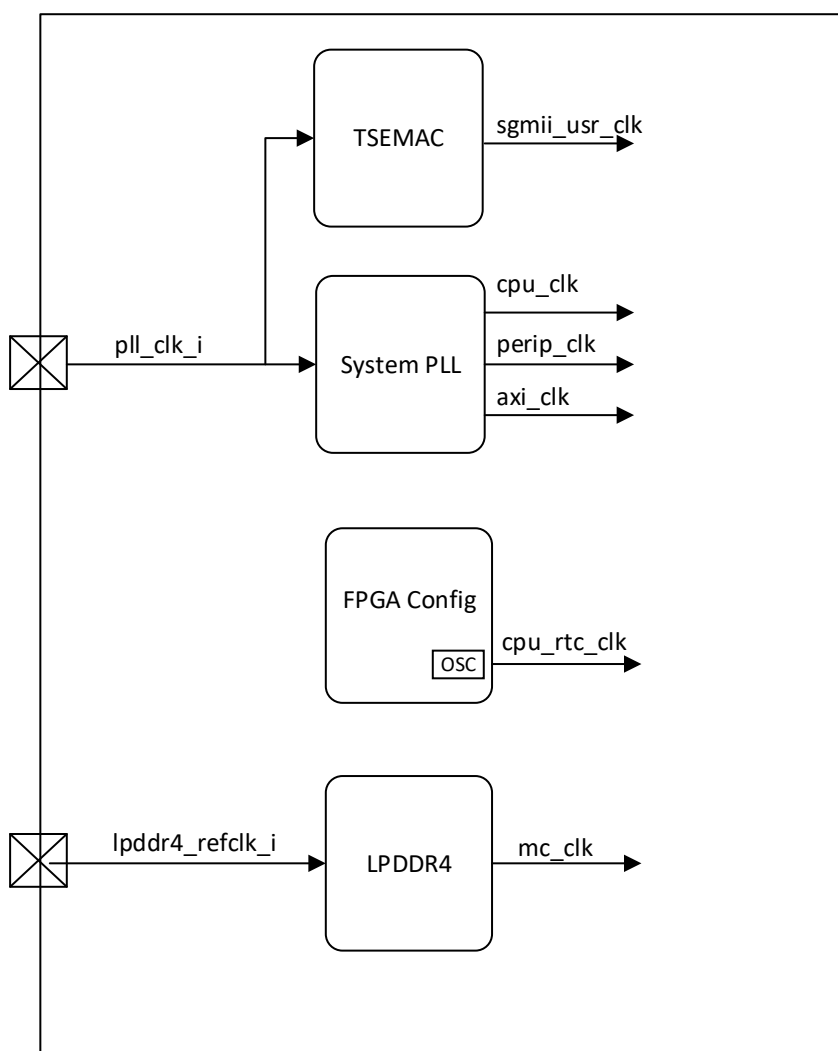


Figure 2.2. GHRD Clock Overview

Table 2.2. GHRD Clocking Overview

Sr No	Clock Name	Clock Freq	Clock Source	Destination
1	lpddr4_refclk_i	100 MHz	On board oscillator U6	LPDDR4 Memory Controller IP PLL
2	pll_clk_i	125 MHz	On board oscillator U4	System PLL and TSE IP
3	cpu_clk	125 MHz	system_pll[CLKOP]	RISC-V RX CPU system clock, System Memory, AXI IC S0, S3, and M3 ports
4	perip_clk	100 MHz	system_pll[CLKOS]	AXI IC M0, M1 Ports, TSE IP, SGDMA Controller, UART, I2C, GPIO, and Multi-boot IP
5	axi_clk	125 MHz	system_pll[CLKOS2]	AXI IC clock, SGDMA Controller
6	sgmii_usr_clk	125 MHz	TSE IP	External CDC FIFO
7	cpu_rtc_clk	32 kHz	Multi-boot IP	RISC-V RX CPU RTC clock
8	mc_clk	133.33 MHz	LPDDR4 Memory Controller	LPDDR4 Memory Controller IP and AXI4 IC (Interconnect) M2 port

2.3. GHRD Reset Overview

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

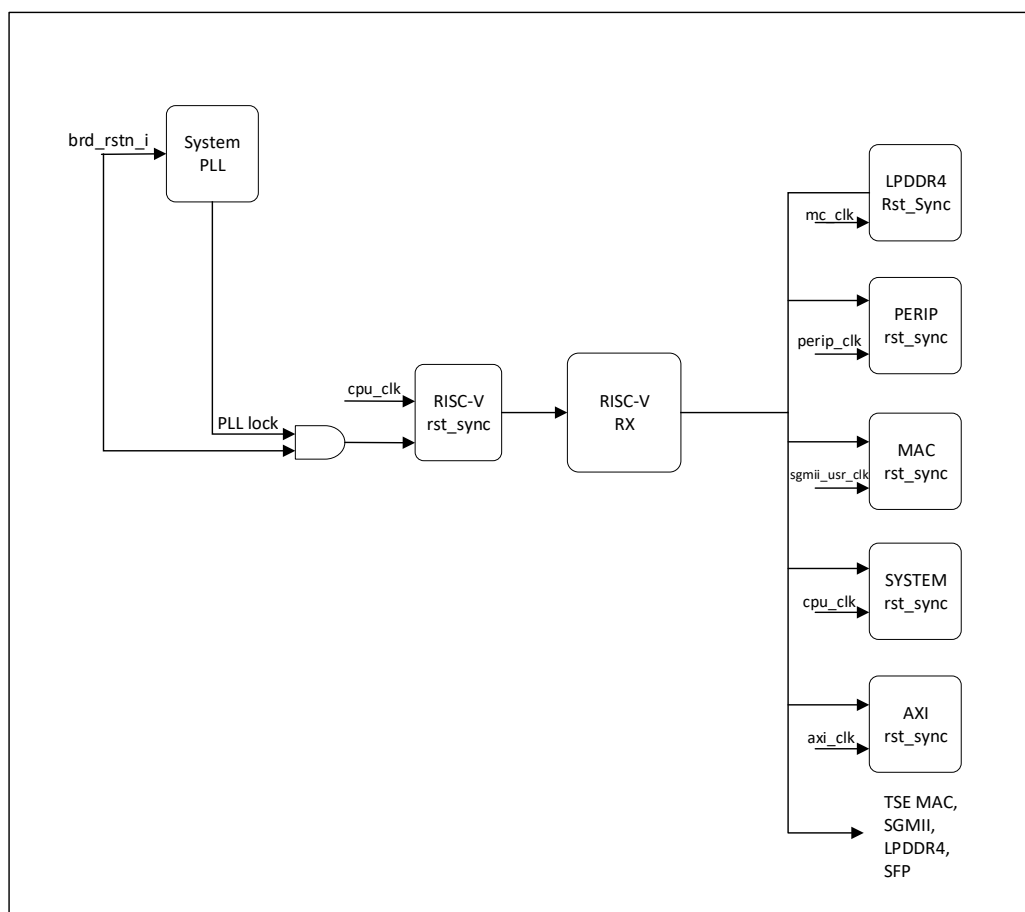


Figure 2.3. GHRD Reset Overview

Table 2.3. GHRD Reset Overview

Reset Signal	Source	Destination	Description
system_rstn_i	Board Pushbutton SW3	PLL reset input pin	Reset the PLL and system reset
cpu_rstn_i	Reset sync output port	RISC-V CPU reset input	Reset pin of CPU
system_resestn_o	RISC-V CPU output	All components in system	RISC-V CPU output reset pin, provides reset to all components in the design. This also triggers the reset during CPU OCD debugging mode.
sys_rstn_o	CPU output reset pin	AXI IC S0, S3, AXI IC M3 and System Memory reset input pin	AXI IC CPU subordinates, System Memory, AXI IC M3
perip_rstn_o	CPU output reset pin	AXI IC M0, M1 and all peripheral IPs reset input pin	Octal SPI controller, APB IC, AXI4 to APB bridge, GPIO, UART, I2C, SGDMA Controller, TSE IP, LPDDR4 Memory Controller and Multi-boot IP
axi_rstn_o	CPU output reset pin	AXI IC S1, S2 and SGDMA Controller MM reset input pin	SGDMA Controller MM and BD
mac_rstn_o	CPU output reset pin	AXI Stream RX-TX FIFO reset input pint	External Stream FIFO for CDC.
lpddr4_rstn_o	CPU output reset pin	AXI IC M2 reset input pin	LPDDR4 Memory Controller

2.4. GHRD Interrupts Overview

Table 2.4. GHRD Interrupt Overview

Sr No	RISC-V CPU IRQ Line	Interrupt Source
1	IRQ_2	SGDMA Controller MM2S IRQ
2	IRQ_3	SGDMA Controller S2MM IRQ
3	IRQ_4	TSEMAC IRQ
4	IRQ_5	OSPI IRQ
5	IRQ_6	GPIO IRQ
6	IRQ_7	LPDDR IRQ
7	IRQ_8	UART IRQ
8	IRQ_9	I2C IRQ

For more information about the platform level interrupt controller information, refer to the Platform Level Interrupt Controller section in the [RISC-V RX CPU IP User Guide \(FPGA-IPUG-02298\)](#) document.

2.5. 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.

2.5.1. RISC-V RX CPU

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 memory and peripherals for control.

For more information about the IP core including memory map information, refer to [RISC-V RX CPU IP User Guide \(FPGA-IPUG-02298\)](#).

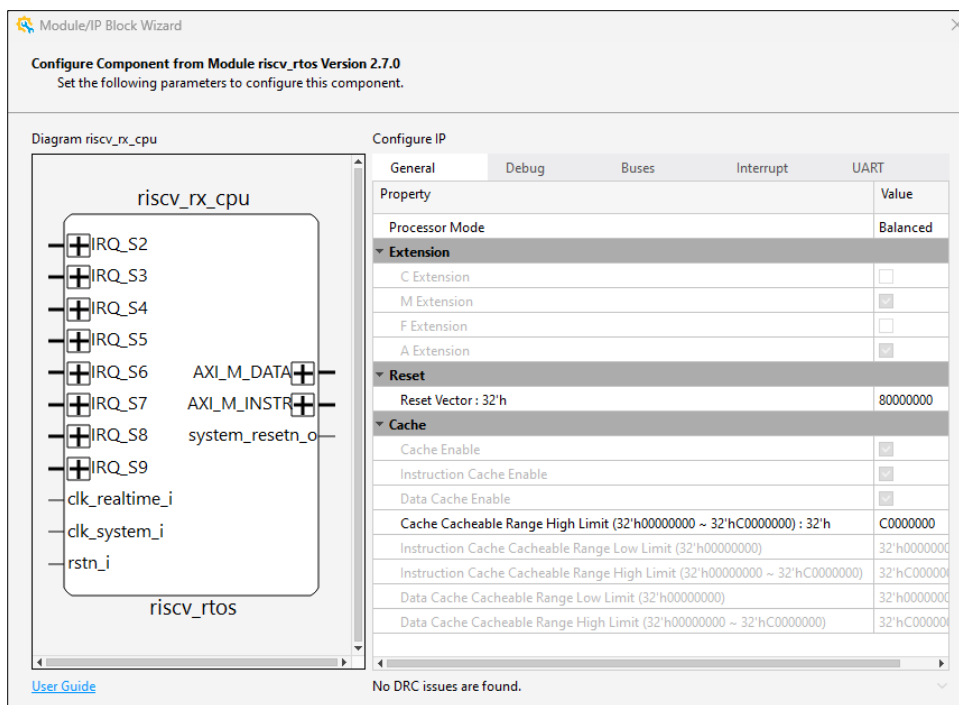


Figure 2.4. RISC-V RX CPU IP Configuration – General

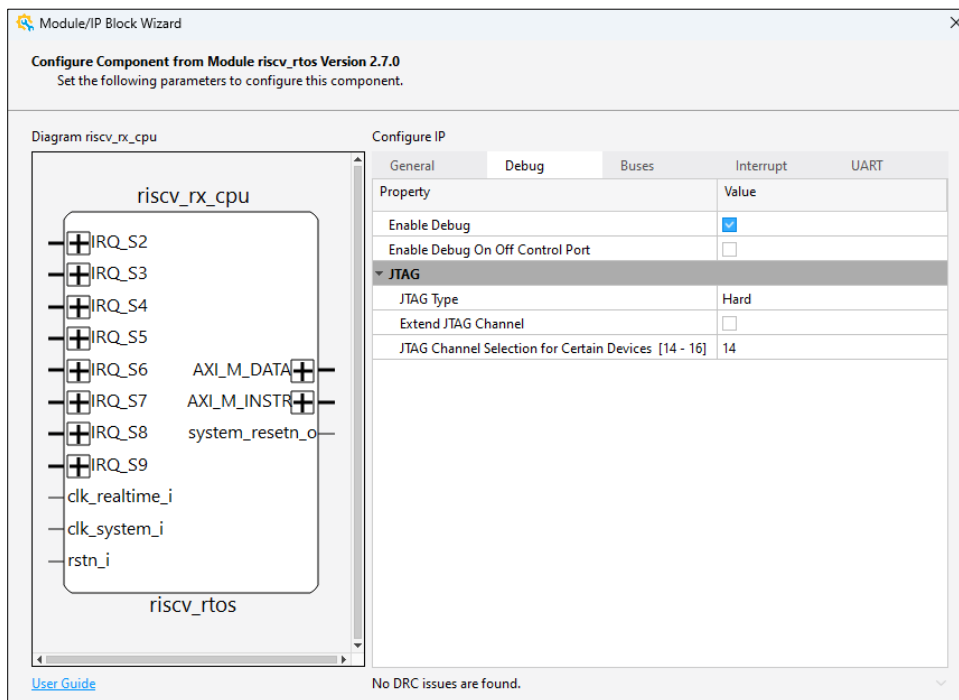


Figure 2.5. RISC-V RX CPU IP Configuration – Debug

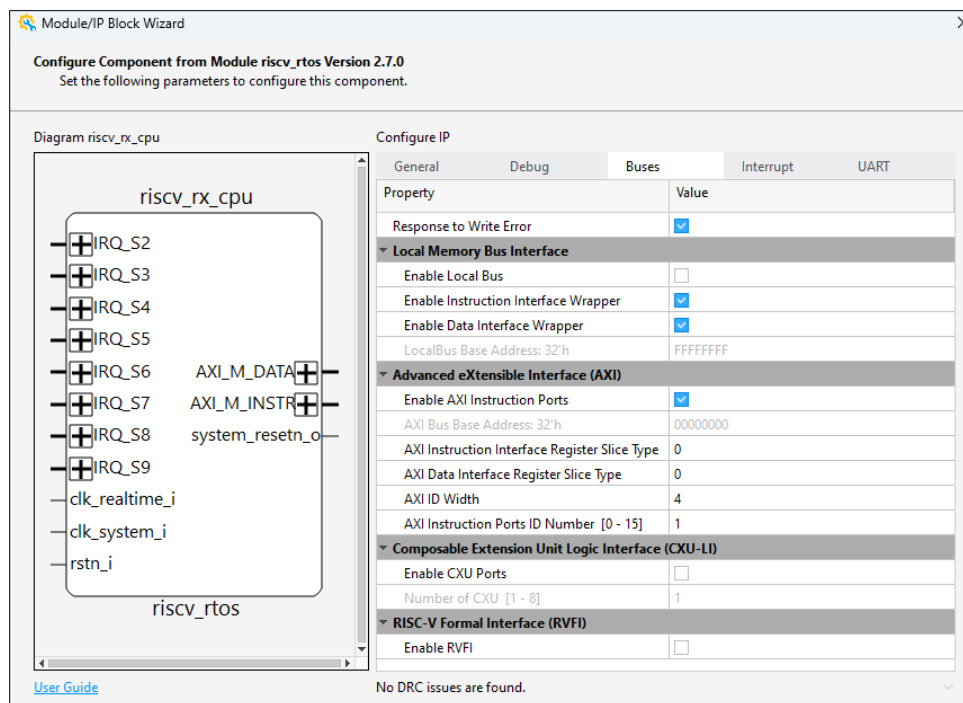


Figure 2.6. RISC-V RX CPU IP Configuration – Buses

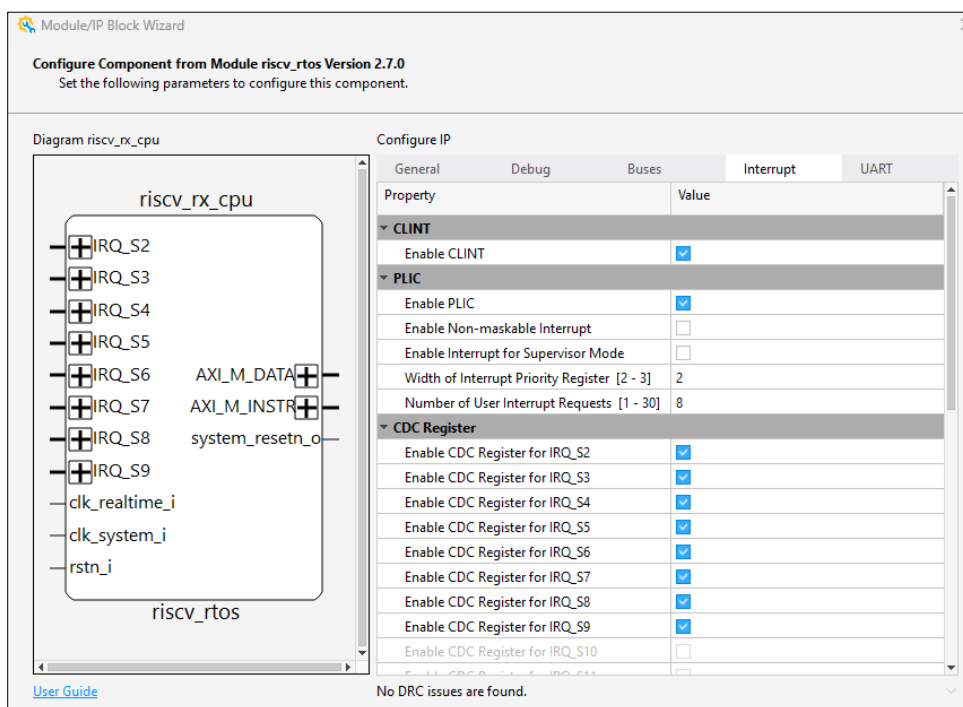


Figure 2.7. RISC-V RX CPU IP Configuration – Interrupt

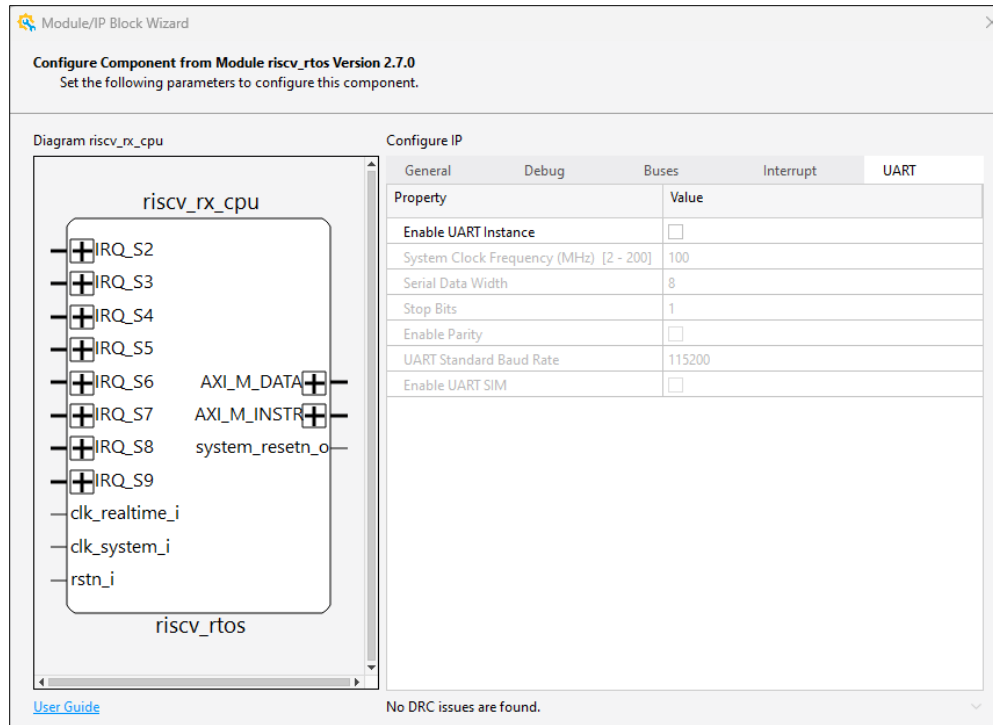


Figure 2.8. RISC-V RX CPU IP Configuration – UART

2.5.2. AXI Interconnect

The AXI Interconnect IP is a key component in system that uses the AMBA AXI4 protocol. It acts as a communication hub between multiple AXI managers and subordinates, enabling efficient data transfer and system scalability.

For more information about the IP core, refer to [AXI Interconnect IP User Guide \(FPGA-IPUG-02196\)](#).

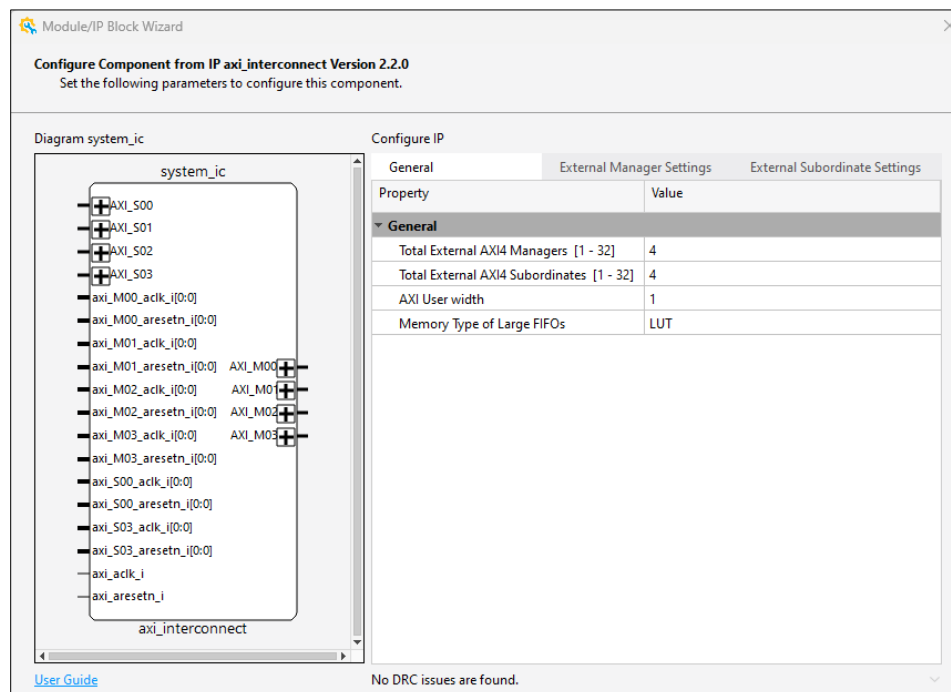


Figure 2.9. AXI Interconnect IP Configuration – General

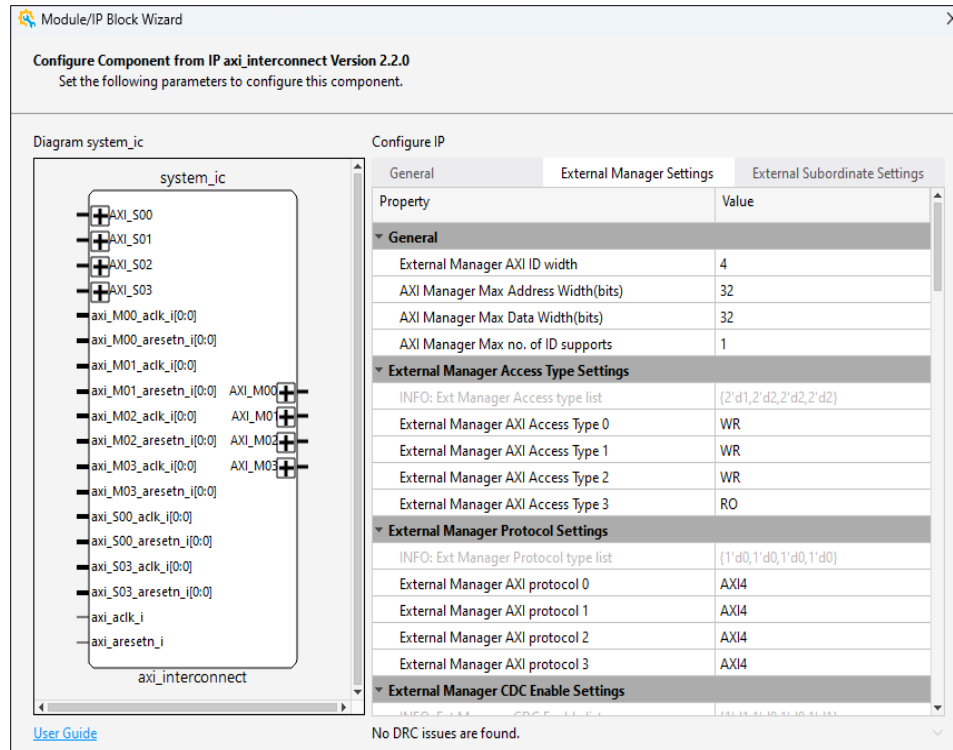


Figure 2.10. AXI Interconnect IP Configuration – External Manager Settings

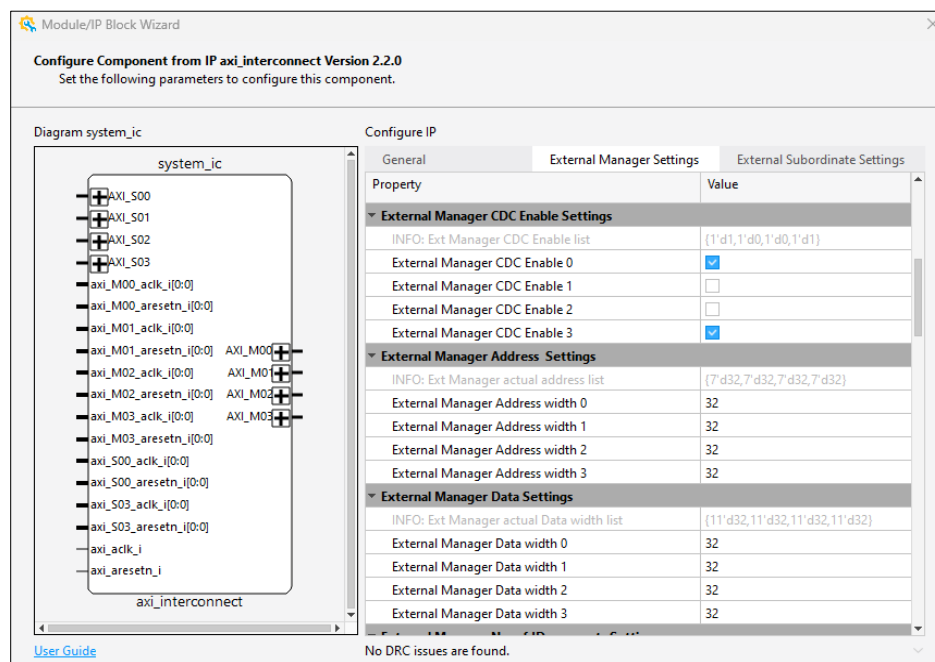


Figure 2.11. AXI Interconnect IP Configuration – External Manager Settings

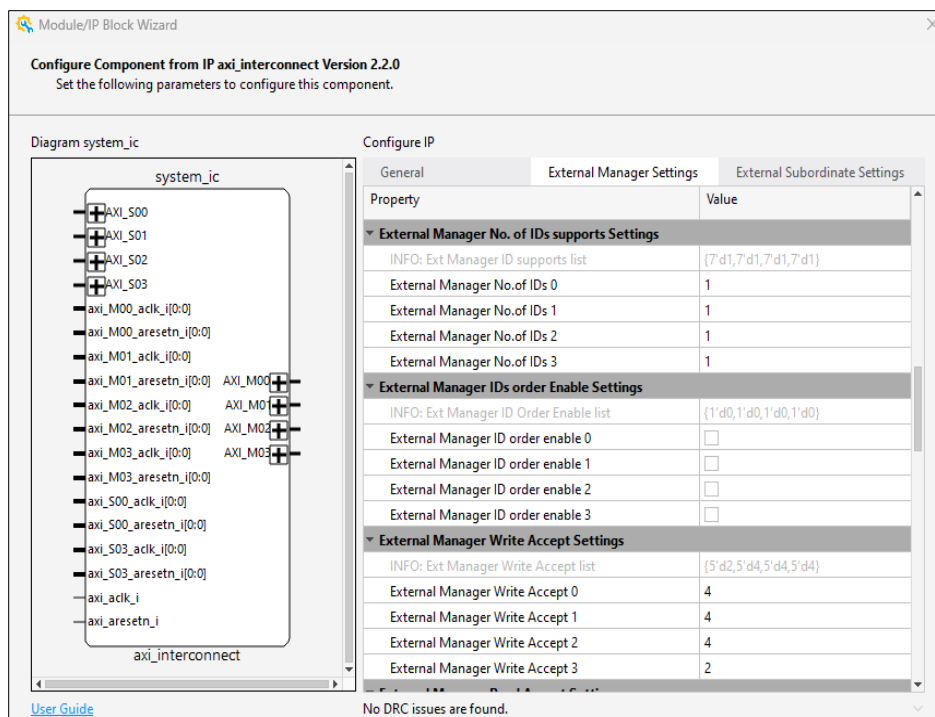


Figure 2.12. AXI Interconnect IP Configuration – External Manager Settings

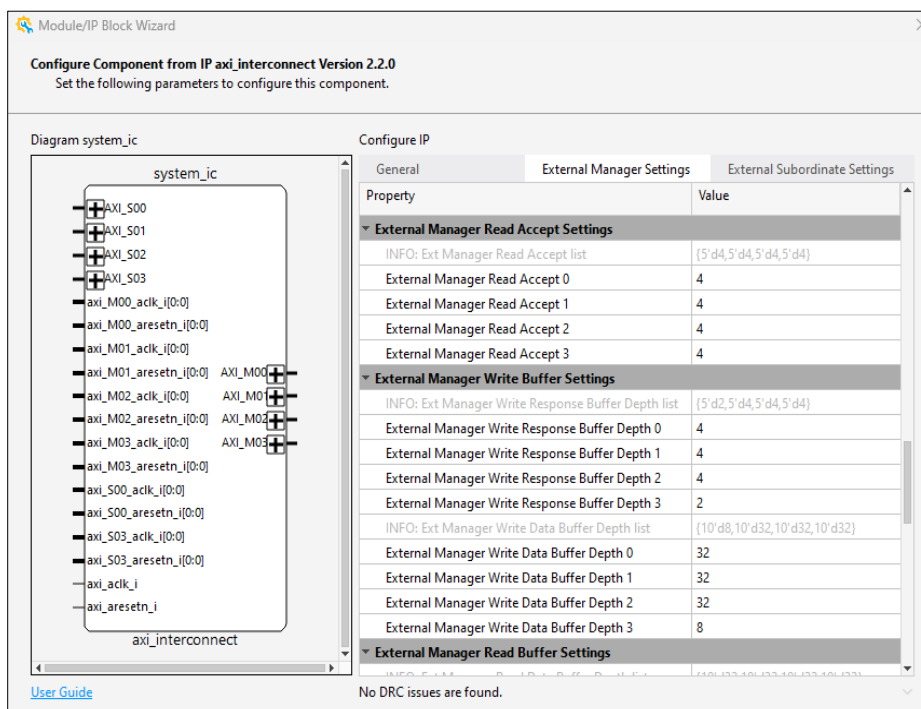


Figure 2.13. AXI Interconnect IP Configuration – External Manager Settings

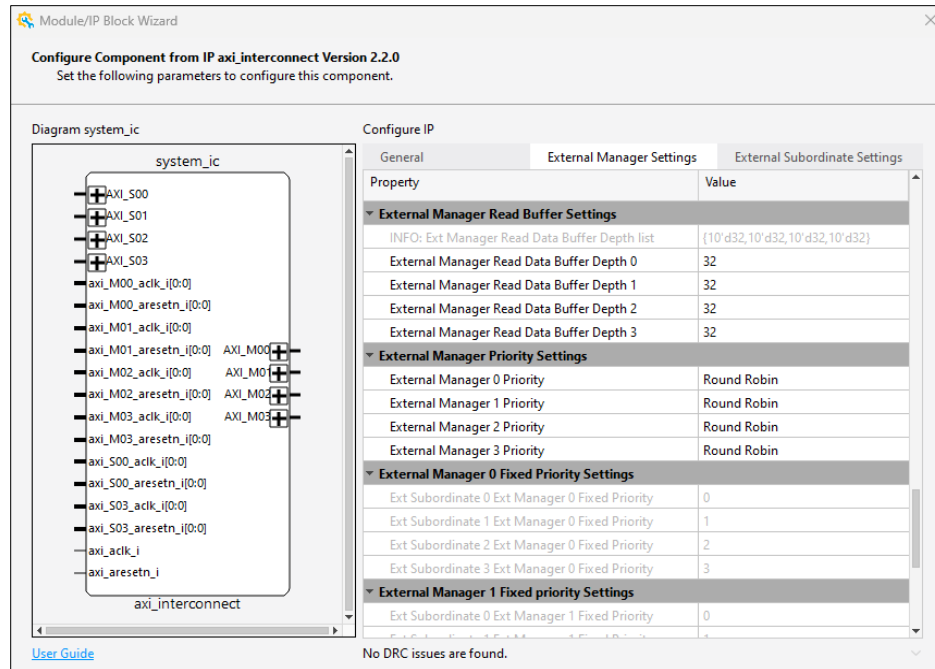


Figure 2.14. AXI Interconnect IP Configuration – External Manager Settings

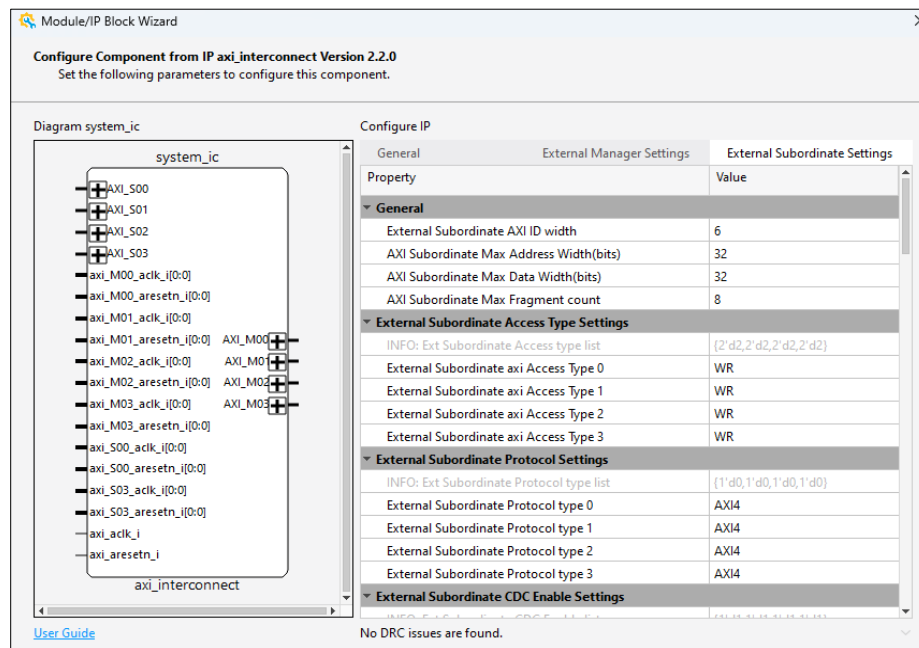


Figure 2.15. AXI Interconnect IP Configuration – External Subordinate Settings



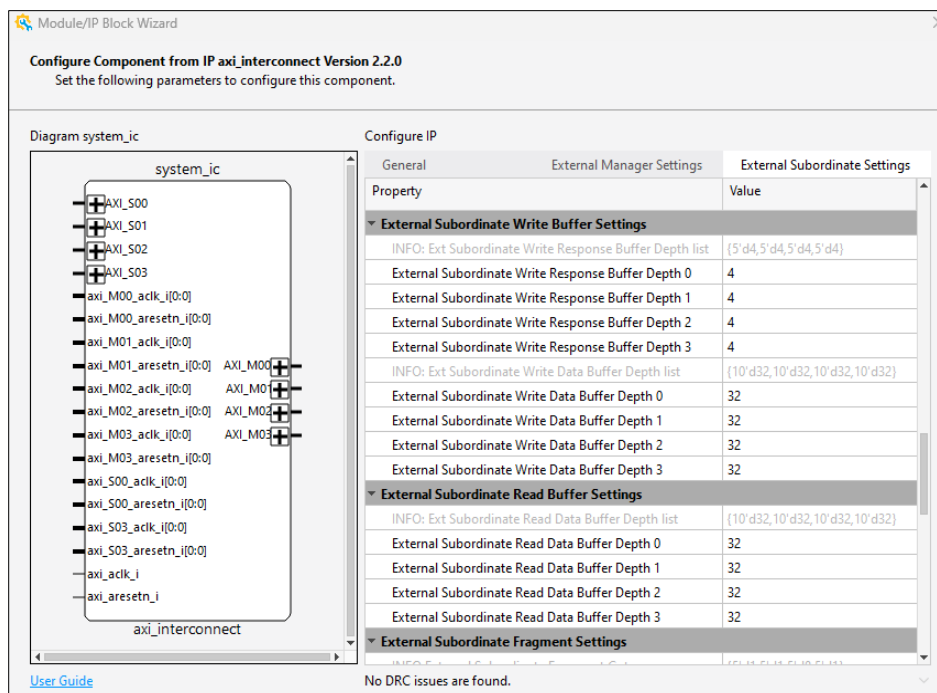


Figure 2.18. AXI Interconnect IP Configuration – External Subordinate Settings

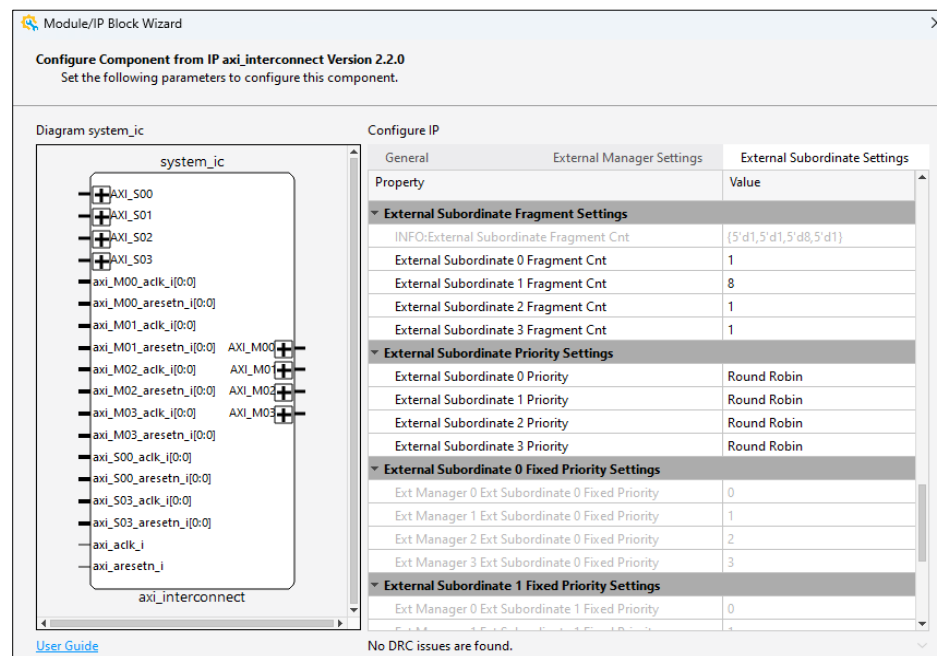


Figure 2.19. AXI Interconnect IP Configuration – External Subordinate Settings

2.5.3. APB Interconnect

The APB Interconnect IP is used in system for connecting low-bandwidth, low-power peripherals to the main system bus.

For more information about the IP core , refer to [APB Interconnect IP User Guide\(FPGA-IPUG-02054\)](#).

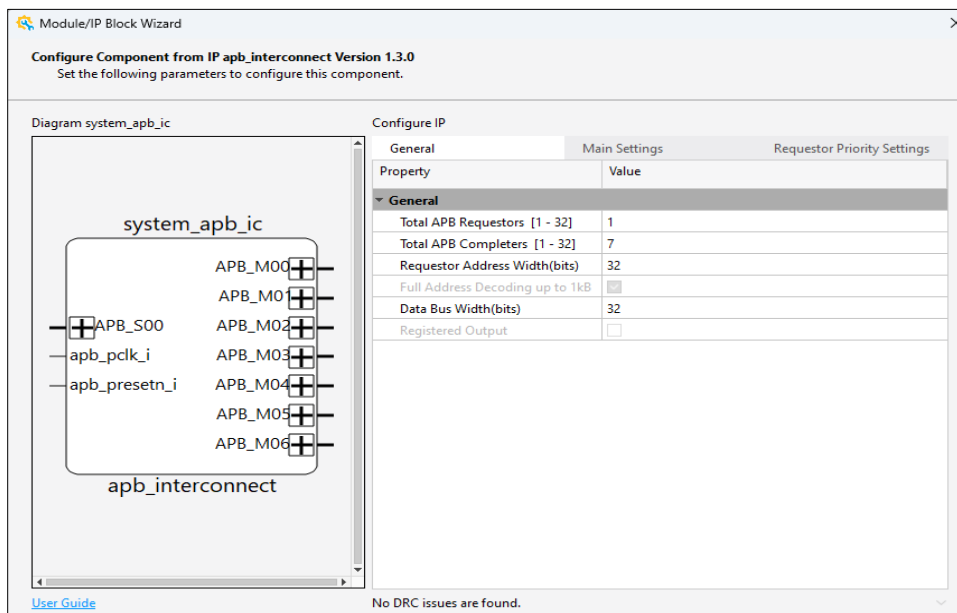


Figure 2.20. APB Interconnect IP Configuration – General

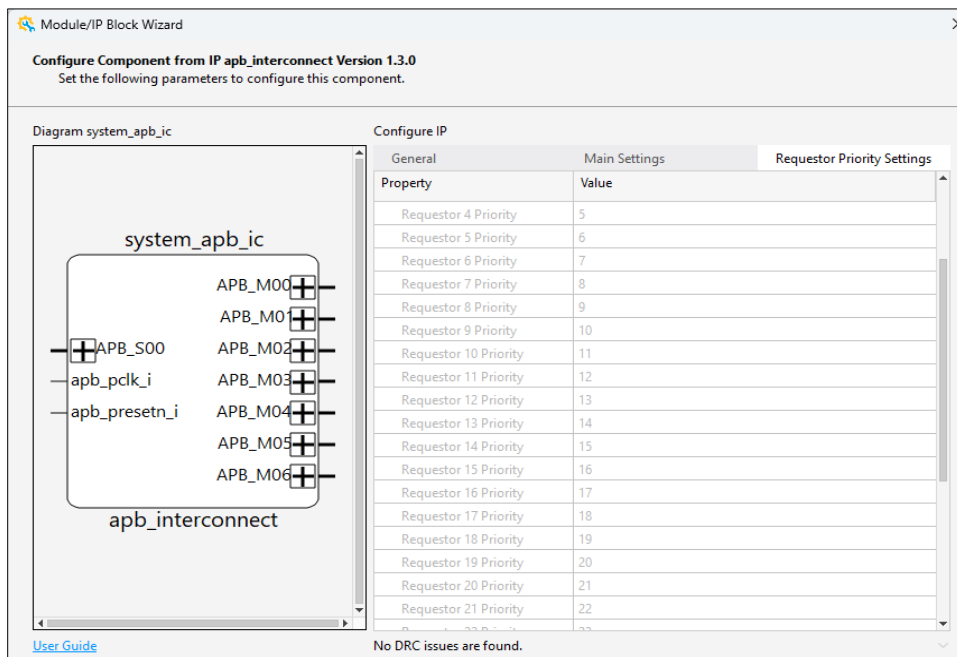


Figure 2.21. APB Interconnect IP Configuration – Requestor Priority Settings

2.5.4. AXI4 to APB Bridge

The AXI4 to APB Bridge IP is used to convert the AXI4 bus transaction into low-speed APB bus transaction. For more information about the IP, refer to [AXI4 to APB Bridge IP User Guide \(FPGA-IPUG-02198\)](#).

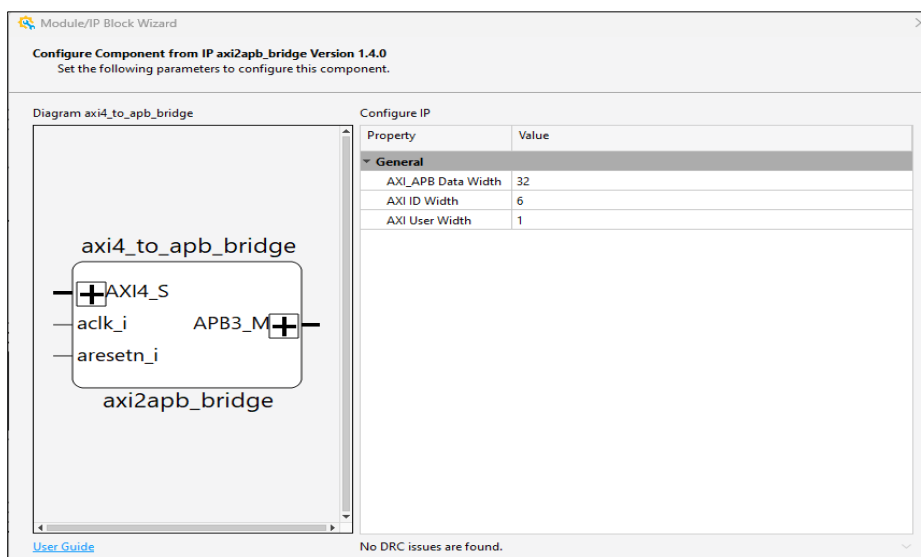


Figure 2.22. AXI4 to APB Bridge IP Configuration – General

2.5.5. LPDDR4 Memory Controller

The LPDDR4 Memory Controller IP enables access to the external LPDDR4 memory devices. The memory can be used to store CPU software code and data as well as Ethernet packet data.

For more information about the IP core including register map information, refer to [LPDDR4 Memory Controller IP Core for Nexus Devices \(FPGA-IPUG-02127\)](#).

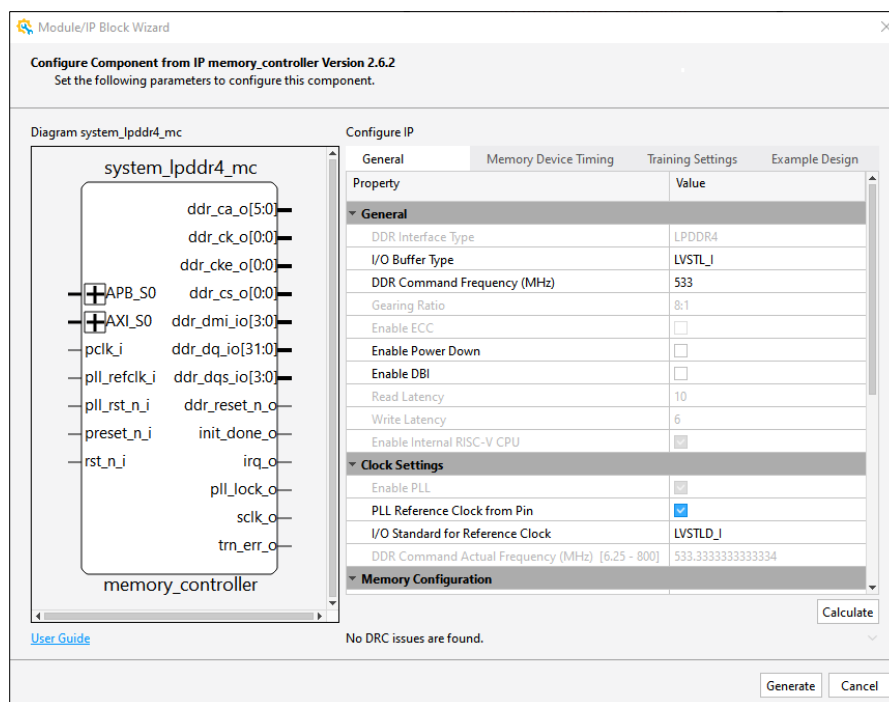


Figure 2.23. LPDDR4 Memory Controller IP Configuration – General

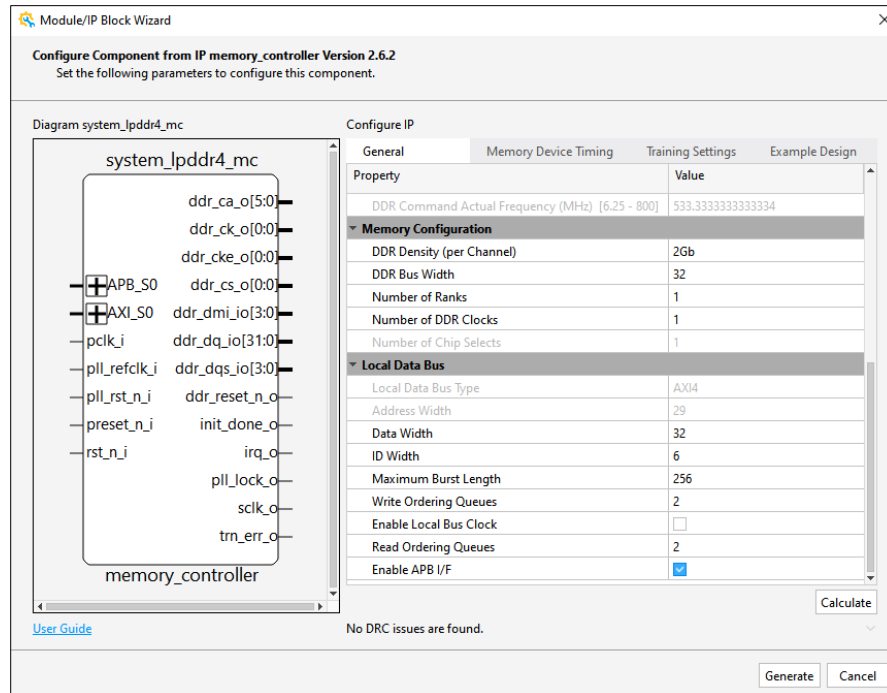


Figure 2.24. LPDDR4 Memory Controller IP Configuration – General

2.5.6. Octal SPI Controller

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

For more information about the IP core including register map information, refer to [Octal SPI Controller IP User Guide \(FPGA-IPUG-02273\)](#).

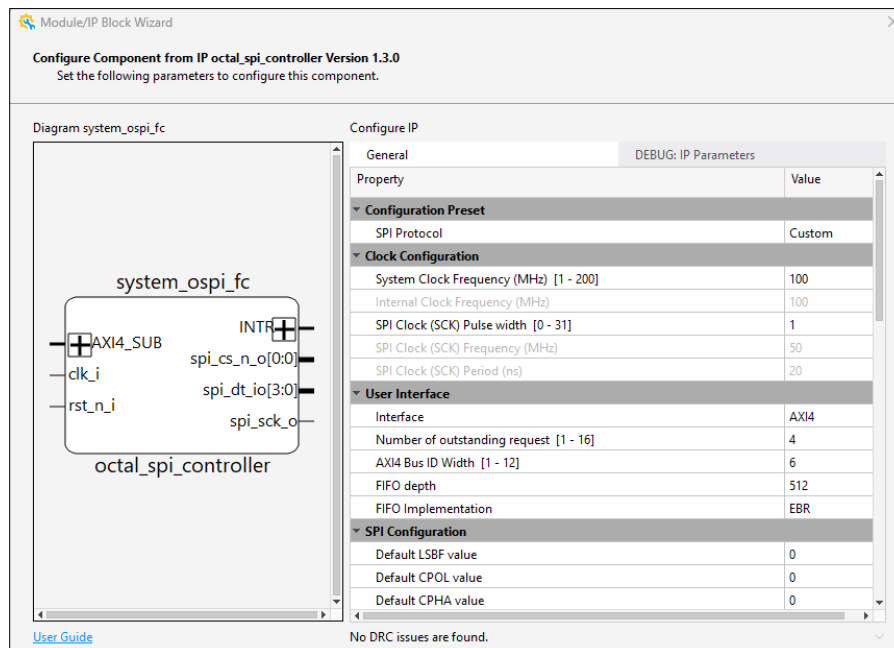


Figure 2.25. Octal SPI Controller IP Configuration – General

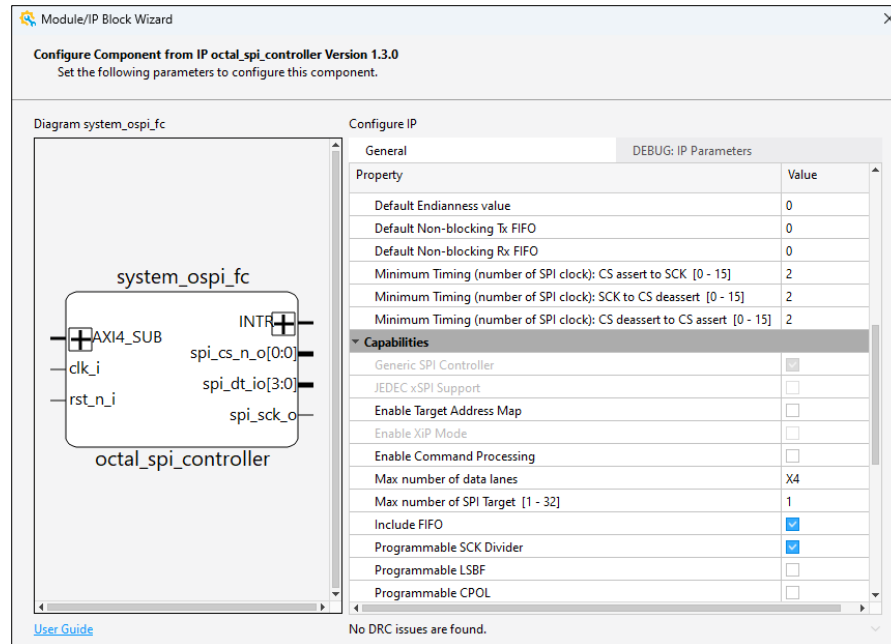


Figure 2.26. Octal SPI Controller IP Configuration – General

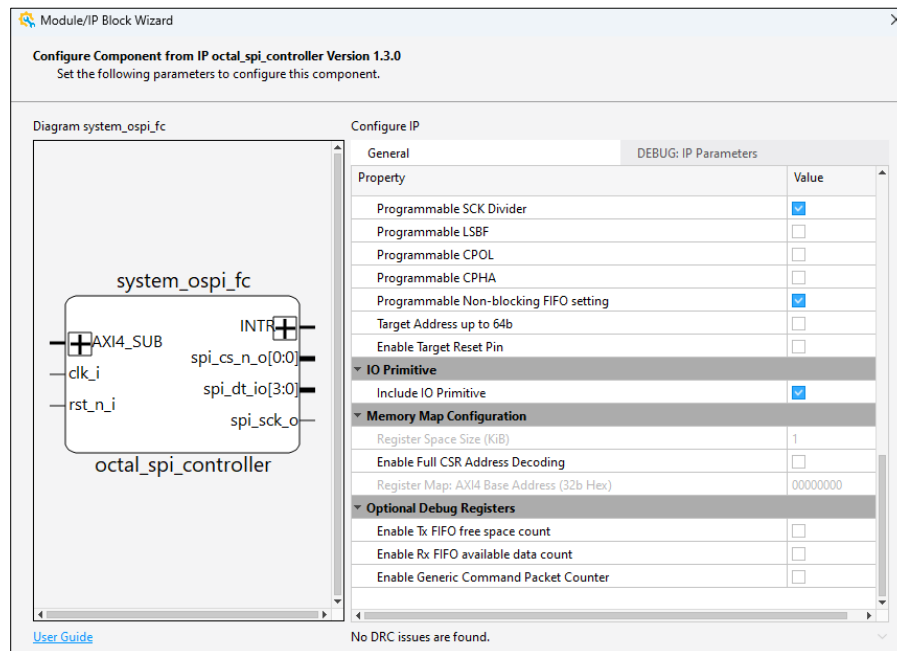


Figure 2.27. Octal SPI Controller IP Configuration – General

2.5.7. Tri-Speed Ethernet MAC + SGMII (SERDES)

The Tri-Speed Ethernet (TSE) IP solution consists of the TSE IP Media Access Controller (MAC) core and the SGMII GbE Physical Coding Sublayer (SGMII PCS) IP core. The TSE IP is a 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 CertusPro-NX GSRD supports 10/100/1000 Mbps network interface as per the IEEE 802.3 standard.

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

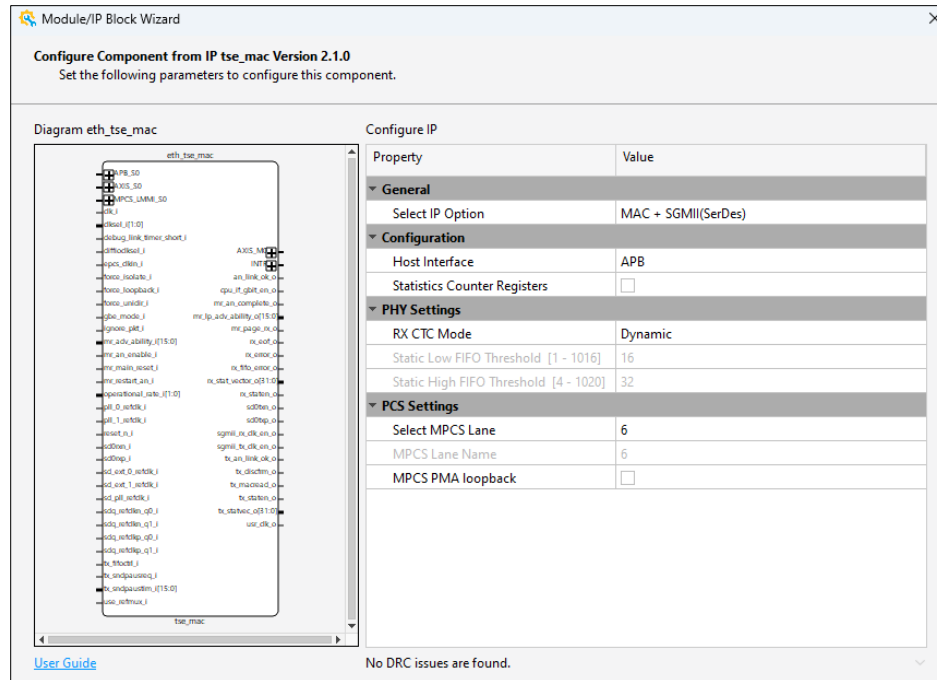


Figure 2.28. TSE IP Configuration

2.5.8. Scatter-Gather DMA Controller

The SGDMA Controller IP core is used to access the main memory independent of the CPU processor. It offloads processor intervention. The processor initiates transfer to SGDMA Controller and receive interrupt 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. The APB CSR interface is used to configure the control and status registers by the RISC-V CPU.

For more information about the IP core including register map information, refer to [SGDMA Controller IP User Guide \(FPGA-IPUG-02131\)](#).

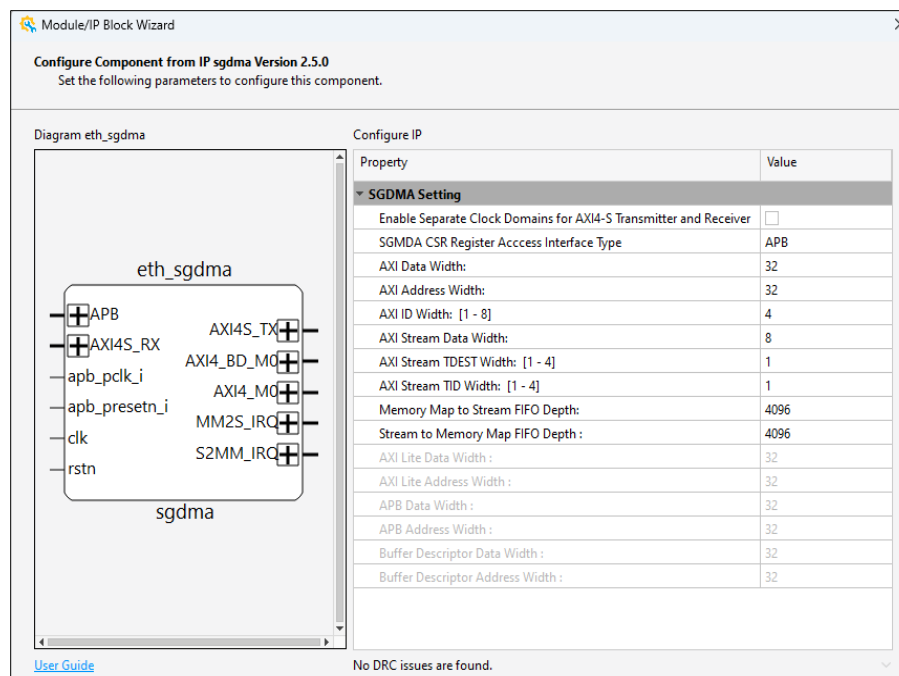


Figure 2.29. SGDMA Controller IP Configuration

2.5.9. UART

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 locator insider the FPGA through an APB interface.

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

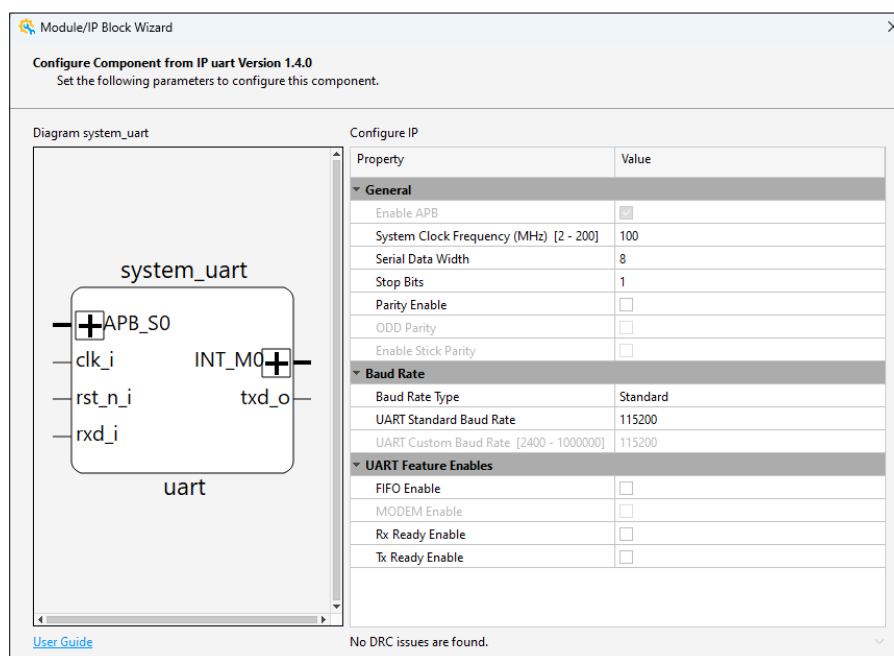


Figure 2.30. UART IP Configuration

2.5.10. GPIO

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.

For more information about the IP core including register map information, refer to [GPIO IP User Guide \(FPGA-IPUG-02076\)](#).

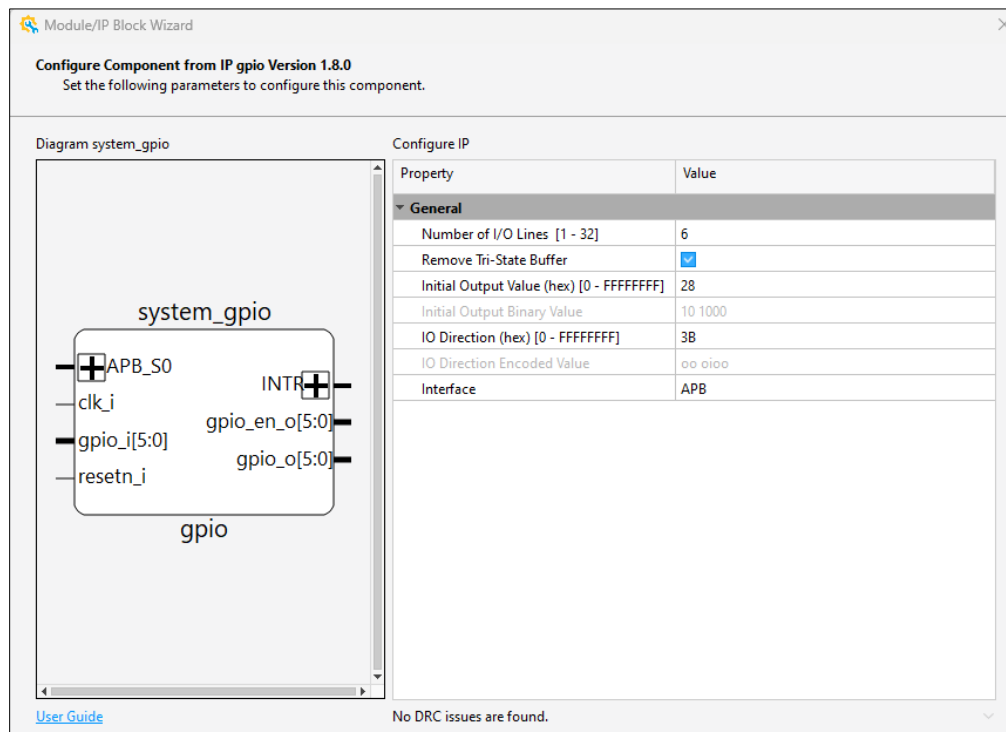


Figure 2.31. GPIO IP Configuration

2.5.11. I2C Controller

The I2C Controller IP core provides a standard two-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 is used to control the Marvell PHY register space inside the SFP connector.

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

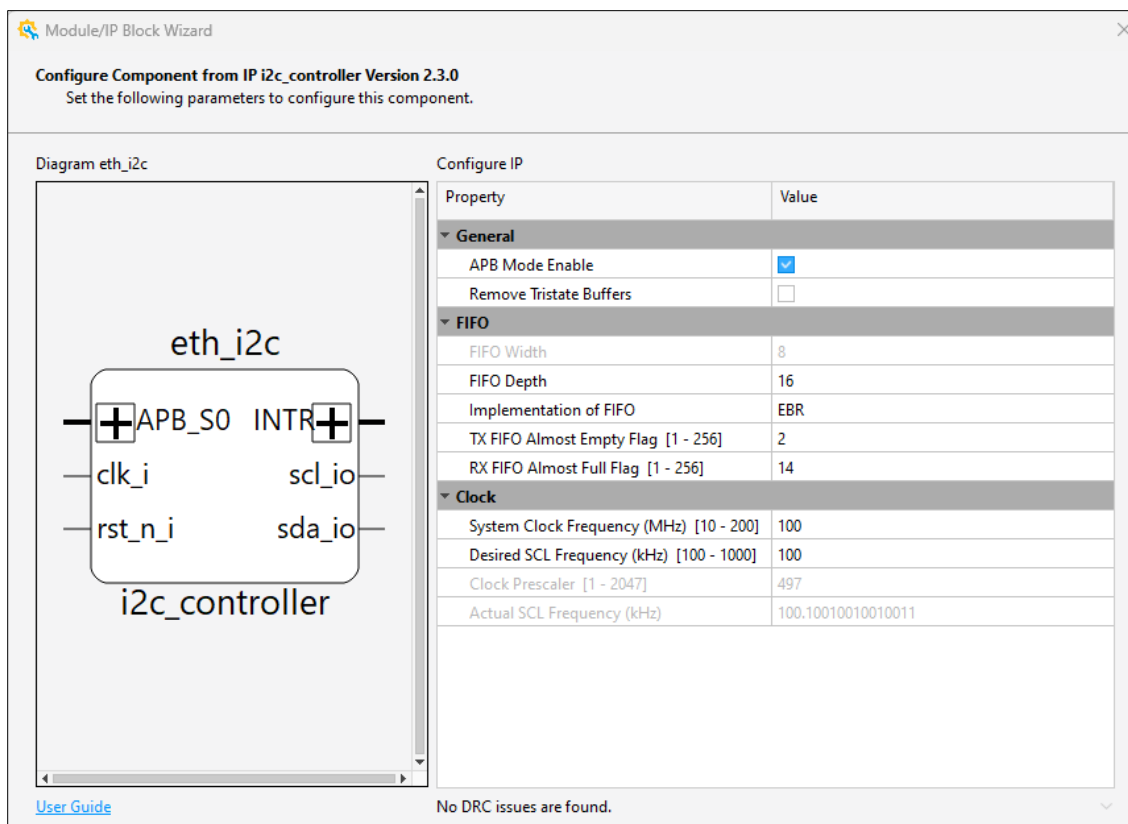


Figure 2.32. I2C IP Configuration

2.5.12. Multi-Boot Configuration Module

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

The OSC IP is instantiated inside module to generate clocks required for logic, LMMI config block as well as low frequency clock for RISC-V CPU.

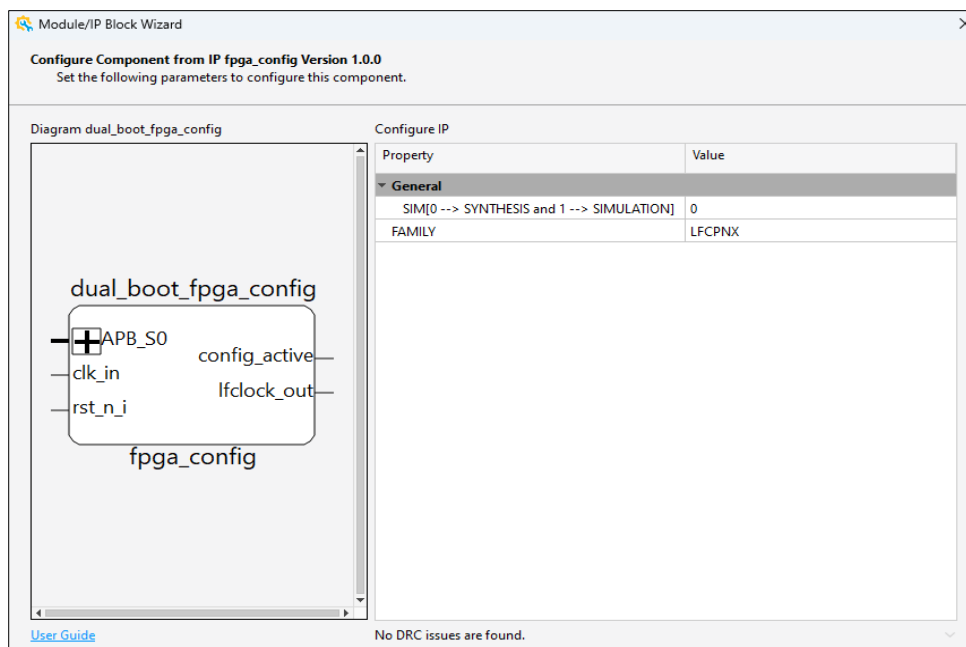


Figure 2.33. Multi-Boot Module Configuration

2.5.13. System Memory

The System Memory implements EBR, LRAM, or Distributed Memory in either single port or dual port AHB-L or AXI4 subordinate. In GSRD, System Memory is configured with LRAM as the memory type and single port AXI4 as the interface. The System Memory in this design is used to store the bootloader. The system's memory size is 128 kB.

For more information about the IP, refer to [System Memory IP User Guide \(FPGA-IPUG-02073\)](#).

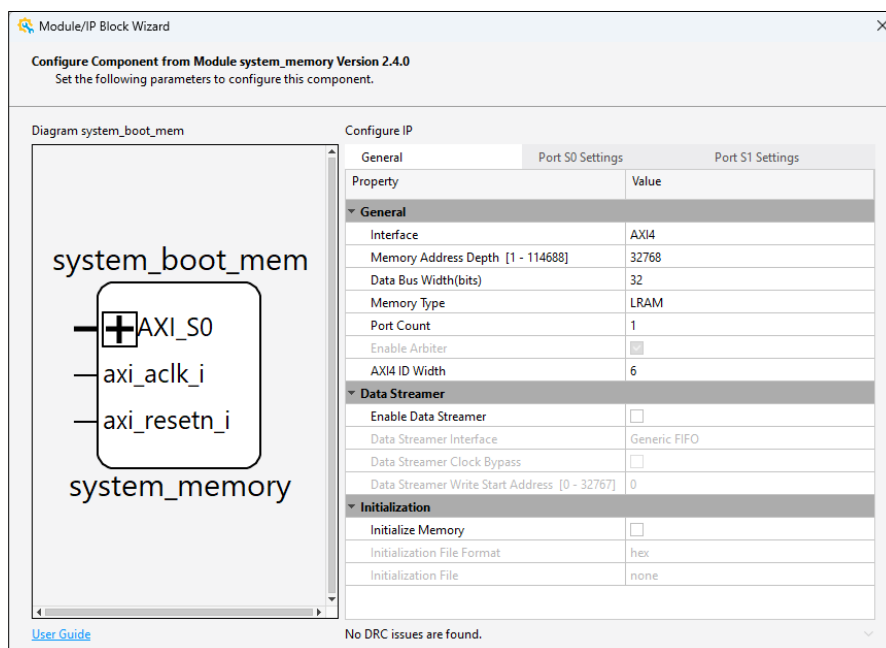


Figure 2.34. System Memory IP Configuration -General

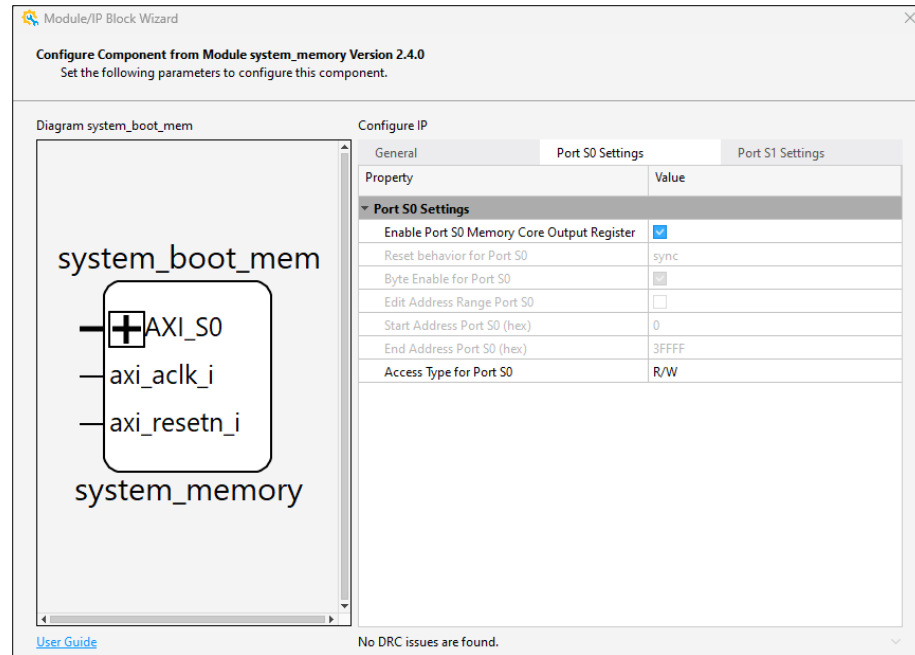


Figure 2.35. System Memory IP Configuration -Port S0 Settings

2.5.14. PLL

The PLL IP is used to generate multiple clocks used in the system. There is one PLL instance used in GSRD, system_pll. For more information about the IP, refer to [PLL Module User Guide \(FPGA-IPUG-02063\)](#).

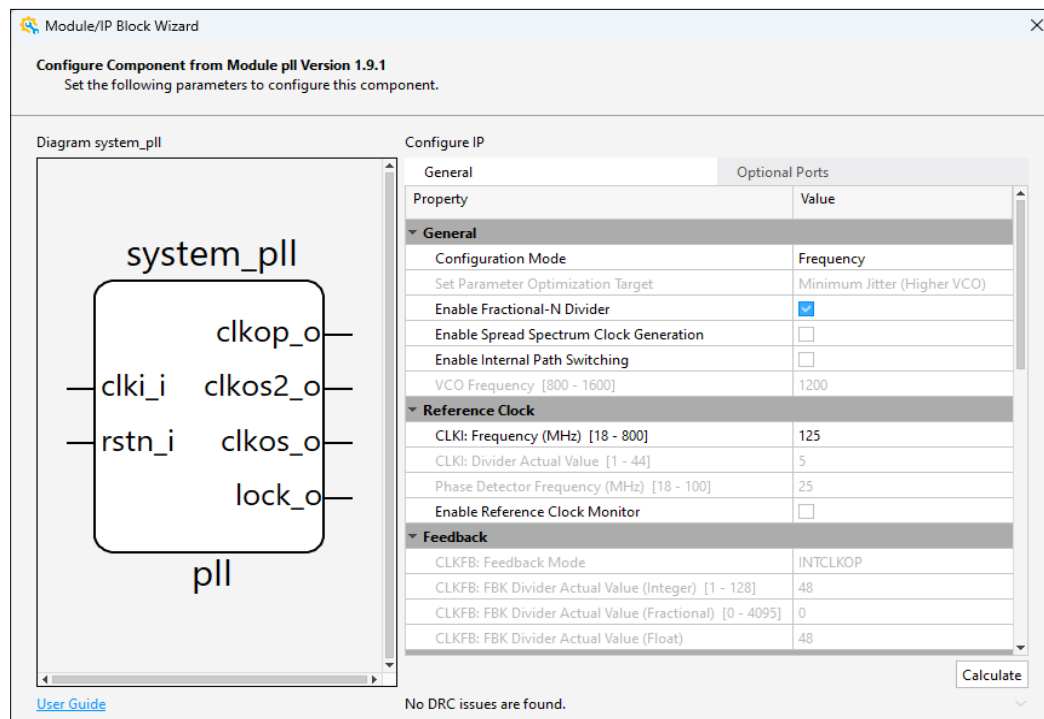


Figure 2.36. PLL IP Configuration -General

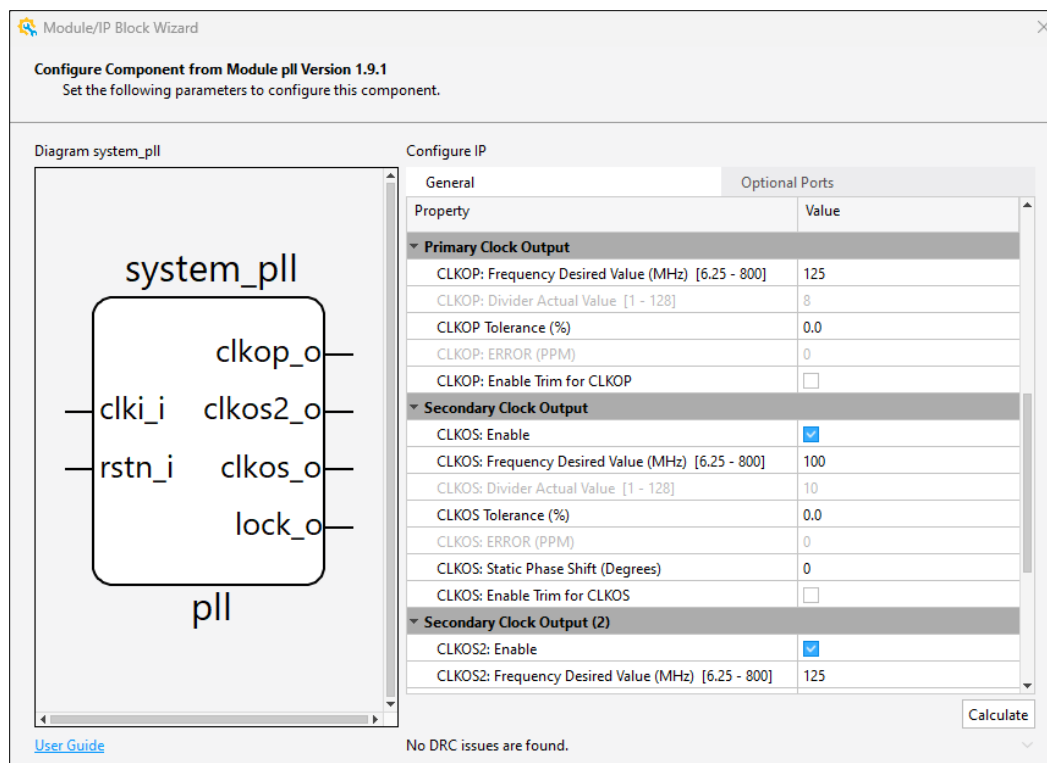


Figure 2.37. PLL IP Configuration – General

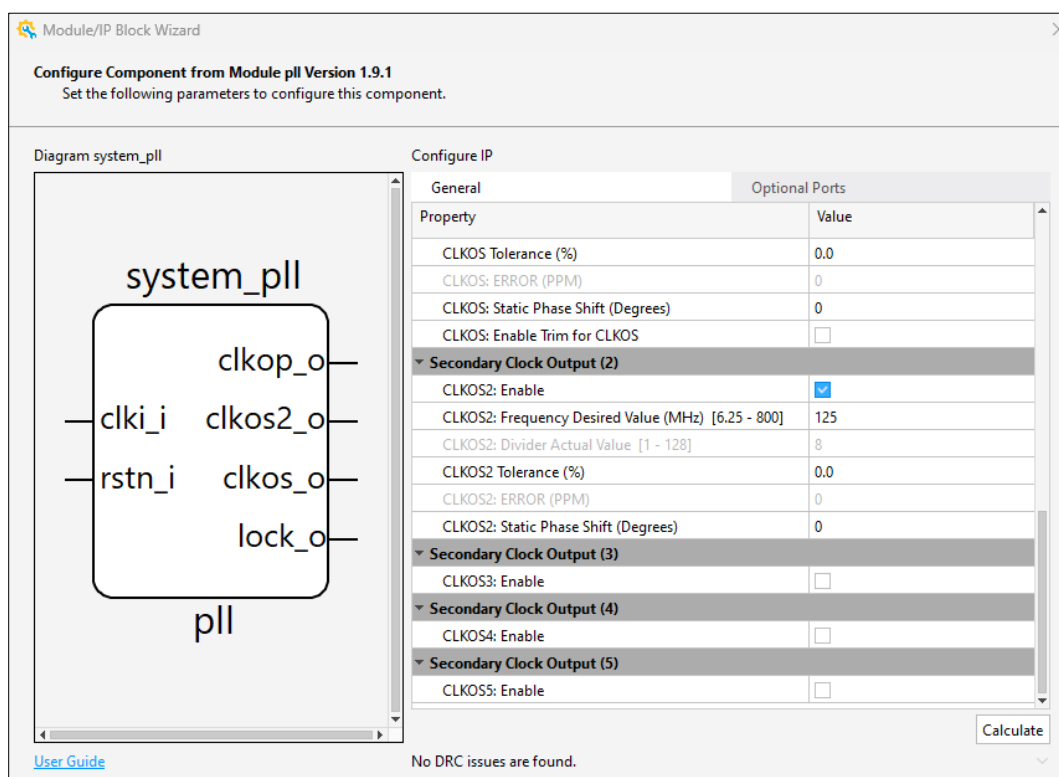


Figure 2.38. PLL IP Configuration – General

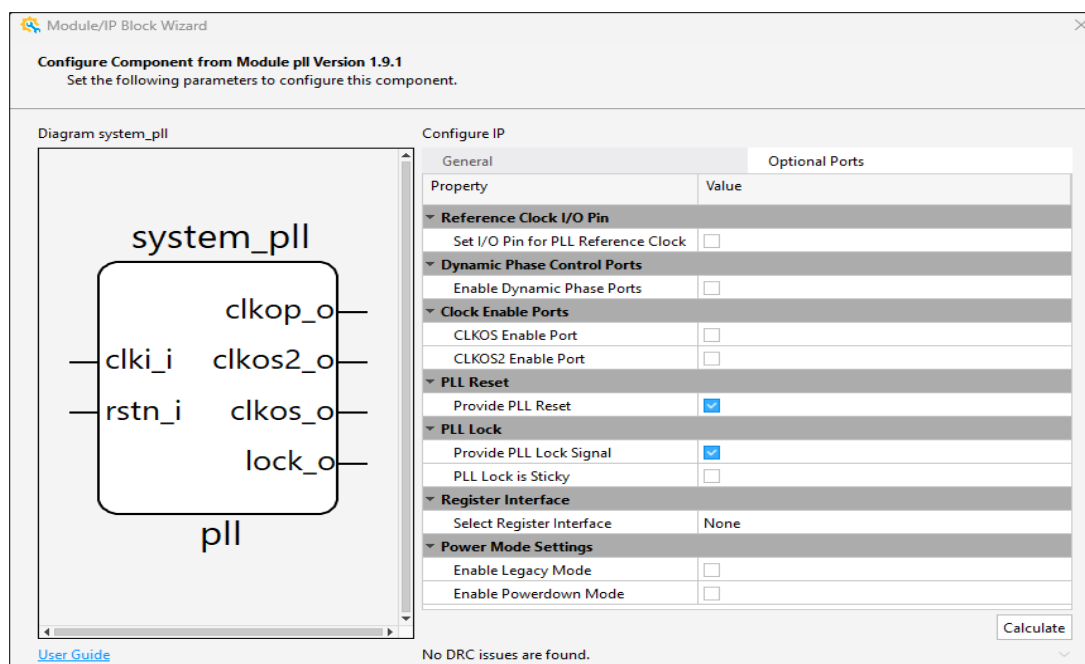


Figure 2.39. PLL IP Configuration – Optional Ports

2.5.15. Reset Modules

A reset synchronizer module ensures that an asynchronous reset signal is safely brought into a synchronous clock domain without causing metastability or glitches on reset signals.

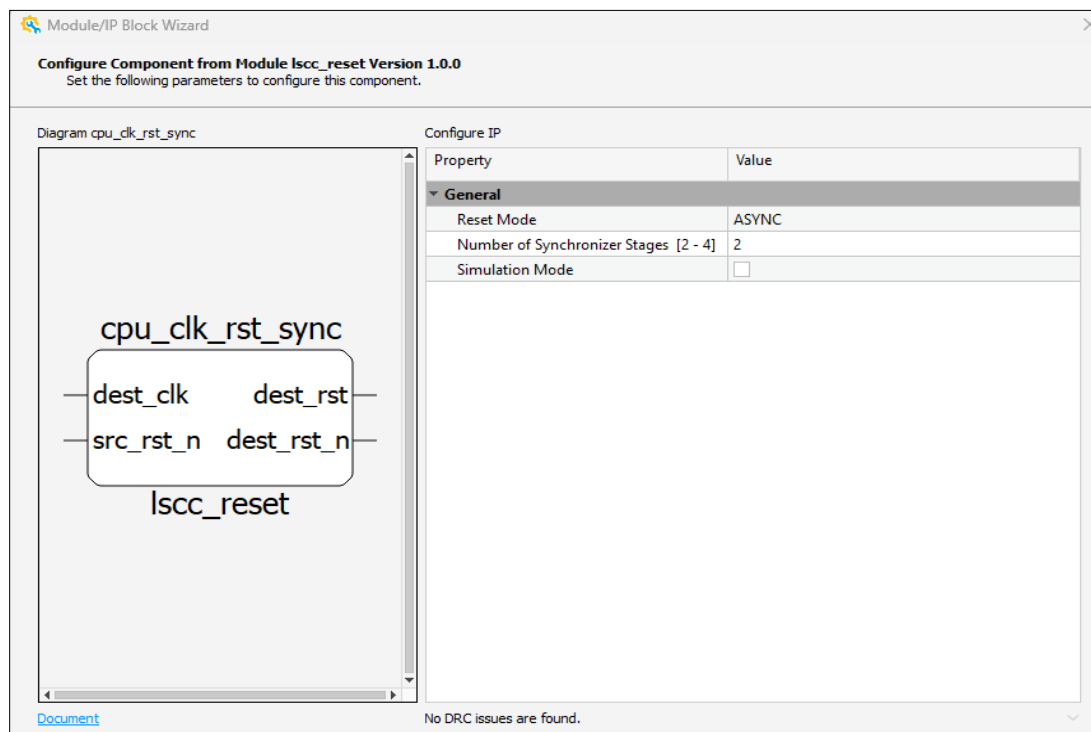


Figure 2.40. Reset Module Configuration – General

2.5.16. AXIS FIFO Module

The AXIS FIFO IP is used to buffer data between two different clock domains while maintaining the AXI stream protocol.

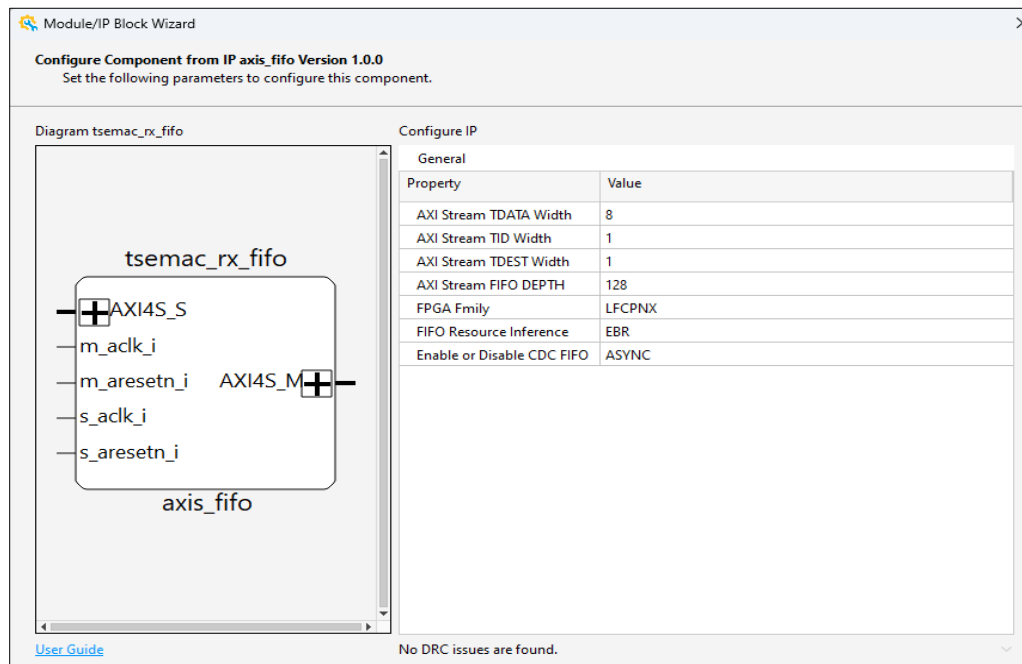


Figure 2.41. AXIS RX FIFO Module Configuration – General

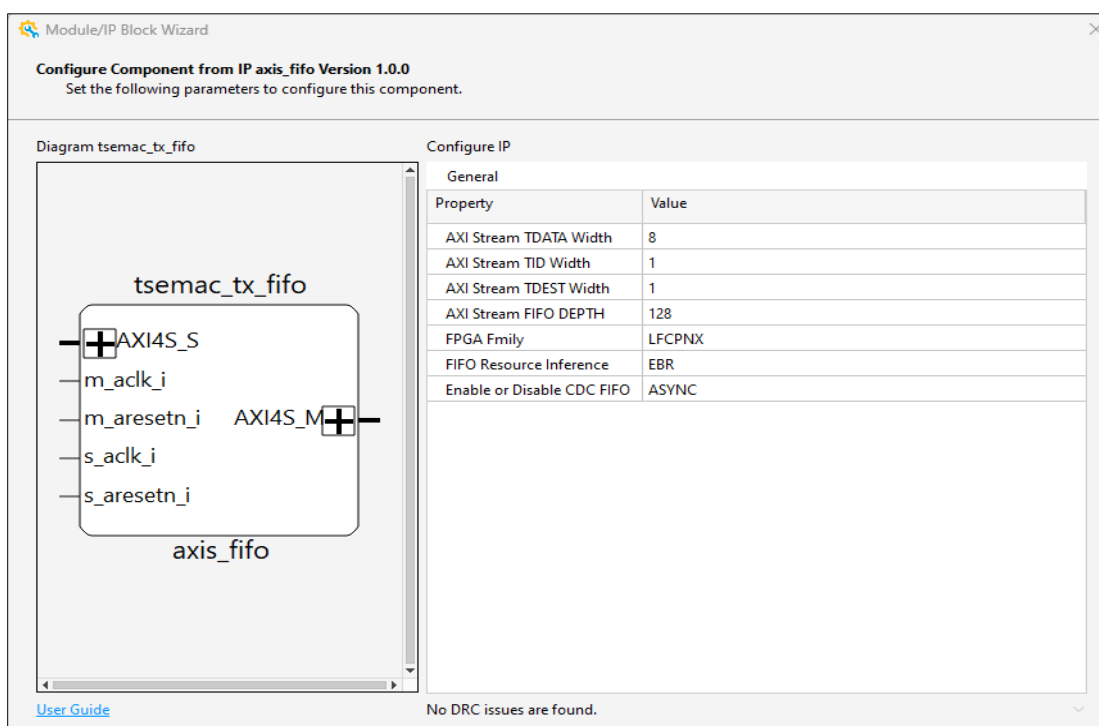


Figure 2.42. AXIS TX FIFO Module Configuration – General

2.6. System Level Interfaces

Table 2.5. System Level Interfaces

Top-Level Interface Name	Supported Protocol	Description
Advanced eXtensible Interface 4	AXI4	Used for Data/Control Interfaces on all IPs
Advanced Peripheral Interface	APB	Used as Control Interface for low-speed IP and LPDDR4 MC
Serial Peripheral Interface	SPI	Used for communication with external SPI Flash
Dual Data Rate	LPDDR4	Used for communication with external LPDDR4 SDRAM
10/100/1000 Mbps Ethernet	SGMII	Used for communication with external SFP module
Inter-Integrated Circuit	I2C	Used for configuring the external PHY in the SFP module
Universal Asynchronous Receiver/Transmitter	UART	Used for CPU printouts
General Purpose I/O	GPIO	Used for LEDs to indicate the SGMII link speed

2.7. SoC Memory/Address Map

Table 2.6. Address Map of GHRD

Base Address	End Address	Size (KB/MB)	Subordinate
0x8000_0000	0x8001_FFFF	128 kB	System Memory
0x0000_0000	0x1FFF_FFFF	512 MB	LPDDR4 AXI
0xC000_0000	0xC000_0FFF	4 kB	UART APB
0xC000_1000	0xC000_1FFF	4 kB	GPIO APB
0xC000_2000	0xC000_2FFF	4 kB	Multi-Boot Config APB
0xC000_3000	0xC000_3FFF	4 kB	SGDMA Controller APB
0xC000_4000	0xC000_7FFF	16 kB	TSE APB
0xC000_8000	0xC000_AFFF	12 kB	Reserved
0xC000_B000	0xC000_BFFF	4 kB	LPDDR4 APB
0xC000_C000	0xC000_CFFF	4 kB	I2C Controller APB
0xC000_D000	0xC3FF_FFFF	—	Reserved
0xC400_0000	0xC400_0FFF	4 kB	Octal SPI Controller
0xC400_1000	0xF1FF_FFFF	—	Reserved
CPU Local Memory			
0xF200_0000	0xF20F_FFFF	1 MB	CLINT
0xFC00_0000	0xFC3F_FFFF	4 MB	PLIC
0xF000_0000	0xF000_03FF	1 kB	Reserved_Space1
0xF000_0400	0xF1FF_FFFF	31 MB	Reserved_Space2
0xF210_0000	0xFBFF_FFFF	159 MB	Reserved_Space3
0xFC40_0000	0xFFFF_FFFF	60 MB	Reserved_Space4

2.8. 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.

Table 2.7. Design Constraints

Sr No	Constraint Type	File name	Comment
1	Clock	constraints.sdc	Lists all generated clock, created clock used by design.
2	Delay	<Project_name>.pdc	Lists inter board delay and false path definitions.
3	I/O	<Project_name>.pdc	Pin locking of all I/O and I/O standard matching board requirement.

2.9. Resource Utilization

Figure 2.43 shows the approximate GHRD resource utilization and Table 2.8 shows the total LUT4, PFU register, I/O buffer, and EBR resource utilization for the LFCPNX-100 device.

impl_1	LUT4	Logic	istributed RAM	Ripple Logic	PFU Registers	IO Registers	IO Buffers	DSP MULT	EBR	Large RAM
▼ soc_golden_gsrd	35903(2)	26807(2)	4482(0)	4614(0)	26501(157)	2(0)	75(16)	6(0)	80(0)	2(0)
▶ tsemac_tx_fifo_inst	74(0)	48(0)	0(0)	26(0)	72(0)	0(0)	0(0)	0(0)	1(0)	0(0)
▶ tsemac_rx_fifo_inst	74(0)	48(0)	0(0)	26(0)	72(0)	0(0)	0(0)	0(0)	1(0)	0(0)
▶ system_uart_inst	236(0)	190(0)	0(0)	46(0)	146(0)	1(0)	0(0)	0(0)	0(0)	0(0)
▶ system_pll_inst	1(0)	1(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ system_ospi_fc_inst	1640(0)	1206(0)	192(0)	242(0)	1231(0)	0(0)	4(0)	0(0)	2(0)	0(0)
▶ system_lpddr4_mc_inst	11733(0)	9301(0)	1260(0)	1172(0)	8513(0)	1(0)	49(0)	0(0)	33(0)	0(0)
▶ system_ic_inst	8705(3)	5275(3)	3006(0)	424(0)	5594(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ system_gpio_inst	90(0)	90(0)	0(0)	0(0)	75(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ system_boot_mem_inst	1263(0)	1179(0)	0(0)	84(0)	658(0)	0(0)	0(0)	0(0)	0(0)	2(0)
▶ system_apb_ic_inst	130(0)	130(0)	0(0)	0(0)	8(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ sys_clk_rst_sync_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ riscv_rx_cpu_inst	4823(1)	3951(1)	6(0)	866(0)	3595(0)	0(0)	0(0)	6(0)	18(0)	0(0)
▶ perip_clk_rst_sync_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ mac_clk_rst_sync_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ lpddr4_scl_rst_sync_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ jtaghub_top_inst	34(33)	20(19)	0(0)	14(14)	53(53)	0(0)	4(4)	0(0)	0(0)	0(0)
▶ eth_tse_mac_sgmii_inst	3885(0)	2985(0)	18(0)	882(0)	3469(0)	0(0)	0(0)	0(0)	5(0)	0(0)
▶ eth_sgdma_inst	2287(0)	1633(0)	0(0)	654(0)	2090(0)	0(0)	0(0)	0(0)	18(0)	0(0)
▶ eth_i2c_inst	581(0)	455(0)	0(0)	126(0)	500(0)	0(0)	2(0)	0(0)	2(0)	0(0)
▶ dual_boot_fpga_config_inst	55(0)	55(0)	0(0)	0(0)	55(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ cpu_clk_rst_sync_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ axi_clk_rst_sync_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ axi4_to_apb_bridge_inst	290(0)	238(0)	0(0)	52(0)	201(0)	0(0)	0(0)	0(0)	0(0)	0(0)

Figure 2.43. LFCPNX-100 Device GHRD Approximate Resource Utilization

Table 2.8. GSRD Total Approximate Resource Utilization

Resource	Approximate Usage	Approximate Percentage Utilization
LUT4 (Logic + Distributed RAM + Ripple Logic)	35903	44.95%
PFU Register	26501	33.17%
I/O Buffers	75	25.08%
EBR	80	38.46%
LRAM	2	29%

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
pll_clk_i	Input	1	Reference clock input for internal PLLs
lpddr4_pll_refclk_i	Input	1	Reference clock input for LPDDR4 MC internal PLL
system_rstn_i	Input	1	Active low reset input for design. Activate by pressing push button SW3 on the board.
GPIO			
gpio_o	Input/Output	2	General Purpose I/O signals connect to LED D65 and D66 on board to indicate SGMII link speed. gpio_o[1:0] (D66:D65) 00: 10 Mbps D65 = OFF, D66 = OFF 01: 100 Mbps. D65 = OFF, D66 = ON 10: 1000 Mbps. D65 = ON, D66 = OFF 11: Not defined. D65 = ON, D66 = ON
UART			
uart_txd_o	Output	1	UART transmits output. Connects to the board TXD signal.
uart_rxd_i	Input	1	UART receives input. Connects to the board RXD signal.
Octal SPI Controller			
ospi_clk	Output	1	Serial clock to SPI Flash
ospi_ss_n_o	Output	1	SPI Flash chip selects
ospi_dt_io	Input/Output	4	Serial data between FPGA and external SPI Flash
LPDDR4 Memory Controller			
lpddr4_ca_o	Output	6	LPDDR4 command/address
lpddr4_ck_o	Output	1	LPDDR4 clock
lpddr4_cke_o	Output	1	LPDDR4 clock enable
lpddr4_cs_o	Output	1	LPDDR4 chip select
lpddr4_dmi_io	Input/Output	4	LPDDR4 data mask
lpddr4_dq_io	Input/Output	32	LPDDR4 Data
lpddr4_dqs_io	Input/Output	4	LPDDR4 data strobe
lpddr4_reset_n_o	Output	1	External Memory chip reset signal
lpddr4_init_done_o	Output	1	Connects to LED D64 for LPDDR4 initialization status check 0: Initialization and training are in progress. LED = OFF 1: Initialization and training are completed. LED = ON
lpddr4_pll_lock_o	Output	1	Connects to LED D67 for LPDDR4 PLL lock status check 0: PLL is unlocked. LED = OFF 1: PLL is locked. LED = ON
lpddr4_trn_err_o	Output	1	Connects to LED D105 for LPDDR4 training status check 0: Training passes. LED = OFF 1: Training fails. LED = ON
SGMII PCS/GbE			
sgmii_sd0rxp_i/ sgmii_sd0rxn_i	Input	1	SGMII PCS receive data from SFP
sgmii_sd0txp_o/ sgmii_sd0txn_o	Output	1	SGMII PCS transmits data from SFP
sgmii_an_link_ok_o	Output	1	Connects to LED D63 Ethernet auto-negotiation link status check 0: Link down. LED = OFF 1: Link up. LED = ON

Signal Name	I/O Type	I/O Width	Description
I2C			
sfp_i2c_scl	Input/Output	1	I2C Clock
sfp_i2c_sda	Input/Output	1	I2C Data
sfp_disable_o	Output	1	Connects to SFP module to TX disable pin
Multi-Boot			
config_active_o	Output	1	Connects to LED D68 to check Multi-Boot Configuration block status 0: Configuration is done. LED = OFF 1: Configuration is in progress. LED = ON

4. Software Components

The Golden System Reference Design (GSRD) is enabled by RISC-V core based on FreeRTOS and Lattice FPGA IP drivers. Lattice developed Board Support Package (BSP) 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.

Lattice GSRD uses the open-source lightweight TCP/IP (lwIP) Ethernet stack. It supports ICMP ping in lwIP TCP/IP Ethernet stack. This example demonstrates how GSRD receives an Internet Control Message Protocol (ICMP) packet Echo Request and sends an Echo Reply to the target which is Windows based or Linux based PC. The IP addresses for both the PC and GSRD are hardcoded to 192.168.1.2 and 192.168.1.4 respectively.

During a ping session, the PC sends out an ICMP echo request packet and wait for an ICMP echo reply from Lattice GSRD. The request packet is sent with a specific timestamp, and upon receiving a valid ICMP echo reply, the PC generates metrics such as packet loss, elapsed time, and other relevant data.

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 Memory Controller configuration, it first reads if the PLL Lock status is set and then initiates Memory Training and waits until training is completed. Configures the Octal SPI Controller to read the application software stored into external flash while FIFO is disabled using API and copies it into LPDDR4 SDRAM. It then calculates the CRC value on the entire application software copied into LPDDR4 SDRAM by 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

The Primary and Golden applications are implemented in FreeRTOS and lwIP with the following functions:

- Executed by RISC-V RX CPU from LPDDR4 SDRAM where it initializes the BSP and Operating System.
- Initialize the TSE IP by setting up the auto-negotiated Ethernet link speed (10/100/1000 Mbps), MAC address and full duplex mode based on the selected configuration.
- 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 lwIP with the IP address (default to 192.168.1.4), and setup the netmask and gateway address for ping connection.
- Configure the SGDMA Controller MM2S (Memory-Mapped to Streaming) interface for transferring the Ethernet ping packets to TSE MAC and set up S2MM (Streaming to Memory Mapped) interface to receive ping packets from the TSE MAC.
- Runs FreeRTOS scheduler and Task Handler. Every incoming Ethernet ping packet is displayed with some minor dots' indicator on the UART terminal.

5. Theory of Operation

The GSRD system comprises two System on Chip (SoC) designs and corresponding software stacks, called as Primary and Golden images. Each SoC design is linked with its respective bootloader and FreeRTOS application software. These SoCs are built using Lattice Propel Builder, and Lattice Radiant software tools. The Lattice Propel SDK provides a software development environment for firmware development. The FPGA image comprises 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 in 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 initializes 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 the Octal SPI Controller to copy the FreeRTOS application software from external SPI Flash into the external LPDDR4 memory device for software execution. Once the application software is copied, the RISC-V checks the CRC on the 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 software images, and the same process follows. Once CRC check passes for either Primary or Golden application software, the RISC-V program counter jumps to the application instruction's starting address and starts the FreeRTOS application software execution.

5.1. Boot-Up Sequence

This section describes the RISC-V RX CPU boot up sequence in detail that configures IP drivers, operating modes and bootloader.

- 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 into the external SPI Flash. Upon power-on boot up, the Primary bitstream is loaded to the SRAM by Config Engine to configure the FPGA. Then, the bootloader configures the peripherals and GSRD building blocks such as UART, GPIO, LPDDR4 Memory Controller and Octal SPI Controller.
- The bootloader fetches the respective Primary application software through the Octal SPI Controller into System memory.
- As the application software needs to be executed from external memory, the RISC-V CPU loads the application software into LPDDR4 memory device.
- This application software is stored at the beginning of the LPDDR4 memory mapped address. After the application software is loaded, the RISC-V CPU calculates the CRC of the application code in LPDDR4.
- The CRC of the application software in LPDDR4 memory device is calculated and validated against the reference CRC embedded in SPI flash.
- If the calculated CRC matches the reference CRC, the RISC-V CPU jumps to FreeRTOS execution in LPDDR4 memory device.
- If the calculated CRC mismatches the reference CRC, the RISC-V CPU issues a FPGA REFRESH command to load the Golden bitstream and application software from the SPI Flash.
- Octal SPI Controller performs a self-diagnostic check for read, write and erase operations. Refer to [Appendix D. Using Octal SPI Controller for Read, Write, and Erase Self-Diagnostic Check \(FreeRTOS Application Software\)](#) for more information.
- FreeRTOS initializes SGDMA Controller, TSE IP, I2C Controller IP, and uses I2C Controller to configure PHY settings in SFP module for Ethernet connection.
- LwIP is enabled with the configuration in TSE IP for Ethernet connection.
- FreeRTOS executes the task scheduler.

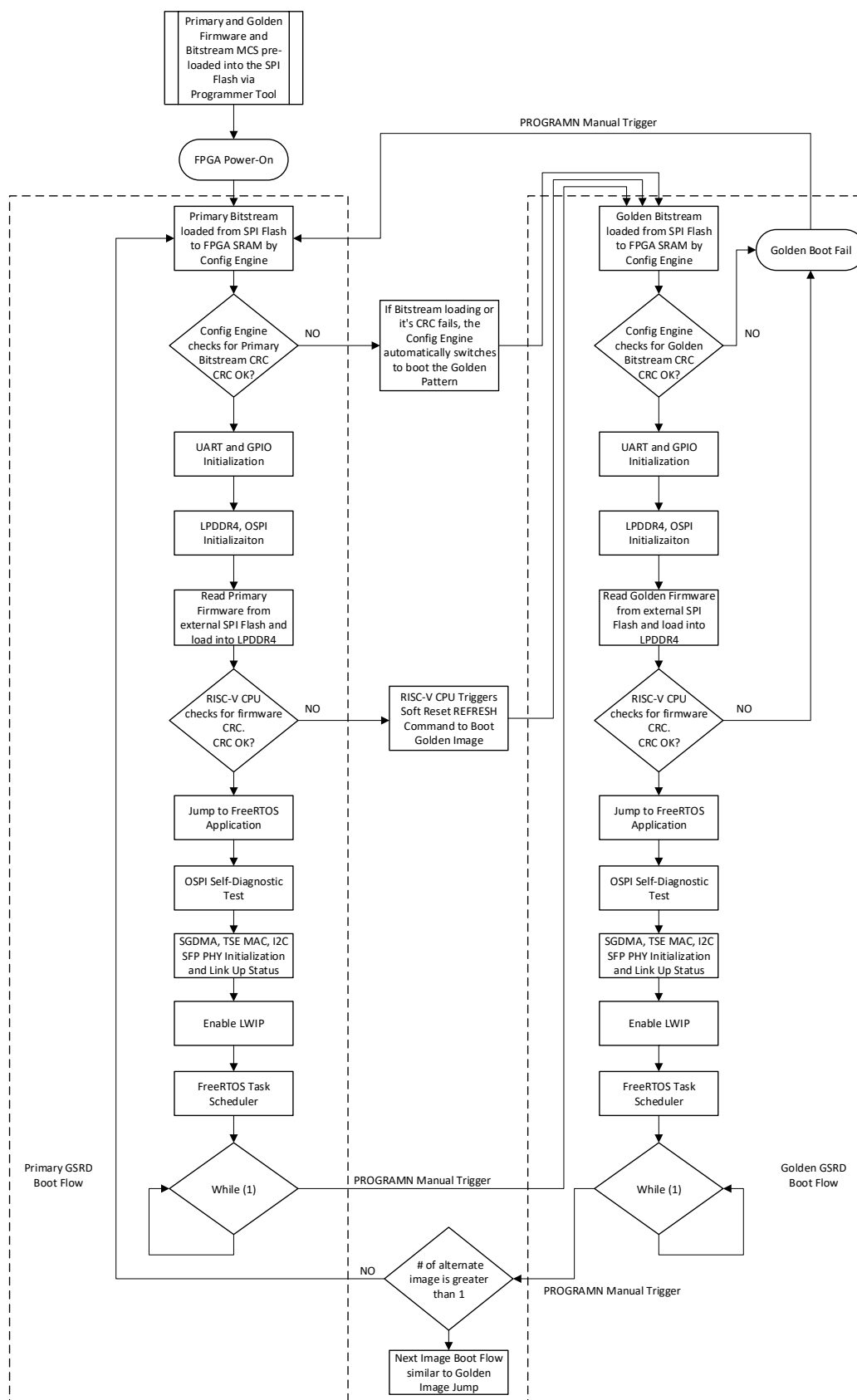


Figure 5.1. GSRD Boot-Up Sequence

5.2. Data Movement

This section describes the RISC-V RX CPU FreeRTOS application software execution in detail.

- Upon the execution of the correct Primary or Golden application software from LPDDR4 SDRAM, 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 Controller accordingly with receive buffer's address, data length and other configuration modes to route the incoming Ethernet packets.
- When the Ethernet frame is received by the TSE IP, it forwards it to SGDMA Controller to transfer the data to receive buffer in LPDDR4 SDRAM.
- When data is written into the LPDDR4 SDRAM, SGDMA Controller triggers interrupt to RISC-V CPU to acknowledge the data transfer is completed. RISC-V CPU then clears the SGDMA Controller's interrupt status.
- Once interrupt is serviced, the RISC-V goes back to the main task scheduler to execute the next task.

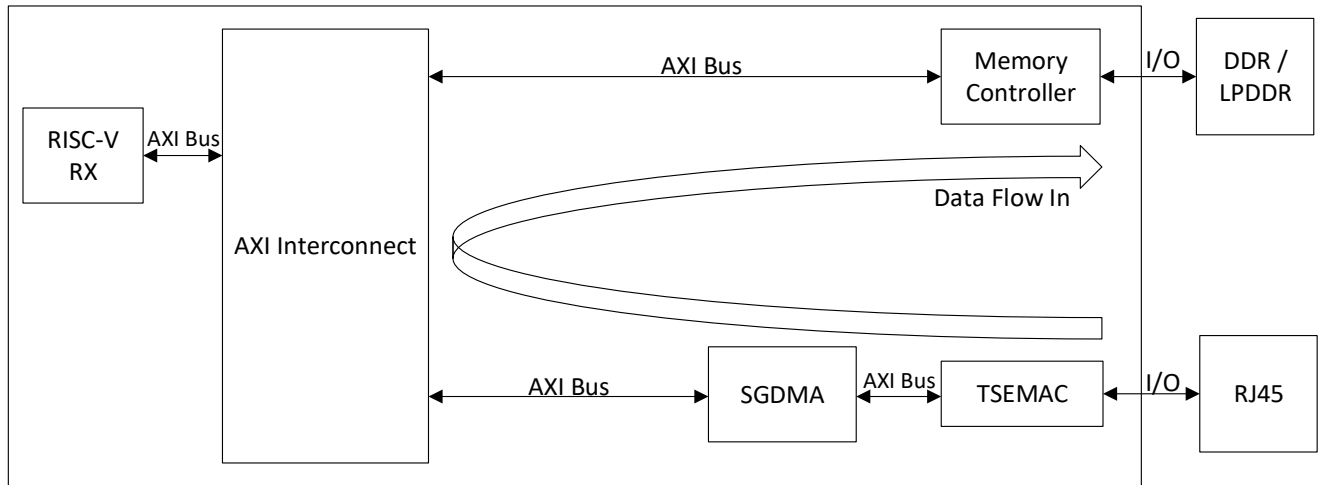


Figure 5.2. Ethernet Data RX Flow

- For outgoing data, data is fetched from a location inside LPDDR4 SDRAM by RISC-V CPU. The SGDMA Controller transfers the data to TSE IP for transmission outside the FPGA.
- When no data activity occurs over Ethernet, the RISC-V CPU continues running idle task in FreeRTOS.

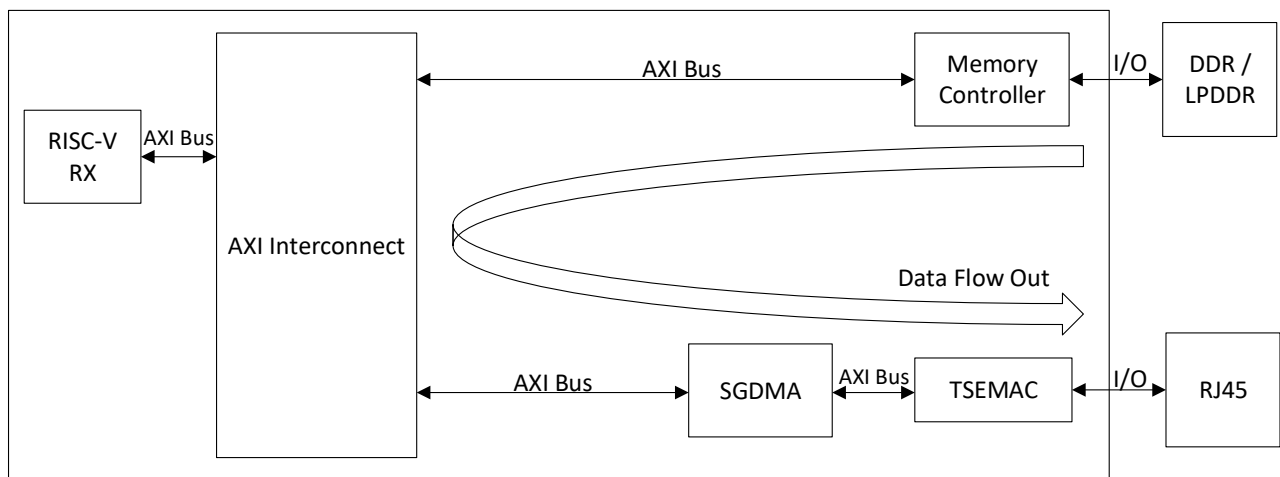


Figure 5.3. Ethernet Data TX Flow

6. Running the GSRD Demonstration

This section describes the procedure for running the GSRD demonstration using the pre-built executables and binary files in the design package. You can skip this chapter if you do not want to run the GSRD demonstration or reference design on hardware.

6.1. Executables

This section provides the directory structure, file names and locations of the executables (SPI Flash) required for running the GSRD demonstration. Your CertusPro-NX Versa Board has either flash device from Winbond or Macronix as stated in [Table 6.1](#). Refer to the [Hardware Requirements](#) section to identify which flash device is mounted on your board.

Table 6.1. Supported Flash Devices

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

- 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 V3.0 – 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 6.1](#) and [Table 6.2](#).

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

Table 6.2. Executable Files for Winbond Flash

File Description	File Name	Starting Address in SPI Flash
Primary Software with CRC	c_primary_appcrc_wb.bin	0x028A 0000
Primary FPGA Bitstream	soc_primary_gsr_d_wb_impl1.bit	0x0000 0000
Golden Software with CRC	c_golden_appcrc_wb.bin	0x0280 0000
Golden FPGA Bitstream	soc_golden_gsr_d_wb_impl1.bit	0x0000 0000
Multi-Boot MCS File (Golden + Primary Bitstream)	multiboot_system_wb.mcs	0x0000 0000

Table 6.3. Executable Files for Macronix Flash

File Description	File Name	Starting Address in SPI Flash
Primary Software with CRC	c_primary_appcrc_mcrx.bin	0x028A 0000
Primary FPGA Bitstream	soc_primary_gsr_d_mcrx_impl1.bit	0x0000 0000
Golden Software with CRC	c_golden_appcrc_mcrx.bin	0x0280 0000
Golden FPGA Bitstream	soc_golden_gsr_d_mcrx_impl1.bit	0x0000 0000
Multi-Boot MCS File (Golden + Primary Bitstream)	multiboot_system_mcrx.mcs	0x0000 0000

6.2. Setting Up the Hardware

This section provides the procedure for setting up the CertusPro-NX Versa Evaluation board for GSRD demonstration as shown in [Figure 6.1](#) and [Figure 6.2](#).

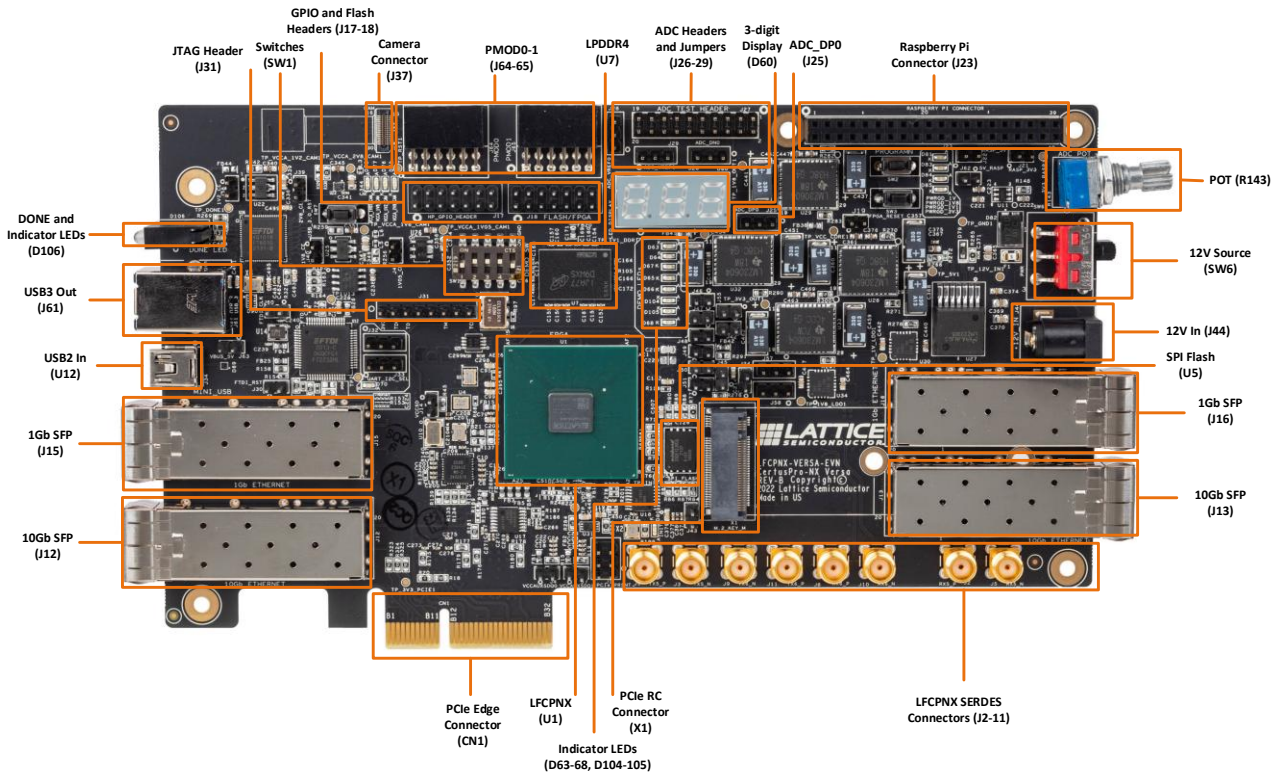


Figure 6.1. CertusPro-NX Versa Evaluation Board

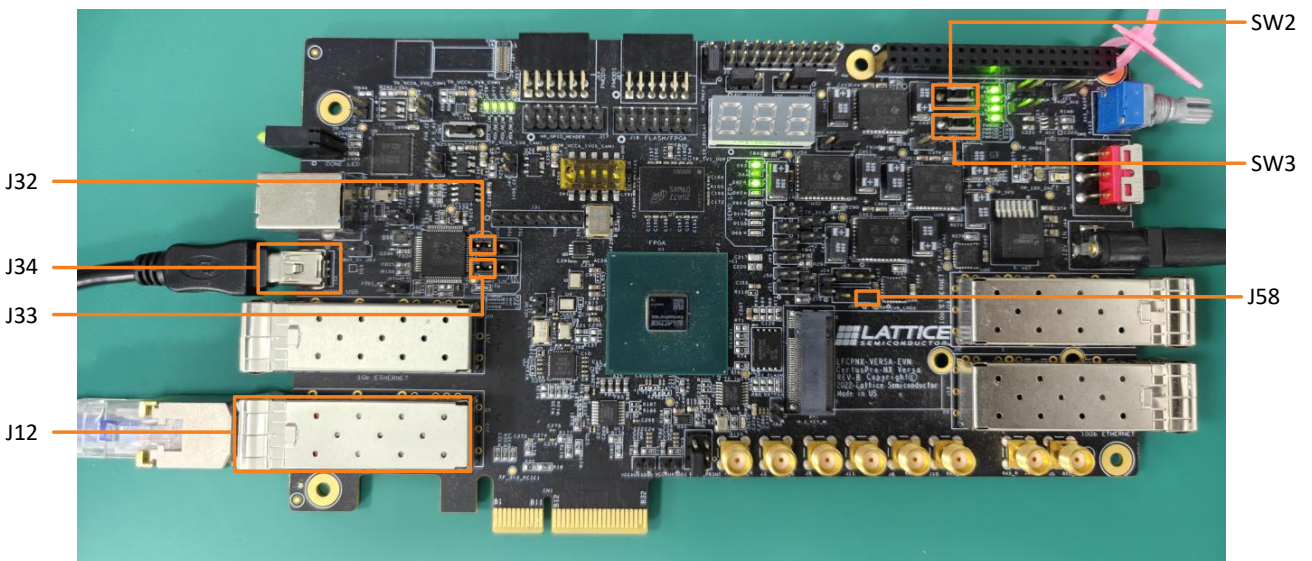


Figure 6.2. Connections, Jumpers and Switches Needed for Demonstration

To set up the hardware, perform the following:

1. Connect the 12 V power adapter to J44 DC input supply jack.
2. Connect the Mini-USB Type-A cable from PC to J34.
3. Connect jumpers on Pin 1 and Pin 2 on both J32 and J33 headers to enable UART.
4. Connect jumper on Pin 1 and Pin 2 on J58 header to enable 3.3 V V_{CCIO} on Bank 0.

5. Ensure the rest of the jumpers are placed at the default positions as stated in [CertusPro-NX Versa Evaluation Board User Guide \(FPGA-EB-02053\)](#).
6. Insert the Ethernet SFP module onto J12 connector.
 - a. Connect the Ethernet RJ45 cable from the host PC cable to the Ethernet SFP module.
7. Turn on power switch SW6.

6.3. Setting Up the UART Terminal

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

To set up the UART terminal, perform the following:

1. Connect the Lattice CertusPro-NX Versa Evaluation board to the PC/Laptop using USB Type-A UART cable.
2. Open Propel SDK tool.
3. Double-click on the terminal icon shown in [Figure 6.3](#).

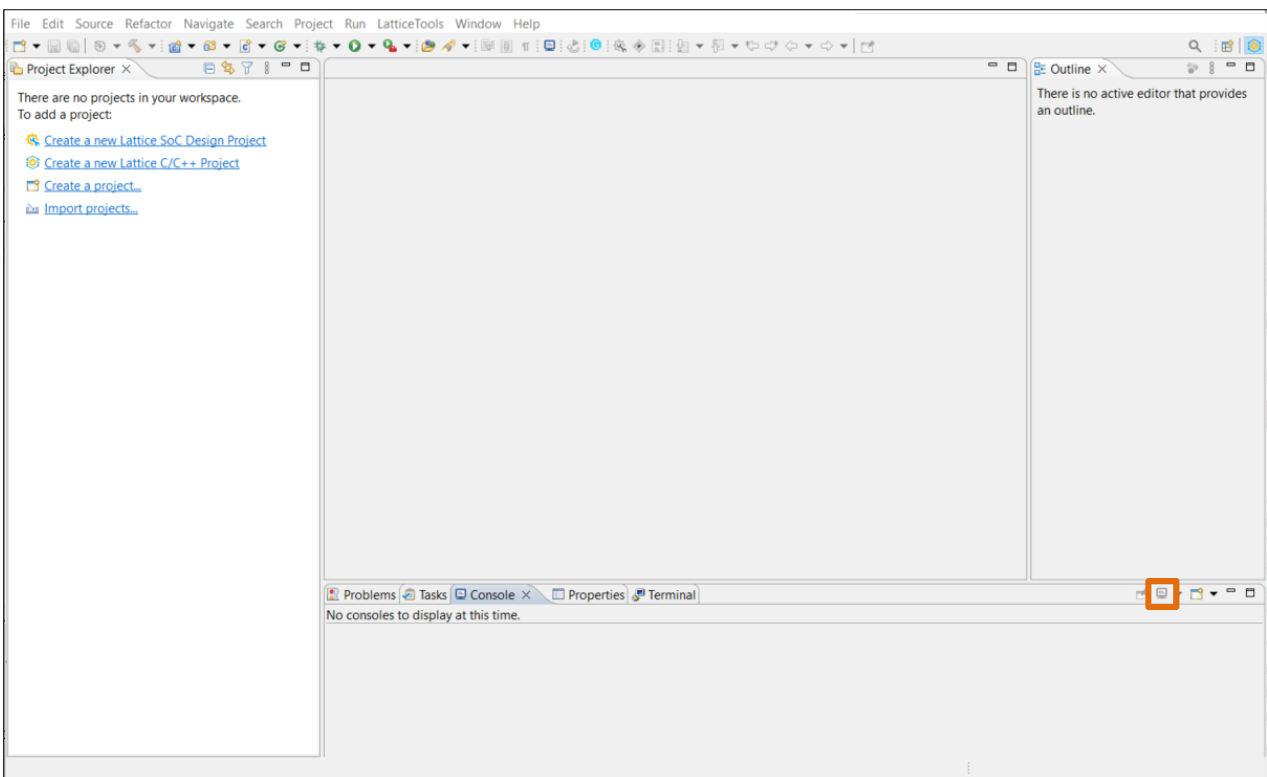


Figure 6.3. UART Terminal Icon on Propel SDK Window

4. Select **Serial Terminal** as shown in [Figure 6.4](#). In Serial port dropdown list, select the last COM in the list as shown in [Figure 6.5](#).

Note: This detail can also be found under the Ports (COM and LPT) section on 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.

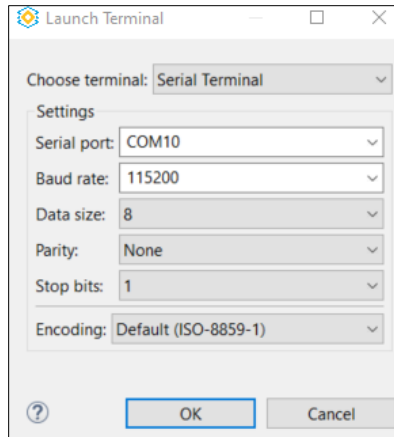


Figure 6.4. UART Launch Terminal Window

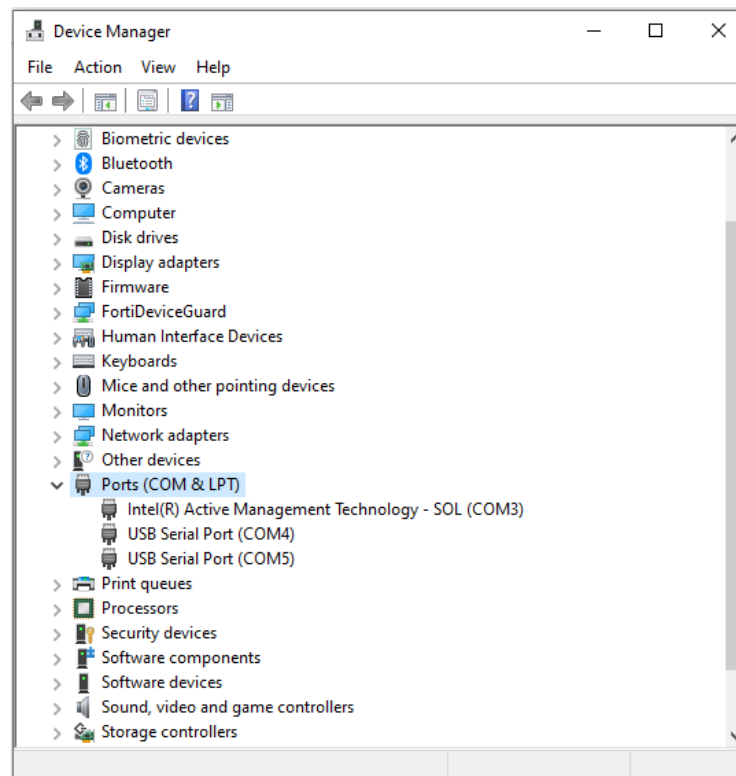


Figure 6.5. Device Manager Window on PC

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

6.4. 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 taken 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.

For more information, refer to [Appendix A. Changing the SPIM Settings in the NV Register](#).

6.5. Programming Standalone Golden or Primary GSRD Bitstream and Application Software

To program the standalone golden or primary GSRD bitstream and application software, perform the following:

1. Connect the CertusPro-NX Versa Evaluation Board to a PC using USB cable as per the hardware setup mentioned in [Setting Up the Hardware](#) section. Make sure jumpers at J32 and J33 are installed 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 the downloaded executables folder.

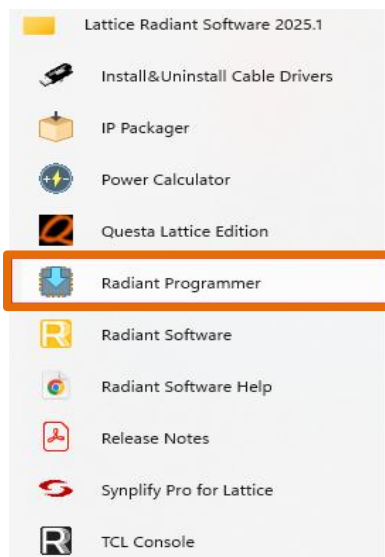


Figure 6.6. Launch Radiant Programmer from Windows Start

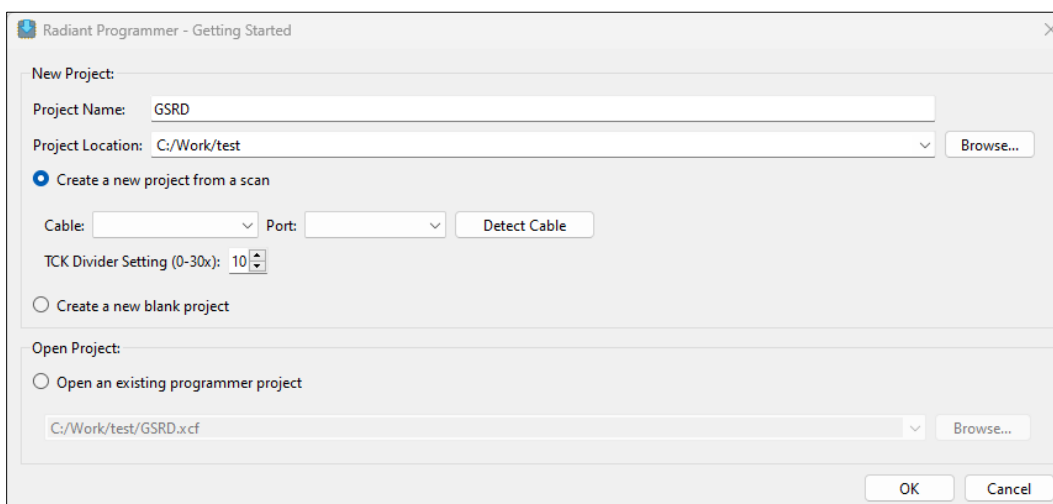


Figure 6.7. Radiant Programmer Start Window

4. Click **OK**.

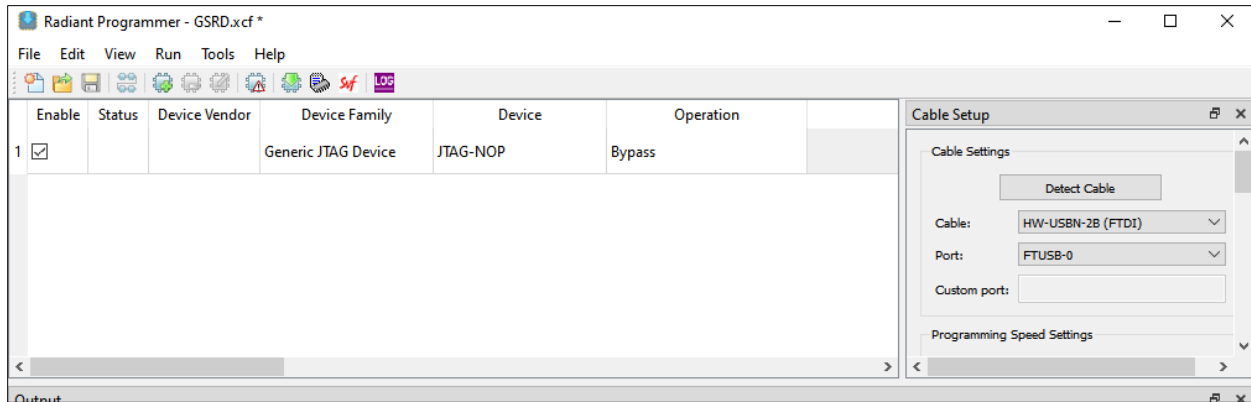


Figure 6.8. Radiant Programmer. xcf Window

- If the **Device Family** shows as **Generic JTAG Device**, click **Scan Device**, as shown in Figure 6.10, to update the **Device Family** information automatically.

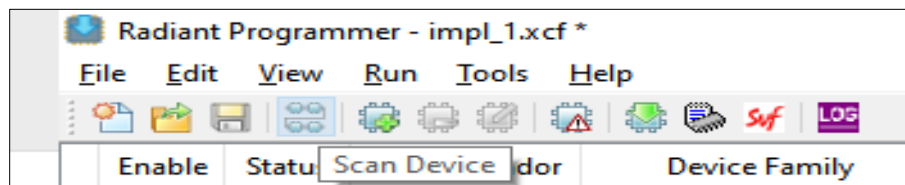


Figure 6.9. Scan Device Icon on Radiant Programmer

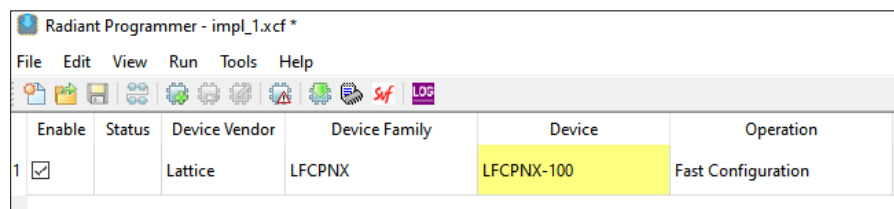


Figure 6.10. Select Device for Programming

- Click on the highlighted item under the **Device** field to un-highlight it as shown in Figure 6.11.

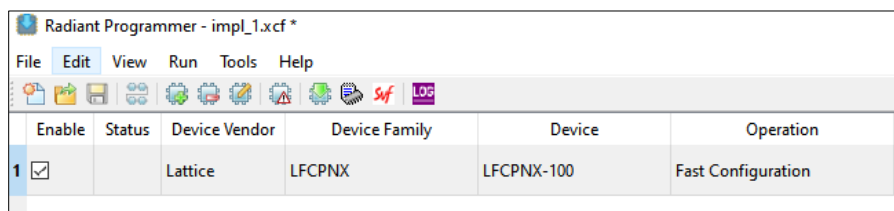


Figure 6.11. Device Selected for Programmer

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

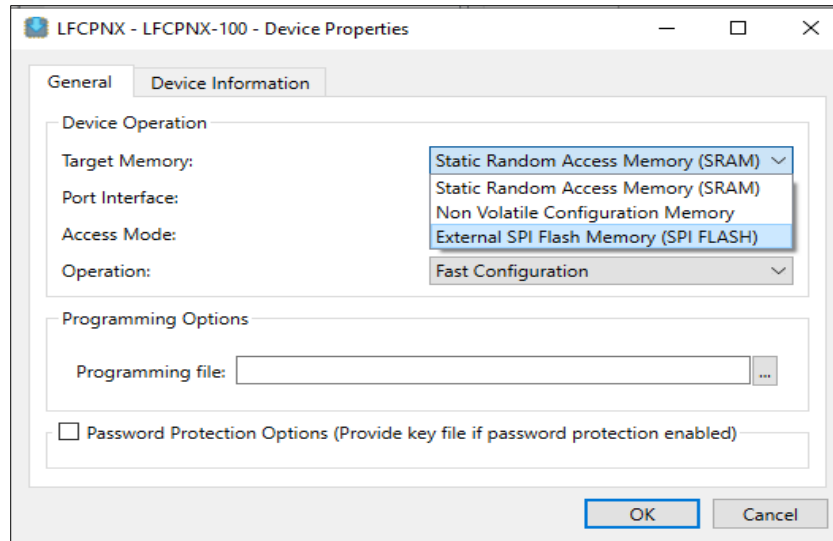


Figure 6.12. Select the Target Memory for Programming – SPI Flash

8. Before programming, you need to erase the entire SPI Flash Memory by applying the settings shown in [Figure 6.13](#) and [Figure 6.14](#).

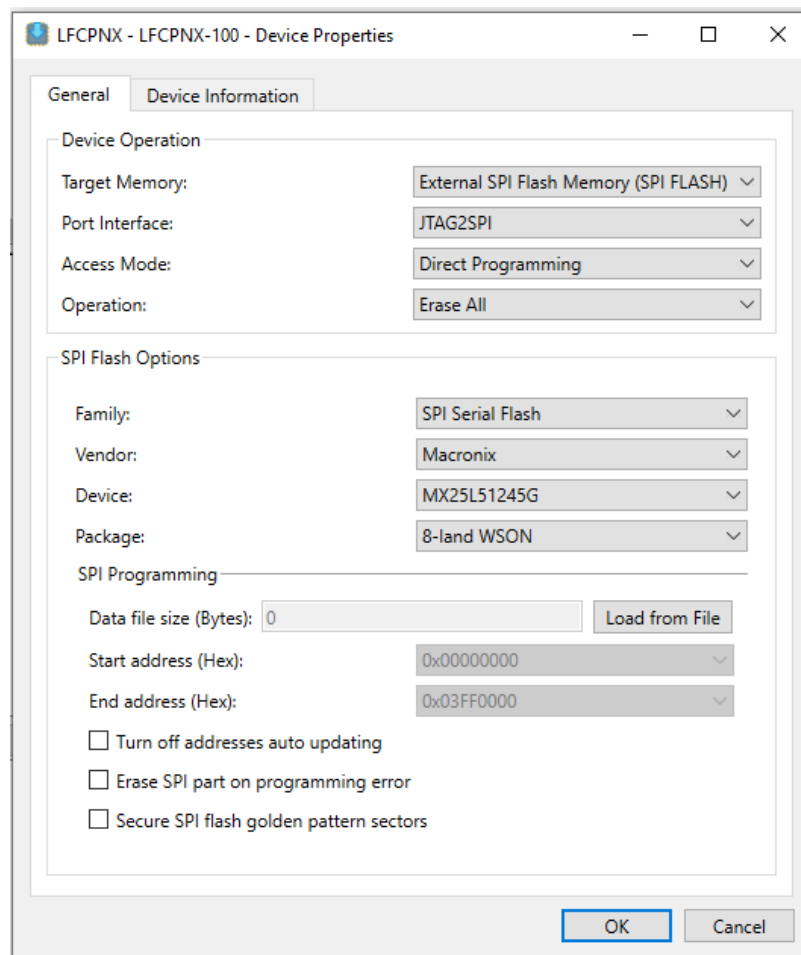


Figure 6.13. Device Properties to Erase the Macronix SPI Flash

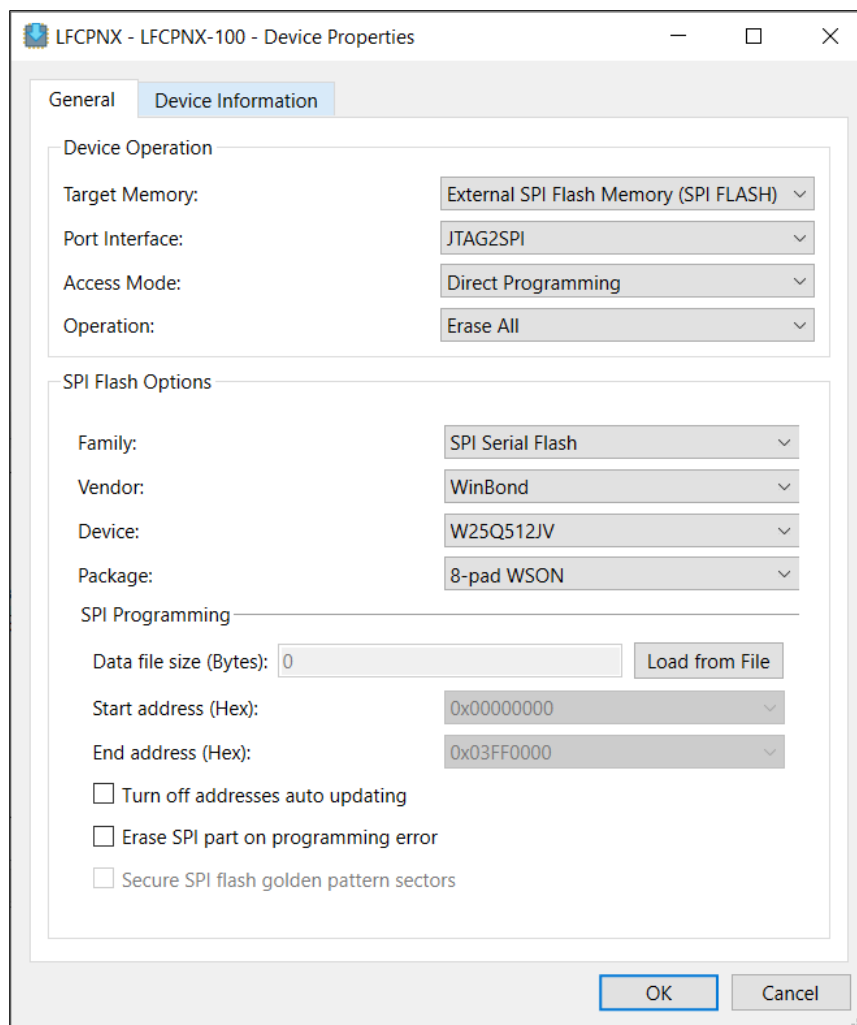


Figure 6.14. Device Properties to Erase the Winbond SPI Flash

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 be completed.

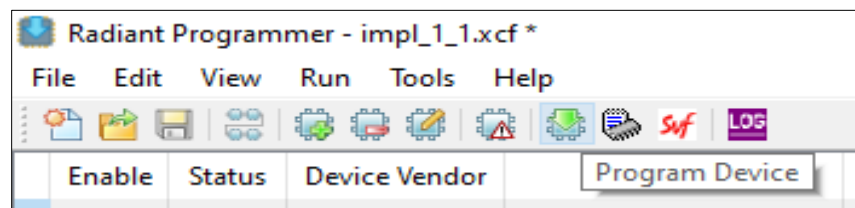


Figure 6.15. Program Button to Program the SPI Flash

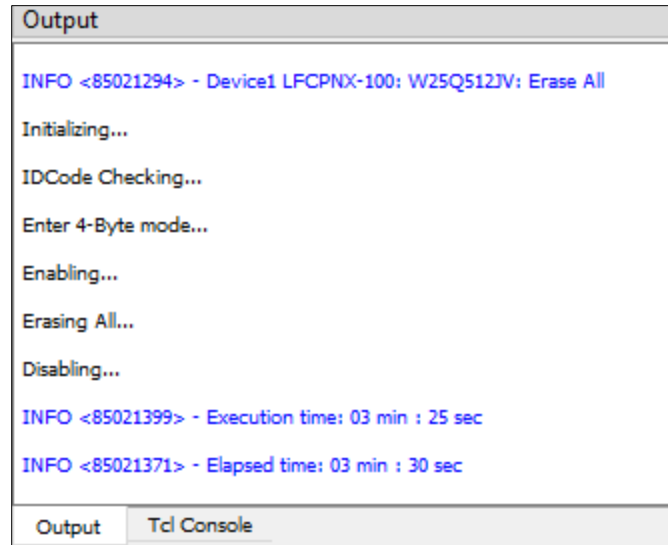


Figure 6.16. Output After Erase All

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

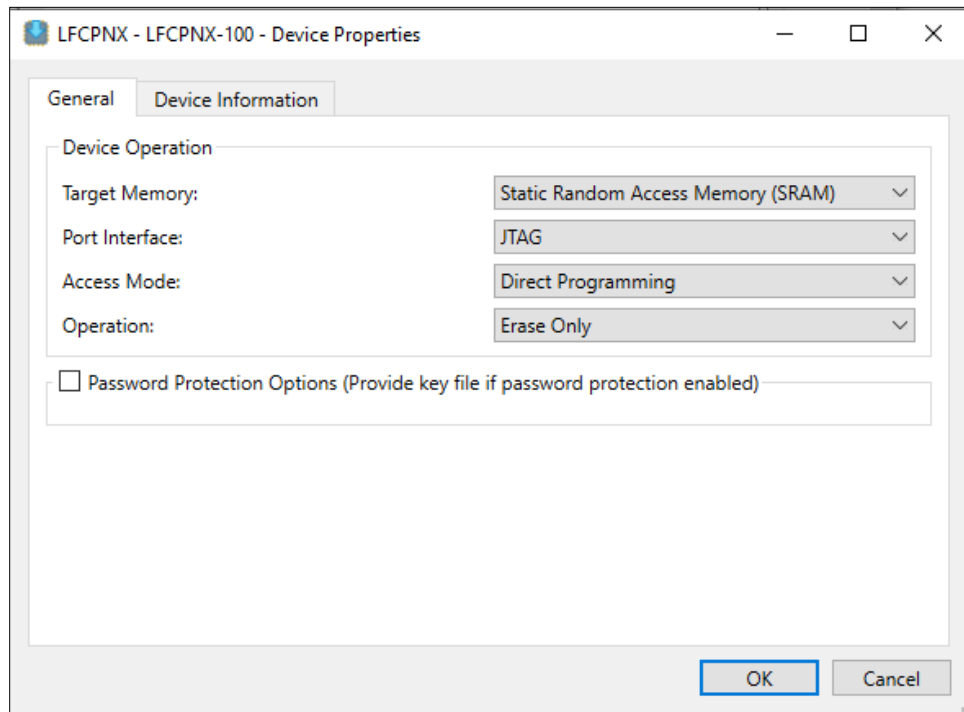


Figure 6.17. Erase Only Operation for SRAM Programming

12. To program the **c_golden_appcrc_wb.bin** or **c_golden_appcrc_mcrx.bin** file into the SPI Flash, follow the settings as shown in [Figure 6.18](#) or [Figure 6.19](#).

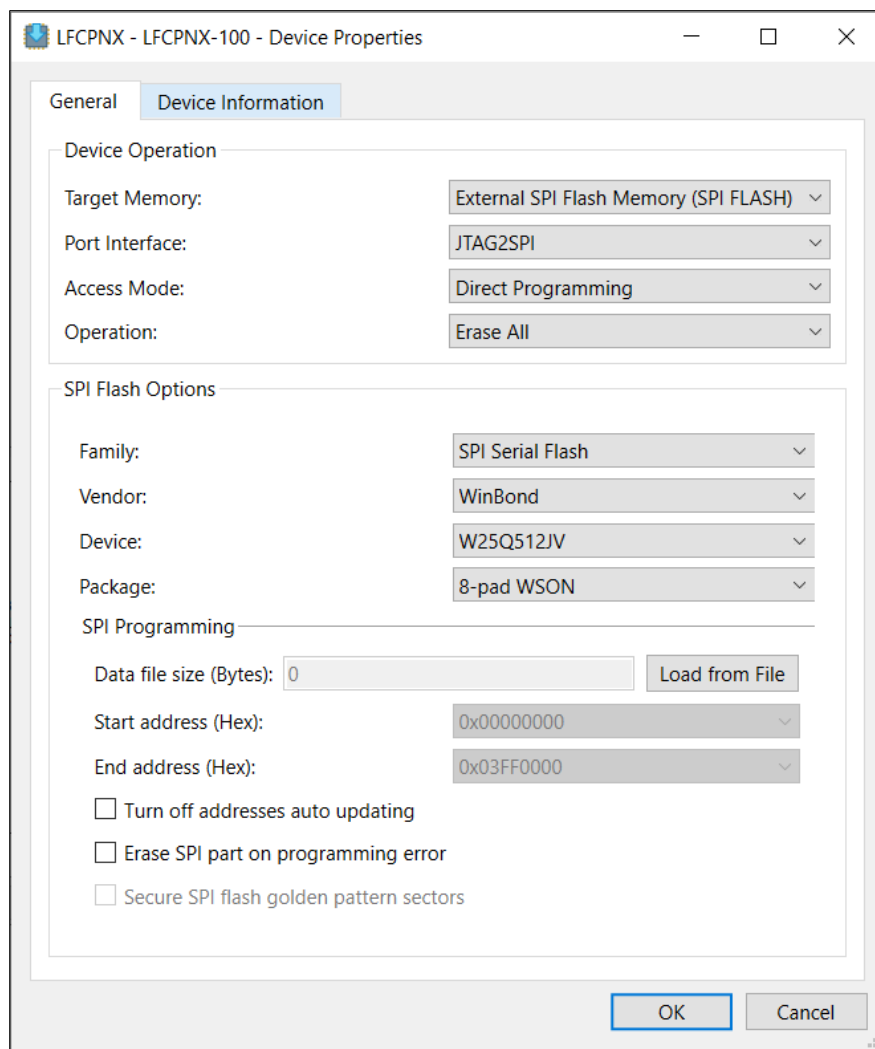


Figure 6.18. Device Properties to Program the Winbond SPI Flash

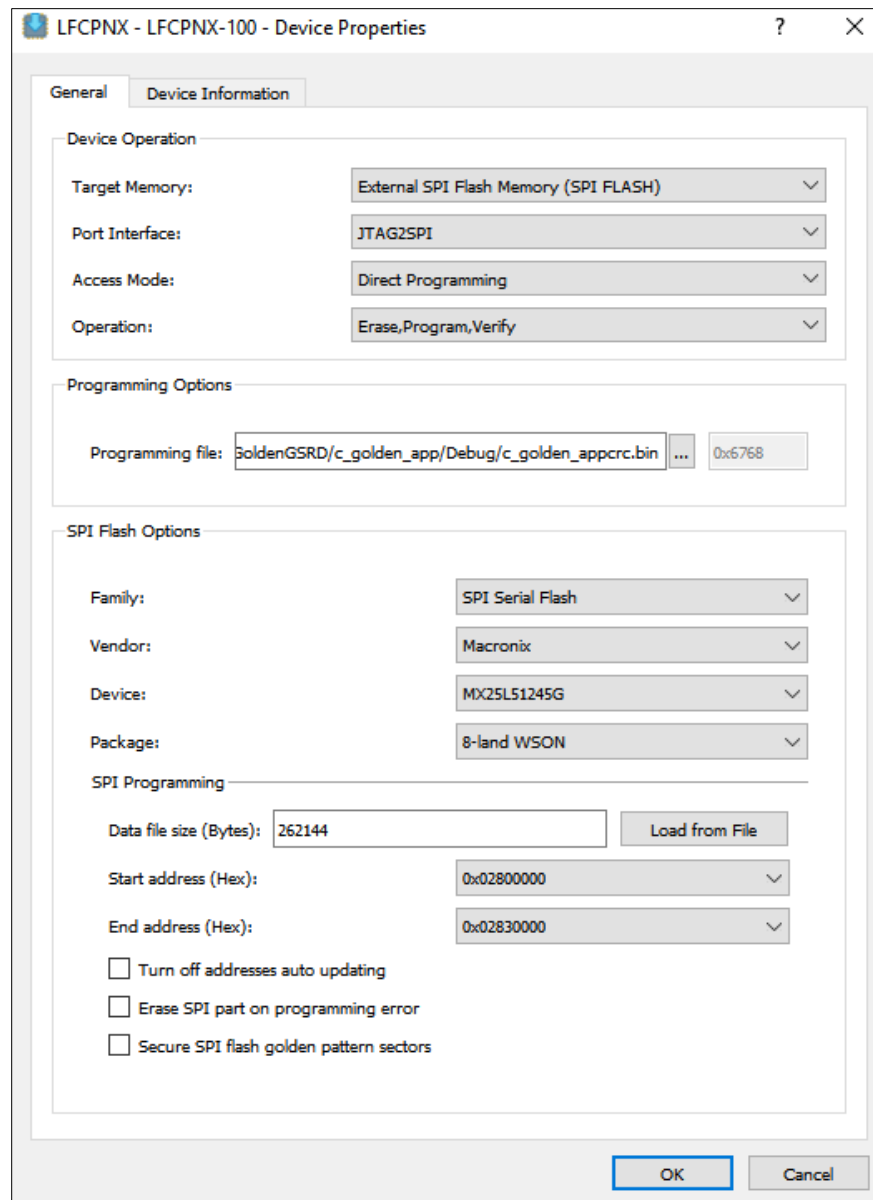


Figure 6.19. Device Properties to Program the Macronix SPI Flash

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

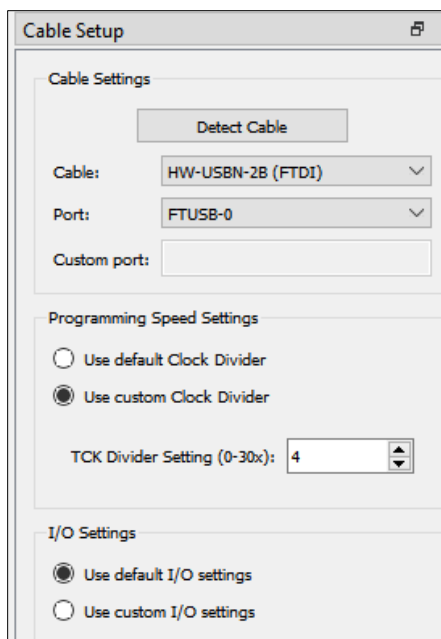


Figure 6.20. Cable Settings for Device Programming

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

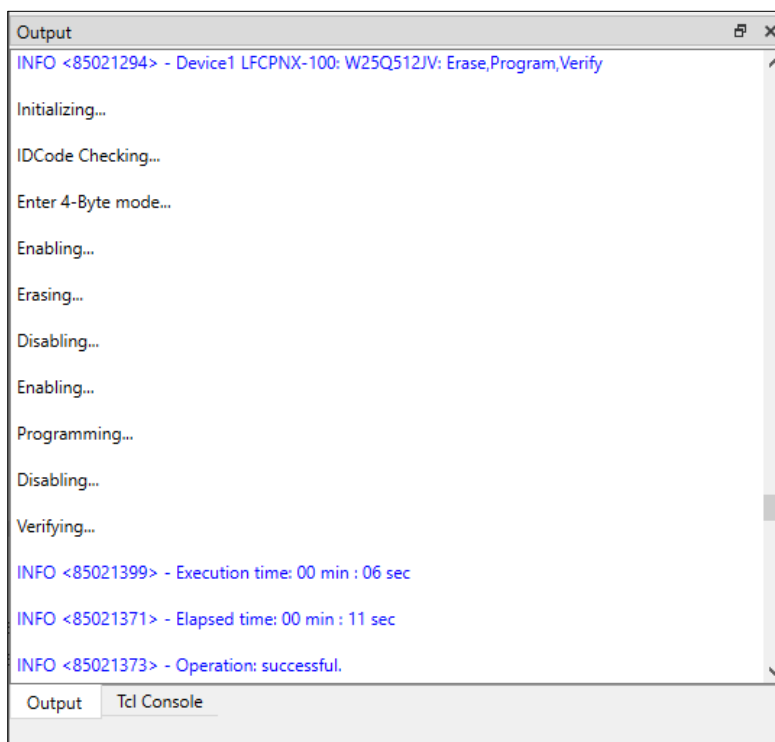


Figure 6.21. Radiant Programmer Console Output after Programming the SPI Flash

16. Power cycle the CertusPro-NX Versa Board.

17. To program the FPGA Golden GSRD bitstream, double-click on the **Operation** tab to update the selections as shown in [Figure 6.22](#). Make sure to provide the path to the .bit file location on your local machine where you have unzipped the executables.

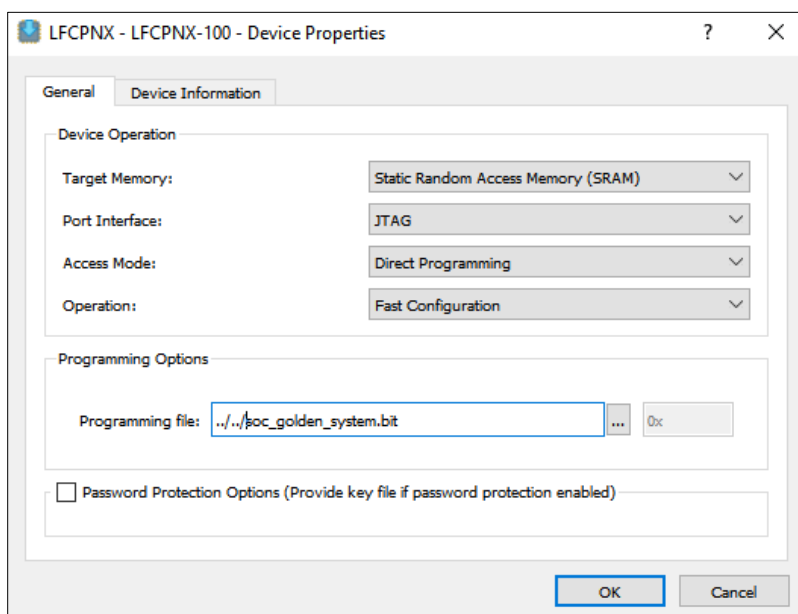


Figure 6.22. Device Properties to Program the FPGA Bitstream in SRAM

18. 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 6.23](#).

```
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 6.23. Radiant Programmer Console Output after Bitstream is Programmed

19. Set up the UART terminal as mentioned in [Setting Up the UART Terminal](#) section.
20. Press **SW3 Reset** button on the board as highlighted in [Figure 6.24](#).

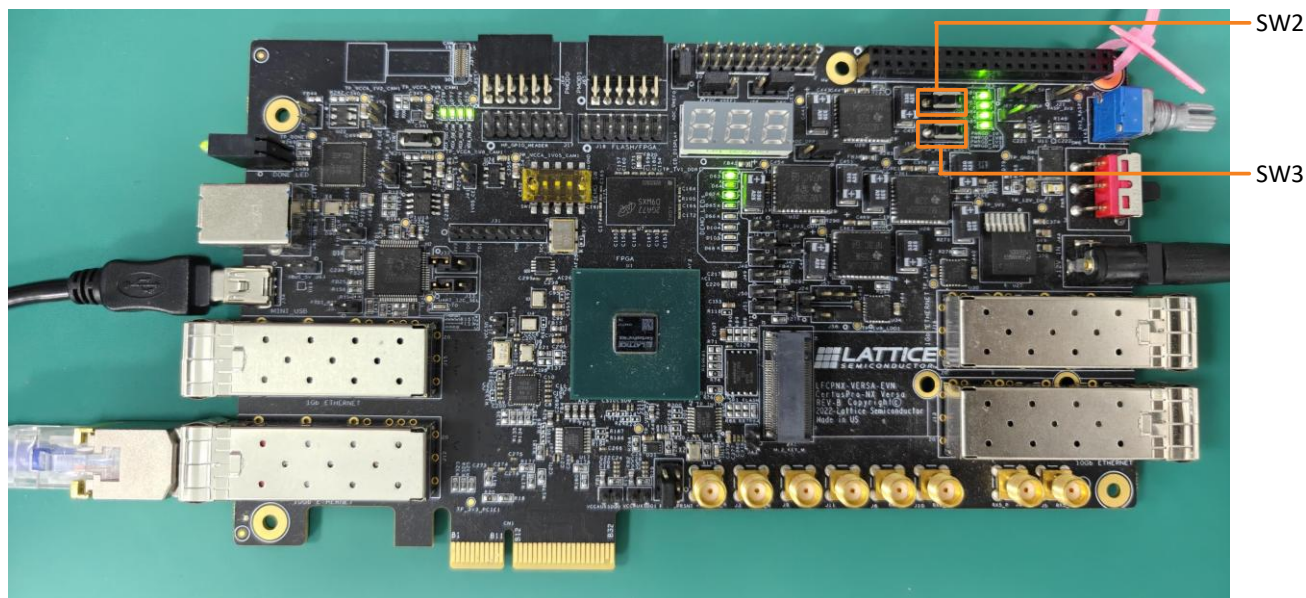


Figure 6.24. SW3 Reset Button and SW2 PROGRAMN Button

21. The results of running the Golden GSRD are as follows:

- The console should appear with messages as shown in [Figure 6.26](#) and [Figure 6.27](#).
- Hardware LED status as shown in [Figure 6.25](#):
 - D63 – Auto negotiation link status. LED = OFF: Link down. LED = ON: Link up.
Note: Reset the FPGA by pressing SW3 whenever the SFP module is removed and re-inserted.
 - D65 and D66 – SGMII link speed

Table 6.4. SGMII Link Speed Definition

D65	D66	Link Speed
OFF	OFF	10 Mbps
OFF	ON	100 Mbps
ON	OFF	1000 Mbps
ON	ON	Not defined

- D64 – LPDDR4 initialization status. LED = OFF: Initialization and training are in progress. LED = ON: Initialization and training are completed.
- D67 – LPDDR4 PLL Lock status. LED = OFF: PLL is unlocked. LED = ON: PLL is locked.
- D105 – LPDDR4 training status. LED = OFF: Training passes. LED = ON: Training fails.
- D68 – Multi-boot configuration status. LED = OFF: Configuration is done. LED = ON: Configuration is in progress.



Figure 6.26. Golden GSRD - Output on UART Terminal for Bootloader and FreeRTOS Start

```
*****
***      GSRD Golden FreeRTOS on RISC-V CPNX      ***
*****
Octal SPI Init Done.
[test_erase4k_prog_read_random] Passed !

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

INFO: Configure Auto-negotiation advertisement register

INFO: Executing PHY software reset to take in register settings changes

INFO: Retrying...If the SFP module takes longer to initialize and respond, consi
der increasing the delayCycle value.

INFO: Retrying...If the SFP module takes longer to initialize and respond, consi
der increasing the delayCycle value.

INFO: Speed and Duplex Resolved: 0xbc 0x0c

INFO: Speed: 0x02 Detected

INFO: Speed: 1000Mbps Detected

INFO: Full-duplex Detected
Configured to 1Gbps: gpio_read: 3f

PHY Initialization Complete.

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

INFO: PHY Model: 0x01410cc2

INFO: I2C Copper Status Register 1: 79 6d

The granularity of pmp is 0.
#####

pmp entry0: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0

pmp entry1: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0

pmp entry2: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0

pmp entry3: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0

#####

ethernet_enable_mac_interrupt
lwip_tcpip_init
Starting lwIP, local interface IP is 192.168.1.4

lwip_netif_init
lwip_udp_init
.....
```

Figure 6.27. Golden GSRD – Output on UART Terminal for FreeRTOS Running

22. Follow the same steps 5 to Step 20 to load the Primary GSRD Software and Bitstream. Refer to the [Executables](#) section for the folder and file names.
23. The results of running the Primary GSRD are as shown in [Figure 6.28](#) and [Figure 6.29](#).

```
*****
***      GSRD Primary Bootloader LFCPNX      ***
*****
Initializing OSPI Controller...Octal SPI init Done.

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

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

CRC matches successfully !!

Jumping to FreeRTOS application ...
```

Figure 6.28. Primary GSRD Bootloader– Output on UART Terminal

```
*****
***      GSRD Primary FreeRTOS on RISC-V CPNX      ***
*****
Octal SPI Init Done.
[test_erase4k_prog_read_random] Passed !

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
INFO: Configure Auto-negotiation advertisement register
INFO: Executing PHY software reset to take in register settings changes
INFO: Retrying...If the SFP module takes longer to initialize and respond, consi
der increasing the delayCycle value.
INFO: Retrying...If the SFP module takes longer to initialize and respond, consi
der increasing the delayCycle value.
INFO: Speed and Duplex Resolved: 0x02 0x4c
INFO: Speed: 0x02 Detected
INFO: Speed: 1000Mbps Detected
INFO: Full-duplex Detected
Configured to 1Gbps: gpio_read: 3f

PHY Initialization Complete.

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

INFO: PHY Model: 0x01410cc2
INFO: I2C Copper Status Register 1: 79 6d

The granularity of pmp is 0.
#####

pmp entry0: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0
pmp entry1: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0
pmp entry2: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0
pmp entry3: mode=0x00, perm=0x00, addr=0x00000000(*4)=0x00000000, locked=0
#####

ethernet_enable_mac_interrupt
lwip_tcpip_init
Starting lwIP, local interface IP is 192.168.1.4

lwip_netif_init
lwip_udp_init
.....
```

Figure 6.29. Primary GSRD FreeRTOS – Output on UART Terminal

6.6. Programming the Golden, Primary Software, and MCS file

This section demonstrates the multi-boot capability of GSRD by manually booting both Primary and Golden software with CRC checking through programming the MCS file.

To program the golden, primary software, and MCS file, perform the following:

1. Follow Steps 1 to 7 from [Programming Standalone Golden or Primary GSRD Bitstream and Application Software](#) section. Do not power-cycle the board.
2. Keep the folder/file of executables handy.
3. To program the **multiboot_system_wb.mcs** or **multiboot_system_mcrx.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 6.30](#).

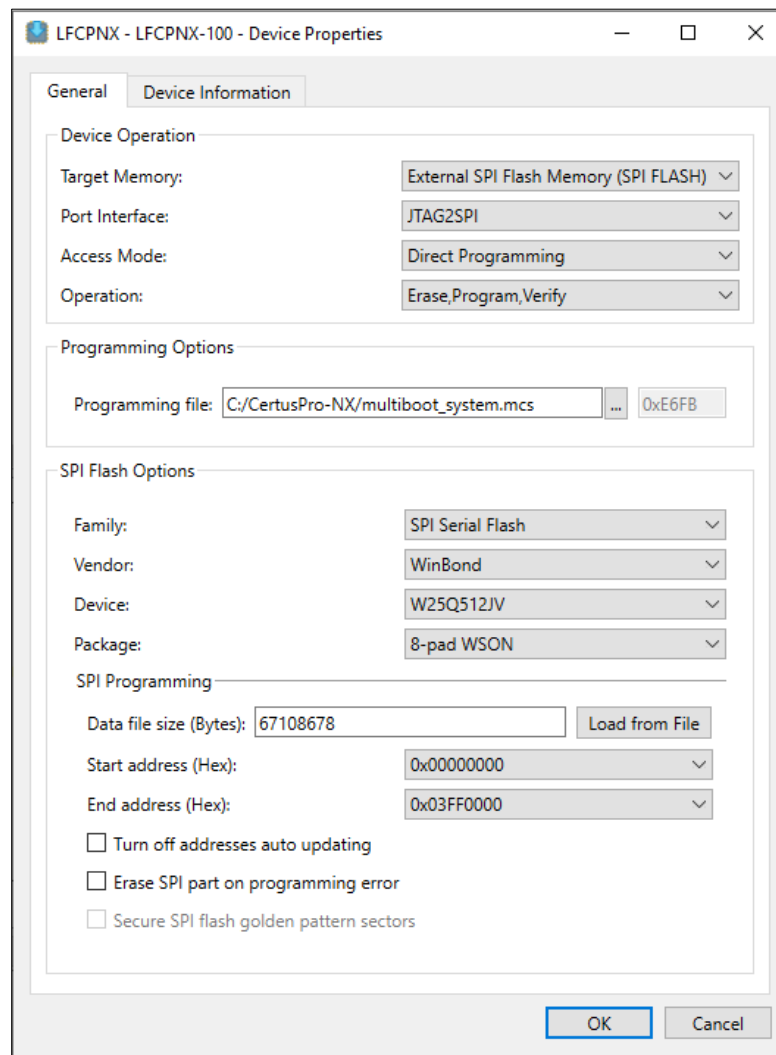


Figure 6.30. Device Properties Window to Setup MCS Programming File

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 **c_primary_apprc_wb.bin** or **c_primary_apprc_mcrx.bin** and **c_golden_apprc_wb.bin** or **c_golden_apprc_mcrx.bin** files mentioned in the [Programming Standalone Golden or Primary GSRD Bitstream and Application Software](#). Make sure to confirm that Starting Address (Hex) for both the binaries as per [Executables](#) section.
7. Once both binaries are programmed, turn off power switch **SW6**.
8. Turn on power switch **SW6**.
9. Set up 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 6.31](#).
 - It loads Primary GSRD software.
 - If you press the **SW3** button again, it loads the same Primary GSRD software.

```
*****
***      GSRD Primary Bootloader LFCPNX      ***
*****
Initializing OSPI Controller...Octal SPI init Done.

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

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

CRC matches successfully !!

Jumping to FreeRTOS application ...
```

Figure 6.31. UART Terminal Output after Power-Cycling Board with MCS and Binaries Programmed

12. For the manual multi-boot, press **SW2 (PROGRAMN)** button mentioned in the [Setting Up the Hardware](#) section on the board just above the **SW3 Reset** button.
13. The results on UART terminal are displayed as shown in [Figure 6.32](#). It switches to the Golden GSRD software and stays there unless the **SW2 PROGRAMN** button is pressed.

```
*****
***      GSRD Golden Bootloader LFCPNX      ***
*****
Initializing OSPI Controller...Octal SPI init Done.

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 2f85
Calculated Firmware CRC value: 2f85

CRC matches successfully !!

Jumping to FreeRTOS application ...

*****
***      GSRD Golden FreeRTOS on RISC-V CPNX      ***
*****
Octal SPI Init Done.
[test_erase4k_prog_read_random] Passed !

PHY Initialization:

INFO: PHY Model: 0x01410cc2

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

INFO: Configure SFP to SGMII mode
```

Figure 6.32. Switches to Golden GSRD upon SW2 PROGRAMN Button

7. Compiling and Running the Reference Design

This section describes the process of compiling the GSRD/GHRD Reference Design. You can always start with the Primary 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. For details, refer to the [Lattice Software Tools Requirements](#) section.

These sections show the typical design and compilation flow for GSRD/GHRD design. Hardware validation is performed after the software image is built at each stage.

Table 7.1. List of Actions and Expected Outputs

Actions	Outputs
Building GHRD SoC Project using Lattice Propel SDK and Builder.	sys_env.xml soc_primary_gsrdsbx soc_golden_gsrdsbx
Synthesizing the RTL files and generating the bitstream using Lattice Radiant Software	soc_primary_gsrdsbx_impl_1.bit soc_golden_gsrdsbx_impl_1.bit
Building the Hello World Program using Lattice Propel SDK and verifying the RISC-V and System Memory SoC design subsystem is built correctly Note: This action is optional	riscv_rtos_helloworld.elf riscv_rtos_helloworld.mem
Building Bootloader binary files using Lattice Propel SDK and verifying the primary or golden bootloader image and SoC design are built correctly	c_primary_bootloader.mem c_primary_bootloader.bin c_golden_bootloader.mem c_golden_bootloader.bin
Building FreeRTOS binary files using Lattice Propel SDK and verifying the primary or golden FreeRTOS image and SoC design are built correctly	c_primary_appcrs.bin c_golden_appcrs.bin
Generating the Multi-Boot MCS File and verifying the multi-boot functionality	multiboot_system.mcs

7.1. Building the GHRD SoC project Using Lattice Propel SDK and Propel Builder

To build the GHRD SoC design using Propel Template:

1. Create a folder for your project on your PC.
2. Launch Propel SDK application.

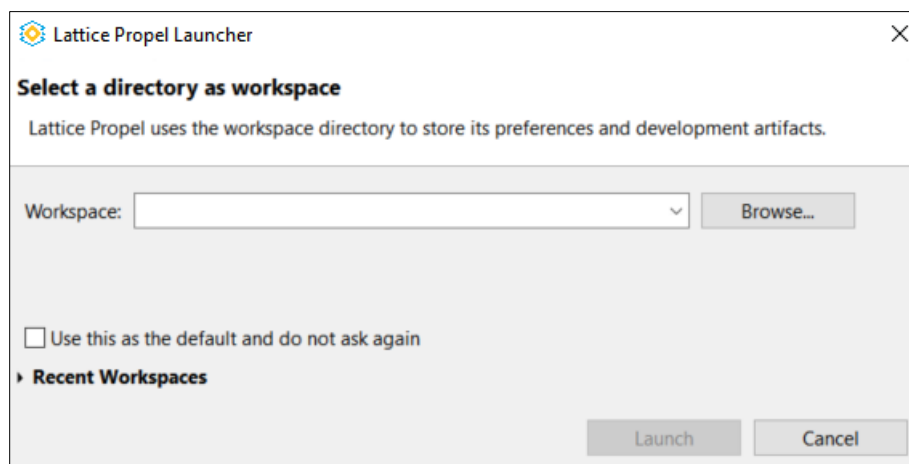


Figure 7.1. Propel SDK Launcher

3. To select the workspace, browse to the created folder by clicking on the **Browse** button as shown in Figure 7.1 and click on **Launch** to create 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.

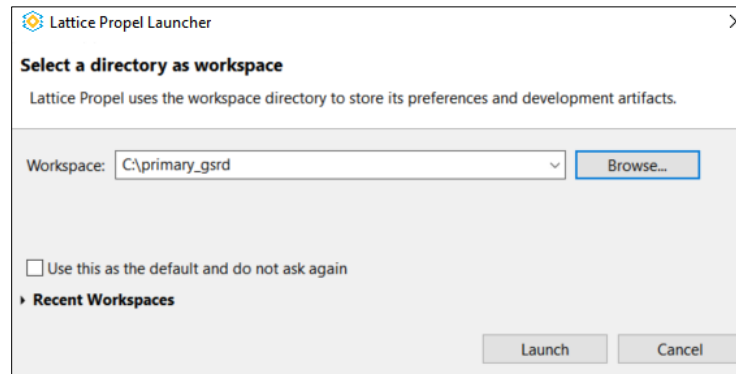


Figure 7.2. Provide Name for the Workspace Directory

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

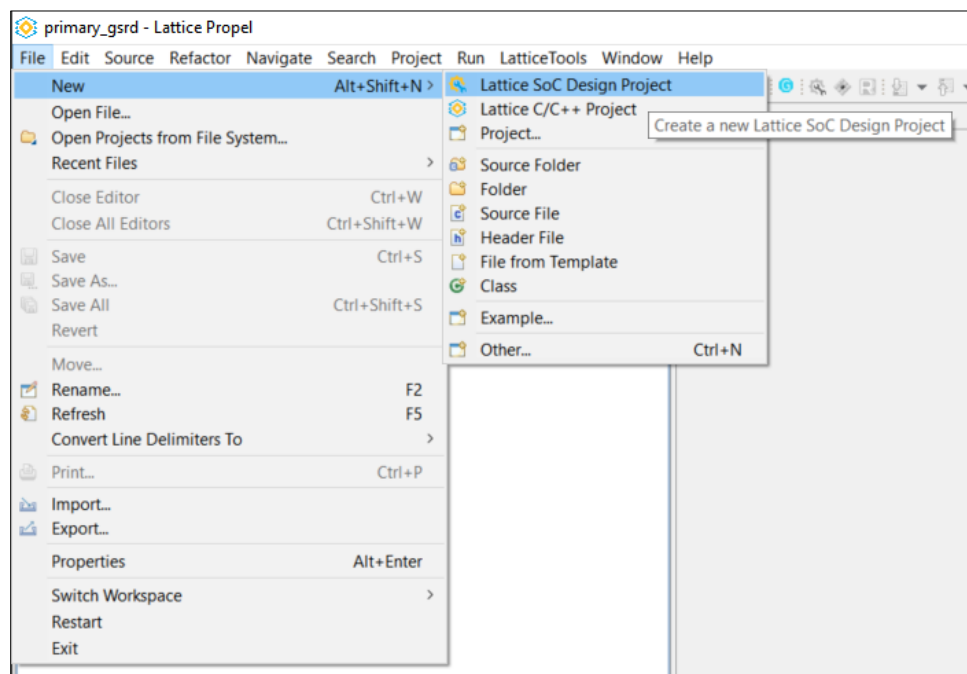


Figure 7.3. Creating Lattice SoC Design Project

5. On the SoC Project window:
 - a. Enter a name for your SoC project.
 - b. Enable **Board** option.
 - c. From **Board Select** section, select **CertusPro-NX Versa Evaluation Board**.
 - d. From **Processor** section, select **RISC-V RX**.
 - e. From **Template Design** section, select **GHRD SoC Project LFCPNX** as shown in Figure 7.4.

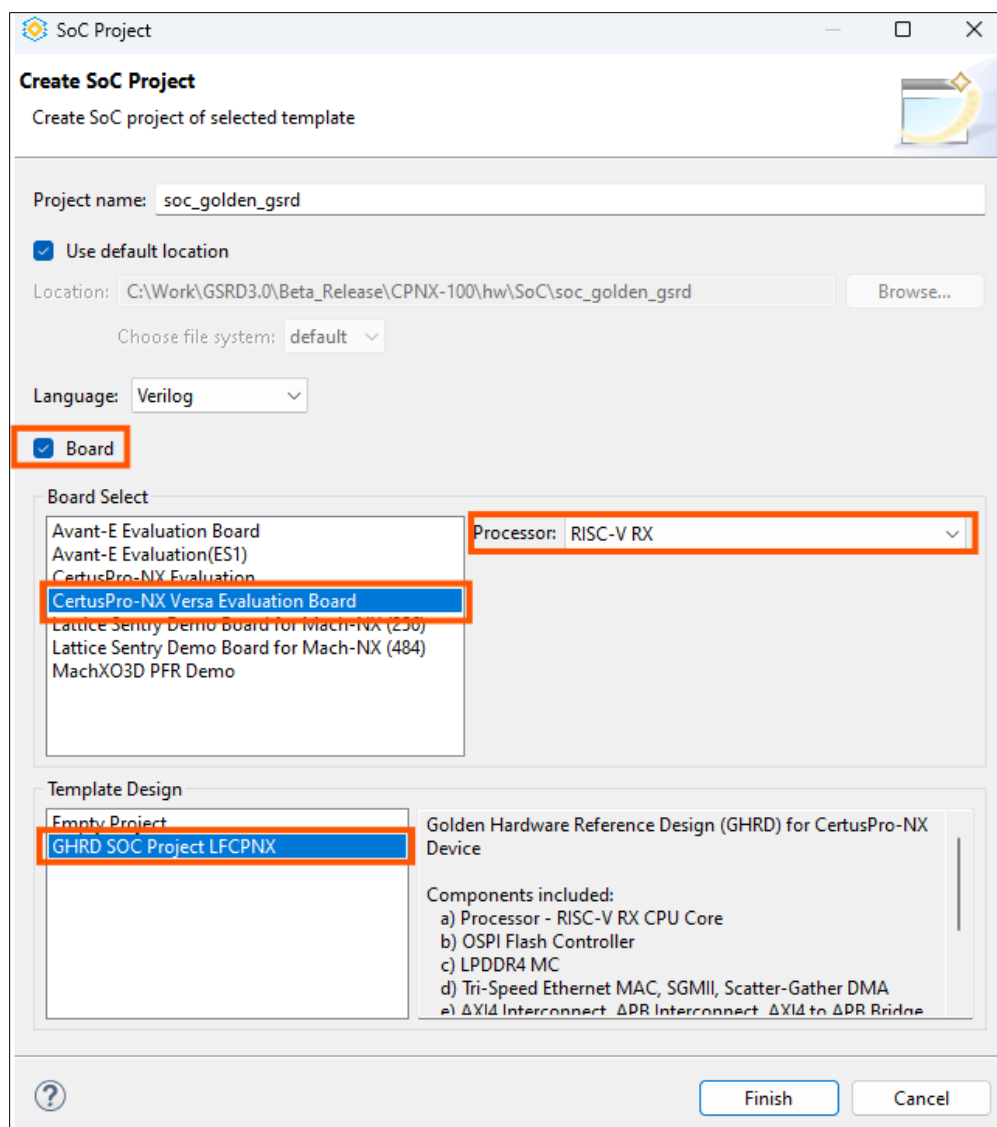


Figure 7.4. SoC Project Window

- Click **Finish** and it launches the Propel Builder Tool and loads the SoC design based on selected template as above. This generates the HDL files for IPs installed in the project.

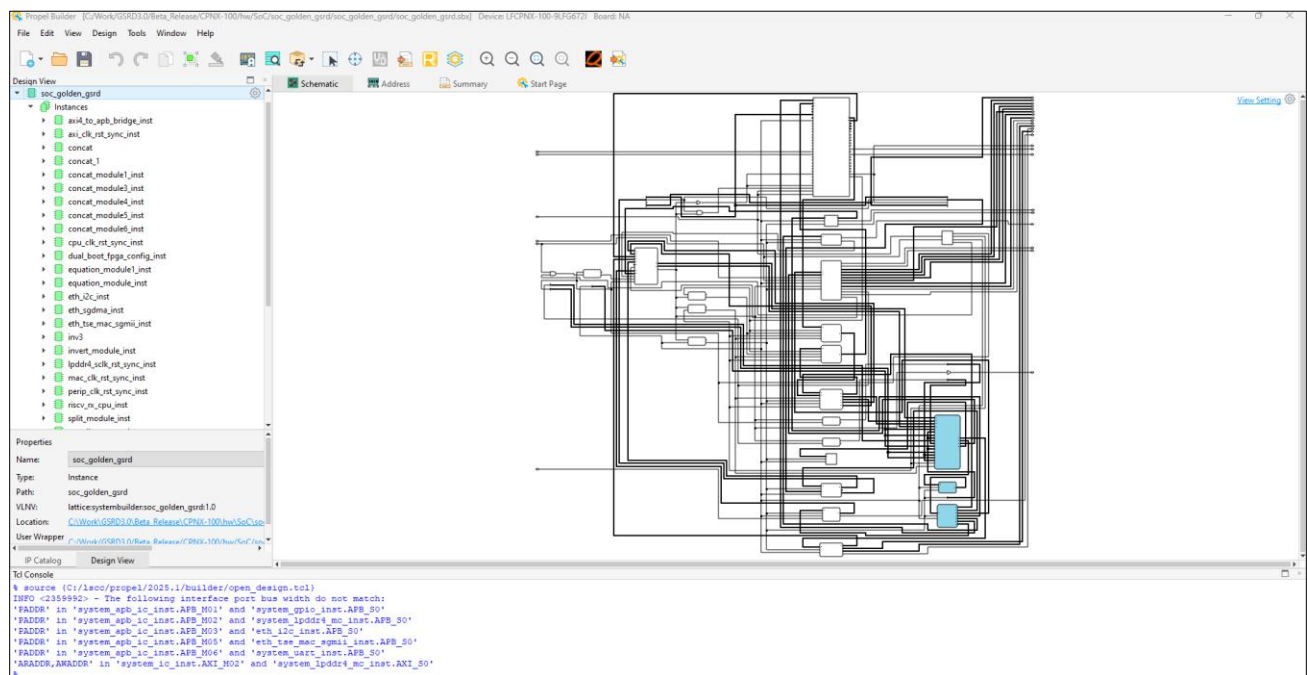


Figure 7.5. Launched SoC in Propel Builder

- Click on **Design > Validate Design** or click on the icon below as highlighted in screenshot.

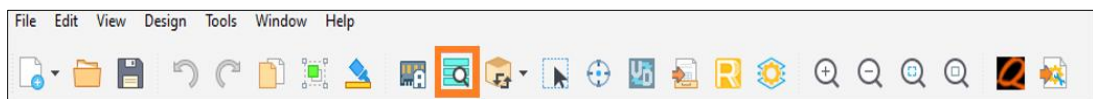


Figure 7.6. Validate Design in Propel Builder

The TCL console must be as follows. You must see no error in the **TCL Console** window. The following **INFO** and **WARNINGS** are expected.

```
Tcl Console
$ sbp_design drc
INFO <2359992> - Start: sbp_design drc.
INFO <2359992> - Dangling inputs are set to default value (ARUSER=0,AWUSER=0,WUSER=0) in system_ic_inst.AXI_500,system_ic_inst.AXI_501,system_ic_inst.AXI_502,system_ic_inst.AXI_503
INFO <2359992> - Dangling inputs are set to default value (BUSER=0,BUSER=0) in system_ic_inst.AXI_M00,system_ic_inst.AXI_M02,system_ic_inst.AXI_M03
INFO <2359992> - Dangling inputs are set to default value (DEST=0,IDI=0) in tsmem_rx_ifc_inst.AXI45_5
WARNING <2359991> - The bus interface port riscv_rx_cpu_inst.IRQ_S7/IRQ is not connected as a part of the interface connection.
INFO <2359137> - Finished successfully: sbp_design drc.
```

Figure 7.7. TCL Console Output after Validating Design

- Click on **Design > Generate** or click on the Icon below as highlighted in below screenshot.



Figure 7.8. Generate in Propel Builder

The TCL Console printout must be as follows. You must see no error in the **TCL Console** window. Note that the **WARNING** and **INFO** messages are expected.

```

Tcl Console
% abp_design generate
INFO <2359189> - Start: abp_design generate.
INFO <2359992> - Dangling inputs are set to default value (ARUSER=0,AMUSER=0,WUSER=0) in system_ic_inst.AXI_S00,system_ic_inst.AXI_S01,system_ic_inst.AXI_S02,system_ic_inst.AXI_S03
INFO <2359992> - Dangling inputs are set to default value (BUSER=0,ROUSER=0) in system_ic_inst.AXI_H00,system_ic_inst.AXI_H02,system_ic_inst.AXI_H03
INFO <2359992> - Dangling inputs are set to default value (TDEST=0,TID=0) in tceasc_rn_rtp_inst.AXI40_5
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_57/IRQ is not connected as a part of the interface connection.
INFO <2359190> - Finished successfully: abp_design generate.
% abp_design save
% abp_design ppe sge -i (C:\Work\GSRD3.0\Beta_Release\CPHX-100\hw\SoC\aooc_golden_gard\aooc_golden_gard.soc) -o (C:\Work\GSRD3.0\Beta_Release\CPHX-100\hw\SoC\aooc_golden_gard\aooc_golden_gard.soc) -no
WARNING <2359991> - (PGE SoCDesign) eth_tce_mac_0gm14_inst.HPC3_MM1_0 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst.IRQ_57 remains unconnected
INFO <2359992> - (PGE FileExistence) No available driver for fpga_config
%
    
```

Figure 7.9. TCL Console Printout after Generating Design

- Notice that after generating the design, the tool creates the `sys_env.xml` file in the project directory as shown in Figure 7.10. The `sys_env.xml` can be used to generate C/C++ project for Hello World (optional), bootloader and FreeRTOS application projects in the next sections.

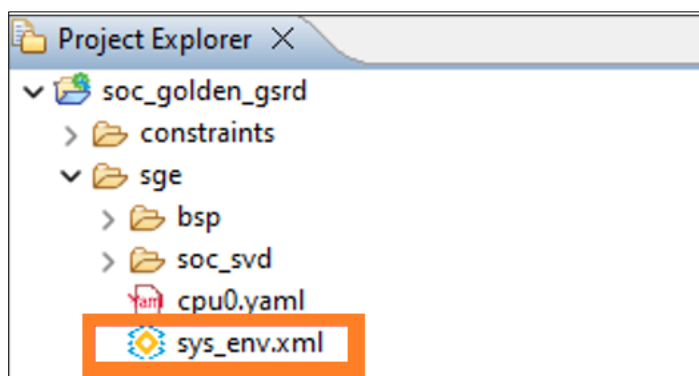


Figure 7.10. sys_env.xml File Created

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

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

- Click the Radiant icon from Propel Builder to launch the Radiant software as shown in Figure 7.11.

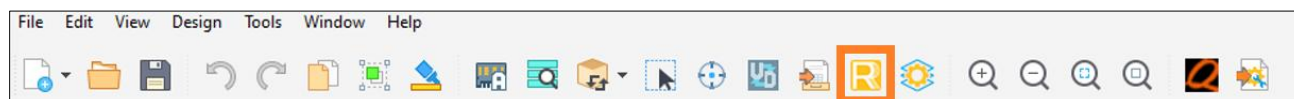


Figure 7.11. Run Radiant Icon

- The Radiant window of your project is launched as shown in Figure 7.12.

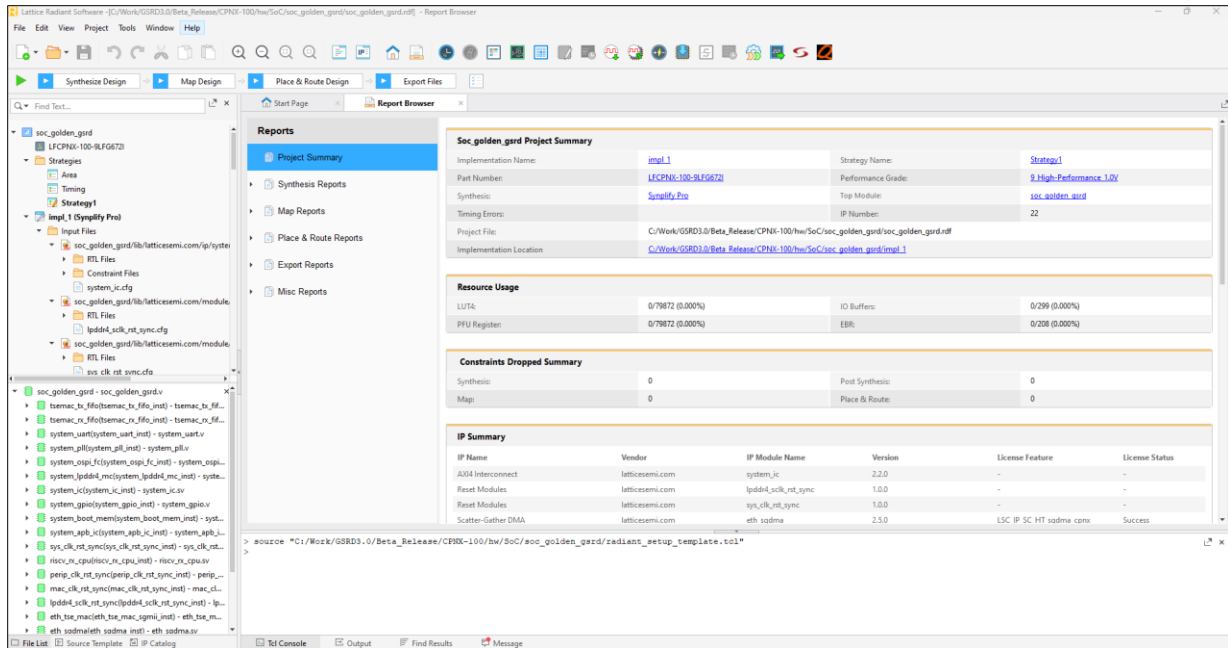


Figure 7.12. Lattice Radiant Window

- Ensure the *constraints.sdc* and *<Project_Name>.pdc* are already part of the project. They must respectively appear under pre-synthesis and post-synthesis constraints directory.

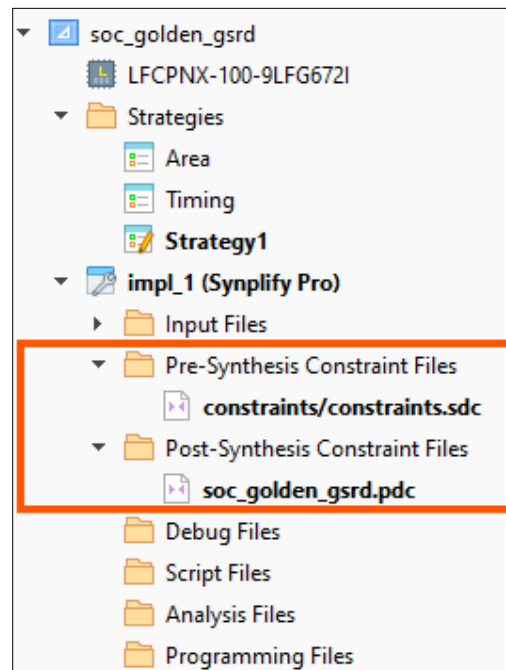


Figure 7.13. Lattice Radiant Window

- Synthesis** and **Place and Route** strategies are used for the GHRD.

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 and route details.

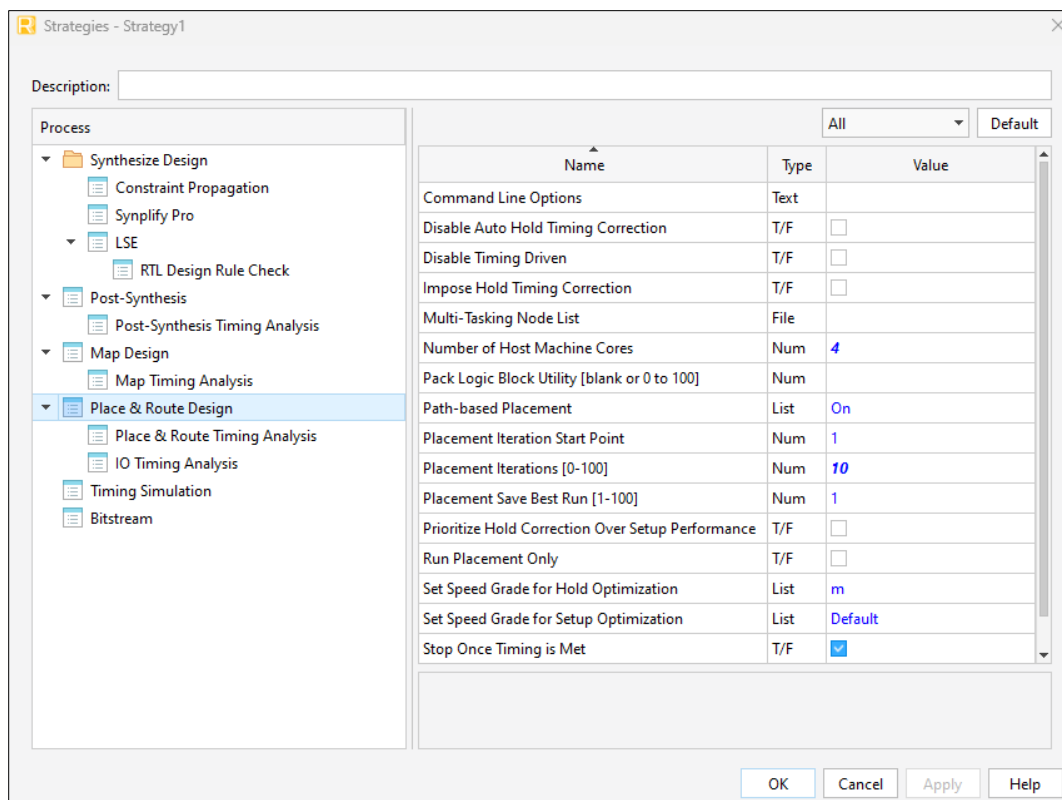
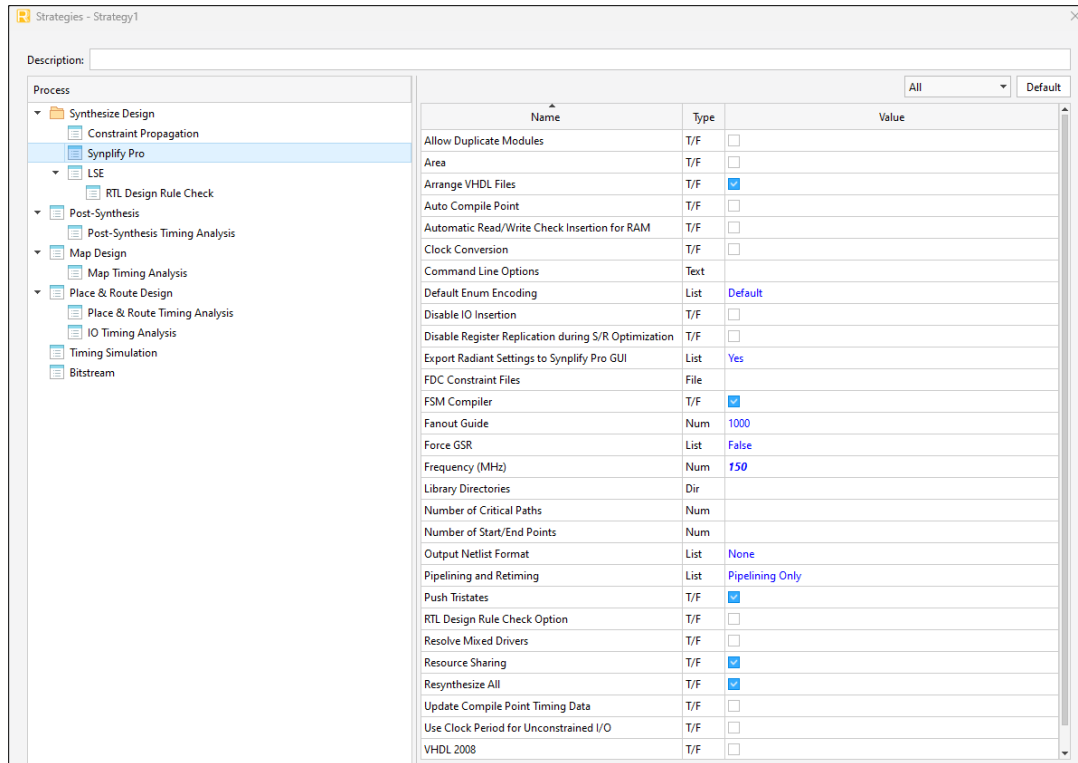


Figure 7.14. Strategies Used for GSRD Testing

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

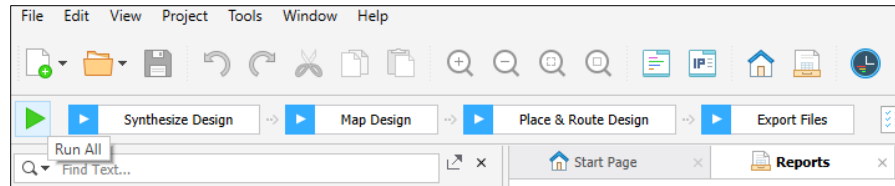


Figure 7.15. Generating the Bit File

6. The compilation flow takes some time to complete. The <project-name_impl_1>.bit file is generated/updated in the project path (<root_directory>/soc_golden_gsr/impl_1) after compilation is completed successfully as shown in Figure 7.16.



Figure 7.16. 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** is 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 Save Best Run** from 1 to 10. This will force tool to run each iteration and increase runtime.
 - c. Unchecked **Stop Once Timing is Met** option.
 - d. Click **OK**.
 - e. Click on **Export Files**.
7. These actions cause the **Place and Route Design** with different incremental start point values. The **Place and Route** reports show all ten placement results and select the best timing result.

7.3. Building the Hello World Program Using Lattice Propel SDK (Optional)

For quick start on SoC to boot up, you can create a basic Hello World program by using the template with the Lattice Propel SDK. Note that this action is optional.

To build the Hello World program using the Lattice Propel SDK, perform the following:

1. Create a folder for your project on your PC. For example, HelloWorld.
2. Launch the Propel SDK application.
3. Proceed to the Propel SDK window. Click on **File > New > Lattice C/C++ Project**.

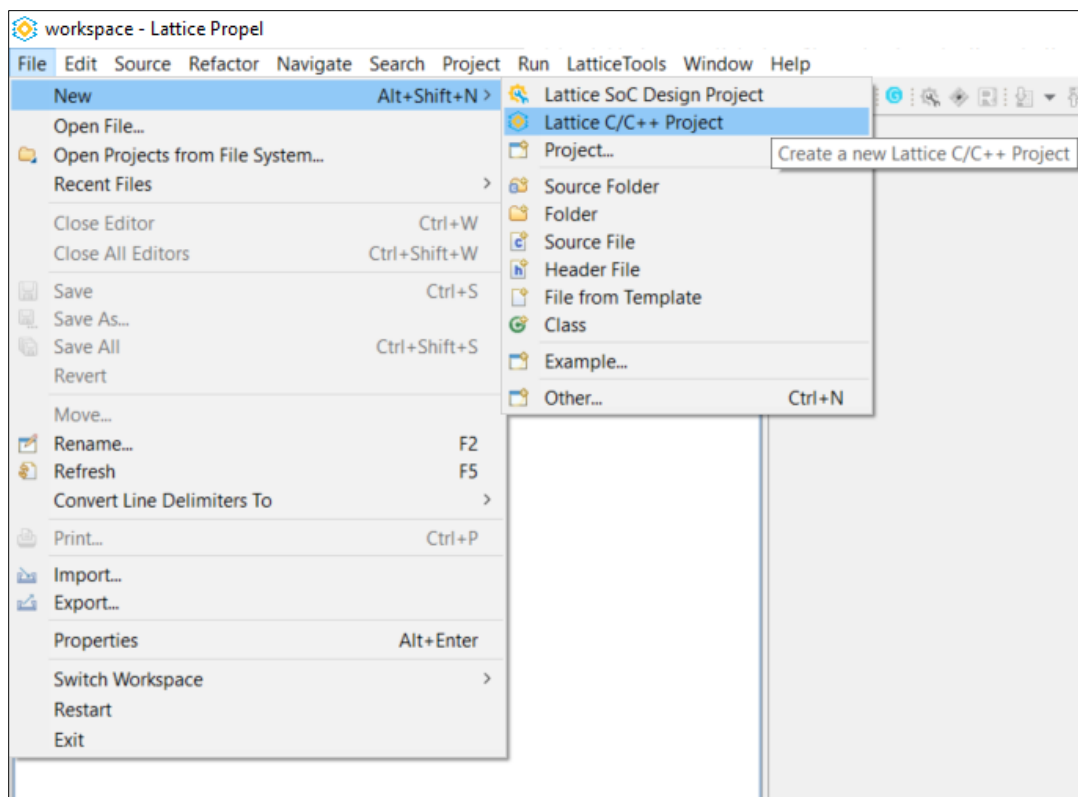


Figure 7.17. Creating Lattice C/C++ Project for Hello World

4. In C/C++ Project window, select the generated `sys_env.xml` file from previous GHRD SoC section. In the *Select Example Application* section, select the **Hello World Project** and provide a name for the project as shown in [Figure 7.18](#).

C/C++ Project

Load System and BSP

Load lattice system environment file and BSP package

Select system environment file and BSP package

System env: C:\HelloWorld\sys_env.xml Browse...

Select processor core to create C/C++ Project

Core selected: riscv_rx_cpu_inst

Project type: C

System information

Device Family	CPU Name	Instance Name
LFCPNX	riscv_rtos	riscv_rx_cpu_inst

Select Example Application

FreeRTOS-LTS minimal Project
FreeRTOS-LTS PMP-Blinky Project
Code Coverage Project
Timing Profiling Project
Hello World Project
Hardware Interrupt Project
Real Timer Project
RISC-V RX Demo Project

Example-HelloWorld-blink-uart
1.Led blink.
The corresponding SoC project should include gpio instance.
2.Uart print.
There are two ways to support uart print:
a) The corresponding SoC project should include uart instance or local uart.

Project name: riscv_rtos_helloworld

☒ Use default location

Location: C:\lsc\propel\2025.1\workspace\riscv_rtos_helloworld Browse...

Choose file system: default

☐ Build the project

☒ Create a debug launch configuration for OpenOCD

? < Back Next > Finish Cancel

Figure 7.18. Hello World C/C++ Selection

- Click **Next**. The Lattice Toolchain is shown in [Figure 7.19](#) and click **Finish**.

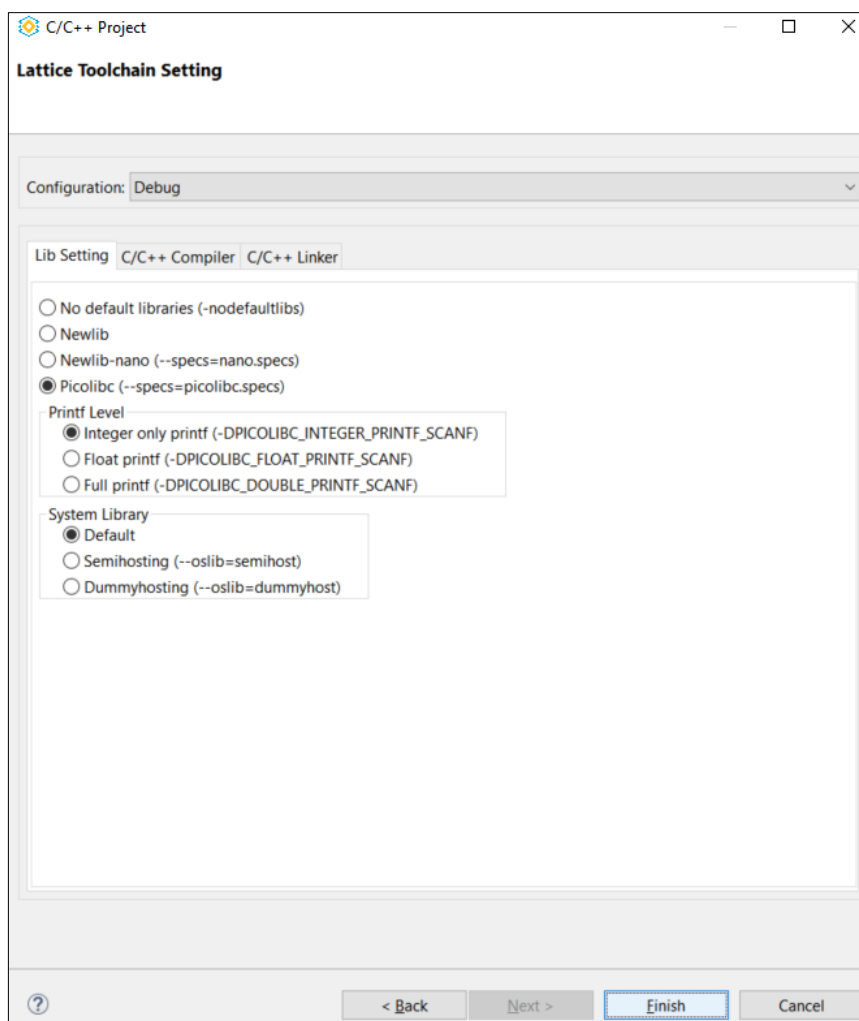


Figure 7.19. C/C++ Lattice Toolchain Setting

- This loads the Hello World project in the workspace as shown in Figure 7.20.

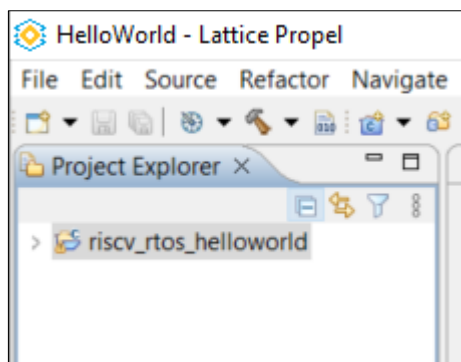


Figure 7.20. Hello World C Project Created

- To build the Hello World project, right-click on project and select **Build Project**.

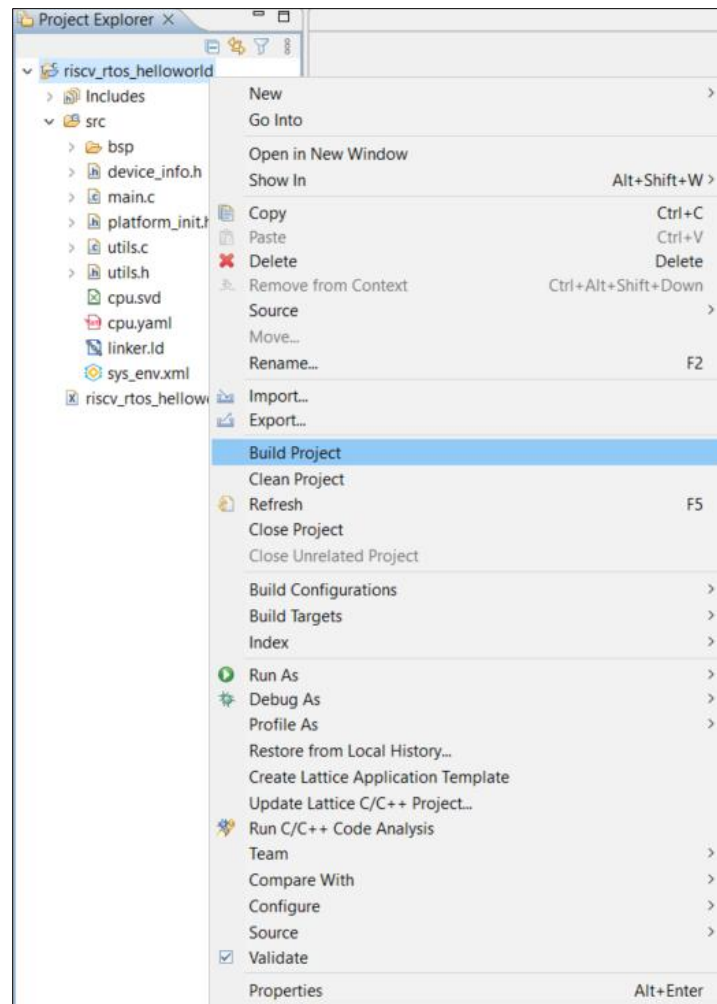


Figure 7.21. Build Hello World Project

8. The console output is displayed as shown in Figure 7.22.

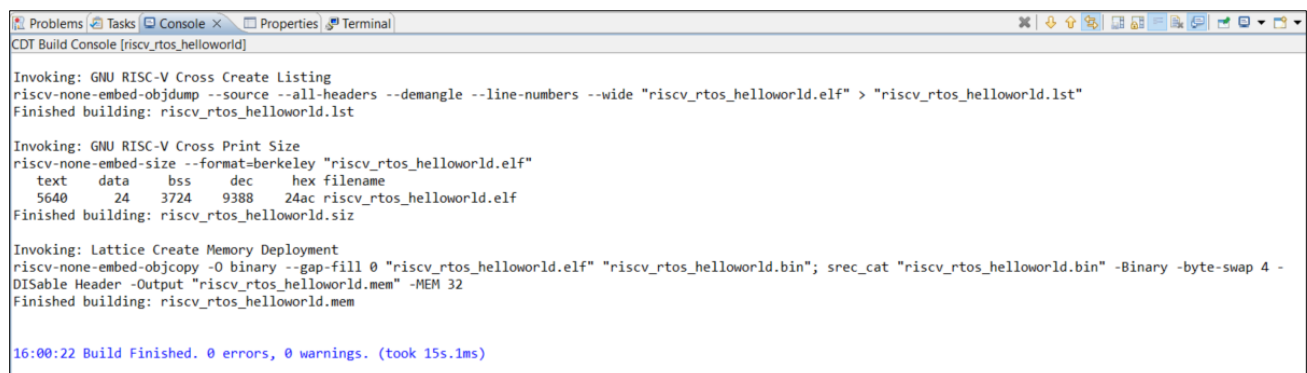


Figure 7.22. Hello World Project Build Console Output

9. The generated *.mem* file can be used to build into the GHRD for quick start-up check on the hardware. Refer to the [Using ECO Editor](#) section to integrate the *.mem* file into the bitstream.

10. Once the bitstream is built with the Hello World program, it can be programmed into the on-board SPI flash for testing. For programming into the flash, refer to the [Programming Standalone Golden or Primary GSRD Bitstream and Application Software](#) section.
11. Once programming into the flash is completed, power cycle the board to load the new image that contains the Hello World project.
12. You must see the *Hello World* message as shown in [Figure 7.23](#).

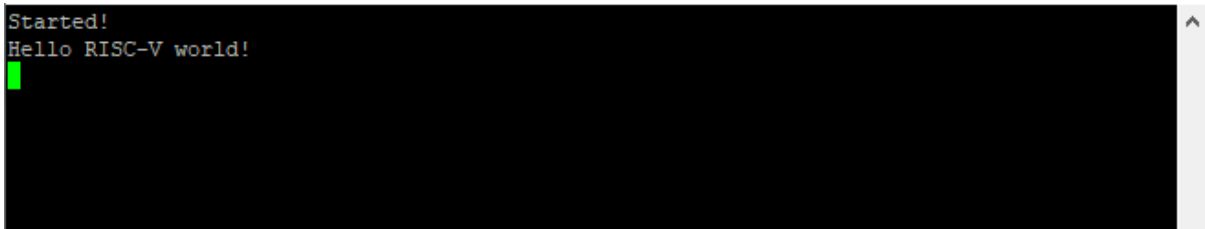


Figure 7.23. Hello World C Program

13. For details on how to create and run the Hello World C program and SoC project, refer to the *Creating a Hello World C Project* section in the [Lattice 2025.1 Propel SDK User Guide \(FPGA-UG-02234\)](#).

7.4. Building the Bare-metal Bootloader Using the Lattice Propel SDK (Primary and Golden)

Once the Hello World program is working, you can proceed to build the Bootloader binary files:

1. Create a folder for your project on your PC. Launch the Propel SDK application.
2. Proceed to the Propel SDK window. Click on **File > New > Lattice C/C++ Project**.

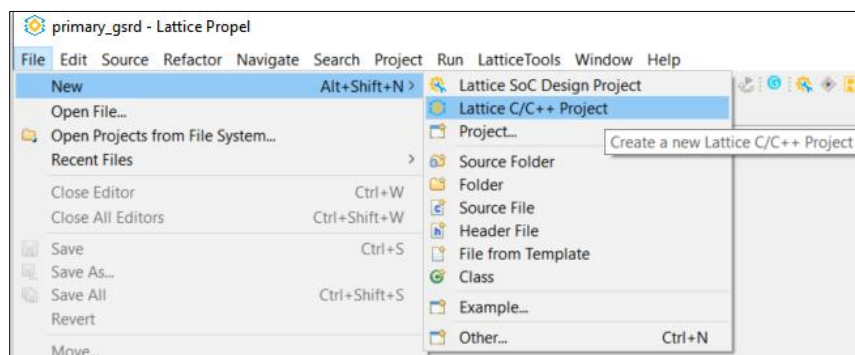


Figure 7.24. Creating Lattice C/C++ Project for Bootloader

3. 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 7.25](#).

C/C++ Project

Load System and BSP
Load lattice system environment file and BSP package

Select system environment file and BSP package
System env: C:\CertusPro-NX\soc_primary_gsrdsge\sys_env.xml Browse...

Select processor core to create C/C++ Project
Core selected: riscv_rx_cpu_inst
Project type: C

System information

Device Family	CPU Name	Instance Name
LFCPNX	riscv_rtos	riscv_rx_cpu_inst

Select Example Application

Example Application	Description
FreeRTOS-LTS minimal Project	
FreeRTOS-LTS PMP-Blinky Project	
Code Coverage Project	
Timing Profiling Project	
GSRD Avant Bootloader	
GSRD CPNX Bootloader	1. Initializes GPIO, UART 2. Configures LPDDR4 MC 3. Configures QSPI Flash Controller 4. Configures Ethernet PHY over I2C interface 5. Copies FreeRTOS Application Software from external SPI Flash to LPDDR4 MC 6. Checks Application Software's integrity by checking the appended CRC
GSRD Avant FreeRTOS	
GSRD CPNX FreeRTOS	
Hello World Project	

Project name: c_primary_bootloader

☒ Use default location
Location: C:\lsc\propel\2025.1\workspace\c_primary_bootloader Browse...
Choose file system: default

☐ Build the project
☒ Create a debug launch configuration for OpenOCD

? < Back **Next >** Finish Cancel

Figure 7.25. Bootloader C/C++ Selection

- Click **Next**. The Lattice Toolchain is shown in [Figure 7.26](#) and click **Finish**.

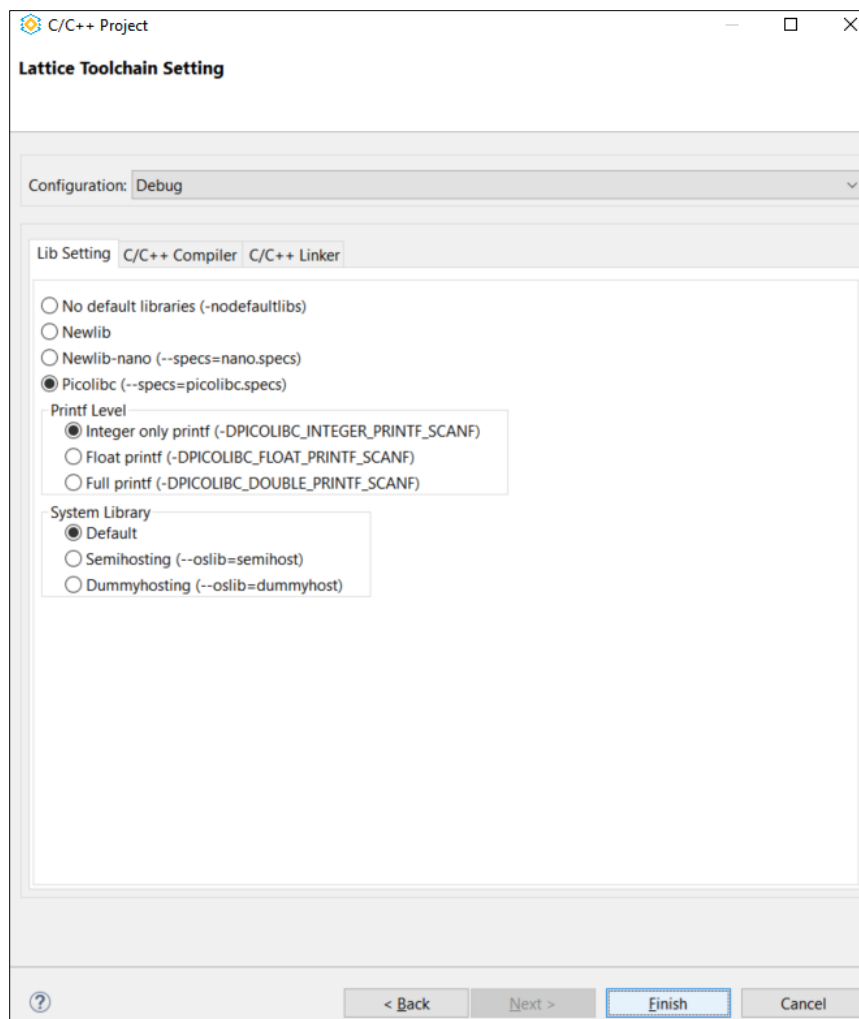


Figure 7.26. C/C++ Lattice Toolchain Setting

- This loads the bootloader project in the workspace as shown in Figure 7.27.

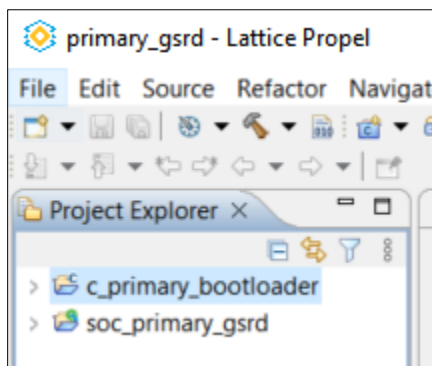


Figure 7.27. Bootloader C Project Created

- To build for the Primary Bootloader, you need to define `_PRIMARY_BUILD_` in the start of the `main.c` file. This enables the SPI address to point to Primary FreeRTOS application software image in the SPI flash. Refer to [Appendix C. Using Different SPI Flash Manufacturer in GSRD Bare-metal Bootloader](#) to modify the Octal SPI Controller driver according to the flash device used.

```
54 /* Set for GOLDEN OR PRIMARY build */
55 #define _PRIMARY_BUILD_
56
```

Figure 7.28. Primary Build Define – Set _PRIMARY_BUILD_ for Primary Build in main.c

7. To create the bootloader binary file (**c_primary_bootloader.mem**), right-click on **c_primary_bootloader** and select **Build Project**.

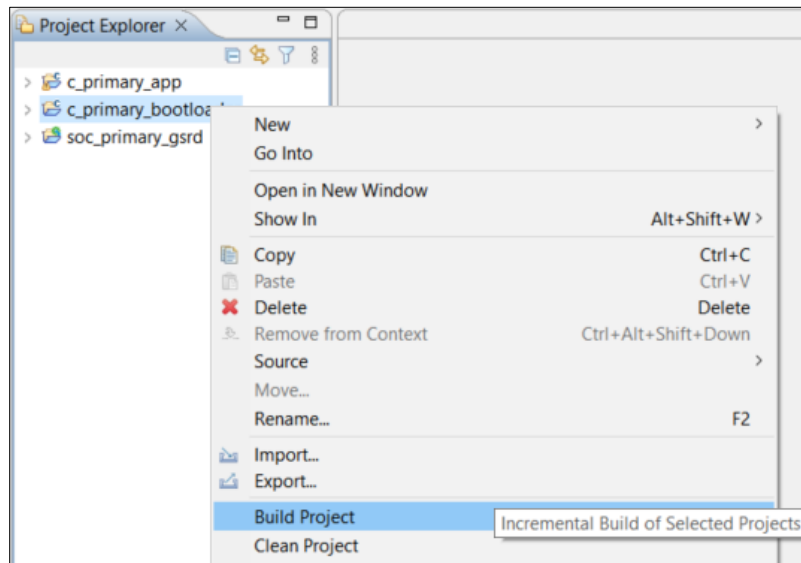


Figure 7.29. Build Bootloader Project

8. The console output is displayed as shown in [Figure 7.30](#).

```
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 7.30. Bootloader Build Project Console Output

9. This creates a debug folder as shown in [Figure 7.31](#). The binary created is named **c_primary_bootloader.mem**. This is as a part of System Memory in Propel Builder SoC design. This can be done through ECO editor or directly from the Propel Builder System Memory user interface.

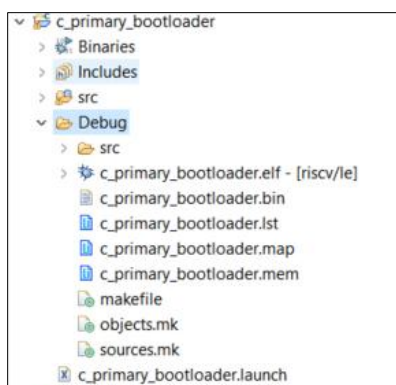


Figure 7.31. Bootloader Binary Created

10. Update the system memory content by using ECO editor. Refer to the [Using ECO Editor](#) section for more information.
11. Once the primary bootloader is run, user can see the following output. The output message is expected, and users can proceed to build FreeRTOS application in the next section.

```

*****
***          GSRD Primary Bootloader LFCPNX          ***
*****
Initializing OSPI Controller...Octal SPI init Done.

Memory Controller Initialization:

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

Memory Controller Initialization Complete.

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: a6e1
Calculated Firmware CRC value: ffff

ERROR: CRC Mis-matched!!!

```

Figure 7.32. Bootloader boots up without FreeRTOS application

12. In the case to create the golden bootloader `c_golden_bootloader.mem`, set the `#define _GOLDEN_BUILD_` in the `main.c` file to build for golden bootloader binary file.

```

54 /* Set for GOLDEN OR PRIMARY build */
55 #define _GOLDEN_BUILD_
56

```

Figure 7.33. Golden Build Define – Set `_GOLDEN_BUILD_` for Golden Build in `main.c`

13. Repeat steps 1 to 11 to create the golden bootloader project.

7.5. Building the FreeRTOS Application Software using Lattice Propel SDK (Primary and Golden)

To build the FreeRTOS application software using Lattice Propel SDK, perform the following:

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

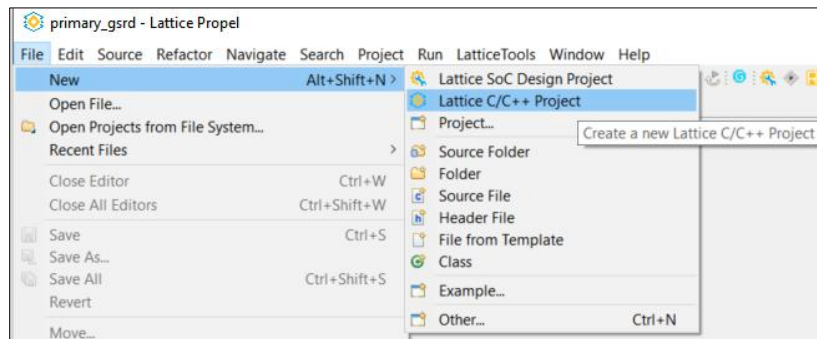


Figure 7.34. Creating C/C++ Project for FreeRTOS

2. In the *Select Example Application*, select the **GSRD CPNX FreeRTOS** project. Provide the project name as shown in Figure 7.35.

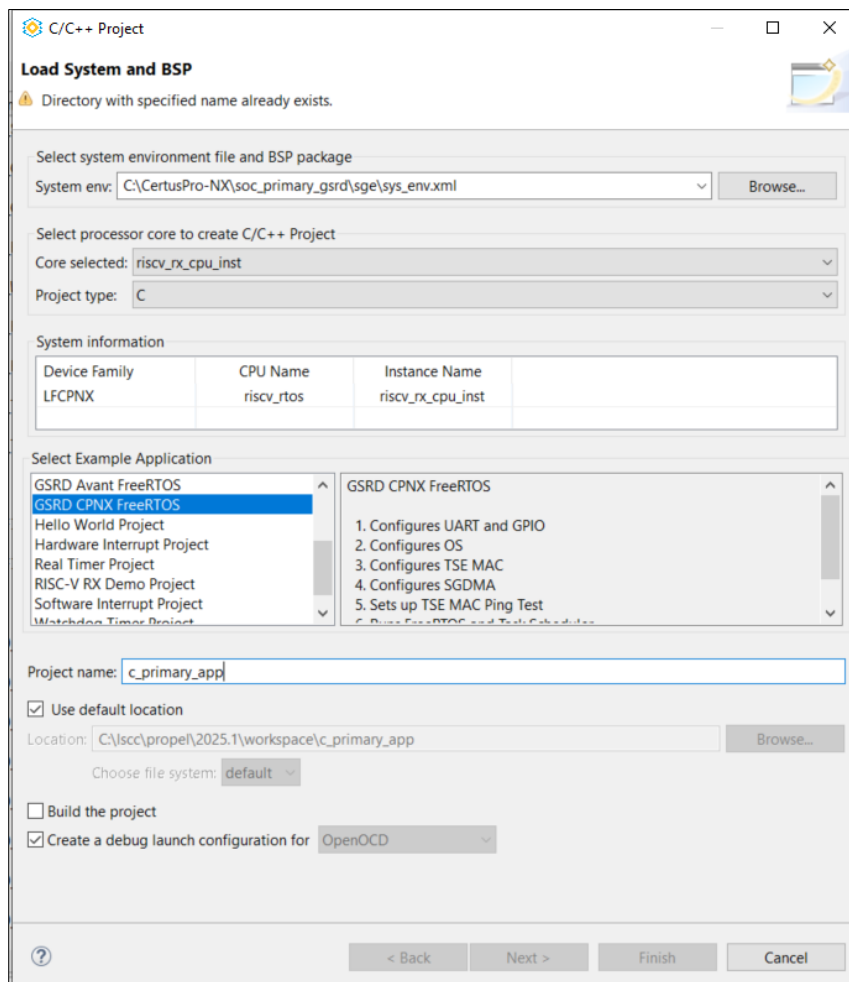


Figure 7.35. FreeRTOS C/C++ Selection

3. 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` and `crc_add_release.txt` as shown in Figure 7.36.

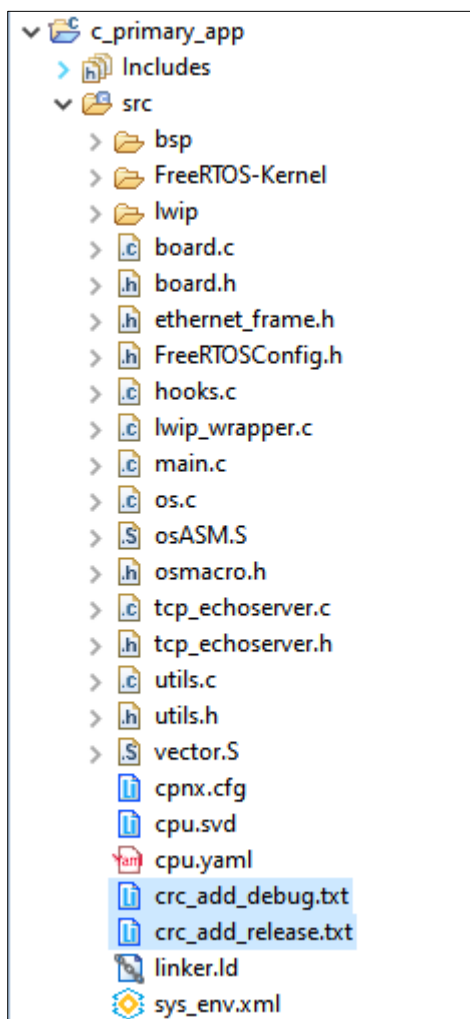


Figure 7.36. FreeRTOS Project Created

4. To avoid any confusion, move the following two files from `c_primary_app > src` to `c_primary_app` directory.

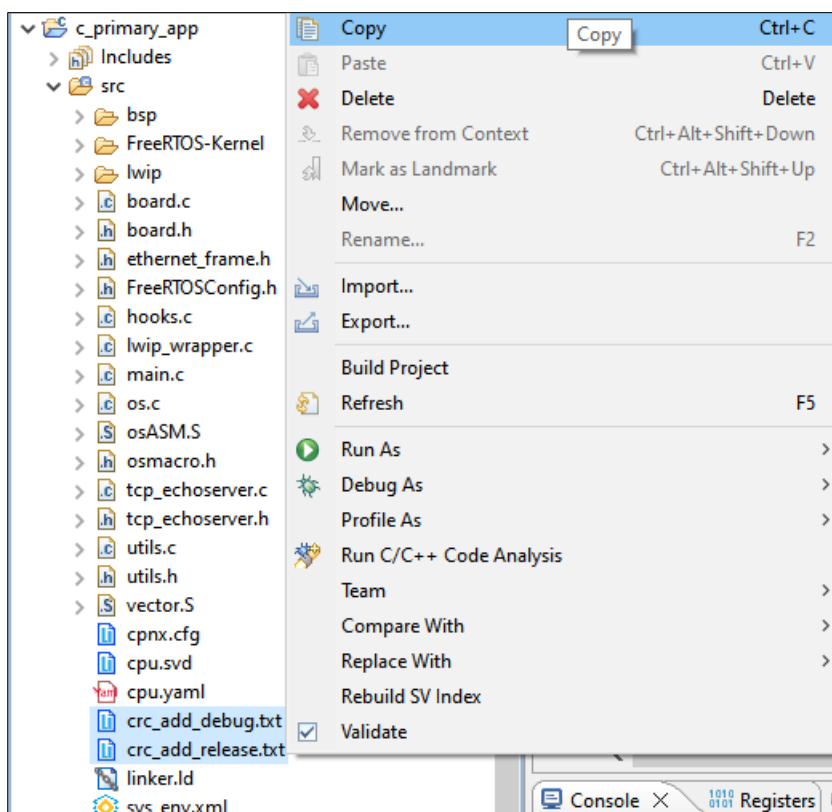


Figure 7.37. Copy the CRC Add files

- Paste it in your main c_primary_app project as shown in Figure 7.37.

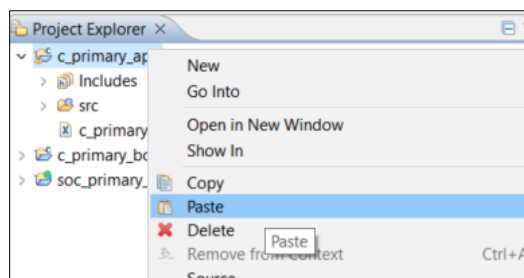


Figure 7.38. Paste in FreeRTOS Application C Project

- The text files are now added under the FreeRTOS App C project as shown in Figure 7.39.

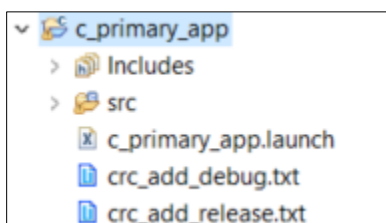


Figure 7.39. Copied Text Files

- To build the Primary GSRD, you need to define `_PRIMARY_BUILD_` in the start of the `main.c` file. This enables the SPI address to point to Primary FreeRTOS image in the SPI flash.

```
54 /* Set for GOLDEN OR PRIMARY build */
55 #define _PRIMARY_BUILD_
56
```

Figure 7.40. Primary Build Define – Set `_PRIMARY_BUILD_` for Primary Build in `main.c`

- In the `c_primary_app` folder, update the `crc_add_debug.txt` or `crc_add_release.txt` file as shown in Figure 7.41. 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 of `c_primary_appcrc.bin`. This output file has the CRC for application software appended at the end of binary file.

```
crc_add_debug.txt X
1# srec_cat command file to add the CRC and produce application file to be flashed
2# Usage: srec_cat @filename
3
4#first: create CRC checksum
5..\Debug\c_primary_app.bin -Binary
6-fill 0xFF 0x0000 0x40000 # fill code area with 0xff
7-crop 0x0000 0x3fffe # just keep code area for CRC calculation below
8-CRC16_Big_Endian 0x3fffe -CCITT # calculate big endian CCITT CRC16 at given address.
9-crop 0x3fffe 0x40000 # keep the CRC itself
10
11#second: add application file
12..\Debug\c_primary_app.bin -Binary
13-fill 0xFF 0x0000 0x3fffe # fill code area with 0xff
14
15# generate a Binary file
16-o ..\Debug\c_primary_appcrc.bin -Binary
```

Figure 7.41. Update `crc_add_debug.txt`

- Open the `Linker.ld` file from `c_primary_app`.

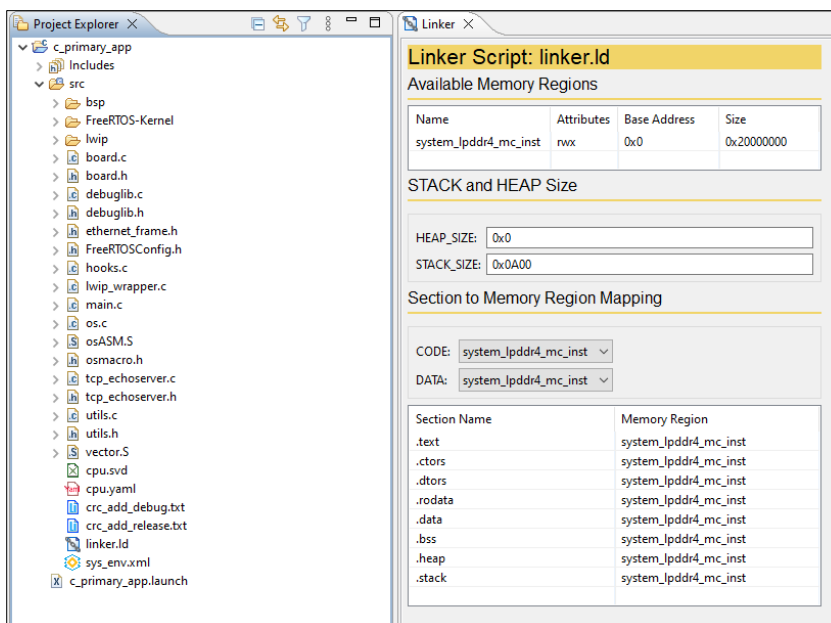


Figure 7.42. Open `linker.ld` File

10. Update the MEMORY org address to **0x0** for FreeRTOS to run from LPDDR4 memory. For the 4Gb LPDDR4, you can use the default linker.ld generated. For the 16G LPDDR4 part, refer to the [Changing LPDDR4 Memory Controller Memory Density](#) section.

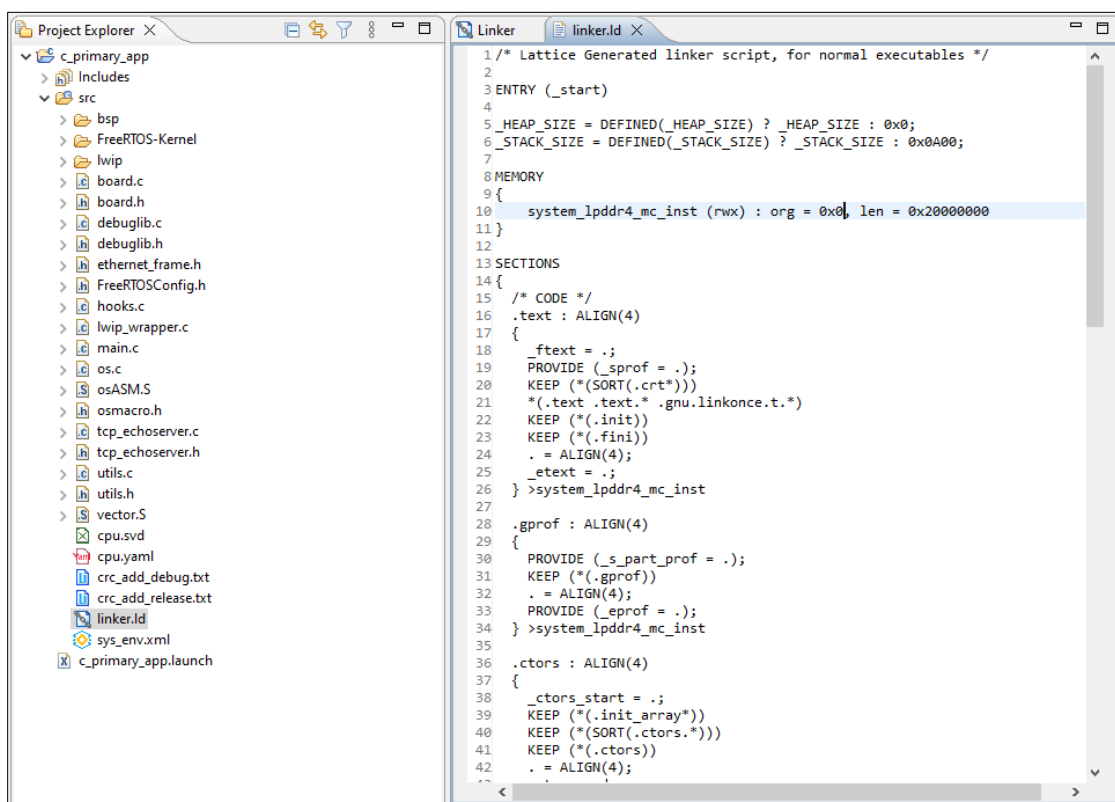


Figure 7.43. Update linker.ld File

11. Press **Ctrl + s** to save the change as shown in [Figure 7.44](#).

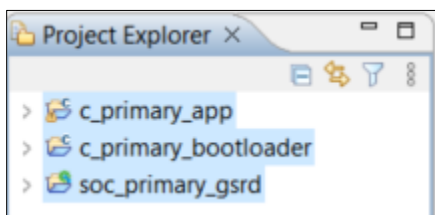


Figure 7.44. Workspace

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

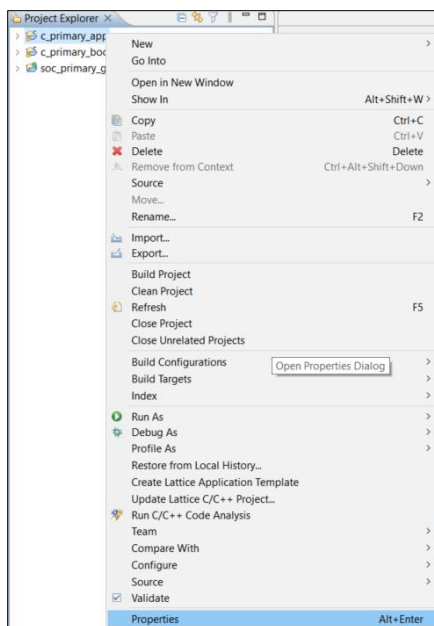


Figure 7.45. Properties

13. In case you wish to create a release folder on the C project build, go to **C/C++ Build > Settings**.

14. Click on **Manage Configurations**, select **Release**. Click **Set Active** and then click **OK**.

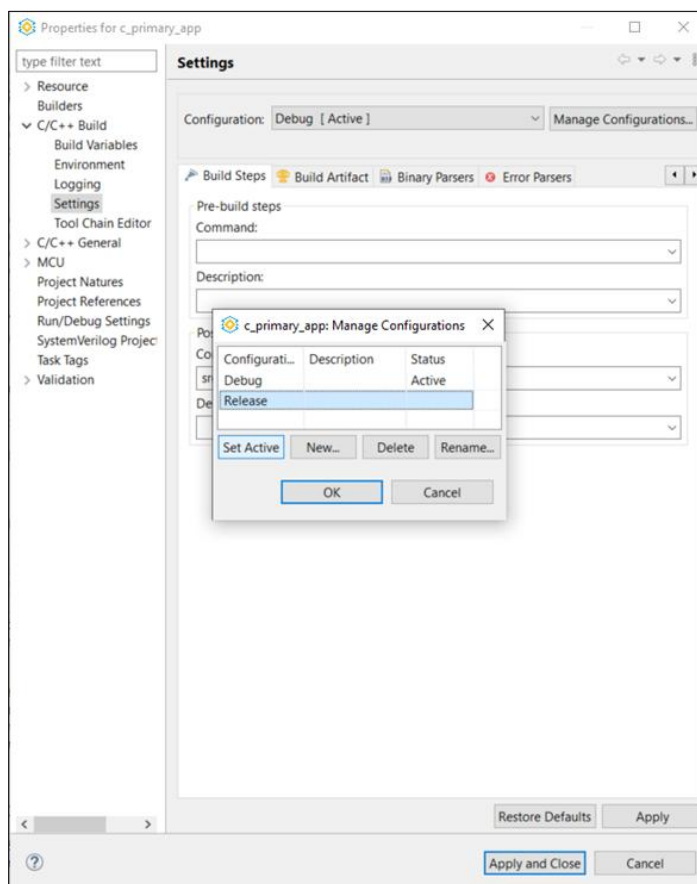


Figure 7.46. Set Release as Active Configuration

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

For Windows system:

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

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

For Linux system:

```
srec_cat "@../crc_add_debug.txt"
```

```
srec_cat "@../crc_add_release.txt"
```

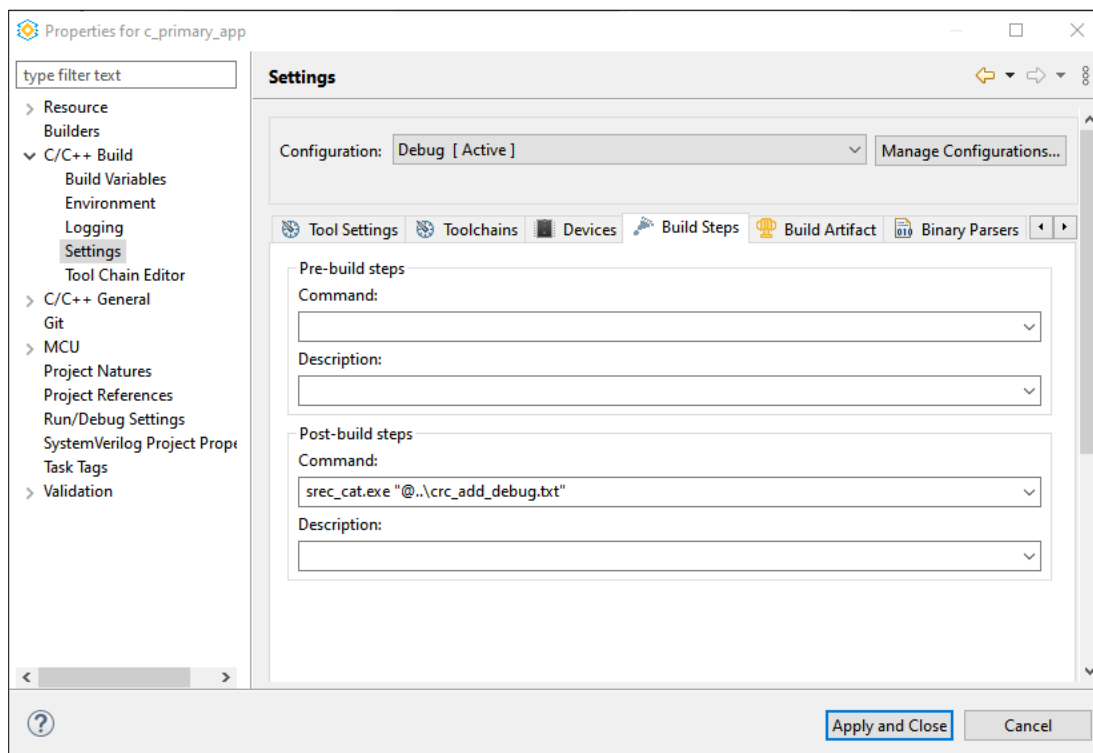


Figure 7.47. Adding Post Build Step for FreeRTOS Application CRC Binary Append (Windows)

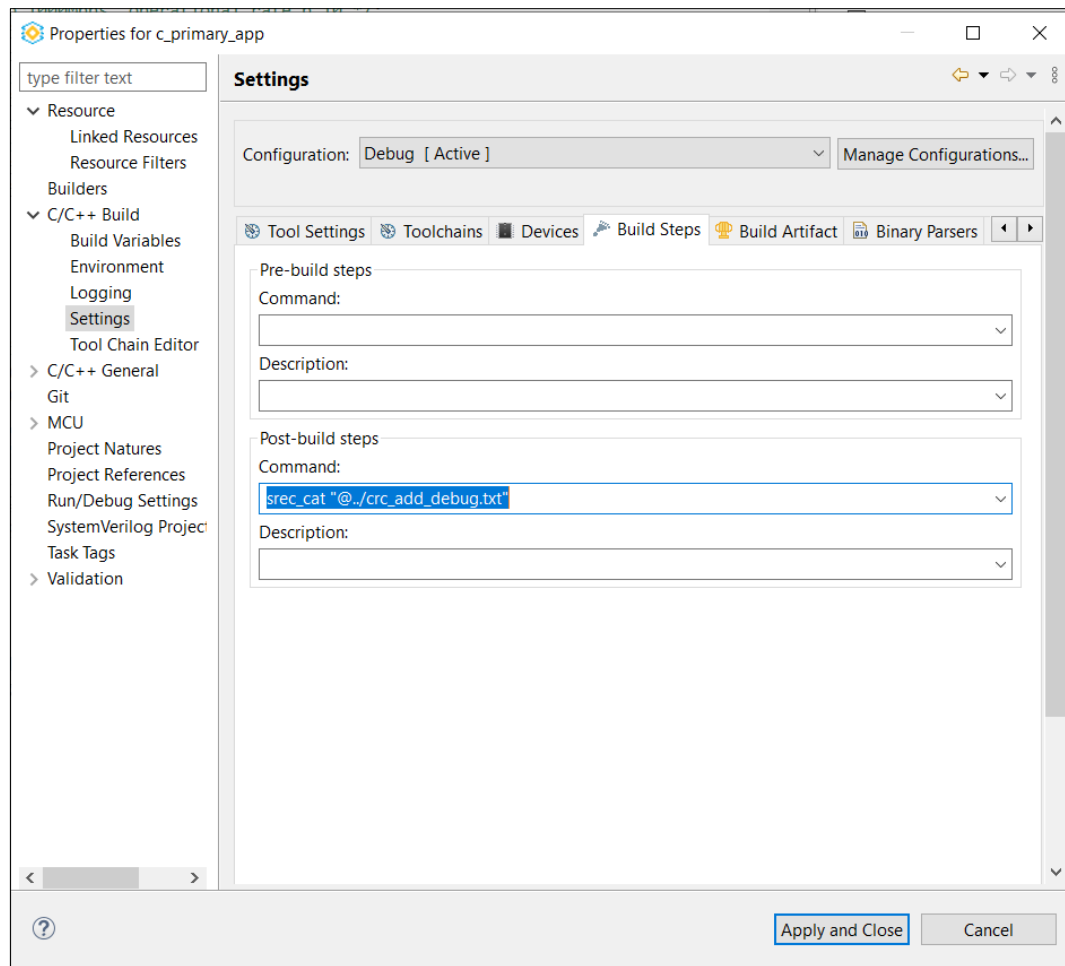


Figure 7.48. Adding Post Build Step for FreeRTOS Application CRC Binary Append (Linux)

16. For Linux system, update the path within the crc script to match Linux path format.

```
# srec_cat command file to add the CRC and produce application file to be flashed
# Usage: srec_cat @filename

#first: create CRC checksum
../Debug/c_golden_app.bin -Binary
-fill 0xFF 0x0000 0x40000          # fill code area with 0xff
-crop 0x0000 0x3ffff             # just keep code area for CRC calculation below
-CRC16_Big_Endian 0x3ffff -CCITT  # calculate big endian CCITT CRC16 at given address
-crop 0x3ffff 0x40000            # keep the CRC itself

#second: add application file
../Debug/c_golden_app.bin -Binary
-fill 0xFF 0x0000 0x3ffff         # fill code area with 0xff

# generate a Binary file
-o ../Debug/c_golden_appcrc.bin -Binary
```

Figure 7.49. Update the CRC Script to Match Linux Path Format

17. Click **Apply and Close**.
18. Right-click on **c_primary_app** and click on **Build Project**.

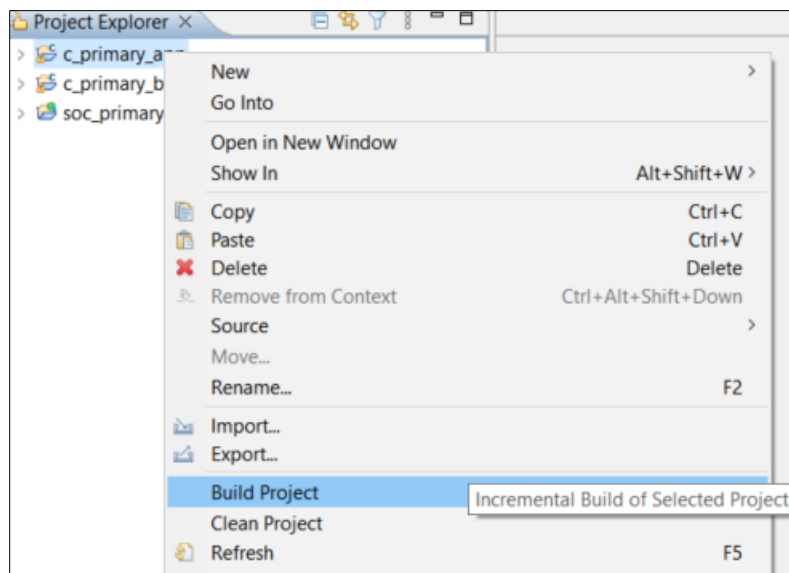


Figure 7.50. Build c_primary_app C/C++ Project

19. The console output is displayed as shown in Figure 7.51.

```
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"
Invoking: GNU RISC-V Cross Print Size
riscv-none-embed-size --format-berkeley "c_primary_app.elf"
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
text data bss dec hex filename
66040 100 308828 374968 5b8b8 c_primary_app.elf
Finished building: c_primary_app.siz
Finished building: c_primary_app.lst
Finished building: c_primary_app.mem

srec_cat.exe c_primary_app.bin -Binary -fill 0xFF 0x0000 0x40000 -crop 0x0000 0x3ffff -CRC16_Big_Endian 0x3ffff -CCITT -crop 0x3ffff 0x40000 c_primary_app.bin -Binary -fill 0xFF 0x0000 0x3ffff -o c
14:58:20 Build Finished. 0 errors, 0 warnings. (took 10s.728ms)
```

Figure 7.51. FreeRTOS App Build Project Console Output

20. This creates the debug folder and the following file with **c_primary_appcrc.bin** binary with CRC as shown in Figure 7.52.

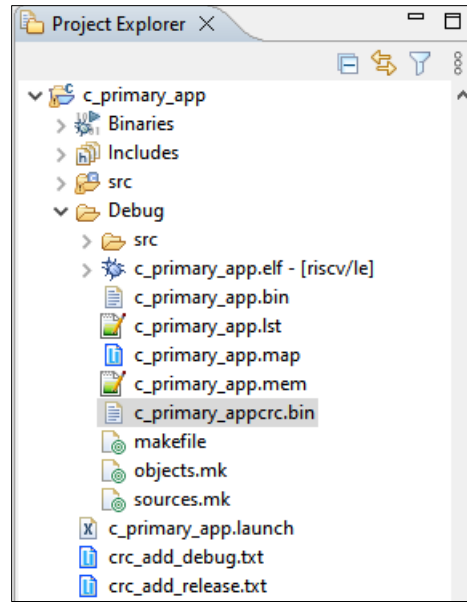


Figure 7.52. FreeRTOS App Binaries Created with CRC

21. For programming into the flash and confirming both primary bootloader and FreeRTOS projects are functioning correctly, refer to the [Programming Standalone Golden or Primary GSRD Bitstream and Application Software](#) section.
22. To create the following files with **c_golden_appcrc.bin** binary, set the **#define _GOLDEN_BUILD_** in the *main.c* file to build for golden bootloader and golden app binaries.

```
54 /* Set for GOLDEN OR PRIMARY build */
55 #define _GOLDEN_BUILD_
56
```

Figure 7.53. Golden Build Define – Set **_GOLDEN_BUILD_** for Golden Build in *main.c*

23. Repeat steps 1 to 21 to create golden bootloader and golden app project.

7.6. 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 Radiant Programmer tool from Lattice Radiant as shown in [Figure 7.54](#).

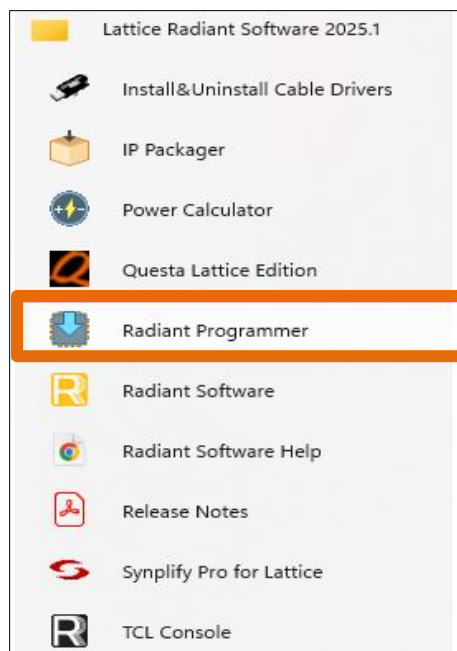


Figure 7.54. Launch Radiant Programmer from Windows Start

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

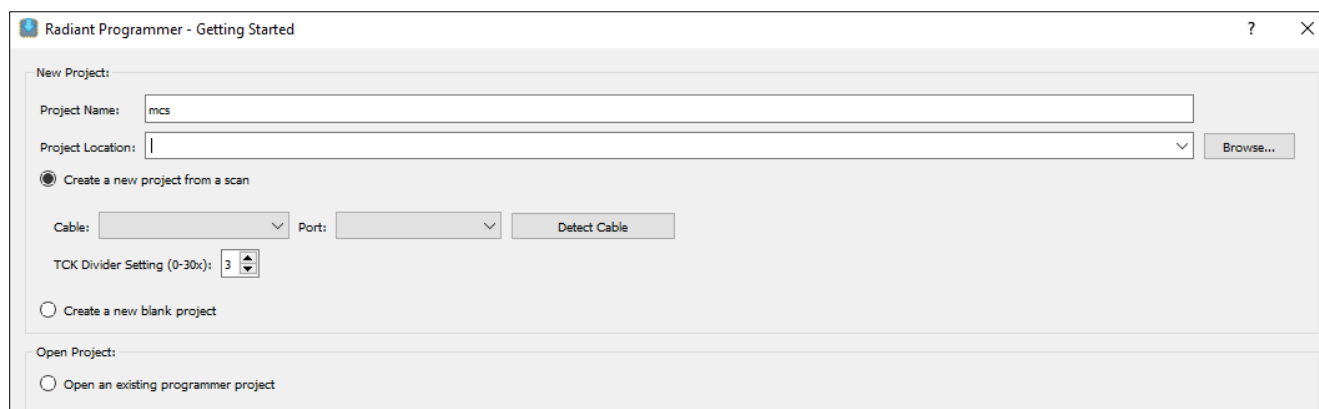


Figure 7.55. Radiant Programmer Getting Started Window

Note: This displays the error message shown in [Figure 7.56](#) since there is no hardware board connected. This error message can be ignored.

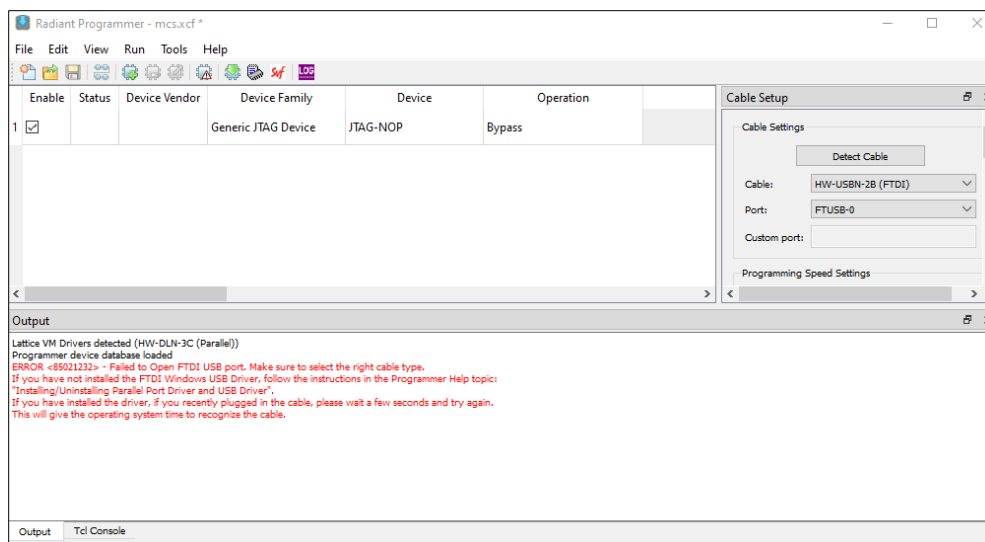


Figure 7.56. Error if No Board is Connected

- Click **Tools > Deployment Tool**.

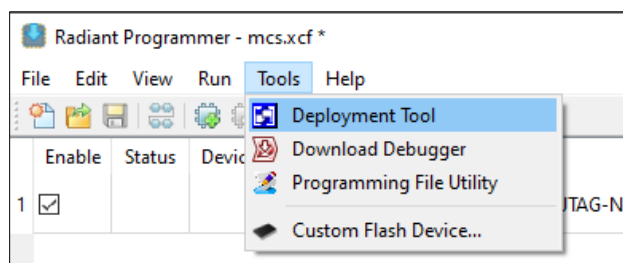


Figure 7.57. Open Deployment Tool from Radiant Programmer

- This opens the Deployment Tool window as shown in [Figure 7.58](#).

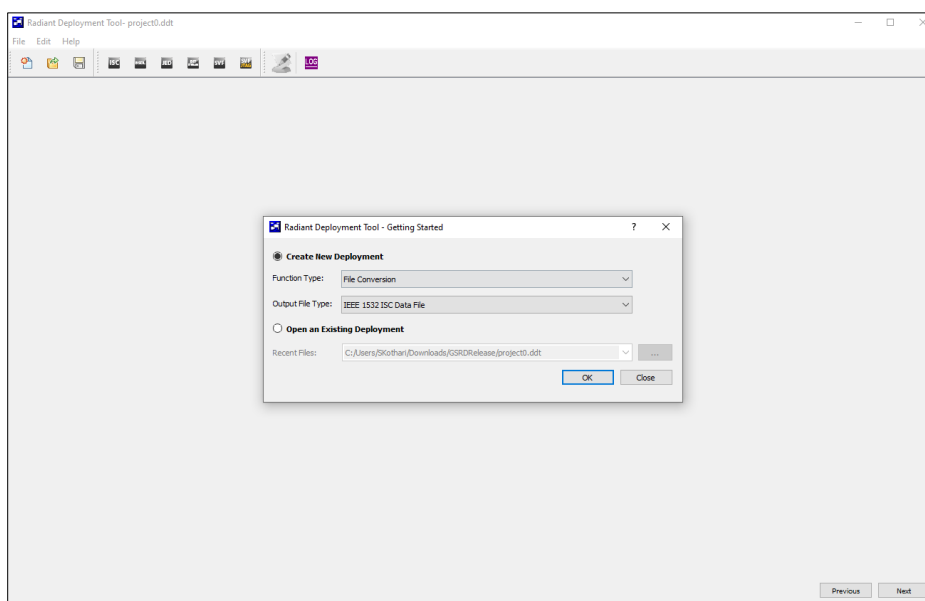


Figure 7.58. Deployment Tool Start Window

5. Apply the settings as shown in [Figure 7.59](#) and click **OK**.

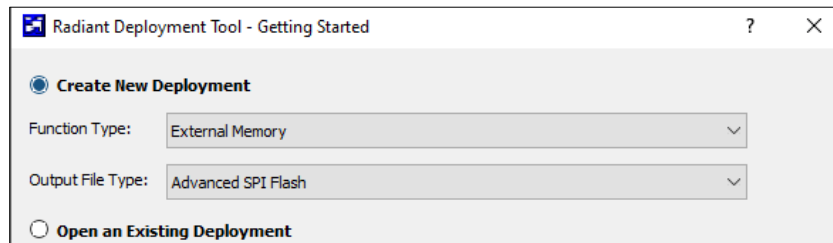


Figure 7.59. Options for Creating New Deployment

6. In **Step 1 of 4: Select the Input File(s)** window, as shown in [Figure 7.60](#):
 - a. Click the **File Name** field to browse and select the primary *.bit* file from your primary soc project.
 - b. The **Device Family** and **Device** fields auto-populate based on the bitstream *.bit* file selected.
 - c. Click **Next**.

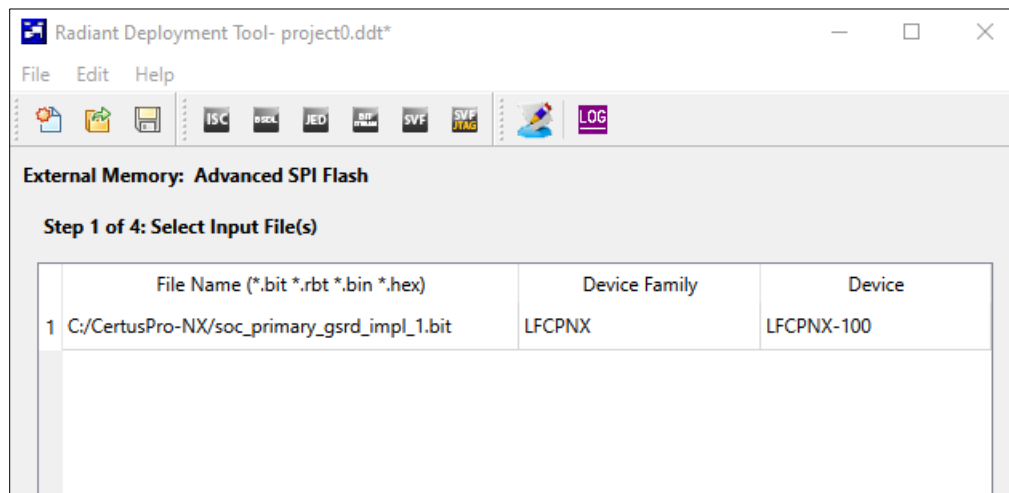


Figure 7.60. External Memory Step 1 of 4: Select Input Files

7. In **Step 2 of 4: Advanced SPI Flash Options**, select the fields as shown in [Figure 7.61](#).
 - a. Under **Options** tab, choose Output Format as **Intel Hex** and SPI Flash Size (Mb) as **512**.
 - b. Select **Quad I/O SPI Flash Read** from the SPI Flash Read Mode dropdown menu.

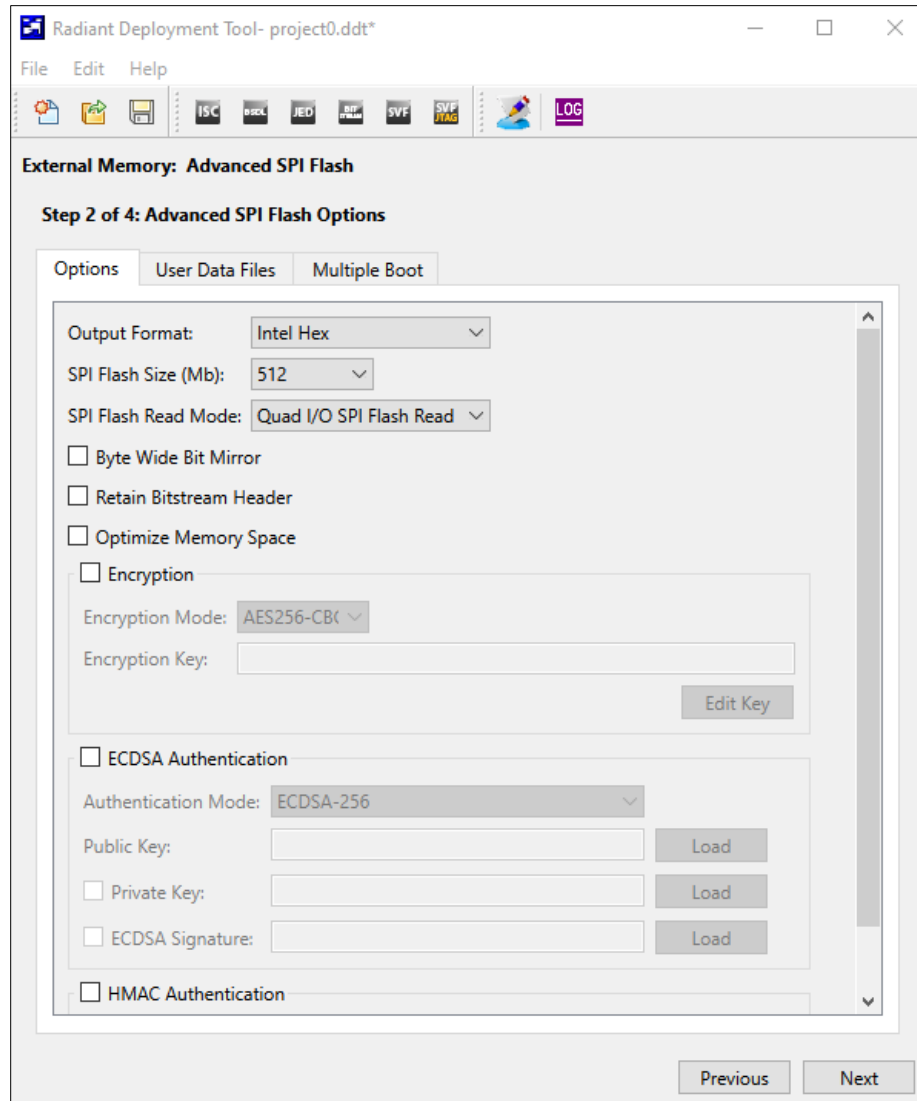


Figure 7.61. External Memory Step 2 of 4: Select Options

- c. No changes needed in **User Data Files** section.
- d. 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 SoC bitstream (that is *soc_golden_gsr_impl_1.bit*) file. The **Starting Address** of **Golden Pattern** is automatically assigned.
 - iv. Under **Alternate Pattern 1** field, click on the browse button and select the golden SoC 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**.

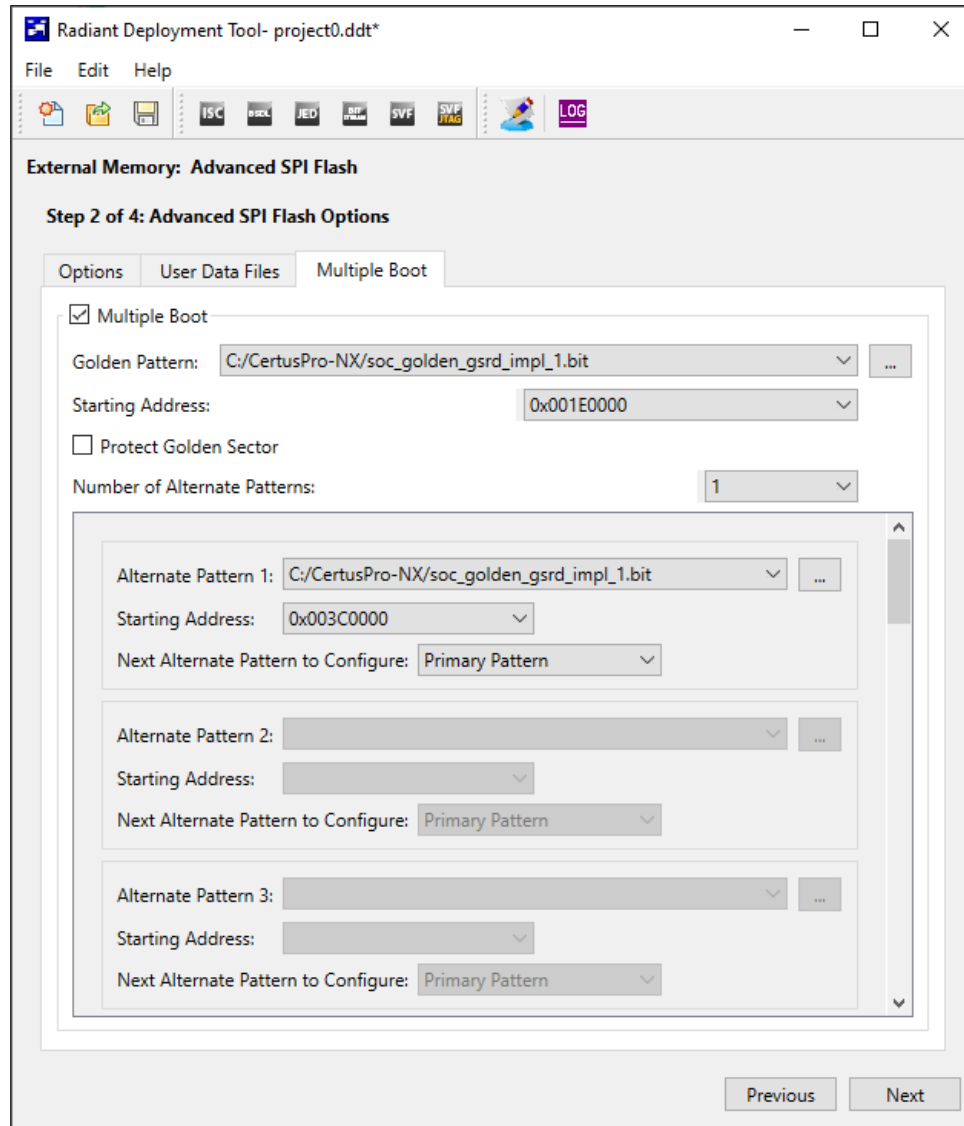


Figure 7.62. External Memory Step 2 of 4: Multiple Boot

8. In **Step 3 of 4: Select the Output File(s)** as shown in [Figure 7.63](#). Choose the location on your machine to generate a .mcs file.

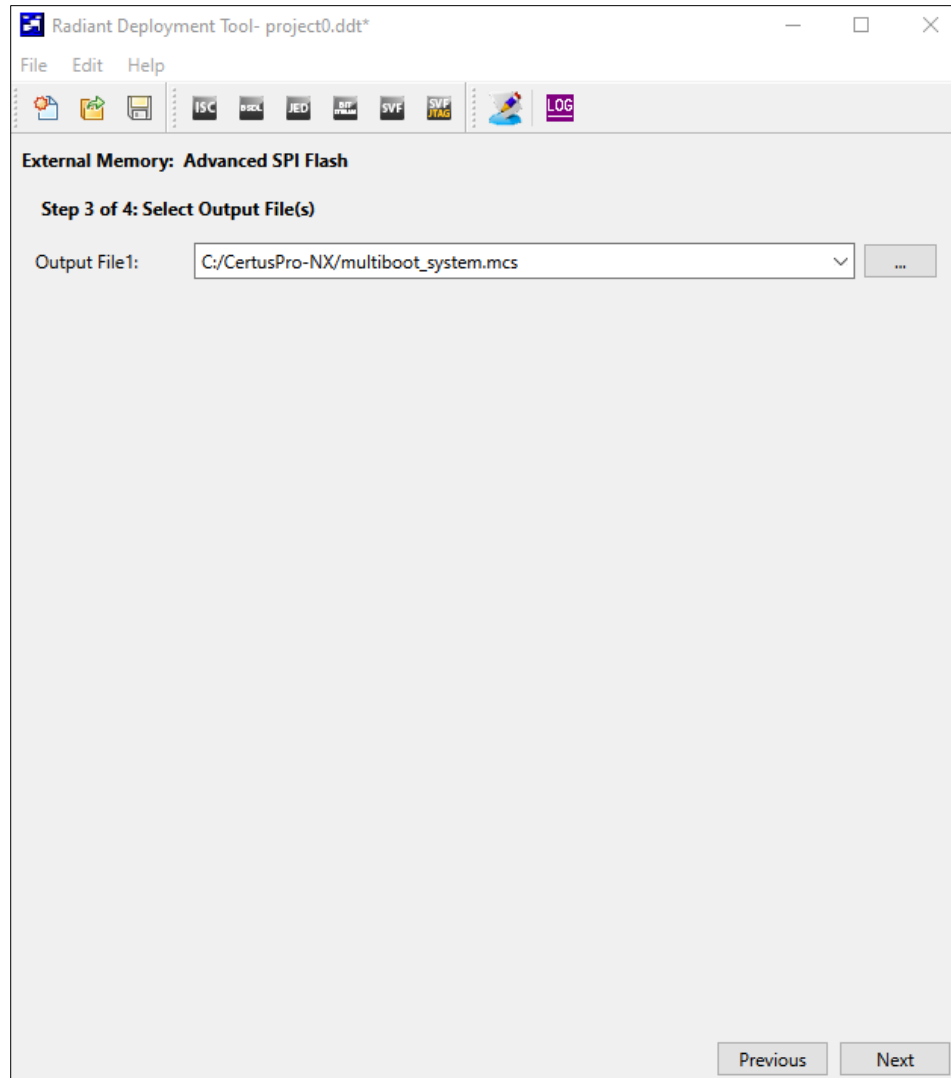


Figure 7.63. External Memory Step 3 of 4: Select Output File(s)

9. Click **Next**.
10. In **Step 4 of 4: Generate Deployment**, click **Generate** at the bottom right corner as shown in [Figure 7.64](#).

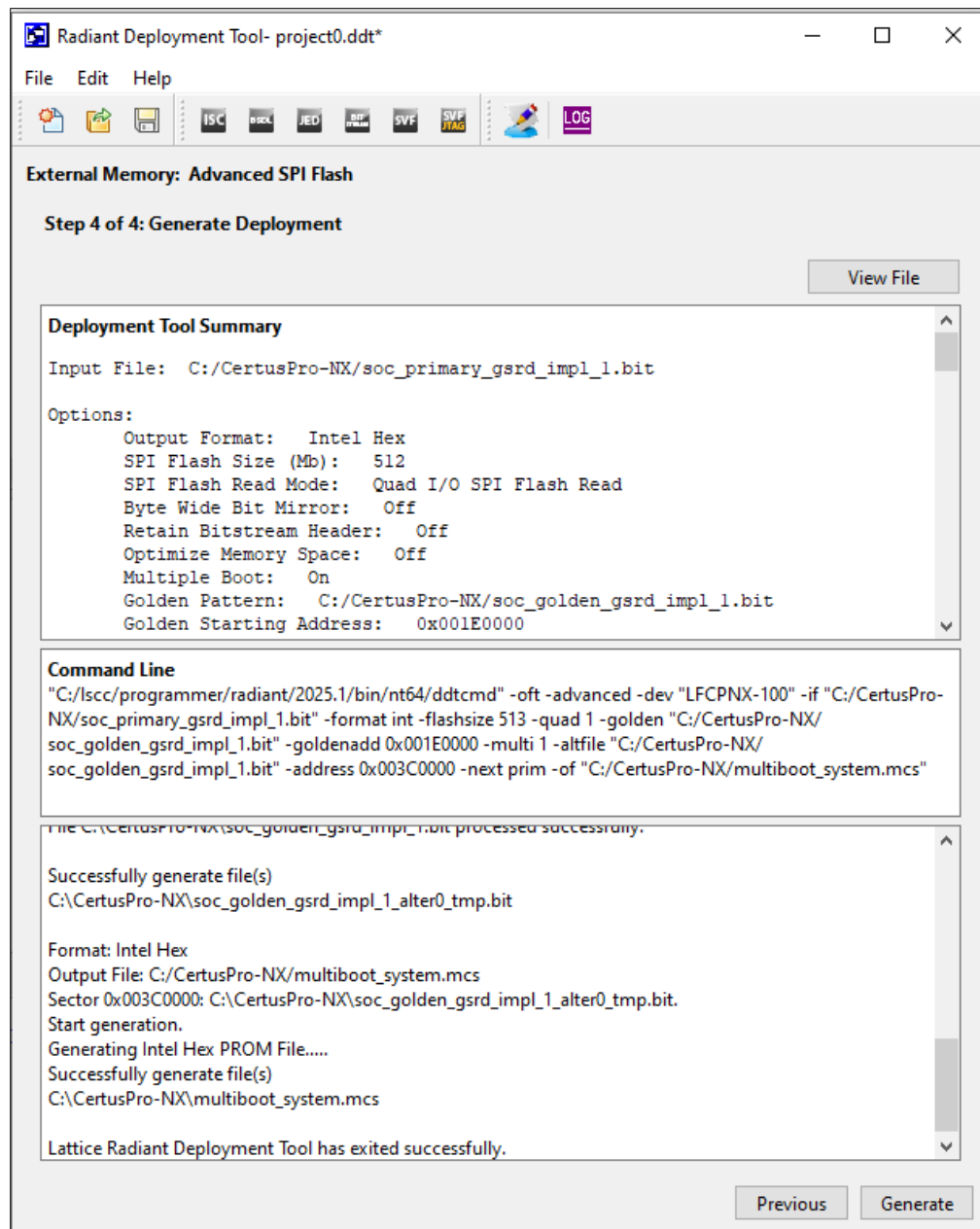


Figure 7.64. External Memory Step 4 of 4: General Development

11. Check for the following output as shown in [Figure 7.65](#).

Lattice Radiant Deployment Tool has exited successfully.

Figure 7.65. 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.
14. For programming into the flash and confirming the MCS file is built correctly, refer to the [Programming Standalone Golden or Primary GSRD Bitstream and Application Software](#) Programming the Golden, Primary Software, and MCS file section.

7.7. Setup Host Machine for Ping Test (Windows)

The setup is based on the setup on Windows 11 Host Machine to the evaluation board. For Linux-based machine, refer to the setup procedure in the [Linux Setup IP Address](#).

1. Search **Ethernet settings** from the Windows Menu.

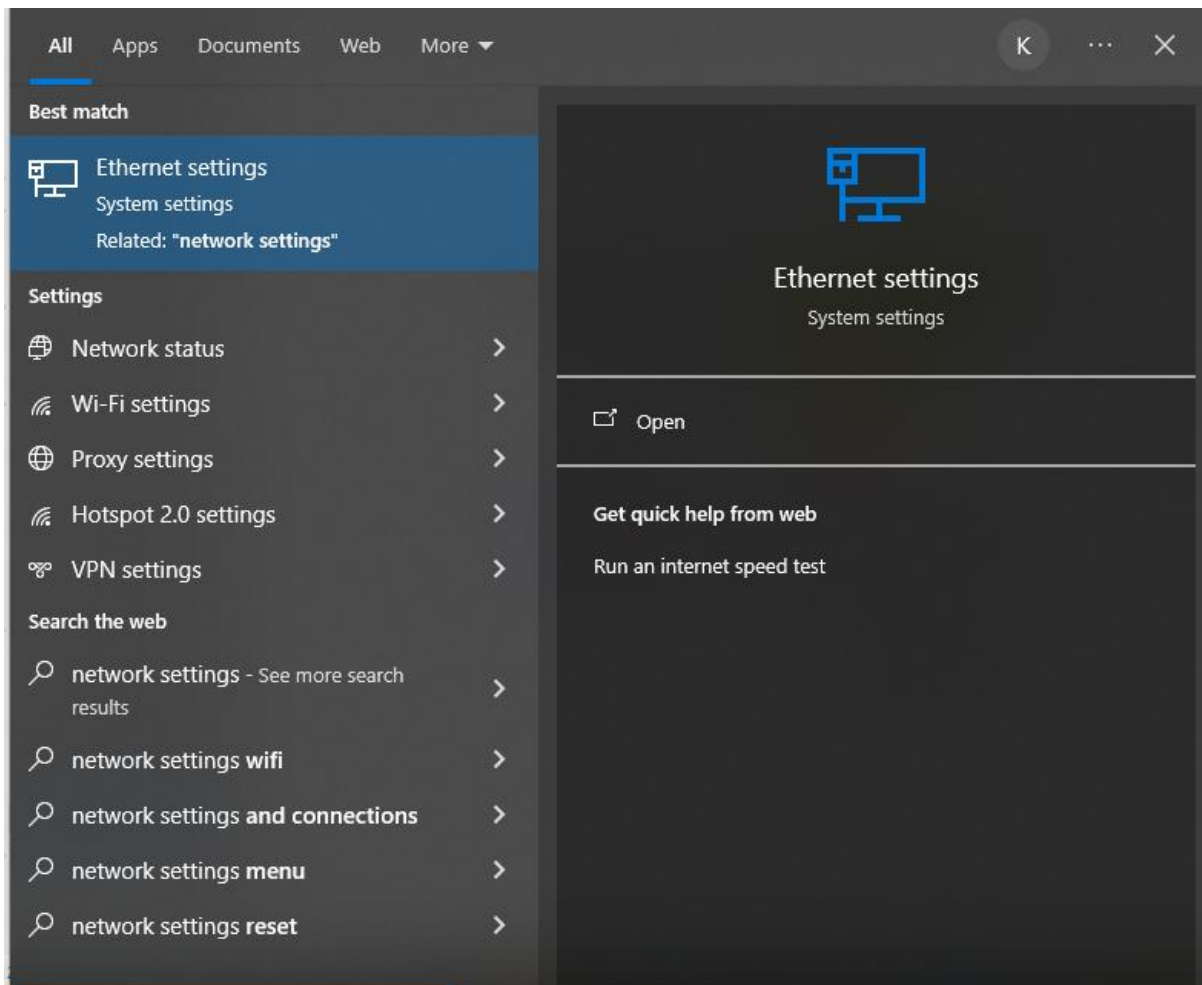


Figure 7.66. Configure Ethernet Settings

2. Select the **Ethernet** on the left panel and identify the Ethernet port connected to the evaluation board.

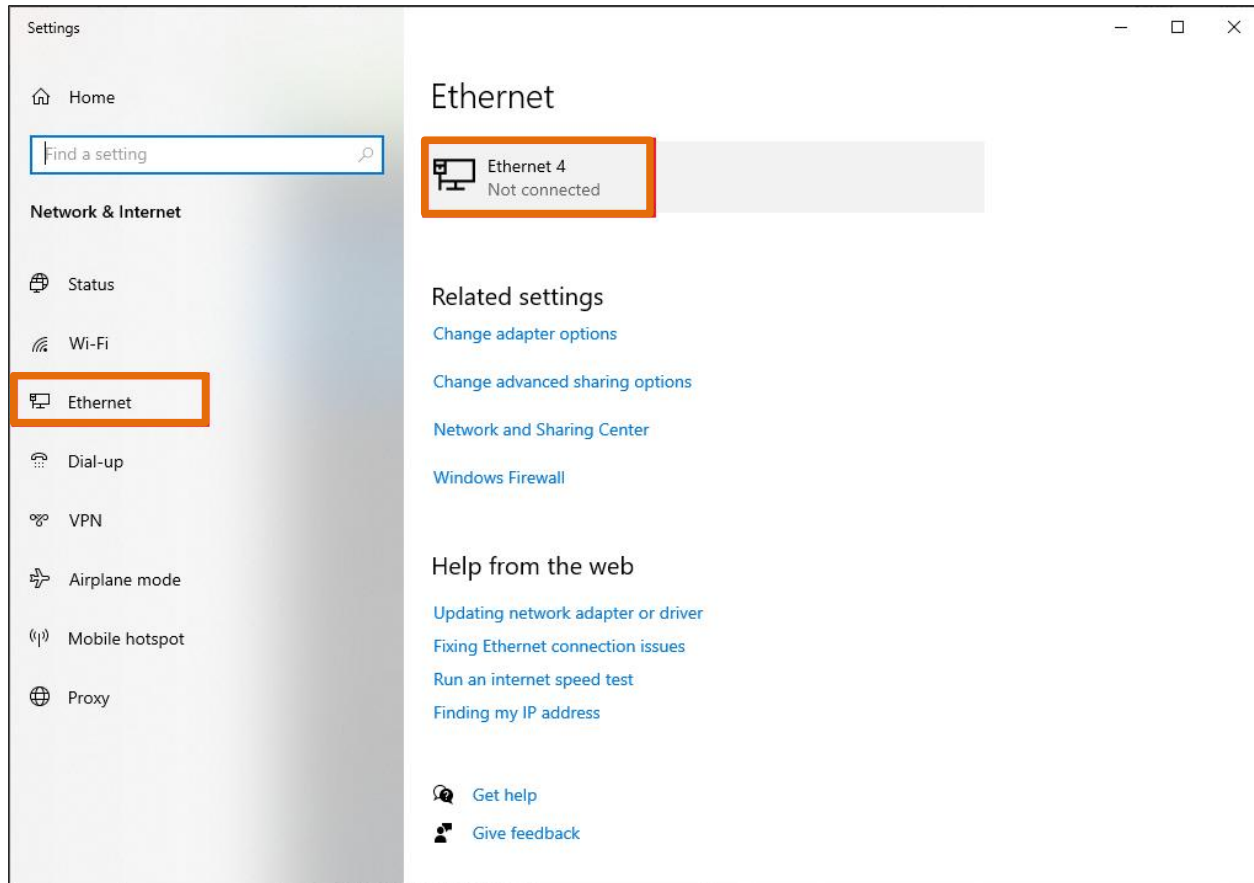


Figure 7.67. Select Ethernet Port

3. Click the **Ethernet** port connected to the evaluation port and click **Edit** at the **IP settings** section.

Note: The status of **Ethernet 4** is **Not connected** because this port is not yet configured.

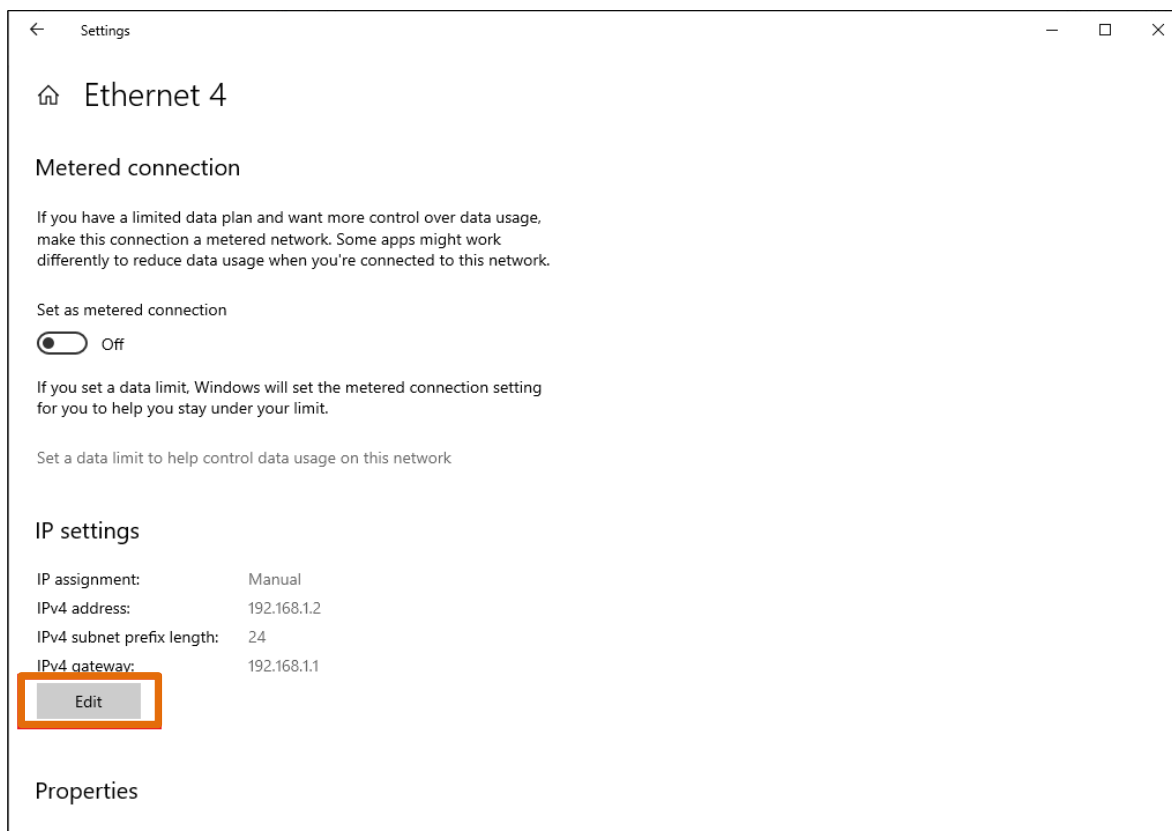


Figure 7.68. Edit IP settings

4. Set up the IP address, Subnet prefix length, and Gateway. Click **Save**.
 - **Edit IP Settings:** Manual
 - **IP Address:** 192.168.1.2
 - **Subnet Prefix Length:** 24
 - **Gateway:** 192.168.1.1

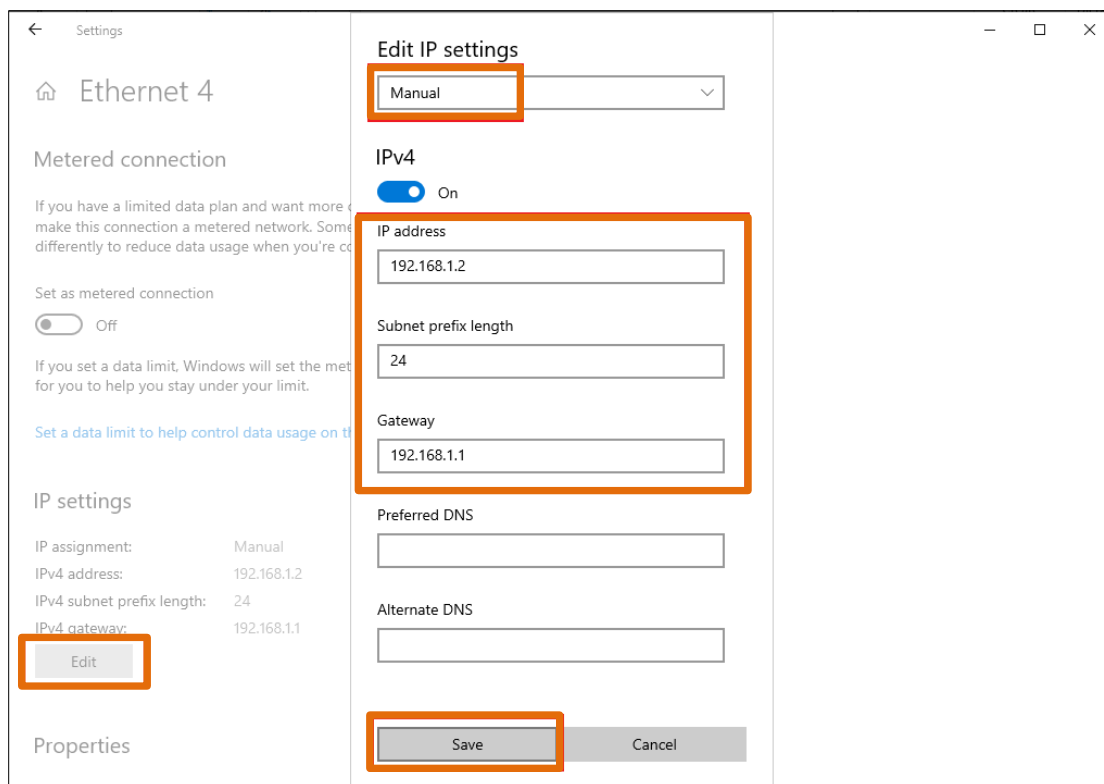


Figure 7.69. Set the IP Settings to Manual, IP address, Subnet Prefix Length, and Gateway

- Once the Ethernet settings are set up, open **Command Prompt** on the host PC.
- Type **ipconfig** at the **Command Prompt** to check the connection between the host PC and evaluation board as shown in [Figure 7.70](#).

```
Ethernet adapter Ethernet 4:

Connection-specific DNS Suffix  . : 
Description . . . . . : Intel(R) Ethernet Connection (23) I219-LM
Physical Address. . . . . : FC-5C-EE-B7-7E-7D
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . : Yes
IPv4 Address. . . . . : 192.168.1.2(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
NetBIOS over Tcpip. . . . . : Enabled
```

Figure 7.70. Check Ethernet Connection

- Run the ping command: **ping 192.168.1.4**. A successful ping to the evaluation board should indicate the number of packets sent equal to the number of packets received with zero packet loss.

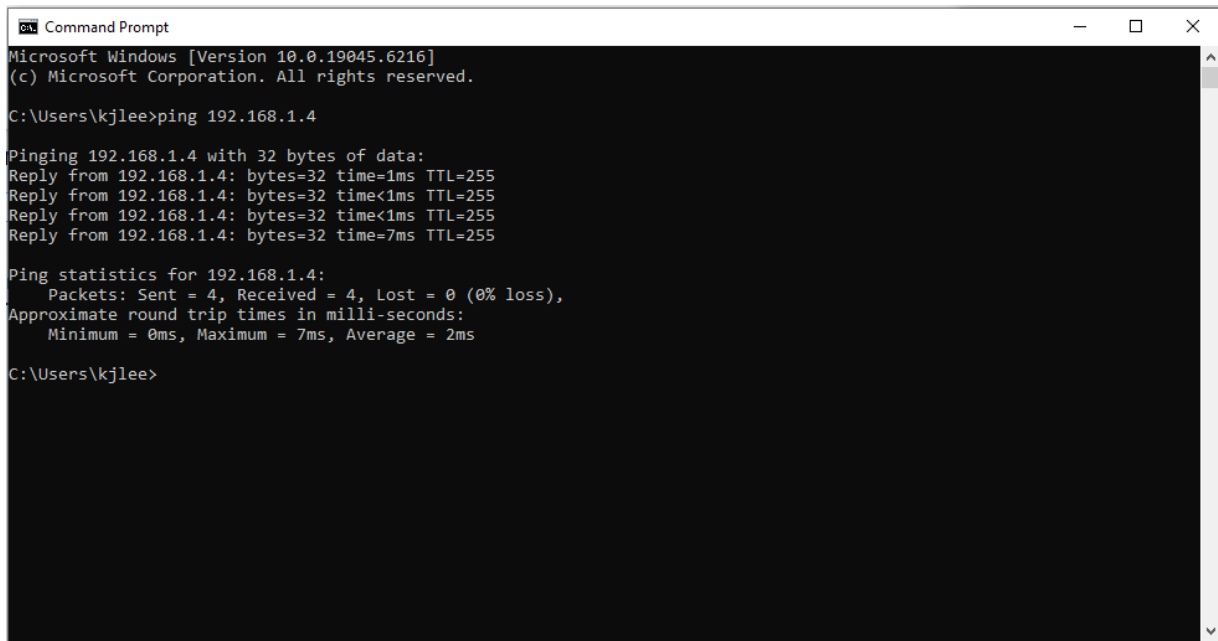


Figure 7.71. Ping the Device

8. Check the UART terminal on the evaluation board. Once the ping packets are received, dots are printed on the UART terminal.

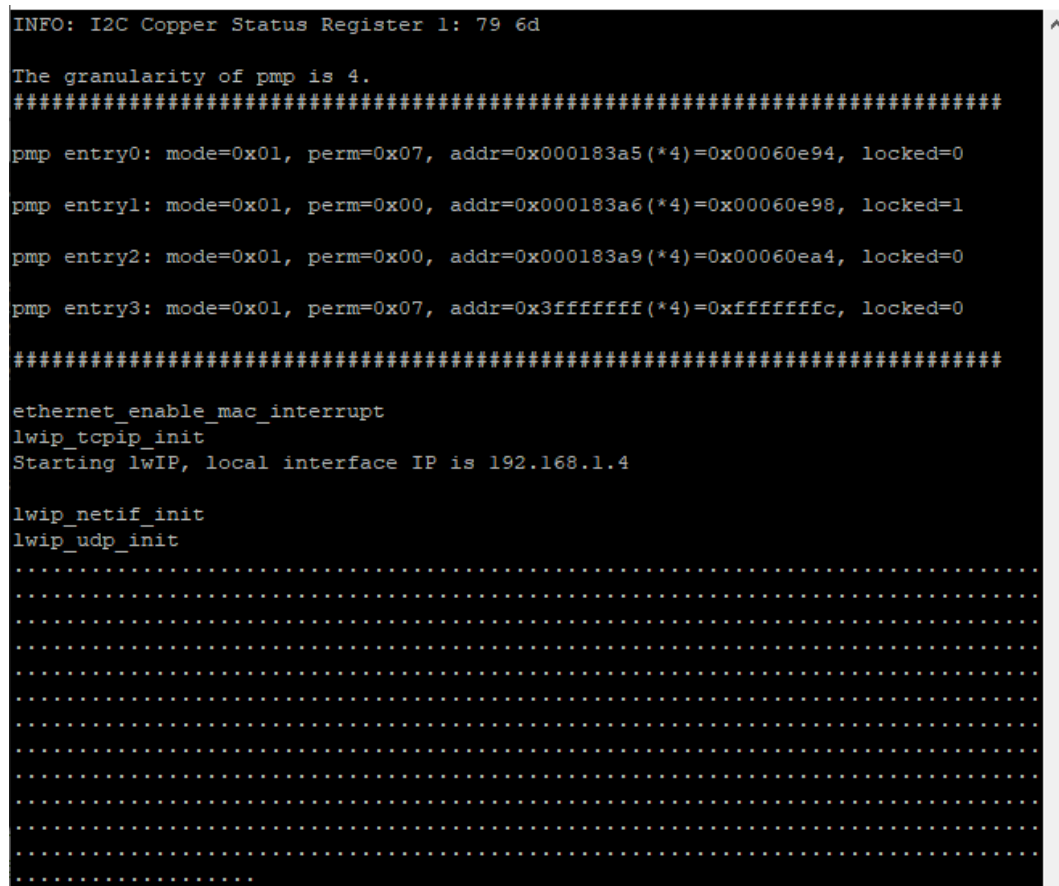


Figure 7.72. UART Terminal Showing the Printout when Ping Packet Received

8. Customizing the GSRD

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

8.1. Adding Component to the GSRD

This reference design can be used as a base design to add components or IPs that your project requires. You can perform this in Propel Builder using the Schematic view. The following procedure shows the design flow in general.

8.1.1. Hardware Flow

To add a component or IP, perform the following:

1. In **Propel Builder**, select and add IP from the **IP Catalog** window. Complete the IP configuration using the wizard and generate.
Note: If you need to create a custom IP, refer to [Lattice IP Packager 2025.1 \(FPGA-UG-02236\)](#).
2. Connect the newly added IP to the RISC-V CPU or other IPs in the system. There are three primary interface types:
 - AXI4 or AXI-lite – Add new Manager/Subordinate interface in system_ic_inst (depending on the interface type on the new IP). Connect the newly added IP to the newly added interface on the interconnect.
 - APB – Add new Requestor/Completer interface in system_apb_ic_inst. Connect the newly added IP to the newly added interface on the interconnect.
 - AHB-Lite – Add new Manager/Subordinate interface in system_ic_inst. Since the newly added IP has AHB-Lite interface, you need to add AXI4 to AHB-Lite Bridge IP in between.
3. Connect the clock and reset signals, and data buses of the newly added IP.
4. Export the I/O from newly added IP to top level module (if applicable).
5. Go to **Address** view to assign the base address for the newly added IP.
6. If you made changes to the bootloader, follow these steps to update the System Memory's initialization file. Otherwise, skip this step.
 - a. Select the System Memory instance from the **Design View** in the left column and it highlights the system_boot_mem_inst.
 - b. Double-click on the highlighted component shown in the schematic and it opens the **Module/IP Block Wizard**.
 - c. Click on the three dots at the **Initialization File's Value** field to locate and select your bootloader file.
 - d. Click **Generate** at the bottom-right corner of the **Module/IP Block Wizard** as shown in [Figure 8.1](#).

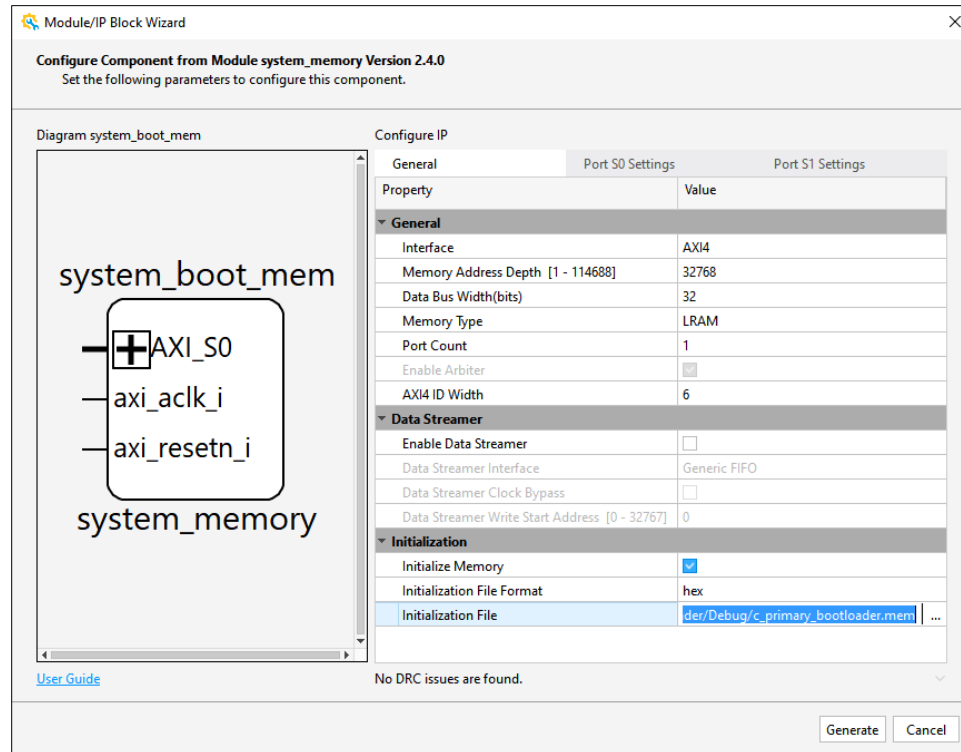


Figure 8.1. Bootloader File Updated in System Memory

- Upon completion, select **Design > Validate Design** and make sure no DRC error.
- Select **Design > Generate > Generate** to update the SoC design.
- In **Lattice Radiant**, assign pins for the newly added IP (if applicable).
- Click **Export Files** to generate the updated bitstream.

8.1.2. Software Flow

If the newly added IP contains software driver, update the Board Support Package (BSP) in the software project. To update the BSP, perform the following:

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

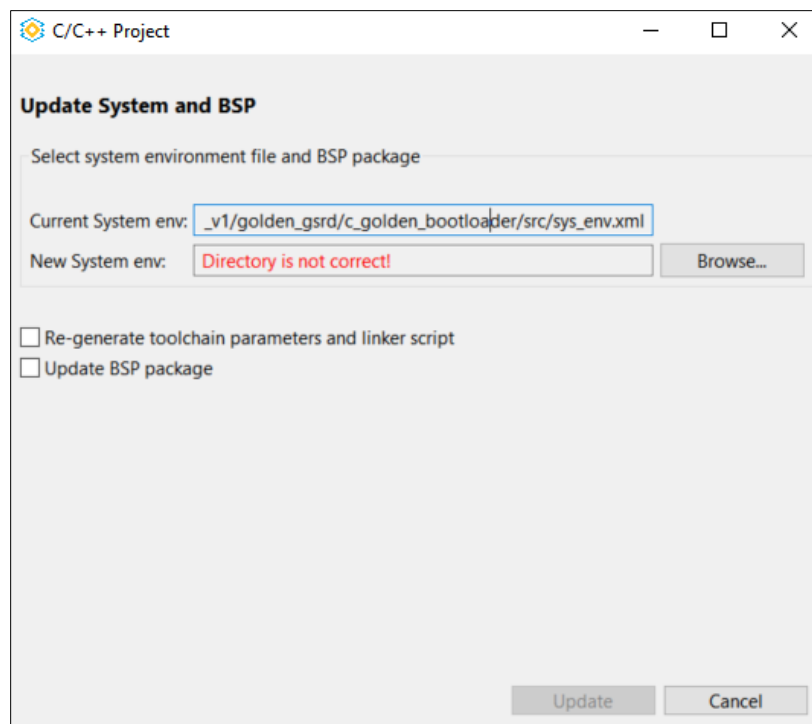


Figure 8.2. New System ENV Error

2. Click the **Browse** button to navigate to the new system env path in your PC.
For example: `<my_work_dir>/sge/sys_env.xml`.

Note: This step updates the software project to point to the Propel Builder system env file located on your PC. The correct path is displayed instead of the previous error.

3. Select the **Update BSP package** box. This shows the newly added IP driver and version available for update.

Note: Do NOT select **Re-generate toolchain parameters and linker script** box. The software projects provided in this reference design contain modified linker script. Selecting this option overwrites the modifications.

4. Click **Update** to complete the process.
5. Build the software project to obtain updated executable files (.elf, .mem, and .bin).

8.2. Changing LPDDR4 Memory Controller Memory Density

The LPDDR4 memory controller IP parameters are set according to the LPDDR4 SDRAM available on the CertusPro-NX Versa Evaluation Board, which can be differentiated by the BRI number at the back side of the board as indicated in [Figure 8.3](#).

Table 8.1. Types of LPDDR4 SDRAM in CertusPro-NX Versa Evaluation Board

BRI Number	LPDDR4 SDRAM Density	Micron Part Number
LSC-2309-001	4 Gb	MT53E128M32D2DS-053-WT:A
Other BRI numbers	16 Gb	MT53E512M32D1ZW-046-WT:B

Both SDRAMs support 533 MHz DDR Command frequency. You can change the IP parameters to suit your board and LPDDR4 memory device in your project.

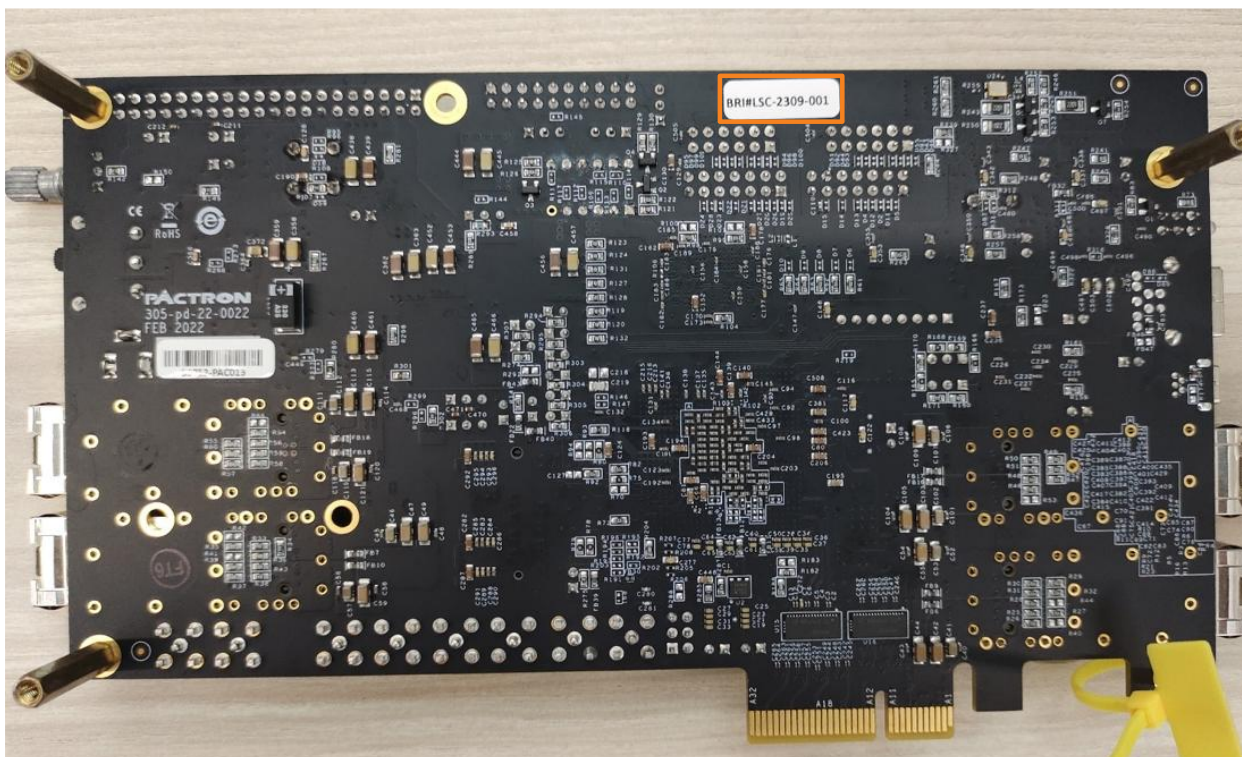


Figure 8.3. Identifying LPDDR4 Memory Device through BRI Number

To modify the IP parameters, perform the following:

1. In Propel Builder, double-click on the *system_lpddr4_mc_inst* block.

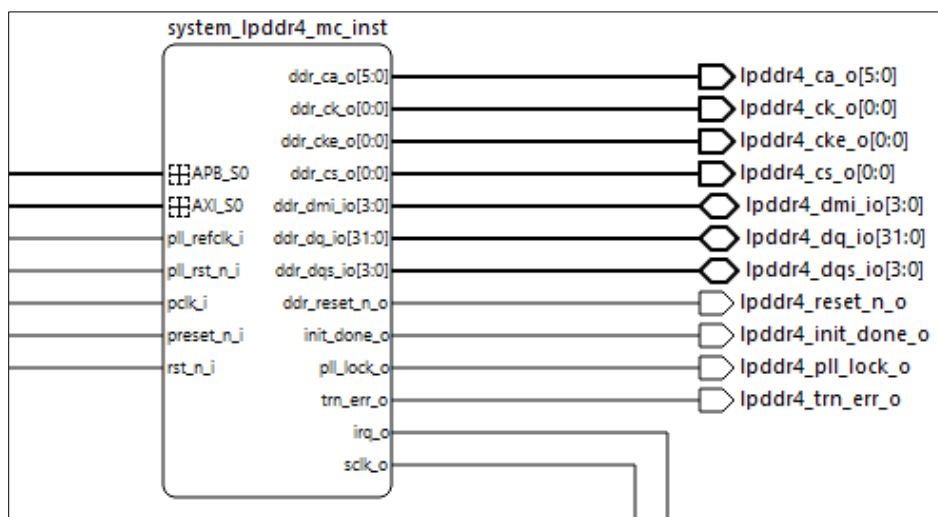


Figure 8.4. IP Block

2. In the Module/IP Block Wizard, make the desired changes to the parameters that suit your board and LPDDR4 memory device. For example, you want to change anything or all.
 - The following example demonstrates the changing of DDR Density (per channel) from 2 Gb to 8 Gb as well as DDR Command Frequency from 533 MHz to 400 MHz.

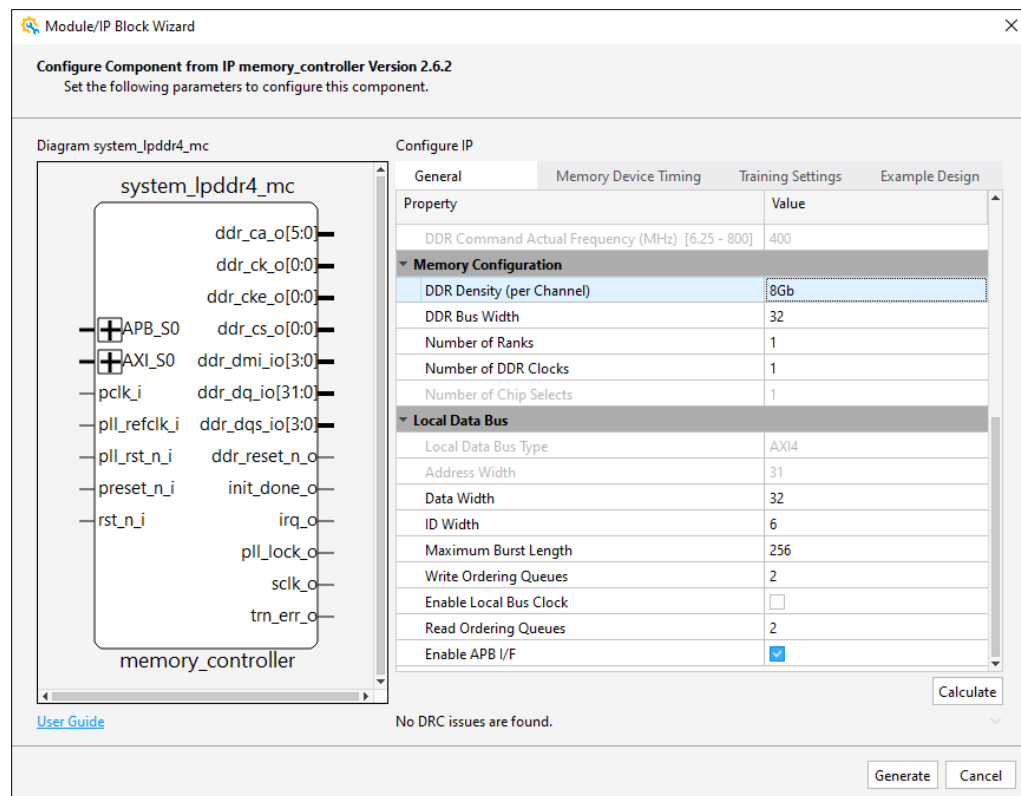
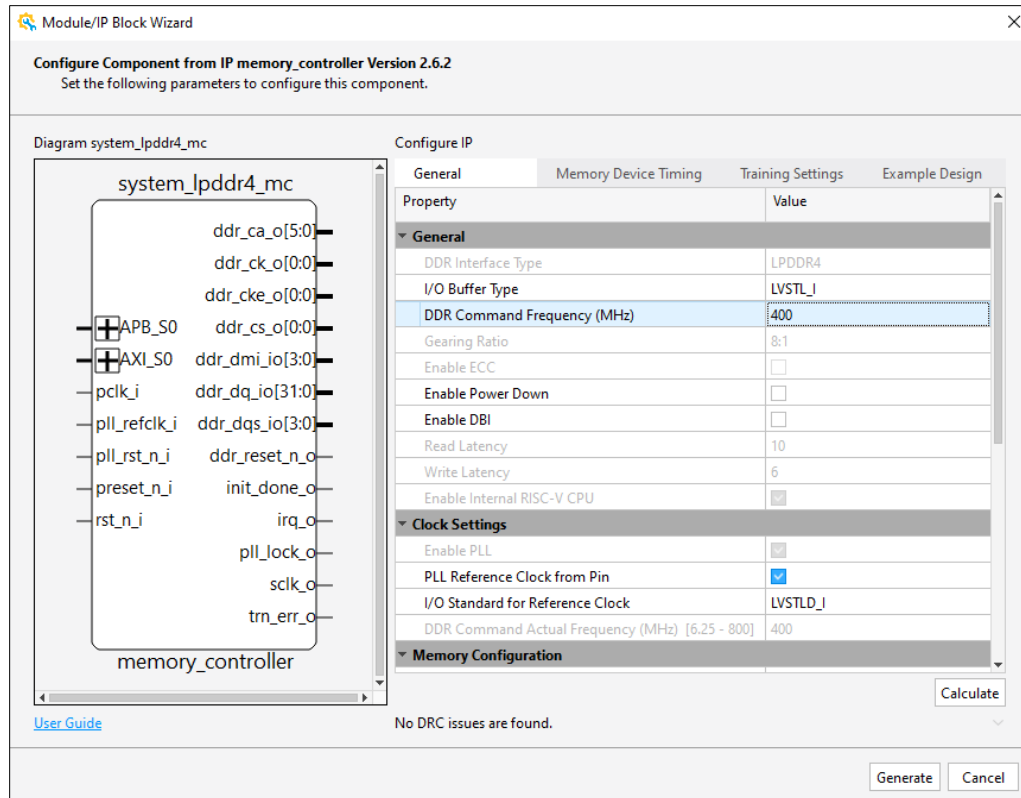


Figure 8.5. Customize IP

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

4. You can see the message below in the TCL Console as shown in [Figure 8.6](#). Ensure that there are no errors.

```

$ sop_configure_ip -vlnv {latticesemi.com:ip:system_ipddr4_mc:2.6.1} -meta_vlnv {latticesemi.com:ip:memory_controller:2.6.1} -cfg_value {AXI_DATA_WIDTH:32,AXI_ID_WIDTH:6,CA_SLASH_RATE:100,DATA_CLK_ENFORCE:0,DOR_CHD_FREQ:100,DOR_DENSITY:1,
DOR_WIDTH:32,MAX_BUFFER_LEN:256}
$ sop_replace -vlnv {latticesemi.com:ip:system_ipddr4_mc:2.6.1} -name {system_ipddr4_mc_inst} -component {soc_golden_gsrdr/system_ipddr4_mc_inst}
INFO <2359992> - The following interface port bus width do not match:
'ARADDR,AWADDR' in 'system_ic_inst.AXI_M02' and 'system_ipddr4_mc_inst.AXI_S0'
INFO <2359992> - The following interface port bus width do not match:
'PADDR' in 'system_apb_ic_inst.APB_M02' and 'system_ipddr4_mc_inst.APB_S0'

```

Figure 8.6. TCL Console Output After Desired Customization

5. Address Map is changed from 512 MB to 2 GB.

Cell	Base Address	Range	End Address	Lock
eth_sgdma_inst				
soc_golden_gsrdr/eth_sgdma_inst/SGDMA_BD_Address_Space (32 address bits: 4G)				
dual_boot_fpga_config_inst/APB_S0	0xC0002000	4K	0xC0002FFF	✓
eth_i2c_inst/APB_S0	0xC000C000	4K	0xC000CFFF	✓
eth_sgdma_inst/APB	0xC0003000	4K	0xC0003FFF	✓
eth_tse_mac_sgmmi_inst/APB_S0	0xC0004000	16K	0xC0007FFF	✓
system_boot_mem_inst/AXI_S0	0x80000000	128K	0x8001FFFF	✓
system_gpio_inst/APB_S0	0xC0001000	4K	0xC0001FFF	✓
system_ipddr4_mc_inst/AXI_S0	0x00000000	2G	0x7FFFFFFF	✓
system_ipddr4_mc_inst/APB_S0	0xC000B000	4K	0xC000BFFF	✓
system_ospi_fc_inst/AXI4_SUB	0xC4000000	4K	0xC4000FFF	✓
system_uart_inst/APB_S0	0xC0000000	4K	0xC0000FFF	✓

Figure 8.7. Updated Address Map

6. Update below in the constraints.sdc file.

```

#####
## LPDDR4 IP Output Clocks
#####
create_generated_clock -name {lpddr4_eclk} -source [get_pins {system_ipddr4_mc_inst/lscclpddr4_mc_inst/u_lp4mem/u_pll/gen_no_refclk_mon_u_pll/REFCLK}] -divide_by 4 -multiply_by 16 \
[get_pins {system_ipddr4_mc_inst/lscclpddr4_mc_inst/u_lp4mem/u_pll/gen_no_refclk_mon_u_pll/CLK05}]
#####

```

Figure 8.8. Updated constraints.sdc

7. Click the **Validate** button as shown in [Figure 8.9](#). Ensure that there are no errors.



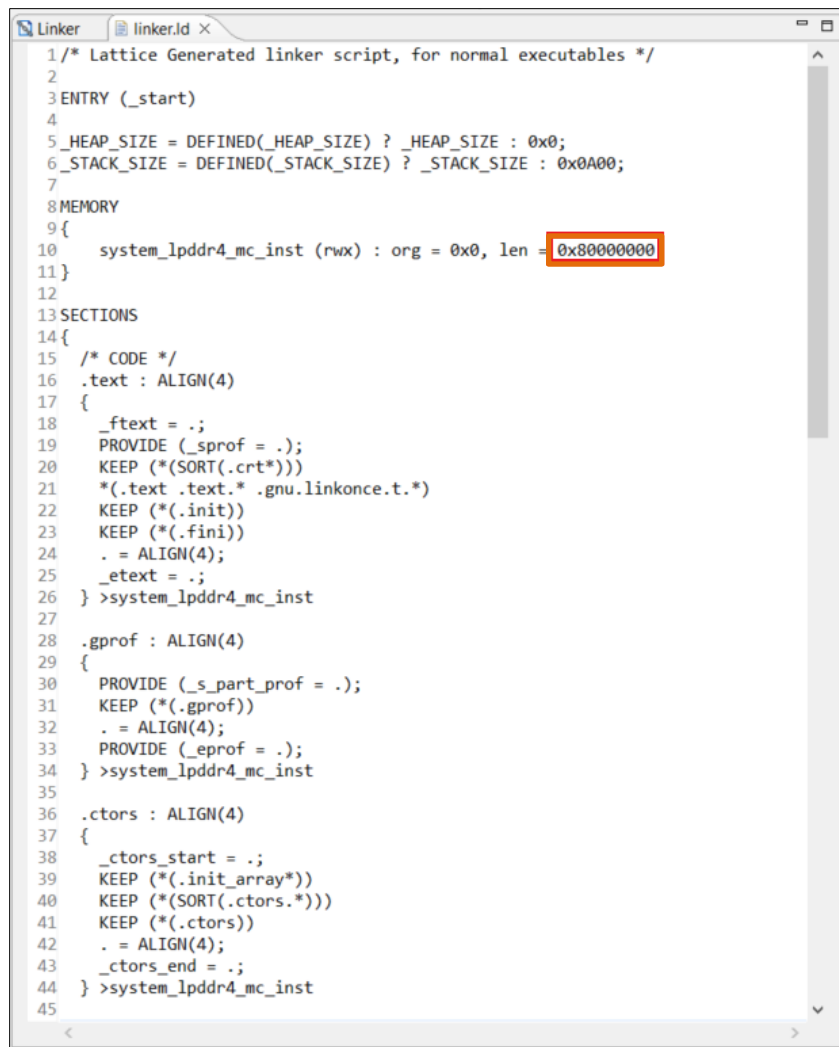
Figure 8.9. Validate Design Again if Any IP is Updated

8. Click the **Generate** button as shown in [Figure 8.10](#). Ensure that there are no errors.



Figure 8.10. Generate Again if any IP is Updated

- Launch the Propel SDK and open the FreeRTOS C/C++ project.
- Edit the linker script with updated LPDDR4 Memory density in C/C++ project.
 - len = 0x80000000 for 16 Gb LPDDR4 SDRAM
 - len = 0x20000000 for 4 Gb LPDDR4 SDRAM



```

1 /* Lattice Generated linker script, for normal executables */
2
3 ENTRY (_start)
4
5 _HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : 0x0;
6 _STACK_SIZE = DEFINED(_STACK_SIZE) ? _STACK_SIZE : 0x0A00;
7
8 MEMORY
9 {
10     system_lpddr4_mc_inst (rw) : org = 0x0, len = 0x80000000
11 }
12
13 SECTIONS
14 {
15     /* CODE */
16     .text : ALIGN(4)
17     {
18         _ftext = .;
19         PROVIDE (_sprof = .);
20         KEEP (*(SORT(.crt*)))
21         *(.text .text.* .gnu.linkonce.t.*)
22         KEEP (*(init))
23         KEEP (*(fini))
24         . = ALIGN(4);
25         _etext = .;
26     } >system_lpddr4_mc_inst
27
28     .gprof : ALIGN(4)
29     {
30         PROVIDE (_s_part_prof = .);
31         KEEP (*(gprof))
32         . = ALIGN(4);
33         PROVIDE (_eprof = .);
34     } >system_lpddr4_mc_inst
35
36     .ctors : ALIGN(4)
37     {
38         _ctors_start = .;
39         KEEP (*(init_array*))
40         KEEP (*(SORT(.ctors.*)))
41         KEEP (*(ctors))
42         . = ALIGN(4);
43         _ctors_end = .;
44     } >system_lpddr4_mc_inst
45

```

Figure 8.11. Update the LPDDR length in linker.ld

- Follow the steps in the [Adding Component to the GSRD](#) section to update the C/C++ project in Propel SDK, regenerate bitstream and program the flash with updated software image.

8.3. Using ECO Editor

During the initial design phase when the bootloader code is being developed, you may need to update the bootloader more frequently. This requires dynamic updating of bootloader file. Typically, the bootloader is loaded into the System Memory with *.mem* file during the bit file generation phase. The Engineering Change Order (ECO) editor tool allows you to update such *c_bootloader.mem* dynamically without requiring the entire Radiant flow to be re-run.

The following example demonstrates how to change/update the *.mem* file dynamically.

- Click on **Tool > ECO Editor** or click on below icon. It opens the ECO Editor windows.

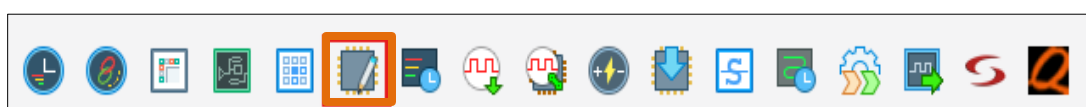
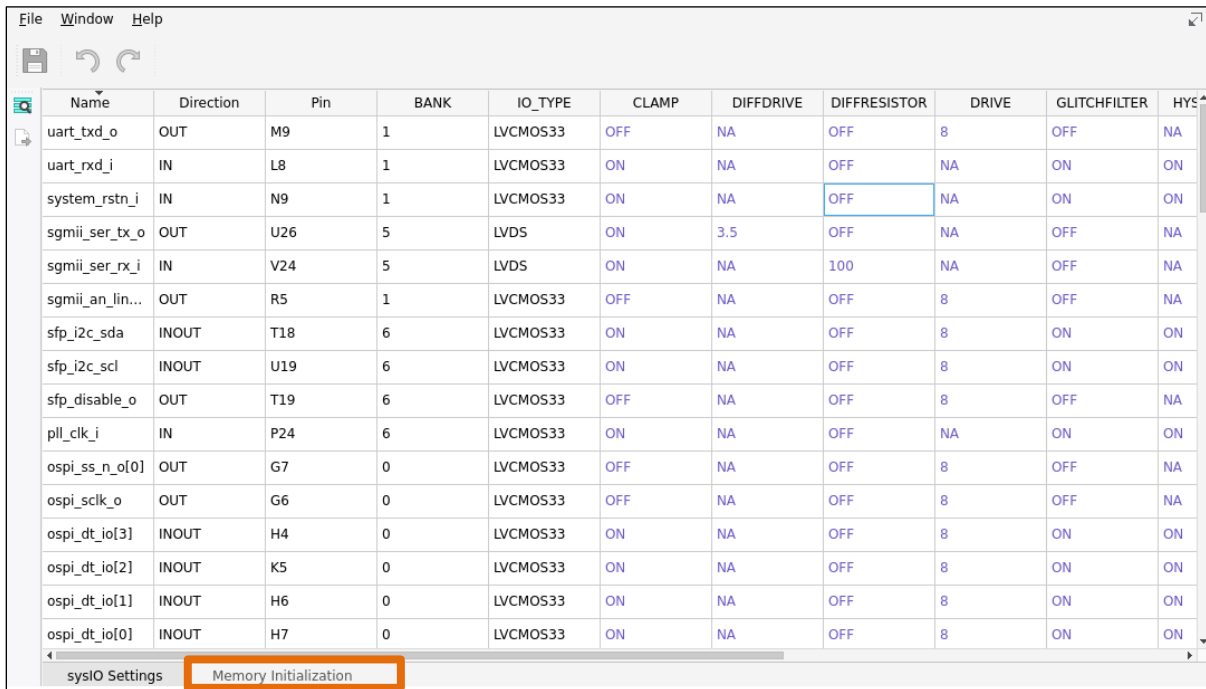


Figure 8.12. ECO Editor Icon in Radiant Software

- Click on **Memory Initialization** tab.

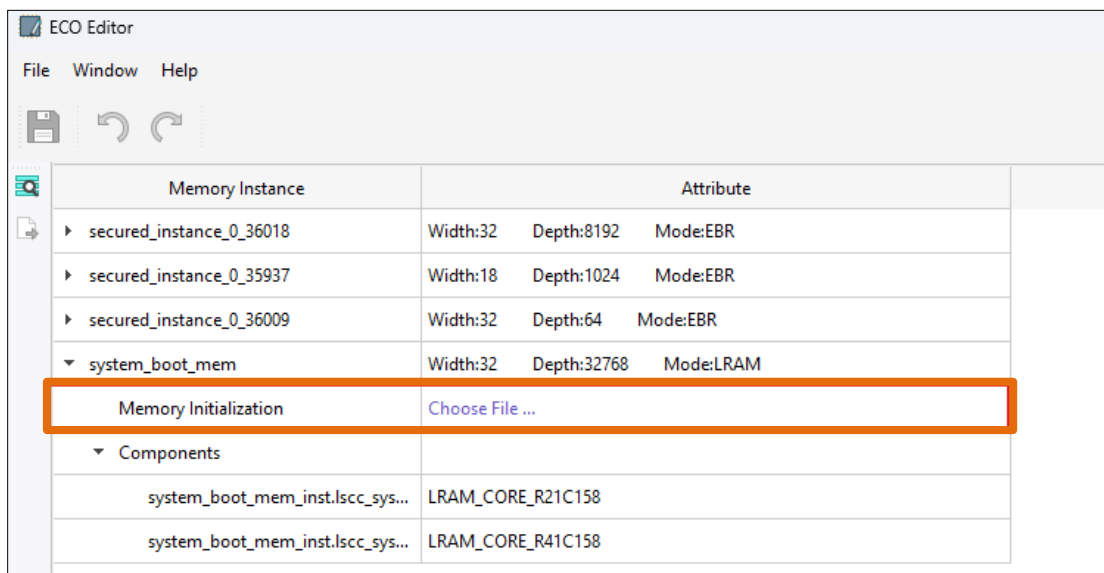


Name	Direction	Pin	BANK	IO_TYPE	CLAMP	DIFFDRIVE	DIFFRESISTOR	DRIVE	GLITCHFILTER	HYS
uart_txd_o	OUT	M9	1	LVC MOS533	OFF	NA	OFF	8	OFF	NA
uart_rxd_i	IN	L8	1	LVC MOS533	ON	NA	OFF	NA	ON	ON
system_rstn_i	IN	N9	1	LVC MOS533	ON	NA	OFF	NA	ON	ON
sgmii_ser_tx_o	OUT	U26	5	LVDS	ON	3.5	OFF	NA	OFF	NA
sgmii_ser_rx_i	IN	V24	5	LVDS	ON	NA	100	NA	OFF	NA
sgmii_an_lin...	OUT	R5	1	LVC MOS533	OFF	NA	OFF	8	OFF	NA
sfp_i2c_sda	INOUT	T18	6	LVC MOS533	ON	NA	OFF	8	ON	ON
sfp_i2c_scl	INOUT	U19	6	LVC MOS533	ON	NA	OFF	8	ON	ON
sfp_disable_o	OUT	T19	6	LVC MOS533	OFF	NA	OFF	8	OFF	NA
pll_clk_i	IN	P24	6	LVC MOS533	ON	NA	OFF	NA	ON	ON
ospi_ss_n_o[0]	OUT	G7	0	LVC MOS533	OFF	NA	OFF	8	OFF	NA
ospi_sclk_o	OUT	G6	0	LVC MOS533	OFF	NA	OFF	8	OFF	NA
ospi_dt_io[3]	INOUT	H4	0	LVC MOS533	ON	NA	OFF	8	ON	ON
ospi_dt_io[2]	INOUT	K5	0	LVC MOS533	ON	NA	OFF	8	ON	ON
ospi_dt_io[1]	INOUT	H6	0	LVC MOS533	ON	NA	OFF	8	ON	ON
ospi_dt_io[0]	INOUT	H7	0	LVC MOS533	ON	NA	OFF	8	ON	ON

sysIO Settings **Memory Initialization**

Figure 8.13. ECO Editor sysIO Settings Tab

- Search for the `system_boot_mem` instance.



Memory Instance	Attribute
▶ secured_instance_0_36018	Width:32 Depth:8192 Mode:EBR
▶ secured_instance_0_35937	Width:18 Depth:1024 Mode:EBR
▶ secured_instance_0_36009	Width:32 Depth:64 Mode:EBR
▼ system_boot_mem	Width:32 Depth:32768 Mode:LRAM
Memory Initialization	Choose File ...
▼ Components	
system_boot_mem_inst.lscs_sys...	LRAM_CORE_R21C158
system_boot_mem_inst.lscs_sys...	LRAM_CORE_R41C158

Figure 8.14. ECO Editor Memory Initialization Tab

- Click on **Choose File ...** to browse for `c_bootloader.mem` file.

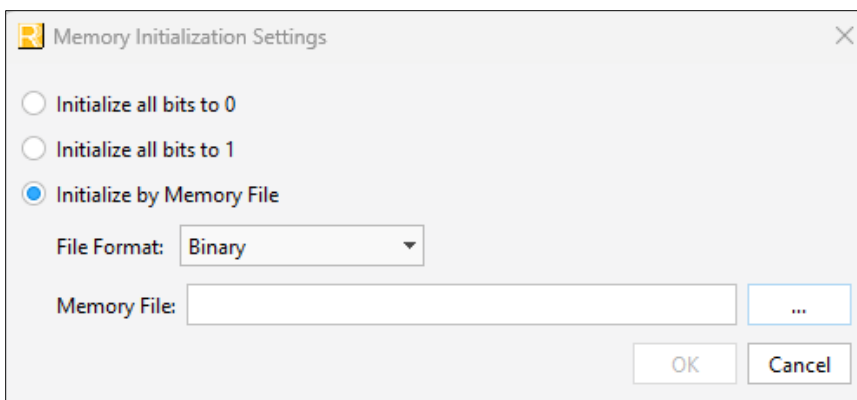


Figure 8.15. Select bootloader in Memory Initialization Settings

- After `c_bootloader.mem` is loaded in the **Memory File** field, change **File format** to **Hexadecimal**. Click **OK**.

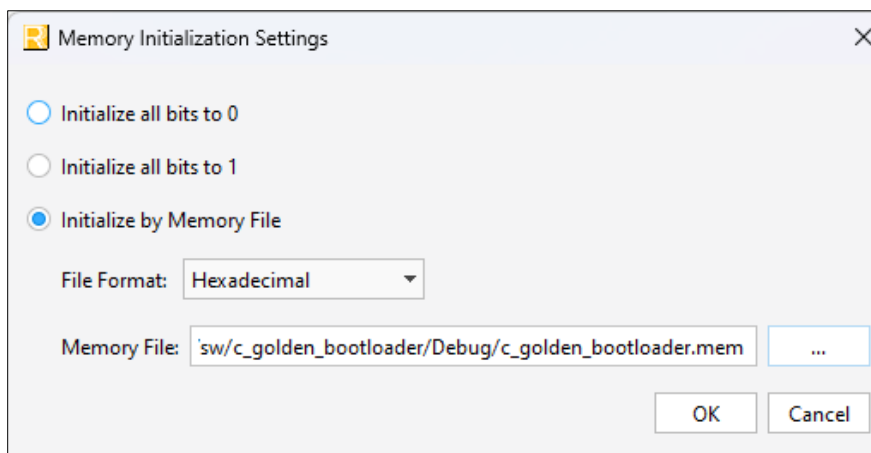


Figure 8.16. Change file format in Memory Initialization Settings

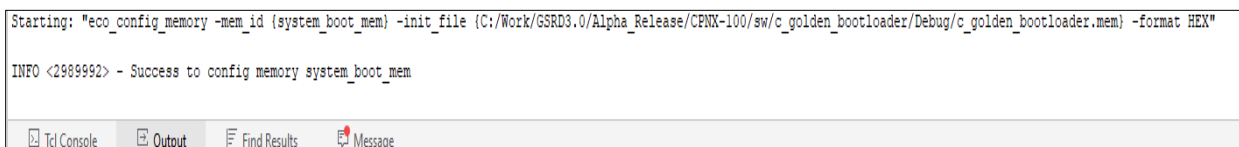


Figure 8.17. Updated System Memory content

- Click **Save** icon and **Save** button to save the ECO changes.

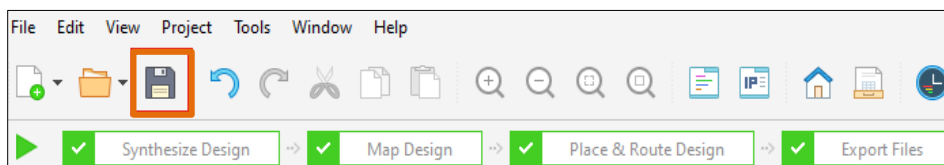


Figure 8.18. Save Icon

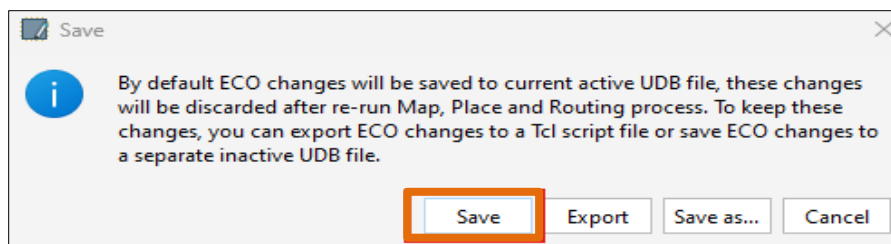


Figure 8.19. Save ECO Changes

7. After saving the project, hit the **Run** icon. The **Post Route Timing Analysis** and **Export Files** phases are re-run.

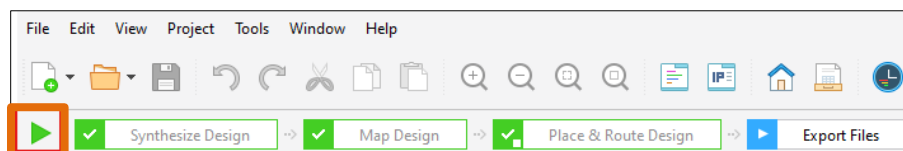


Figure 8.20. Re-run partial Radiant Flow

8. The bitstream is re-generated with updated *.mem* file in the System Memory.

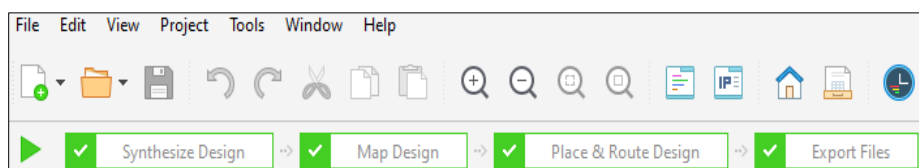


Figure 8.21. Bitstream Generation Flow Completed Successfully



Figure 8.22. Bitstream is Re-Generated

9. Debugging the GSRD

9.1. Debugging Using Reveal Signal

To debug the hardware and software design on the GSRD, the Reveal signals can be added as the trigger to generate waveform to capture the respective signals. Refer to *Chapter 2* and *3* of the [Reveal User Guide for Radiant Software](#) document for the steps on how to run the Reveal Inserter and Reveal Analyzer.

9.2. Debugging Using the OpenOCD Debugger

Software C/C++ project can be debugged by using the GDB OpenOCD debugger embedded inside the Propel SDK. Refer to the *Programming and On-Chip Debugging Flow* section of the [Lattice Propel 2025.1 SDK User Guide \(FPGA-UG-02234\)](#) for the steps on building and loading the symbol file.

9.3. Debugging with Verbosity Level

Refer to the [Appendix B. Enabling Verbosity Level in Software](#) to enable software verbosity level for debugging purposes.

Appendix A. Changing the SPIM Settings in the NV Register

Perform the instructions in this appendix if you are using SPI flash that requires 4-Bytes (32-bit) addressing mode.

To change the SPIM settings, perform the following:

1. 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 do this only once for your board. Subsequent programming does not need to program NV Register 1 again.

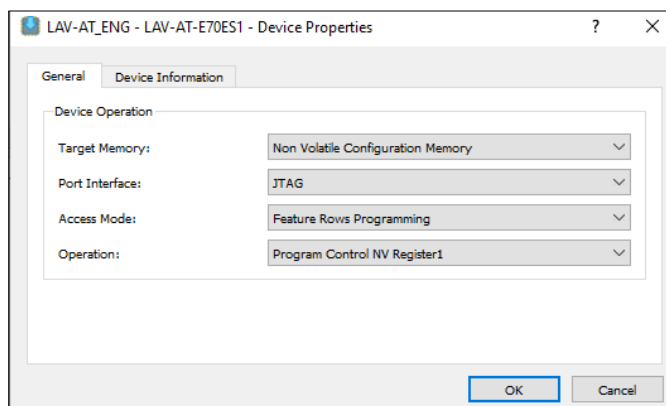


Figure A.1. One-Time Programmable Control NV Register1

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

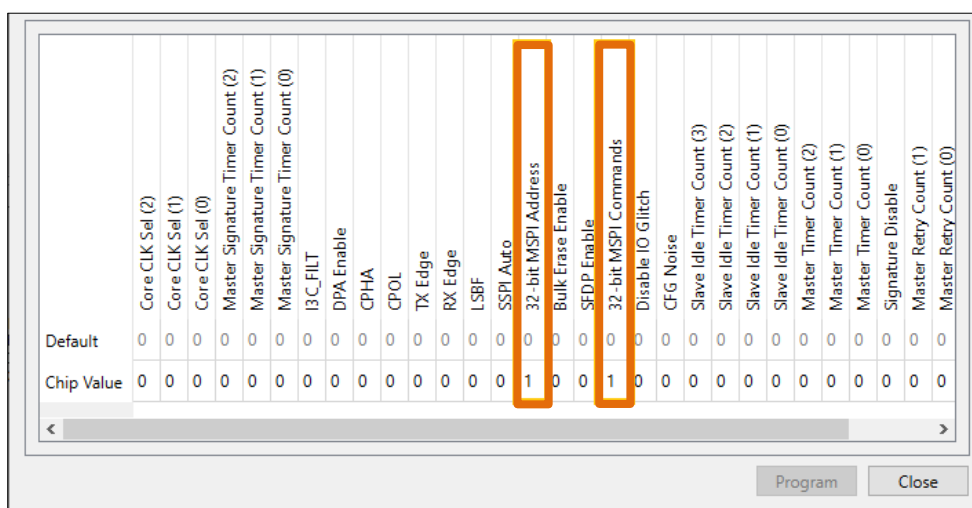


Figure A.2. Settings to Select Chip Value

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

4. Click **Program**.
5. Power cycle the Evaluation board.

Appendix B. Enabling Verbosity Level in Software

Debuglib is added to implement software verbosity level. There are three verbosity levels: DEBUG_ERROR, DEBUG_INIT, and DEBUG_INFO. Debug Build is enabled by default.

Table B.1. Debuglib Verbose Levels

Debug Level	Description
DEBUG_INIT	Used for all initialization printout. Mandatory. Enabled for both Debug and Release build.
DEBUG_INFO	Used for additional information printout for debugging purposes. Enabled for only Debug build.
DEBUG_ERROR	Used for error printout. Enabled for both Debug and Release build.

To enable verbosity in the software, perform the following:

- To enable a logging print, you can call `DEBUG_MSG (PRINTLEVEL, "Messages")`. PRINTLEVEL is the verbosity level selection above. For example, to enable DEBUG_INFO print, you can set the following function:

```
DEBUG_MSG(DEBUG_INFO, "Hello World.");
```

 - For the Debug Build, all three levels of logs are printed.
 - For the Release Build, logging with DEBUG_ERROR and DEBUG_INIT are printed.
- You can enable the Release Build by right-click on `c_primary_app > Build Configurations > Set Active > Release`.

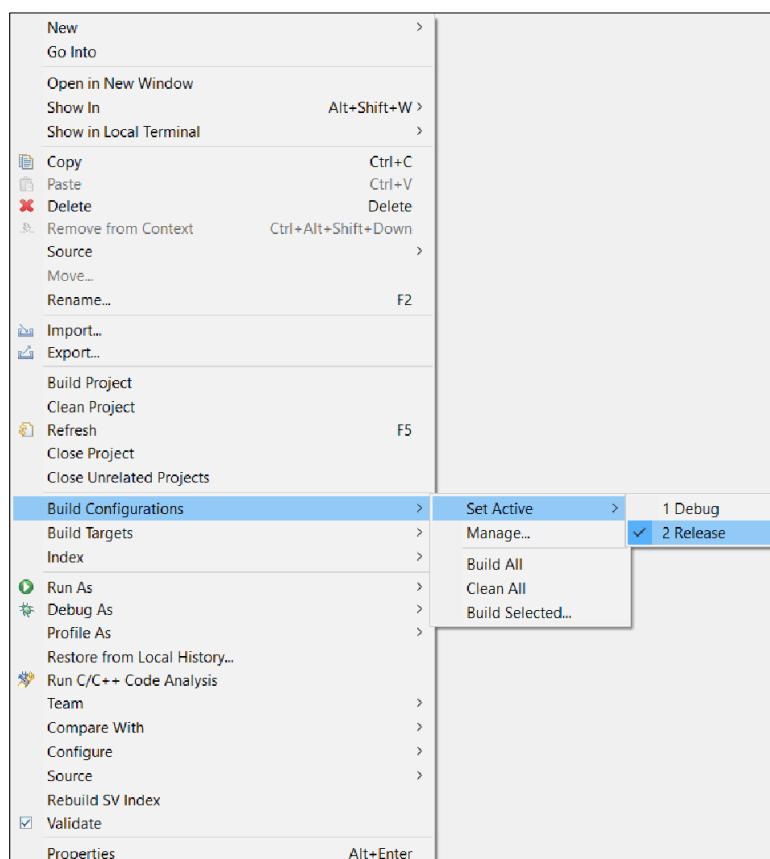


Figure B.1. Set Release Build Configurations

- Set the defined symbols into the toolchain by right-click on **Project > Properties**. In the **C/C++ Build**, select **Settings > GNU RISC-V Cross C Compiler > Preprocessor > Defined symbols (-D) > Add LSCC_RELEASE_BUILD > Apply and Close**.

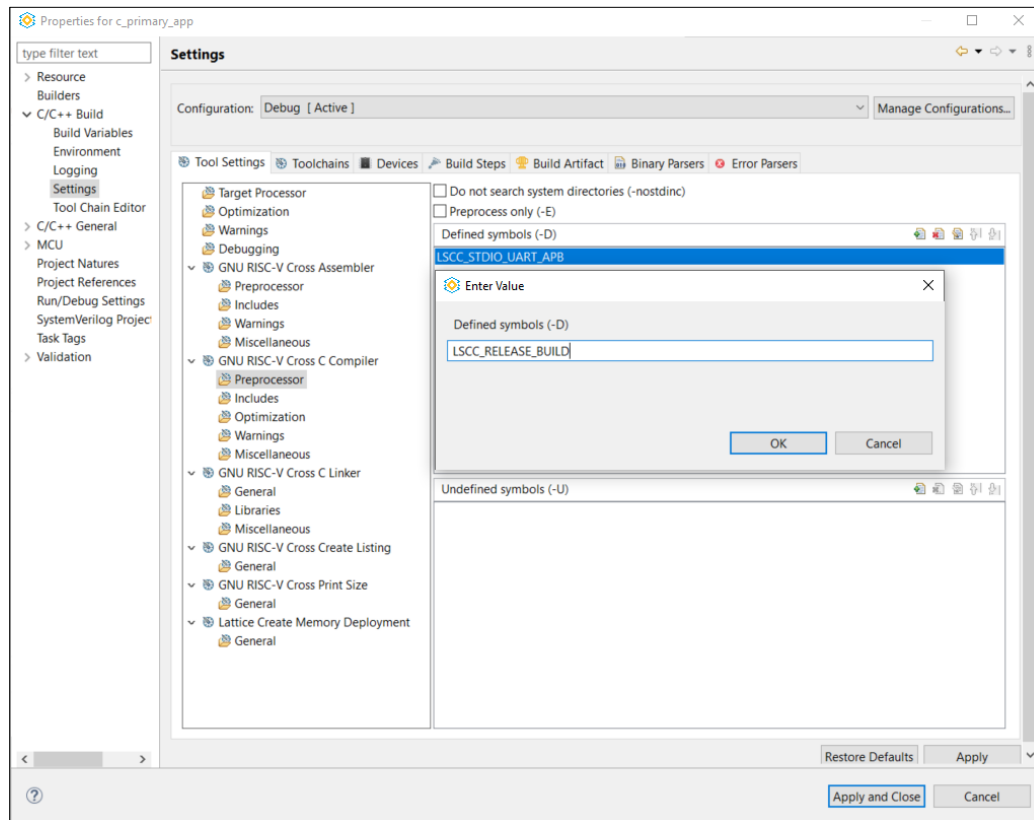


Figure B.2. Add LSCC_RELEASE_BUILD Defined Symbols for Release Build Output

- Right-click on **c_primary_app** and click on **Build Project**.

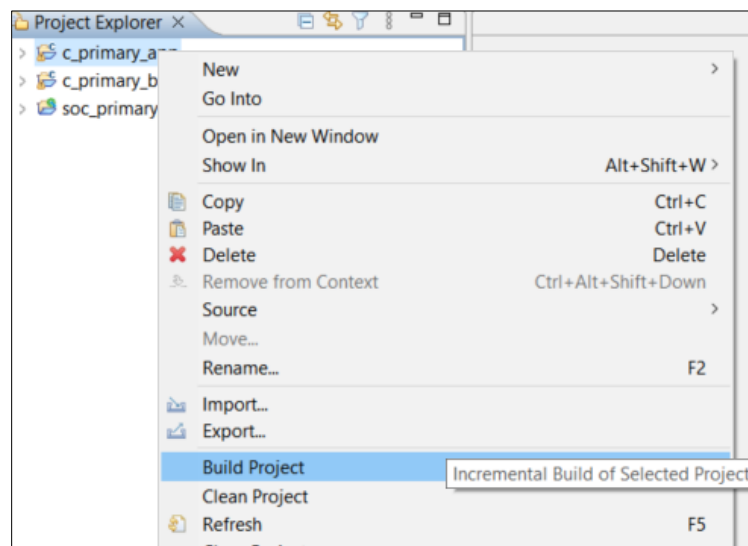


Figure B.3. Build c_primary_app C/C++ Project

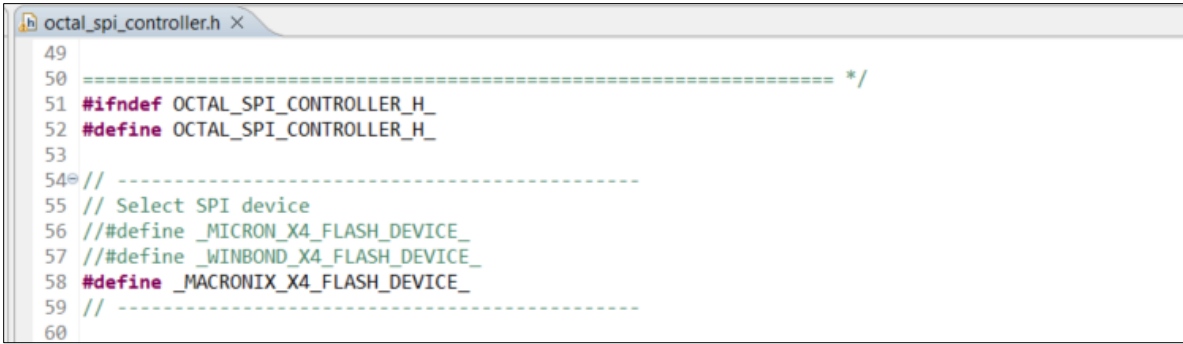
These actions enable verbosity levels **DEBUG_ERROR** and **DEBUG_INIT** in the Release Build.

Appendix C. Using Different SPI Flash Manufacturer in GSRD Bare-metal Bootloader

There are few SPI flash manufacturers supported by Octal SPI 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. Launch the Propel SDK and open the `c_primary_bootloader` or `c_golden_bootloader` project.
2. Change the flash define parameter in `octal_spi_controller.h` to the respective SPI flash model (see [Figure C.1](#)).



```
49
50 ===== */
51 #ifndef OCTAL_SPI_CONTROLLER_H_
52 #define OCTAL_SPI_CONTROLLER_H_
53
54 // -----
55 // Select SPI device
56 // #define _MICRON_X4_FLASH_DEVICE_
57 // #define _WINBOND_X4_FLASH_DEVICE_
58 #define _MACRONIX_X4_FLASH_DEVICE_
59 // -----
60
```

Figure C.1. SPI Flash Manufacturer Changes in `octal_spi_controller.h`

- `_MICRON_X4_FLASH_DEVICE_` = Micron MT25QU128
 - `_MACRONIX_X4_FLASH_DEVICE_` = Macronix MX25L12833F
 - `_WINBOND_X4_FLASH_DEVICE_` = Winbond W25Q512JV
3. Redo the steps in the [Building the Bare-metal Bootloader Using the Lattice Propel SDK \(Primary and Golden\)](#) and [Building the FreeRTOS Application Software using Lattice Propel SDK \(Primary and Golden\)](#) section to update the bootloader and FreeRTOS application software.

Appendix D. Using Octal SPI Controller for Read, Write, and Erase Self-Diagnostic Check (FreeRTOS Application Software)

During the software boot-up in FreeRTOS, Octal SPI Controller performs a self-diagnostic check for read, write, and erase operations (enabled by default).

1. The Octal SPI Controller erases 4 kB first in the SPI flash at the offset 0x2000000.
2. Then, it performs a write of a fixed pattern into the 4 kB region on the same SPI region.
3. Finally, it performs a readback to compare if the data is correct from the write.
4. If you want to disable the self-diagnostic check, comment out the *octal_spi_diag* function call in FreeRTOS application software main.c.

```
spix8_param_init();

if(spix8_flash_ctl_init(octal_spi_c0_inst, SYSTEM_OSPI_FC_INST_OCTAL_SPI_CONTROLLER_AXI4_MAP_BASE_ADDR,
    flash_addr_offset))
{
    octal_spi_c0_inst->init_done = SUCCESS;
    DEBUG_MSG(DEBUG_INIT, "Octal SPI Init Done.\r\n");
}
else
    DEBUG_MSG(DEBUG_INIT, "Octal SPI Init Failed\r\n");

gencmd_flash_set_quad_mode(octal_spi_c0_inst, FLASH_QUADSPI_ENABLE);
/* octal spi self-diagnostic check, comment out to skip */
octal_spi_diag(&cmd_buf, &rsp_buf, FLASH_QUADSPI_ENABLE);
```

Figure D.1. Octal SPI Controller Read, Write and Erase check in FreeRTOS Application Software

```
Octal SPI Init Done.
Index:[0] Data Mismatch! exp_data = beefdead, obs_data = ffffffff
exp_addr = 1d518, obs_addr = 1e584
Index:[1] Data Mismatch! exp_data = beefdeae, obs_data = ffffffff
exp_addr = 1d51c, obs_addr = 1e588
Index:[2] Data Mismatch! exp_data = beefdeaf, obs_data = ffffffff
exp_addr = 1d520, obs_addr = 1e58c
Index:[3] Data Mismatch! exp_data = beefdeb0, obs_data = ffffffff
exp_addr = 1d524, obs_addr = 1e590
Index:[4] Data Mismatch! exp_data = beefdeb1, obs_data = ffffffff
exp_addr = 1d528, obs_addr = 1e594
Index:[5] Data Mismatch! exp_data = beefdeb2, obs_data = ffffffff
exp_addr = 1d52c, obs_addr = 1e598
Index:[6] Data Mismatch! exp_data = beefdeb3, obs_data = ffffffff
exp_addr = 1d530, obs_addr = 1e59c
Index:[7] Data Mismatch! exp_data = beefdeb4, obs_data = ffffffff
exp_addr = 1d534, obs_addr = 1e5a0
Index:[8] Data Mismatch! exp_data = beefdeb5, obs_data = ffffffff
exp_addr = 1d538, obs_addr = 1e5a4
Index:[9] Data Mismatch! exp_data = beefdeb6, obs_data = ffffffff
exp_addr = 1d53c, obs_addr = 1e5a8
Index:[10] Data Mismatch! exp_data = beefdeb7, obs_data = ffffffff
exp_addr = 1d540, obs_addr = 1e5ac
Index:[11] Data Mismatch! exp_data = beefdeb8, obs_data = ffffffff
exp_addr = 1d544, obs_addr = 1e5b0
Index:[12] Data Mismatch! exp_data = beefdeb9, obs_data = ffffffff
exp_addr = 1d548, obs_addr = 1e5b4
Index:[13] Data Mismatch! exp_data = beefdeba, obs_data = ffffffff
exp_addr = 1d54c, obs_addr = 1e5b8
Index:[14] Data Mismatch! exp_data = beefdebb, obs_data = ffffffff
exp_addr = 1d550, obs_addr = 1e5bc
Index:[15] Data Mismatch! exp_data = beefdeb0, obs_data = ffffffff
exp_addr = 1d554, obs_addr = 1e5c0
[test_erase4k_prog_read_random] Failed !
```

Figure D.2. Self-diagnostic Check Failed

```
*****
***      GSRD Primary FreeRTOS on RISC-V CPNX      ***
*****
Octal SPI Init Done.
[test_erase4k_prog_read_random] Passed !
```

Figure D.3. Self-diagnostic Check Passed

Appendix E. Configuring SPI Clock (SCK) Pulse Width

To configure the Octal SPI clock output (spi_sck_o) to operate at a frequency different from the Octal SPI clock input (clk_i), the divider register (sck_rate) must be used.

In the Octal SPI Controller IP user interface, ensure that the **Programmable SCK Divider** option is checked (see Figure E.1).

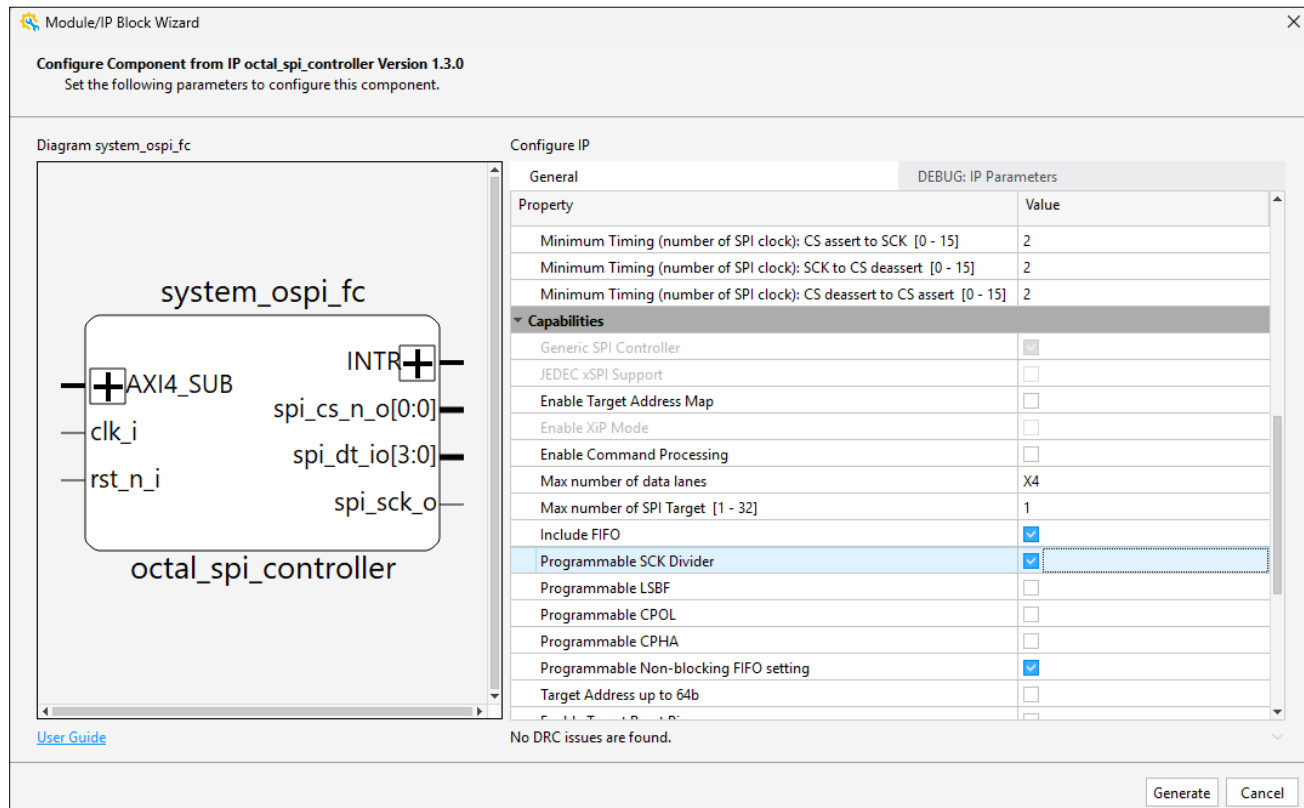


Figure E.1. Programmable SCK Divider Option in the IP User Interface

Refer to the calculations below:

If $sck_rate = 0$:

$spi_sck_o = clk_i$

The SPI clock output directly follows the SPI clock input.

If $sck_rate \neq 0$:

$spi_sck_o = clk_i / (2 \times sck_rate)$

The SPI clock output is derived by dividing the SPI input clock by twice the value of the divider register.

sck_rate parameter description:

0 – Divide by 1 of clk_i

1 – Divide by 2 of clk_i

2 – Divide by 4 of clk_i

3 – Divide by 6 of clk_i

...

Example:

If the clock source of Octal SPI IP is 200 MHz, the `sck_rate` must be a value of 2 to get the SPI output clock (`sck_sck_o`) value of 50 MHz.

Calculation:

SCK Frequency = 200 MHz / (2 × 2^[1]) = 50 MHz

Note [1]: `sck_rate` value of 2 indicates dividing by 4 (2×2) in the calculation.

The `sck_rate` configuration can be found in the `spix8_param_init` function located in the bootloader's `main.c` (see below).

```
void spix8_param_init(void){  
    unsigned int sck_rate;  
    sck_rate = 1;  
  
    octal_spi_c0.init_done          = FAILURE;  
    octal_spi_c0.base_addr         = SYSTEM_OSPI_FC_INST_OCTAL_SPI_CONTROLLER_AXI4_MAP_BASE_ADDR;  
    octal_spi_c0.max_num_lane      = SYSTEM_OSPI_FC_INST_MAX_NUMLANE;  
    octal_spi_c0.sys_clk_freq      = SYSTEM_OSPI_FC_INST_CLKI_FREQ;  
    octal_spi_c0.spi_io_width      = SPIX8_IO_X1;  
}
```

Figure E.2. Octal SPI Controller `sck_rate` Configuration

References

- [CertusPro-NX Versa Evaluation Board User Guide \(FPGA-EB-02053\)](#)
- [APB Interconnect IP User Guide \(FPGA-IPUG-02054\)](#)
- [AXI4 Interconnect IP User Guide \(FPGA-IPUG-02196\)](#)
- [AXI4 to APB Bridge IP User Guide \(FPGA-IPUG-02198\)](#)
- [GPIO IP Core User Guide \(FPGA-IPUG-02076\)](#)
- [Lattice IP Packager 2025.1 \(FPGA-UG-02236\)](#)
- [Lattice Propel 2025.1.1 Builder User Guide \(FPGA-UG-02298\)](#)
- [Lattice Propel 2025.1 SDK User Guide \(FPGA-UG-02234\)](#)
- [Lattice Radiant Timing Constraints Methodology \(FPGA-AN-02059\)](#)
- [LPDDR4 Memory Controller IP Core for Nexus Devices User Guide \(FPGA-IPUG-02127\)](#)
- [Octal SPI Controller IP Core User Guide \(FPGA-IPUG-02248\)](#)
- [RISC-V RX CPU IP User Guide \(FPGA-IPUG-02298\)](#)
- [SGDMA Controller IP Core User Guide \(FPGA-IPUG-02131\)](#)
- [System Memory IP User Guide \(FPGA-IPUG-02073\)](#)
- [Tri-Speed Ethernet IP Core User Guide \(FPGA-IPUG-02084\)](#)
- [UART IP Core User Guide \(FPGA-IPUG-02105\)](#)
- [Lattice Radiant Software 2025.1 User Guide](#)
- [Reveal User Guide for Radiant Software](#)
- [Lattice Propel Design Environment](#) web page
- [Lattice Radiant Software](#) web page
- [Lattice Solutions IP Cores](#) web page
- [Lattice Insights](#) for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

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

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

Revision History

Revision 1.0, October 2025

Section	Change Summary
All	Initial release.



www.latticesemi.com