



Golden System Reference Design and Demo User Guide v1.0 for Certus-NX Devices

Lattice Propel 2025.1.1

Lattice Radiant 2025.1.1

Reference Design

FPGA-RD-02323-1.0

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.....	9
1. Introduction	10
1.1. Quick Facts	10
1.2. Features	11
1.3. Naming Conventions	11
1.3.1. Nomenclature.....	11
1.3.2. Signal Names	11
1.4. Prerequisites	12
1.4.1. Lattice Software Tools Requirements	12
1.4.2. Hardware Requirements	12
2. Functional Description.....	13
2.1. GHRD System Architecture Overview	13
2.2. GHRD Clocking Overview	14
2.3. GHRD Reset Overview	15
2.4. GHRD Interrupts Overview.....	16
2.5. IP Configurations	16
2.5.1. RISC-V MC CPU	16
2.5.2. AHB-Lite Interconnect	19
2.5.3. APB Interconnect.....	19
2.5.4. AHB-Lite to APB Bridge.....	20
2.5.5. AHB-Lite to AXI4 Bridge.....	21
2.5.6. AXI4 to AHB-Lite Bridge.....	21
2.5.7. Octal SPI Controller	22
2.5.8. Tri-Speed Ethernet MAC + RGMII.....	23
2.5.9. Scatter-Gather DMA Controller.....	24
2.5.10 UART	24
2.5.10. GPIO	25
2.5.11. Multi-Boot Configuration Module.....	26
2.5.12. System Memory	26
2.5.13. PLL	27
2.5.14. Reset Modules.....	30
2.5.15. Clock Domain Crossing Module.....	31
2.5.16. AXIS FIFO Module.....	31
2.6. System Level Interfaces.....	33
2.7. SoC Memory/Address Map	33
2.8. Design Constraints	33
2.9. Resource Utilization	34
3. Signal Description	35
4. Software Components.....	36
4.1. Primary and Golden Bootloader	36
4.2. Primary and Golden Application	36
5. Theory of Operation	37
5.1. Boot-Up Sequence	37
5.2. Data Movement	39
6. Running the GSRD Demonstration	40
6.1. Executables	40
6.2. Setting Up the Hardware.....	40
6.3. Setting Up the UART Terminal	42
6.4. Programming Standalone Golden or Primary GSRD Bitstream and Application Software	43
6.5. Programming the Golden, Primary Software, and MCS File	54
7. Compiling and Running the Reference Design	57

7.1.	Building the GHRD SoC project using Lattice Propel SDK and Propel Builder	57
7.2.	Synthesizing the RTL Files and Generating the Bitstream using Lattice Radiant	61
7.3.	Building the Hello World Program Using the Lattice Propel SDK (Optional).....	63
7.4.	Building the Bare-metal Bootloader using Lattice Propel SDK (Primary and Golden)	68
7.5.	Building the Bare-metal Application Software using Lattice Propel SDK (Primary and Golden).....	73
7.6.	Generating the Multi-Boot MCS File	82
8.	Customizing the GSRD	89
8.1.	Adding Component to the GSRD	89
8.1.1.	Hardware Flow	89
8.1.2.	Software Flow.....	90
8.2.	Using the ECO Editor	91
9.	Debugging the GSRD.....	95
9.1.	Debugging Using Reveal Signal	95
9.2.	Debugging Using the OpenOCD Debugger.....	95
9.3.	Debugging with Verbosity Level.....	95
	Appendix A. Enabling Verbosity Level in Software	96
	Appendix B. Using Different SPI Flash Manufacturer in GSRD Bare-metal Bootloader.....	98
	Appendix C. Using the Octal SPI Controller for Read, Write, and Erase Self-Diagnostic Check (Bare-metal Application Software)	99
	Appendix D. Configuring SPI Clock (SCK) Pulse Width	101
	References.....	103
	Technical Support Assistance	104
	Revision History.....	105

Figures

Figure 2.1. GHRD System Architecture	13
Figure 2.2. GHRD Clocking Overview	14
Figure 2.3. GHRD Reset Overview	15
Figure 2.4. RISC-V MC CPU IP Configuration – General	17
Figure 2.5. RISC-V MC CPU IP Configuration – Debug	17
Figure 2.6. RISC-V MC CPU IP Configuration – Interrupt	18
Figure 2.7. RISC-V MC CPU IP Configuration – Buses.....	18
Figure 2.8. AHB-Lite Interconnect IP Configuration – General	19
Figure 2.9. AHB-Lite Interconnect IP Configuration – Max Burst Size Settings	19
Figure 2.10. APB Interconnect IP Configuration – General.....	20
Figure 2.11. AHB-L to APB Bridge IP Configuration – General.....	20
Figure 2.12. AHB-Lite to AXI4 Bridge IP Configuration – General.....	21
Figure 2.13. AXI4 to AHB-Lite Bridge IP Configuration – General.....	21
Figure 2.14. Octal SPI Controller IP Configuration – General	22
Figure 2.15. Octal SPI Controller IP Configuration – General	22
Figure 2.16. Octal SPI Controller IP Configuration – General	23
Figure 2.17. TSE IP Configuration	23
Figure 2.18. SGDMA Controller IP Configuration.....	24
Figure 2.19. UART IP Configuration	25
Figure 2.20. GPIO IP Configuration	25
Figure 2.21. Multi-Boot Module Configuration	26
Figure 2.22. System Memory IP Configuration – General	26
Figure 2.23. System Memory IP Configuration – Port S0 Settings.....	27
Figure 2.24. System Memory IP Configuration – Port S1 Settings.....	27
Figure 2.25. PLL IP Configuration – General for system_pll.....	28
Figure 2.26. PLL IP Configuration – General for system_pll.....	28
Figure 2.27. PLL IP Configuration – General for system_pll.....	29
Figure 2.28. PLL IP Configuration – Optional Ports for system_pll	29
Figure 2.29. PLL IP Configuration – General for eth_pll.....	30
Figure 2.30. PLL IP Configuration – General for eth_pll.....	30
Figure 2.31. Reset Module Configuration – General	31
Figure 2.32. Clock Domain Crossing Module Configuration – General.....	31
Figure 2.33. AXIS RX FIFO Module Configuration – General for tsemac_rx_axis_fifo	32
Figure 2.34. AXIS FIFO Module Configuration – General for tsemac_tx_axis_fifo	32
Figure 2.35. LFD2NX-40 Device GHRD Approximate Resource Utilization	34
Figure 5.1. GSRD Boot-Up Sequence	38
Figure 5.2. Ethernet Data RX Flow	39
Figure 5.3. Ethernet Data TX Flow	39
Figure 6.1. Certus-NX Versa Evaluation Board	41
Figure 6.2. Connections, Jumpers and Buttons needed for Demonstration	41
Figure 6.3. UART Terminal Icon on Propel SDK Window	42
Figure 6.4. UART Launch Terminal Window	43
Figure 6.5. Device Manager Window on PC	43
Figure 6.6. Launch Radiant Programmer from Windows Start.....	44
Figure 6.7. Radiant Programmer Start Window	44
Figure 6.8. Radiant Programmer. xcf Window	44
Figure 6.9. Scan Device Icon on Radiant Programmer.....	45
Figure 6.10. Select Device for Programming	45
Figure 6.11. Select the Target Memory for Programming – SPI Flash	45
Figure 6.12. Device Properties to Erase the Micron SPI Flash	46
Figure 6.13. Program Button to Program the SPI Flash	46
Figure 6.14. Output After Erase All.....	47

Figure 6.15. Erase Only Operation for SRAM Programming.....	47
Figure 6.16. Device Properties to Program the Micron SPI Flash.....	48
Figure 6.17. Cable Settings for Device Programming	49
Figure 6.18. Radiant Programmer Console Output after Programming the SPI Flash.....	49
Figure 6.19. Device Properties to Program the FPGA Bitstream in SRAM.....	50
Figure 6.20. Radiant Programmer Console Output after Bitstream is Programmed.....	50
Figure 6.21. SW3 Reset Button and SW4 PROGRAMN Button	51
Figure 6.22. LED Status	52
Figure 6.23. Golden GSRD - Output on UART Terminal for Bootloader and Bare-metal Application Software Starts.....	52
Figure 6.24. Golden GSRD - Output on UART Terminal for Bare-metal Application Software Running.....	53
Figure 6.25. Primary GSRD Bootloader– Output on UART Terminal	53
Figure 6.26. Primary GSRD Bare-metal Application Software– Output on UART Terminal.....	54
Figure 6.27. Device Properties Window to Setup MCS Programming File	55
Figure 6.28. UART Terminal Output after Power-Cycling Board with MCS and Binaries Programmed	56
Figure 6.29. Switches to Golden GSRD Upon SW4 PROGRAMN Button.....	56
Figure 7.1. Propel SDK Launcher.....	57
Figure 7.2. Provide Name for the Workspace Directory.....	58
Figure 7.3. Creating Lattice SoC Design Project.....	58
Figure 7.4. SoC Project Window	59
Figure 7.5. Launched SoC in Propel Builder.....	59
Figure 7.6. Validate Design in Propel Builder	60
Figure 7.7. TCL Console Output after Validating Design.....	60
Figure 7.8. Generate in Propel Builder	60
Figure 7.9. TCL Console Printout after Generating Design	60
Figure 7.10. sys_env.xml File Created	61
Figure 7.11. Run Radiant Icon.....	61
Figure 7.12. Lattice Radiant Window.....	61
Figure 7.13. Lattice Radiant Window.....	62
Figure 7.14. Strategy Used for Certus-NX GSRD Testing	62
Figure 7.15. Generating the Bit File.....	63
Figure 7.16. Successful Radiant Flow and Bitstream Generation.....	63
Figure 7.17. Creating Lattice C/C++ Project for Hello World.....	64
Figure 7.18. Hello World, C/C++ Selection.....	65
Figure 7.19. C/C++ Lattice Toolchain Setting.....	66
Figure 7.20. Hello World C Project Created.....	66
Figure 7.21. Build Hello World Project	67
Figure 7.22. Hello World, Project Build Console Output	67
Figure 7.23. Hello World C Program	68
Figure 7.24. Creating Lattice C/C++ Project for Bootloader	68
Figure 7.25. Bootloader C/C++ Selection.....	69
Figure 7.26. C/C++ Lattice Toolchain Setting.....	70
Figure 7.27. Bootloader C Project Created	70
Figure 7.28. Primary Build Define – Set _PRIMARY_BUILD_ for Primary Build in main.c.....	71
Figure 7.29. Build Bootloader Project.....	71
Figure 7.30. Bootloader Build Project Console Output.....	71
Figure 7.31. Bootloader Binary Created	72
Figure 7.32. Bootloader Boots Up without Bare-metal Application.....	72
Figure 7.33. Golden Build Define – Set _GOLDEN_BUILD_ for Golden Build in main.c.....	72
Figure 7.34. Creating C/C++ Project for bare-metal application software	73
Figure 7.35. FreeRTOS C/C++ Selection	73
Figure 7.36. Bare-metal Application Software Project Created.....	74
Figure 7.37. Copy the CRC and Add Files	74
Figure 7.38. Paste in Bare-metal Application C Project.....	75
Figure 7.39. Copied Text Files	75

Figure 7.40. Primary Build Define – Set <code>_PRIMARY_BUILD_</code> for Primary Build in <code>main.c</code>	75
Figure 7.41. Update <code>crc_add_debug.txt</code>	76
Figure 7.42. Open <code>Linker.Ld</code> File	76
Figure 7.43. Update <code>Linker.Ld</code> File.....	77
Figure 7.44. Workspace	77
Figure 7.45. Properties	78
Figure 7.46. Set Release as Active Configuration	79
Figure 7.47. Adding Post Build Step for Bare-metal Application CRC Binary Append (Windows).....	80
Figure 7.48. Adding Post Build Step for Bare-metal Application CRC Binary Append (Linux)	80
Figure 7.49. Update the CRC script to match Linux path format.....	81
Figure 7.50. Build <code>c_primary_app</code> C/C++ Project.....	81
Figure 7.51. Bare-metal Application Build Project Console Output	81
Figure 7.52. Bare-metal App Binaries Created with CRC.....	82
Figure 7.53. Golden Build Define – Set <code>_GOLDEN_BUILD_</code> for Golden Build in <code>main.c</code>	82
Figure 7.54. Launch Radiant Programmer from Windows Start.....	83
Figure 7.55. Radiant Programmer Getting Started Window	83
Figure 7.56. Error if No Board is Connected	83
Figure 7.57. Open Deployment Tool from Radiant Programmer	84
Figure 7.58. Deployment Tool Start Window	84
Figure 7.59. Options for Creating New Deployment	84
Figure 7.60. External Memory Step 1 of 4: Select Input Files.....	85
Figure 7.61. External Memory Step 2 of 4: Select Options.....	85
Figure 7.62. External Memory Step 2 of 4: Multiple Boot	86
Figure 7.63. External Memory Step 3 of 4: Select Output File(s)	87
Figure 7.64. External Memory Step 4 of 4: General Development.....	88
Figure 7.65. MCS File Generated Successfully	88
Figure 8.1. Bootloader File Updated in the System Memory	90
Figure 8.2. New System env Error	91
Figure 8.3. ECO Editor Icon in Radiant Software	91
Figure 8.4. ECO Editor sys/I/O Settings Tab	92
Figure 8.5. ECO Editor Memory Initialization Tab	92
Figure 8.6. Select bootloader in Memory Initialization Settings	93
Figure 8.7. Change File Format in Memory Initialization Settings.....	93
Figure 8.8. Updated System Memory Content	93
Figure 8.9. Save Icon.....	93
Figure 8.10. Save ECO Changes.....	94
Figure 8.11. Re-run Partial Radiant Flow	94
Figure 8.12. Bitstream Generation Flow Completed Successfully.....	94
Figure 8.13. Bitstream is Re-generated	94
Figure A.1. Set Release Build Configurations.....	96
Figure B.1. SPI Flash Manufacturer Changes in <code>octal_spi_controller.h</code>	98
Figure C.1. Octal SPI Controller Read, Write and Erase Check-In Bare-metal Application Software.....	99
Figure C.2. Self-Diagnostic Check Failed	99
Figure C.3. Self-Diagnostic Check Passed.....	100

Tables

Table 1.1. Summary of the System	10
Table 1.2. List of Hardware Required by GSRD.....	12
Table 2.1. IP Versions.....	13
Table 2.2. GHRD Clocking Overview	14
Table 2.3. Reset Scheme.....	16
Table 2.4. GHRD Interrupt Overview	16
Table 2.5. System Level Interfaces	33
Table 2.6. GHRD Address Map.....	33
Table 2.7. Design Constraints	33
Table 2.8. GSRD Total Approximate Resource Utilization	34
Table 6.1. Executable Files for Micron Flash	40
Table 7.1. List of Actions and Expected Outputs	57
Table A.1. Debuglib Verbose Levels.....	96

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AHBL	Advanced High-performance Bus-Lite
APB	Advanced Peripheral Bus
API	Application Programming Interface
AXI	Advanced eXtensible Interface
AXI4-Lite	Advanced eXtensible Interface-Lite
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DDR	Double Data Rate
DDR3	Double Data Rate Generation 3
FIFO	First-In-First-Out
GHRD	Golden Hardware Reference Design
GPIO	General Purpose Input/Output
GSRD	Golden System Reference Design
ISR	Interrupt Service Routines
LMMI	Lattice Memory Mapped Interface
MDIO	Management Data Input/Output
OPN	Ordering Part Number
PLL	Phase-Locked Loop
QSPI	Quad Serial Peripheral Interface
RGMII	Reduced Gigabit Media Independent Interface
RISC-V	Reduced Instruction Set Computer-V
RTL	Register Transfer Level
SGDMA	Scatter-Gather Direct Memory Access
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, Watchdog Timer, Tri-Speed Ethernet (TSE), Octal SPI Controller and Scatter-Gather DMA (SGDMA) Controller. All these building blocks are connected through industry standard protocols such as AHB-Lite, 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. Bootloader is built on RISC-V MC 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 Certus™-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	Certus-NX
	SoC Version	1.0
FPGA Device(s)	Targeted Board	Certus-NX Versa Board OPN: LFD2NX-VERSA-EVN
	Targeted Device	LFD2NX-40-8BG256C
	Supported User Interface	AHB-Lite, 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:

- FPGA device supported in this document is LFD2NX-40
- RISC-V MC CPU, SGDMA Controller, TSE IP, and Octal SPI Controller over AHB-Lite and AXI4 Interfaces
- 100 Mbps Ethernet throughput through RGMII support at 25 MHz
- Low-speed peripherals like GPIO and UART
- Primary and Golden bootloader and bare-metal application software
- Application software CRC check by function implemented in RISC-V MC bootloader code
- FPGA bitstream CRC check done by FPGA Configuration Engine
- Bare-metal application software is run on System Memory internal to FPGA
- Manual and Automatic Multi-Boot capability

1.3. Naming Conventions

1.3.1. Nomenclature

The following are the nomenclature used in this document:

- Boot Up – Process of starting the RISC-V MC CPU, loading the bare-metal application software from external SPI Flash into the System Memory and executing the application.
- Bootloader – Code that initializes and configures various peripherals and loads the bare-metal application software into System 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.
- Bare-metal application software – Loaded into System Memory and executed by RISC-V CPU at the end of boot up process.
- SPI Flash – Non-volatile external memory that stores the bitstreams, bare-metal application software and multi-boot MCS bitstream.

1.3.2. 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.

1.4. Prerequisites

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

1.4.1. 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 Certus-NX Golden System Reference Design
 - Download here: [Downloadable Software tab in the GHRD/GSRD Reference Design](#)

1.4.2. 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 Certus-NX Versa Evaluation Board OPN: LFD2NX-VERSA-EVN	1	Certus-NX Versa Evaluation Board web page The Evaluation Board is available in two configurations: OPN: LFD2NX-VERSA-EVN (with Ethernet PHY) OPN: LFD2NX-VERSA-B-EVN (without Ethernet PHY) The Evaluation Board with Ethernet PHY (LFD2NX-VERSA-EVN) is required for this GSRD to operate correctly.
2	Mini-USB Type-A UART cable for programming bitstream, firmware and proper terminal prints	1	Included in Certus-NX Versa Board Evaluation Kit
3	Cat6 RJ45 Ethernet cable to connect Certus-NX board to the Host PC	1	Not included in Certus-NX Versa Board Evaluation Kit.
4	12 V power adapter for board power	1	Included in Certus-NX Versa Board Evaluation Kit

The GSRD/GHRD SoC architecture comprises of a RISC-V MC CPU, SGDMA Controller, Octal SPI Controller, and 100 Mbps TSE IP, interconnected through a combination of high-speed and low-speed bus fabrics such as AHB Lite Interconnect, and APB Interconnect. This architecture enables seamless communication and data exchange between the components, facilitating efficient operation and system performance.

The diagram illustrates the system architecture of the RISC-V MC CPU. Key components and their interconnections include:

- RJ-45**: Connected to the **On-board Eth PHY**.
- On-board Eth PHY**: Interfaces with the **TSE MAC** block via **RGMII** and **MDIO** signals.
- TSE MAC**: Contains **AXI-TX** and **AXI-RX** interfaces, connected to **AXI-S RX FIFO** and **AXI-S TX FIFO** respectively.
- AXI-S RX/TX FIFO**: 8-bit buffers connected to the **SGDMA** block.
- SGDMA**: Contains **AXI-RX**, **AXI-TX**, **AXI-BD**, and **AXI-MM** interfaces, connected to the **APB IC**.
- APB IC**: Manages **Multi Boot**, **GPIO**, **UART**, and **WDT** peripherals. It is connected to the **AHBL to APB Bridge**.
- AHBL to APB Bridge**: 32-bit bridge between the **APB IC** and the **AHB-L IC**.
- AHB-L IC**: Manages **AXI4 to AHB** bridges, **AXI4 OSPI**, and **AHB-L System Memory**.
- AXI4 to AHB**: 32-bit bridges connecting the **AXI4 OSPI** to the **AHB-L IC**.
- AXI4 OSPI**: 32-bit interface connected to the **Ext SPI Flash** via a 4-bit **spi_clk** signal.
- AHB-L System Memory**: 32-bit memory block connected to the **AHB-L IC**.
- RISC-V MC CPU**: The central processing unit, connected to the **AHB-L IC** via **AHBL-DS** and **AHBL-IS** signals.
- Clocks**: **system_clk** is distributed throughout the system. **perip_clk** is used for peripheral clocks.

Legend:
 Manager i/f \longleftrightarrow Subordinate i/f

The design includes the following components:

Soft IP	IP Version
RISC-V MC CPU	2.8.1
System Memory	2.4.0
Octal SPI Controller	1.3.0
General Purpose I/O	1.8.0
UART	1.4.0
AHB Lite Interconnect	1.4.0
APB Interconnect	1.3.0
AHB-Lite to AXI4 Bridge	1.3.0
AHB-Lite to APB Bridge	1.1.2
AXI4 to AHB-Lite Bridge	1.4.0
Tri-Speed Ethernet	2.1.0

Soft IP	IP Version
Scatter-Gather DMA	2.5.0
Multi-Boot Configuration	1.0.0
Watchdog Timer	1.6.0
PLL	1.9.1
Reset Modules	1.0.0
Clock Domain Crossing Modules	1.0.0
AXIS FIFO	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

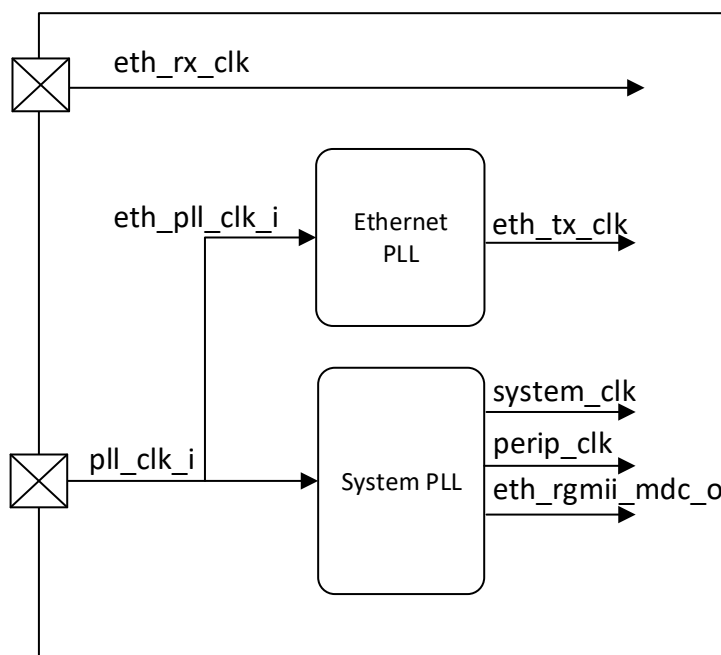


Figure 2.2. GHRD Clocking Overview

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

Table 2.2. GHRD Clocking Overview

Sr No	Clock Name	Clock Freq	Clock Source	Destination
1	pll_clk_i	25 MHz	On board oscillator X2	System and Eth PLL
2	system_clk	90 MHz	system_clk_pll[CLKOP]	RISC-V CPU, SGDMA Controller, AHBL-IC, System Memory, AXI4 to AHBL Bridge, AHBL to AXI4 Bridge and Octal SPI Controller
3	perip_clk	50 MHz	system_clk_pll[CLKOS]	AHBL to APB Bridge, APB IC, UART, WDT, GPIO, SGDMA Controller, TSE IP and Dual Boot IP
4	eth_rgmii_mdc_o	10 MHz	system_clk_pll[CLKOS2]	TSE MDIO and RGMII PHY MDIO
5	eth_tx_clk	25 MHz	eth_pll[CLKOP]	TSE TX path, External TX CDC FIFO

Sr No	Clock Name	Clock Freq	Clock Source	Destination
6	eth_rx_clk	25 MHz	On board RGMII PHY	TSE RX path, External RX CDC FIFO

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

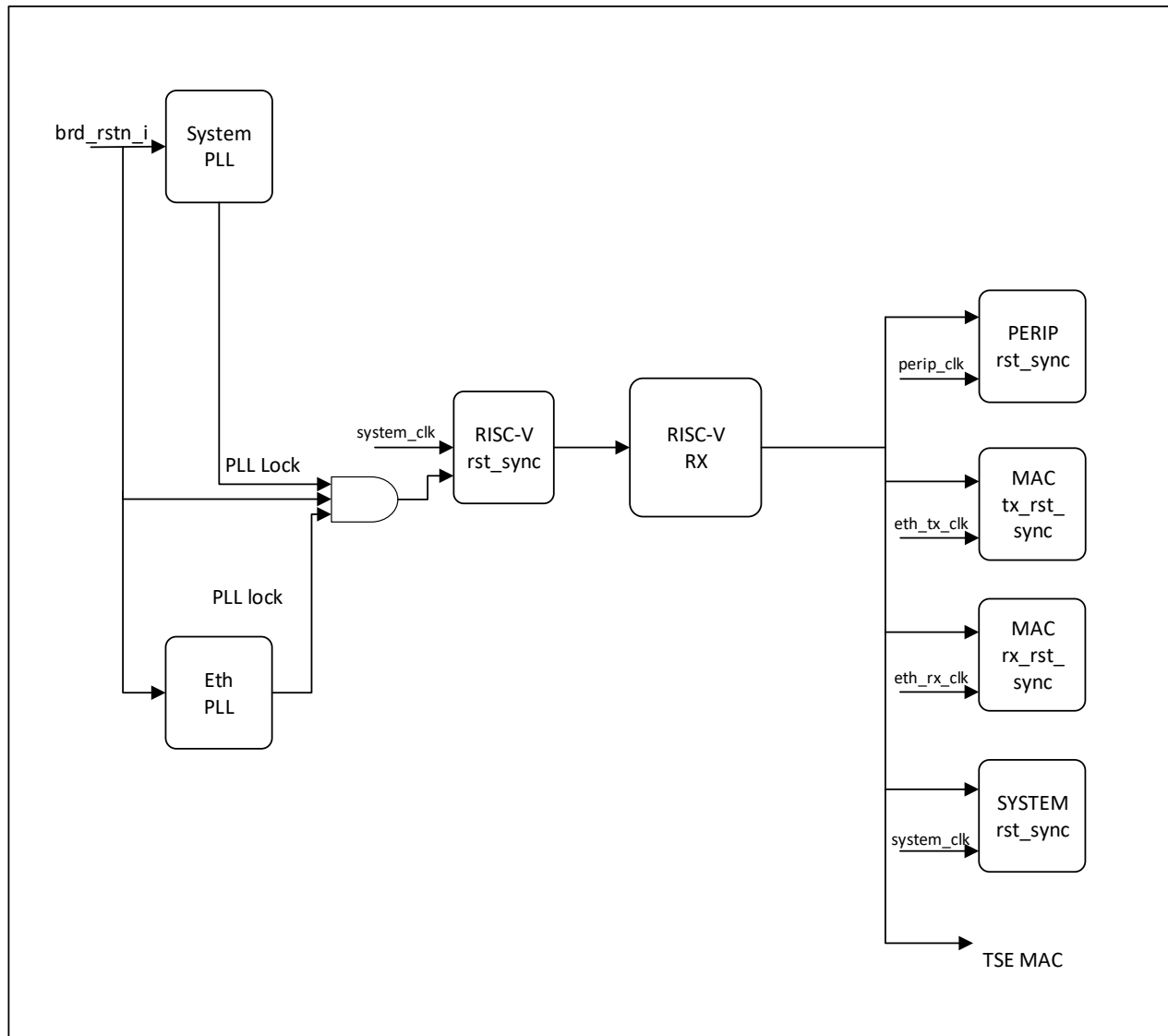


Figure 2.3. GHRD Reset Overview

Table 2.3. Reset Scheme

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, SGDMA Controller, TSE IP, 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_tx_rstn	CPU output reset pin	AXI Stream -TX FIFO reset input pint	External Stream FIFO for CDC.
mac_rx_rstn	CPU output reset pin	AXI Stream RX- FIFO reset input pint	External Stream FIFO for CDC.

2.4. GHRD Interrupts Overview

Table 2.4. GHRD Interrupt Overview

Sr No	RISC-V CPU IRQ Line	Interrupt Source
1	IRQ_0	UART IRQ
2	IRQ_1	WDT IRQ
3	IRQ_2	SGDMA Controller MM2S IRQ
4	IRQ_3	SGDMA Controller S2MM IRQ
5	IRQ_4	TSE MAC IRQ
6	IRQ_5	OSPI IRQ
7	IRQ_6	GPIO IRQ

For more information about the platform level interrupt controller information, refer to the Platform Level Interrupt Controller section, refer to [RISC-V MC CPU IP User Guide \(FPGA-IPUG-02278\)](#).

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 MC CPU

The RISC-V MC CPU IP has AHB-L based instruction and data ports. The instruction ports are connected to the memory that contains the bootloader software and bare-metal 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 MC CPU \(Micro-Controller\) IP Module User Guide \(FPGA-IPUG-02278\)](#).

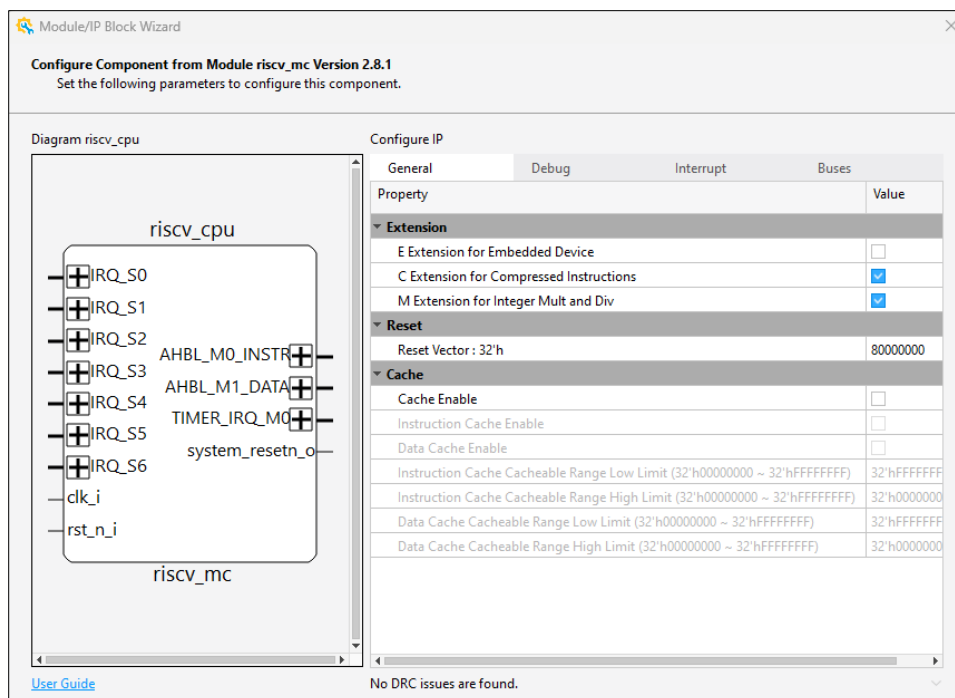


Figure 2.4. RISC-V MC CPU IP Configuration – General

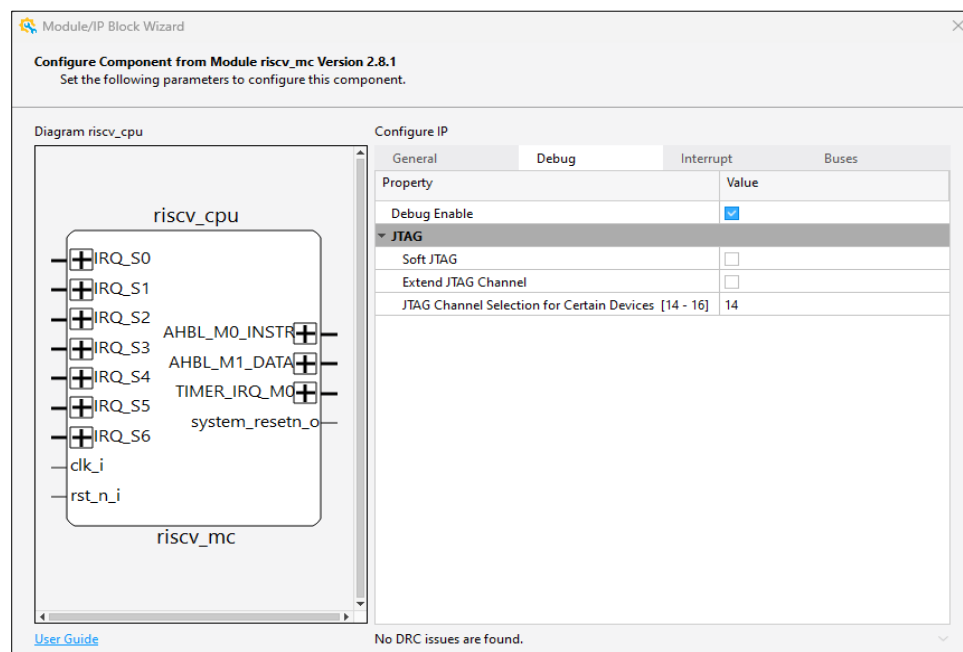


Figure 2.5. RISC-V MC CPU IP Configuration – Debug

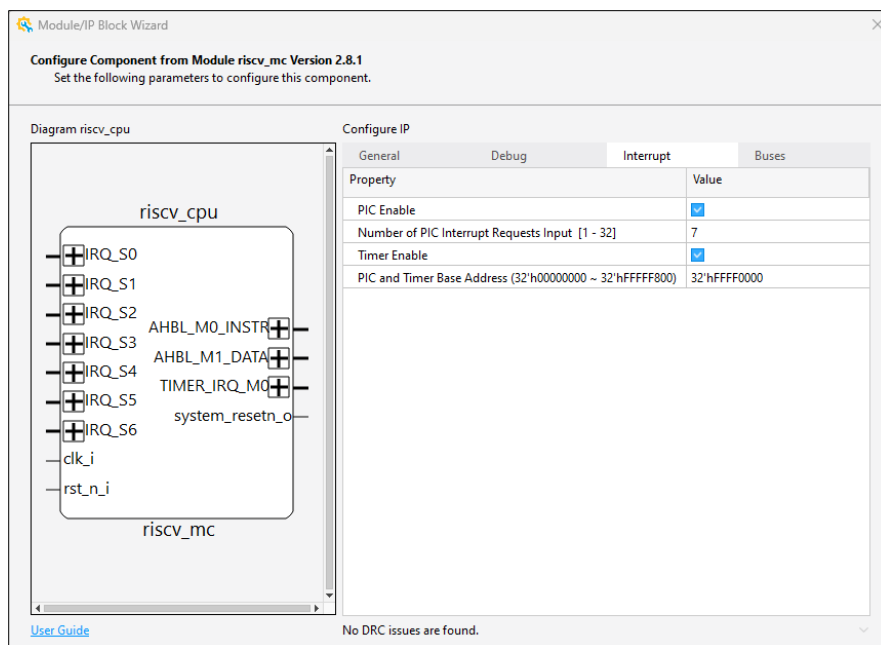


Figure 2.6. RISC-V MC CPU IP Configuration – Interrupt

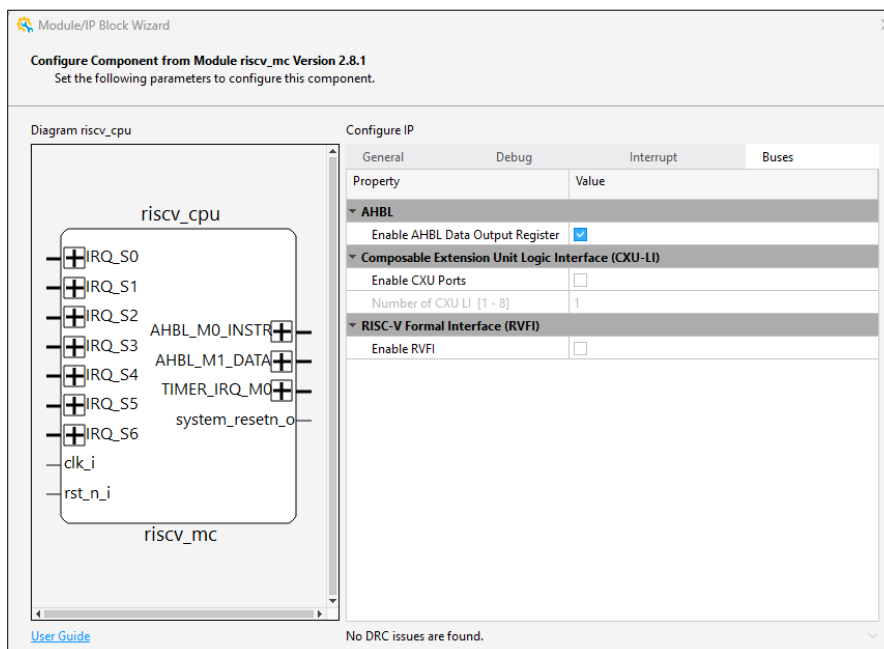


Figure 2.7. RISC-V MC CPU IP Configuration – Buses

2.5.2. AHB-Lite Interconnect

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

For more information about the IP core, refer to [AHB-Lite Interconnect IP User Guide \(FPGA-IPUG-02051\)](#).

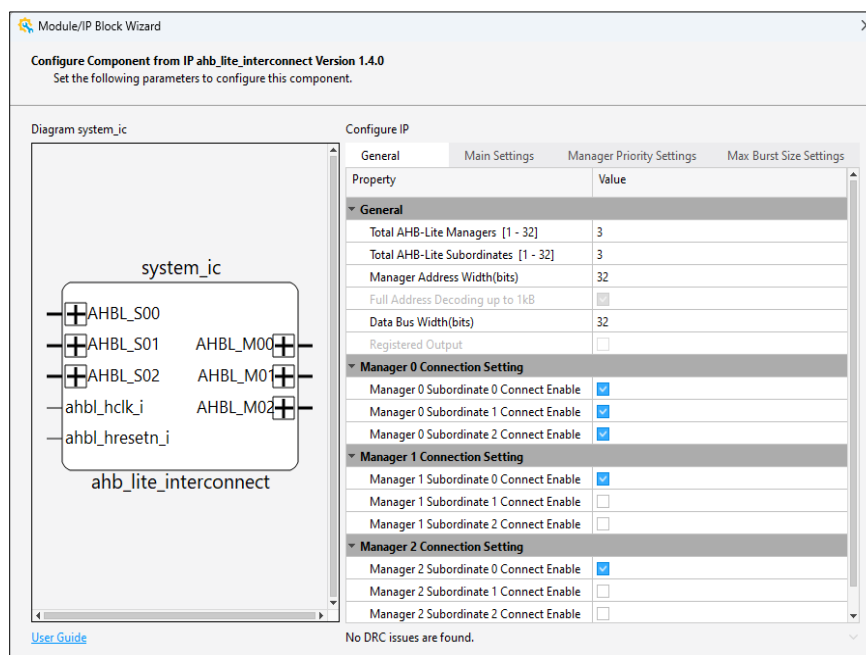


Figure 2.8. AHB-Lite Interconnect IP Configuration – General

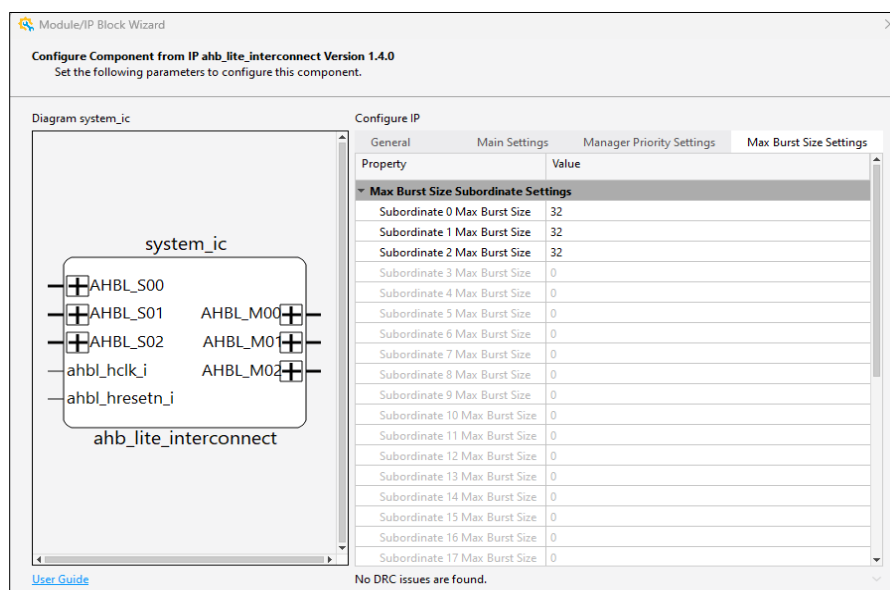


Figure 2.9. AHB-Lite Interconnect IP Configuration – Max Burst Size Settings

2.5.3. APB Interconnect

The Advanced Peripheral Bus (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 \(FPGA-IPUG-02054\)](#).

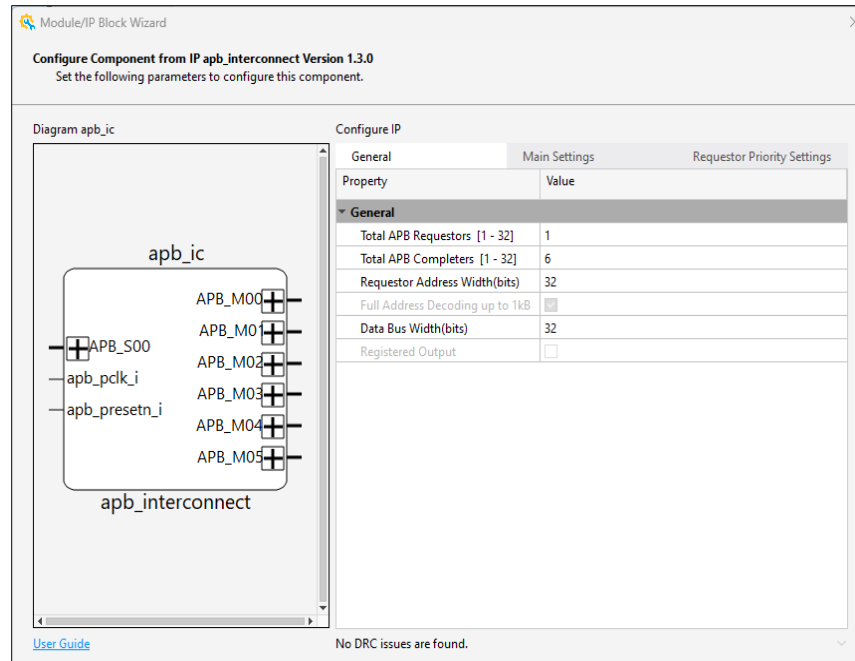


Figure 2.10. APB Interconnect IP Configuration – General

2.5.4. AHB-Lite to APB Bridge

The AHB-Lite to APB Bridge IP is used to convert the AHB-Lite bus transaction into low-speed APB bus transaction.

For more information about the IP, refer to [AHB-Lite to APB Bridge IP User Guide \(FPGA-IPUG-02053\)](#)

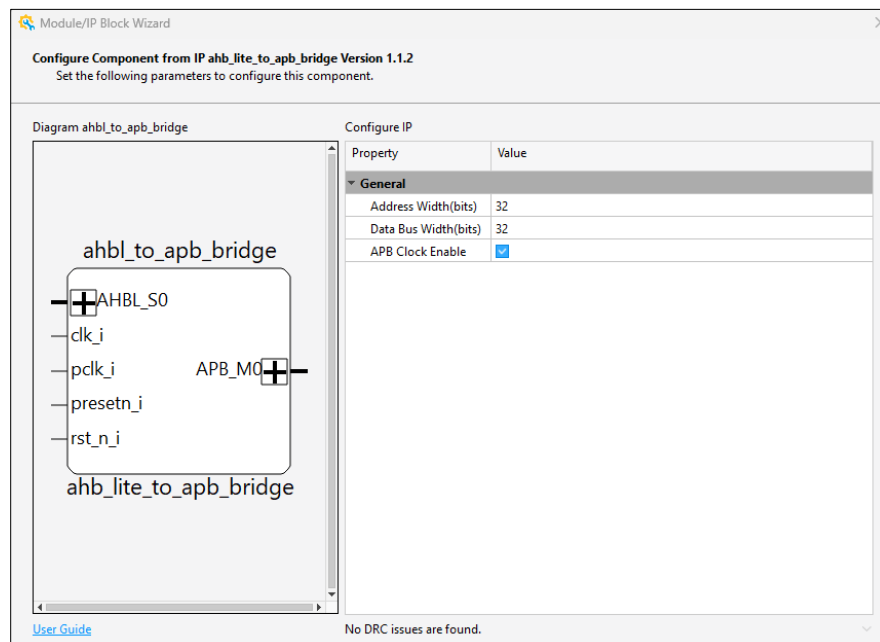


Figure 2.11. AHB-L to APB Bridge IP Configuration – General

2.5.5. AHB-Lite to AXI4 Bridge

The AHB-Lite to AXI4 Bridge IP is used to convert the AHB-Lite bus transaction into AXI4 bus transaction.

For more information about the IP core, refer to [AHB-Lite to AXI4 Bridge IP User Guide \(FPGA-IPUG-02242\)](#).

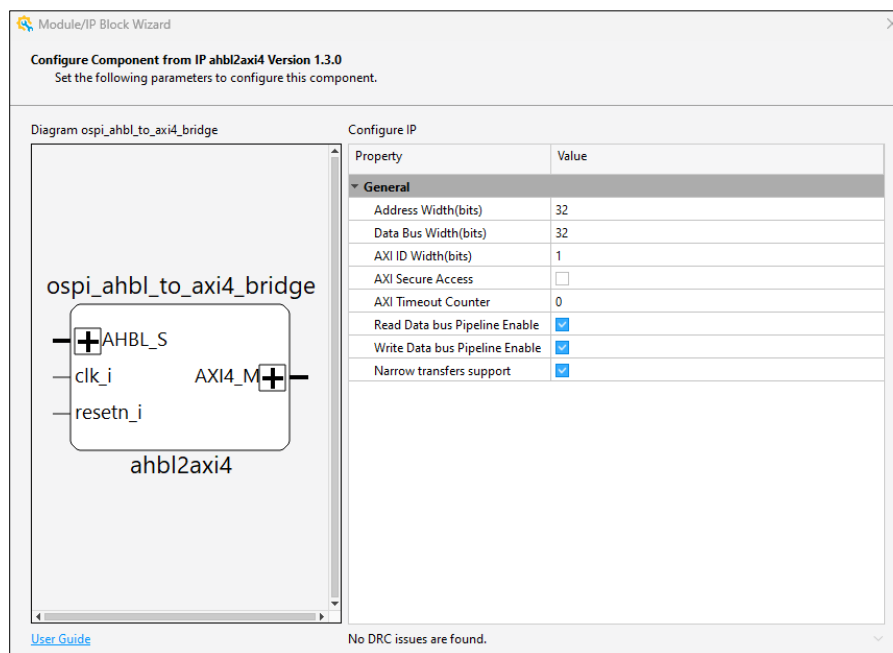


Figure 2.12. AHB-Lite to AXI4 Bridge IP Configuration – General

2.5.6. AXI4 to AHB-Lite Bridge

The AXI4 to AHB-Lite Bridge IP is used to convert the AXI4 bus transaction into AHB-Lite bus transaction.

For more information about the IP core, refer to [AXI4 to AHB-Lite Bridge IP User Guide \(FPGA-IPUG-02199\)](#).

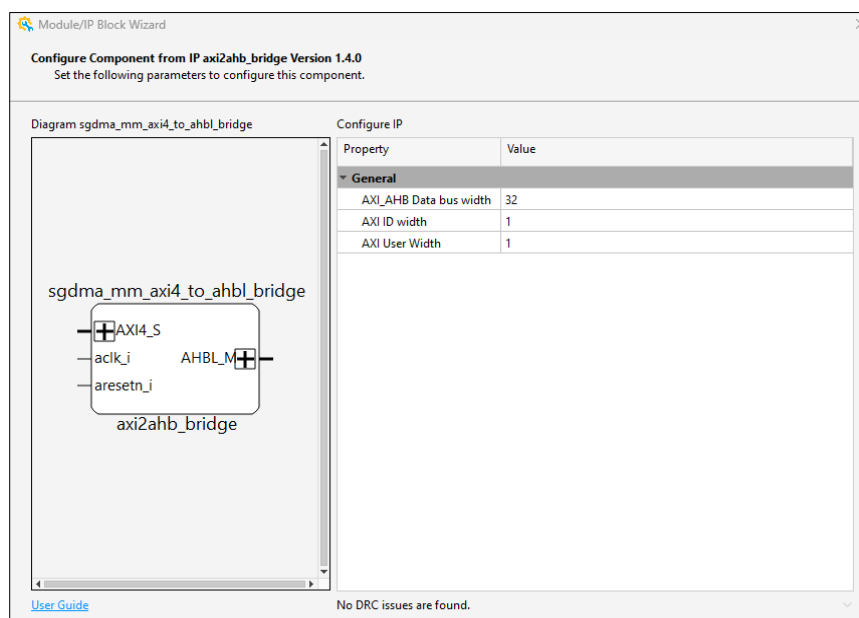


Figure 2.13. AXI4 to AHB-Lite Bridge IP Configuration – General

2.5.7. 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 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 \(FPGA-IPUG-02273\)](#).

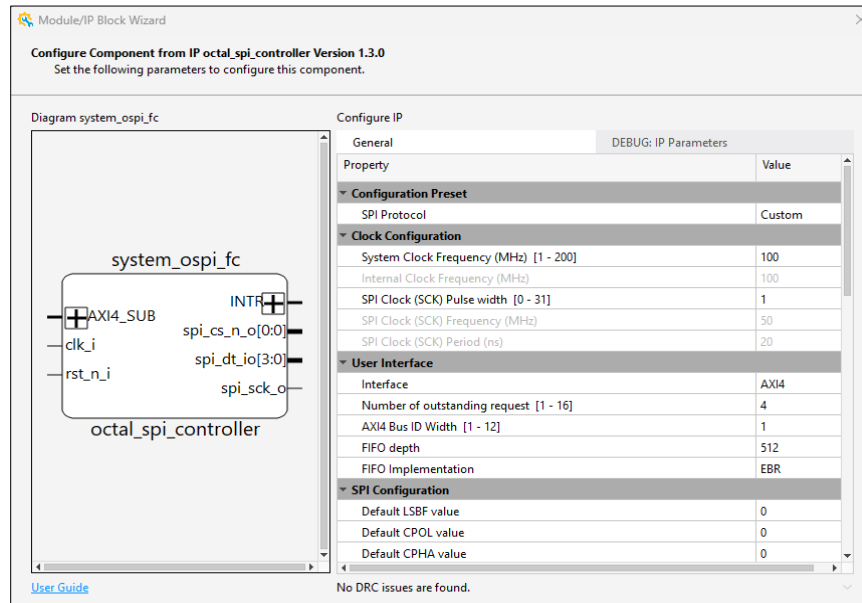


Figure 2.14. Octal SPI Controller IP Configuration – General

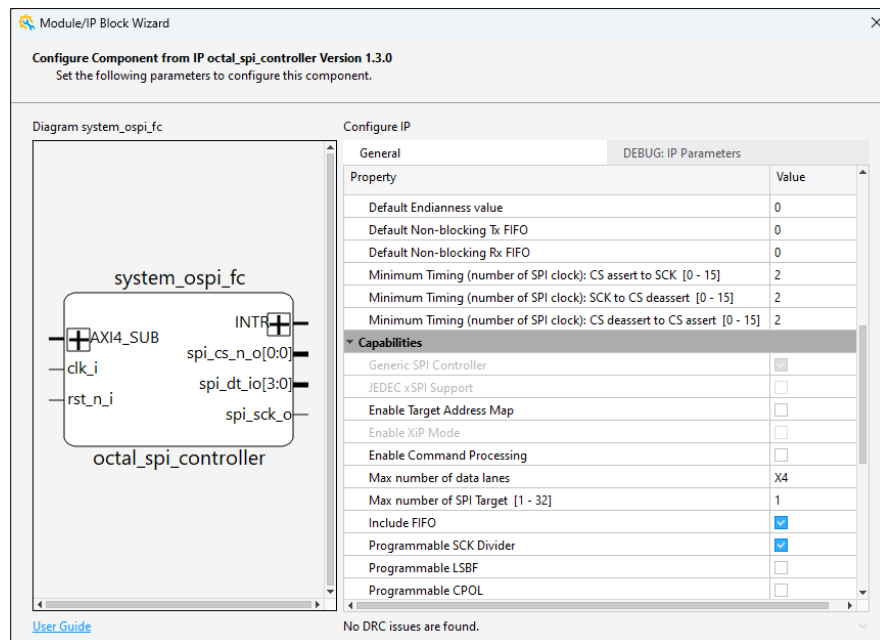


Figure 2.15. Octal SPI Controller IP Configuration – General



The Tri-Speed Ethernet (TSE) IP solution consists of the TSE IP Media Access Controller (MAC) core and RGMII interface. 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 Certus-NX GSRD supports a 100 Mbps network interface as per the IEEE 802.3 standard. The Ethernet PHY is configured through the MDIO interface on the TSE IP.

Module/IP Block Wizard

Configure Component from IP tse_mac Version 2.1.0
Set the following parameters to configure this component.

Diagram eth_tsmac

Configure IP

Property	Value
General	
Select IP Option	MAC only
Configuration	
Select MAC Operating Option	RGMII
Host Interface	APB
Include MIIM Module	<input checked="" type="checkbox"/>
Statistics Counter Registers	<input type="checkbox"/>
RGMII Timing Consideration	
Enable FPGA delay for TX	<input type="checkbox"/>
Enable FPGA delay for RX	<input type="checkbox"/>

[User Guide](#)

No DRC issues are found.

Figure 2.17. TSE IP Configuration

2.5.9. 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 \(FPGA-IPUG-02131\)](#).

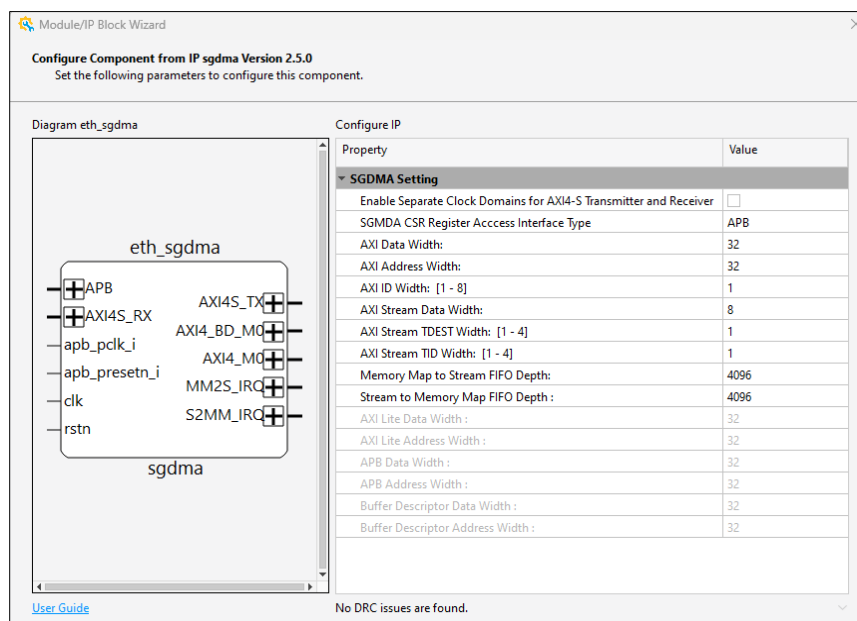


Figure 2.18. SGDMA Controller IP Configuration

2.5.10 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 \(FPGA-IPUG-02105\)](#).

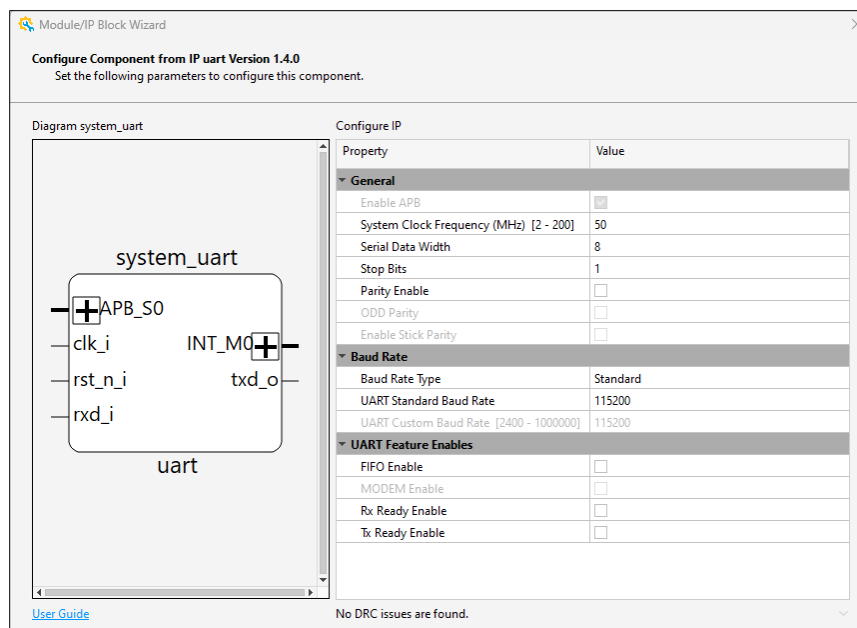


Figure 2.19. UART IP Configuration

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 \(FPGA-IPUG-02076\)](#).

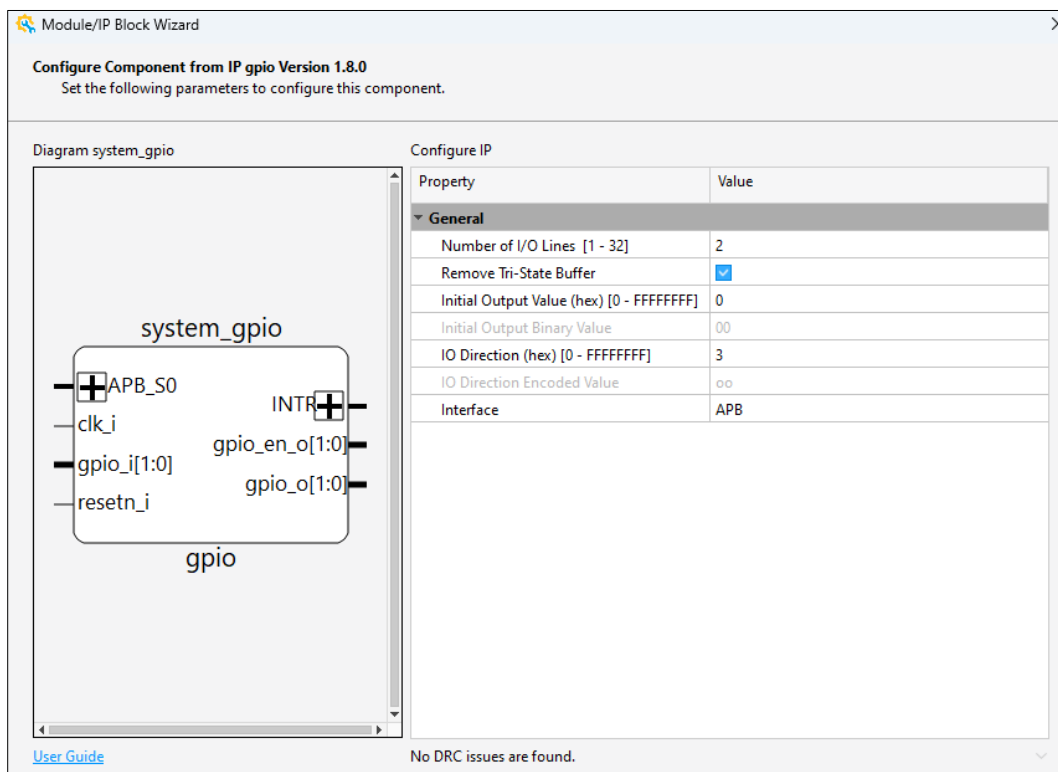


Figure 2.20. GPIO IP Configuration

2.5.11. 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 and LMMI config block.

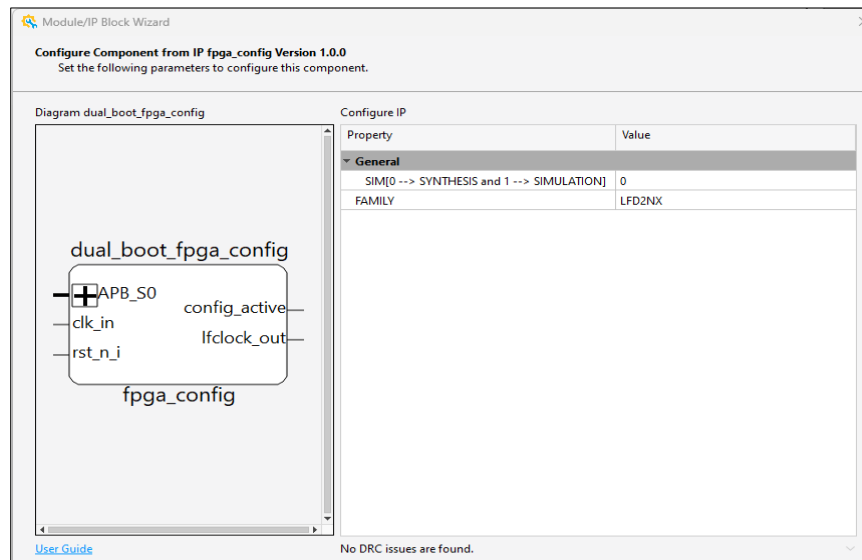


Figure 2.21. Multi-Boot Module Configuration

2.5.12. 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 dual port AHB-L as the interfaces. The System Memory in this design is used to store the bootloader and bare-metal application software. The system's memory size is 64 kB.

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

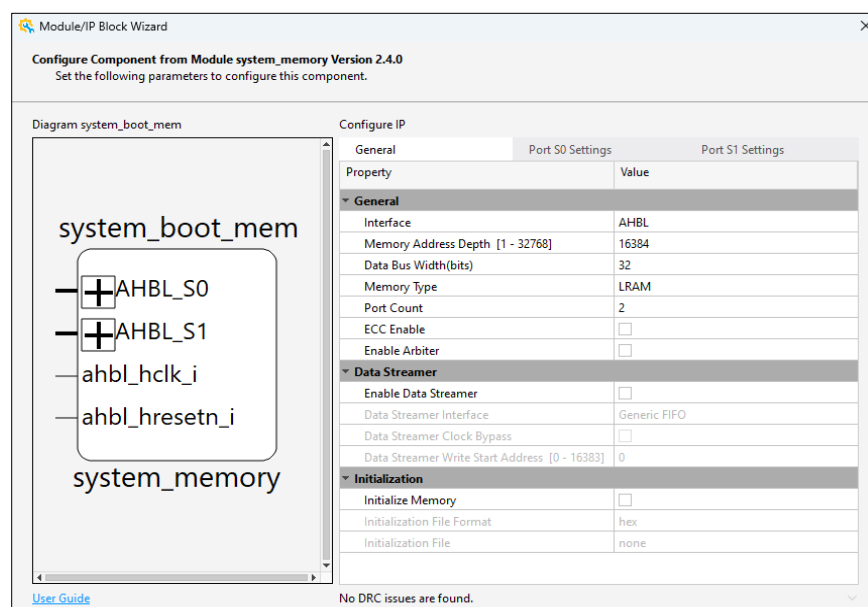


Figure 2.22. System Memory IP Configuration – General

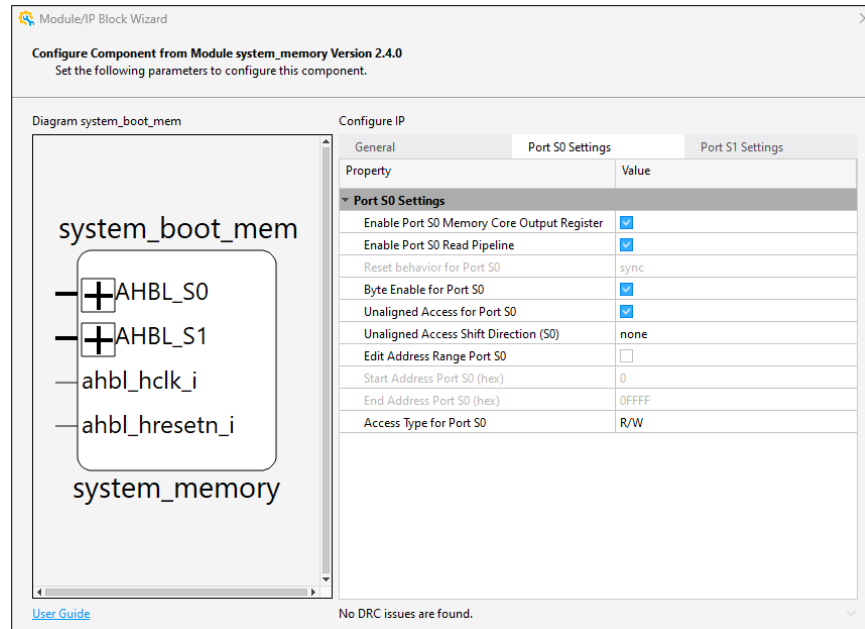


Figure 2.23. System Memory IP Configuration – Port S0 Settings

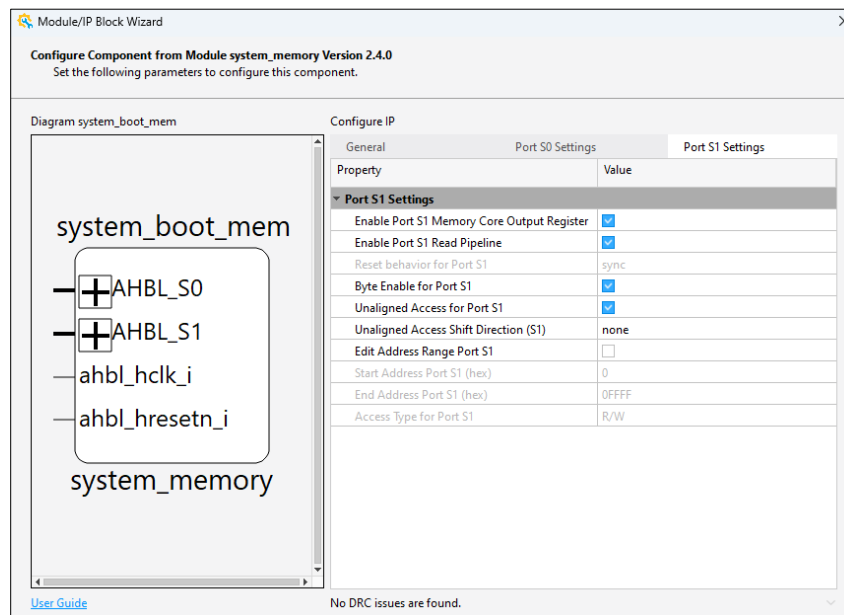


Figure 2.24. System Memory IP Configuration – Port S1 Settings

2.5.13. PLL

The PLL IP is used to generate multiple clocks used in the system. There are two PLL instances used in GSRD, system_pll, and eth_pll.

For more information about the IP, refer to [PLL IP User Guide \(FPGA-IPUG-02063\)](#).

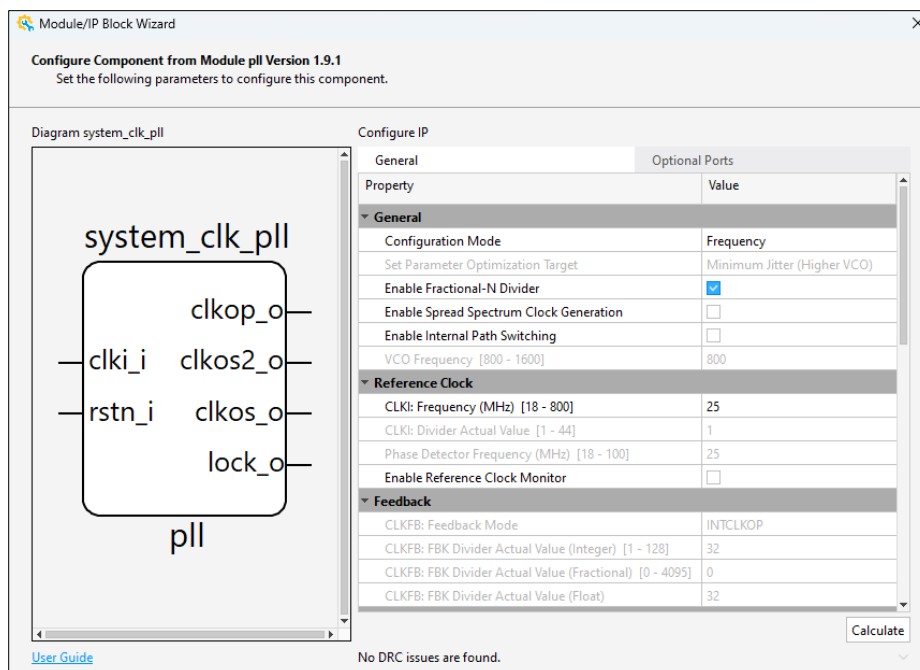


Figure 2.25. PLL IP Configuration – General for system_pll

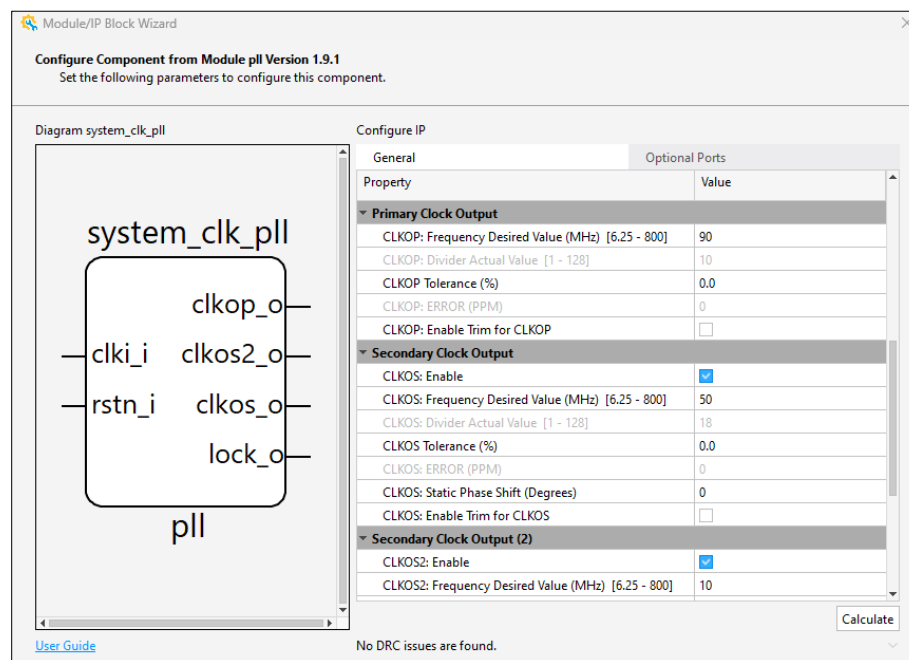


Figure 2.26. PLL IP Configuration – General for system_pll

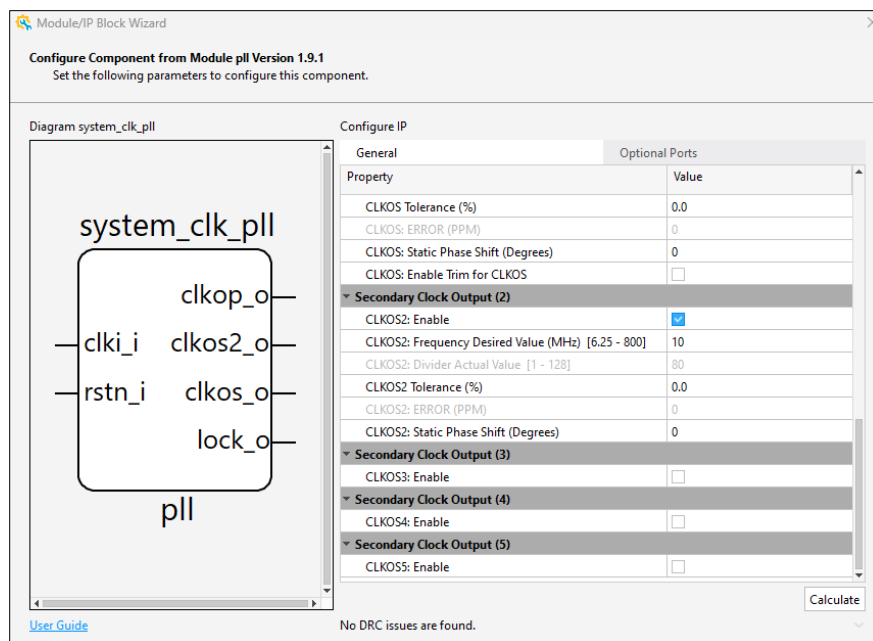


Figure 2.27. PLL IP Configuration – General for system_pll

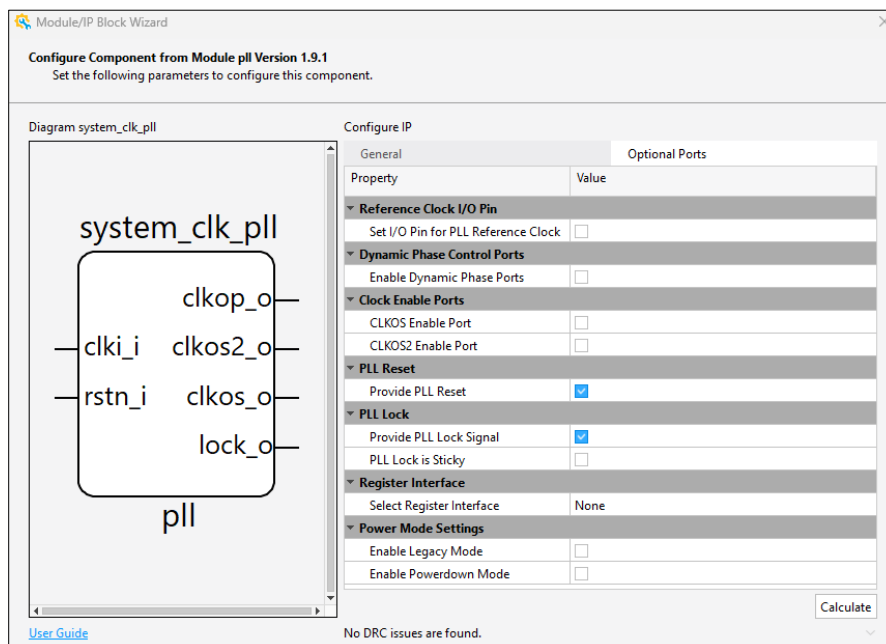


Figure 2.28. PLL IP Configuration – Optional Ports for system_pll

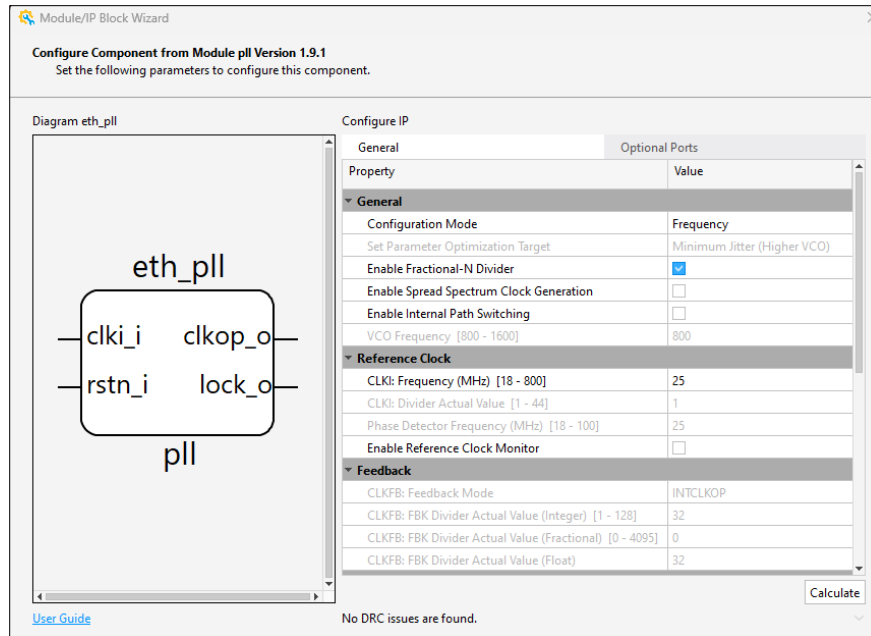


Figure 2.29. PLL IP Configuration – General for eth_pll

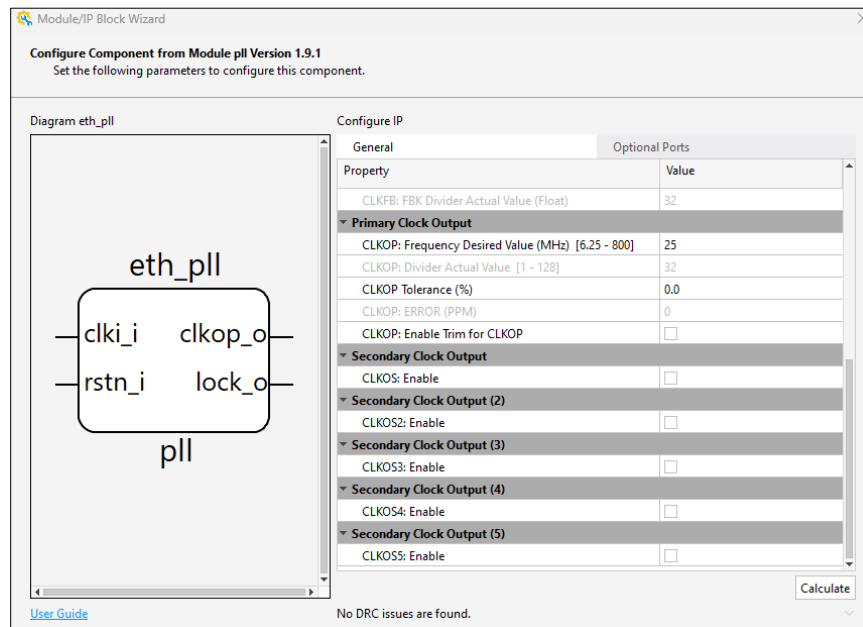


Figure 2.30. PLL IP Configuration – General for eth_pll

2.5.14. 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.

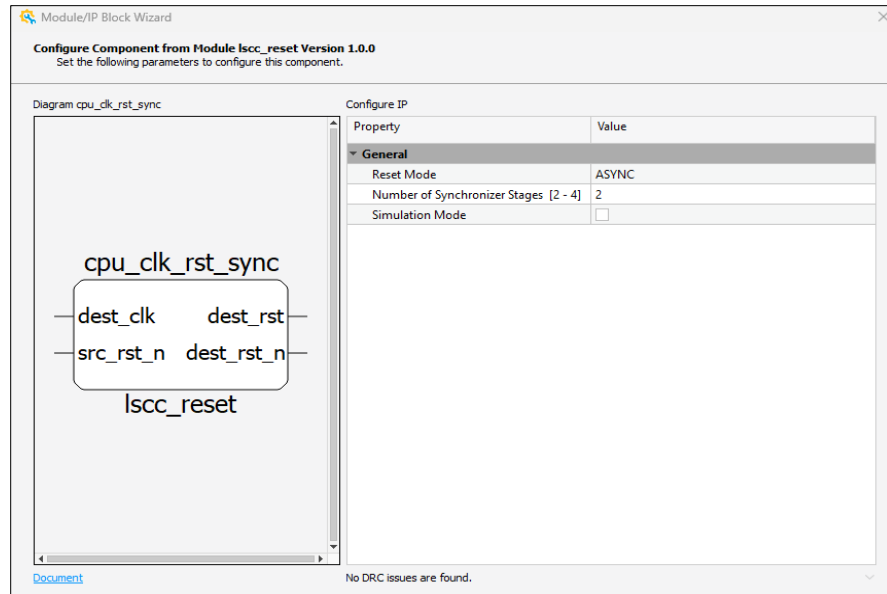


Figure 2.31. Reset Module Configuration – General

2.5.15. Clock Domain Crossing Module

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

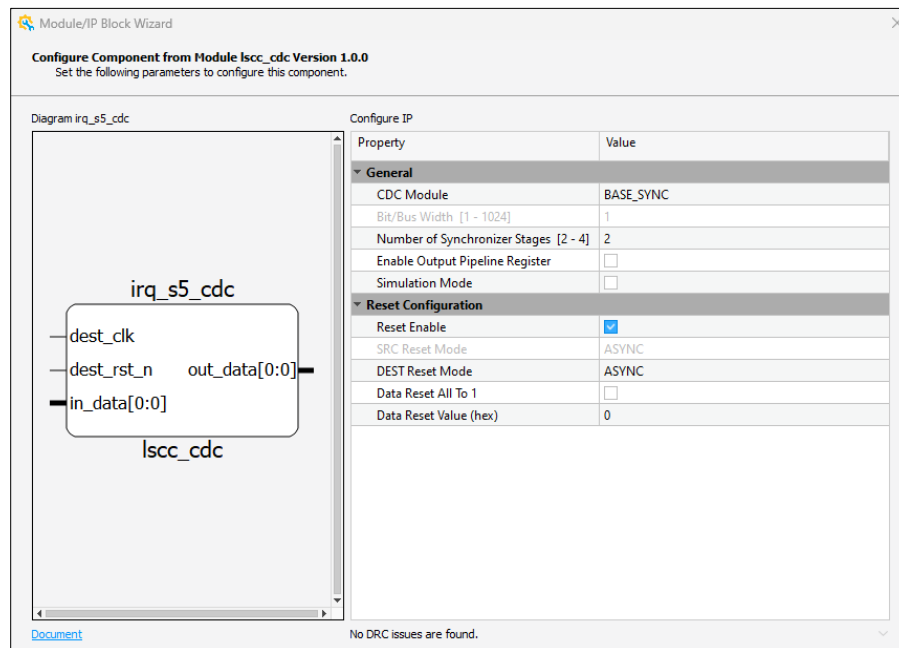


Figure 2.32. Clock Domain Crossing Module Configuration – General

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.

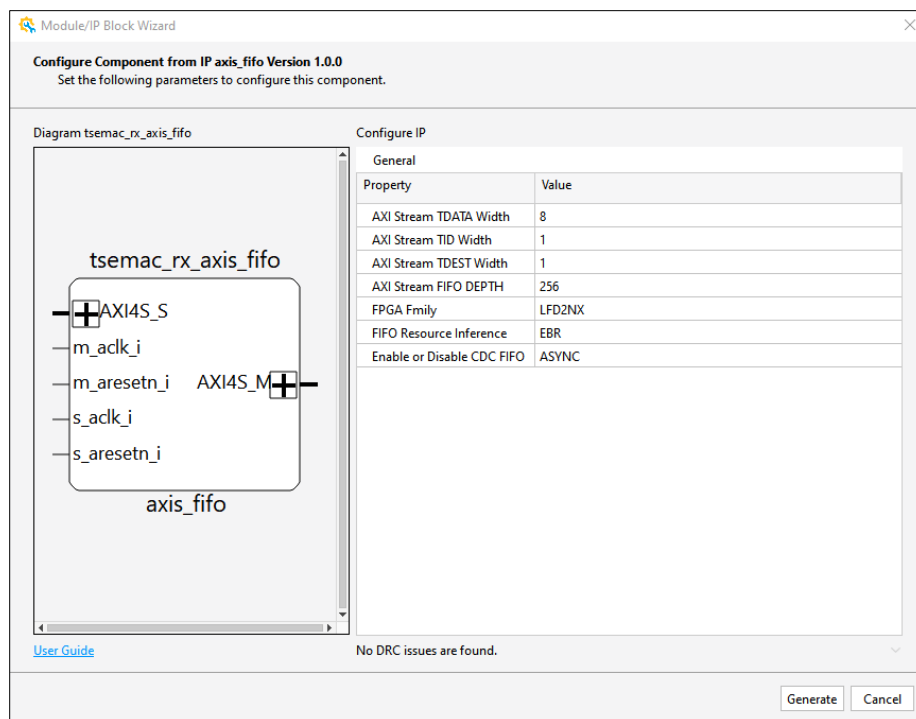


Figure 2.33. AXIS RX FIFO Module Configuration – General for tsemac_rx_axis_fifo

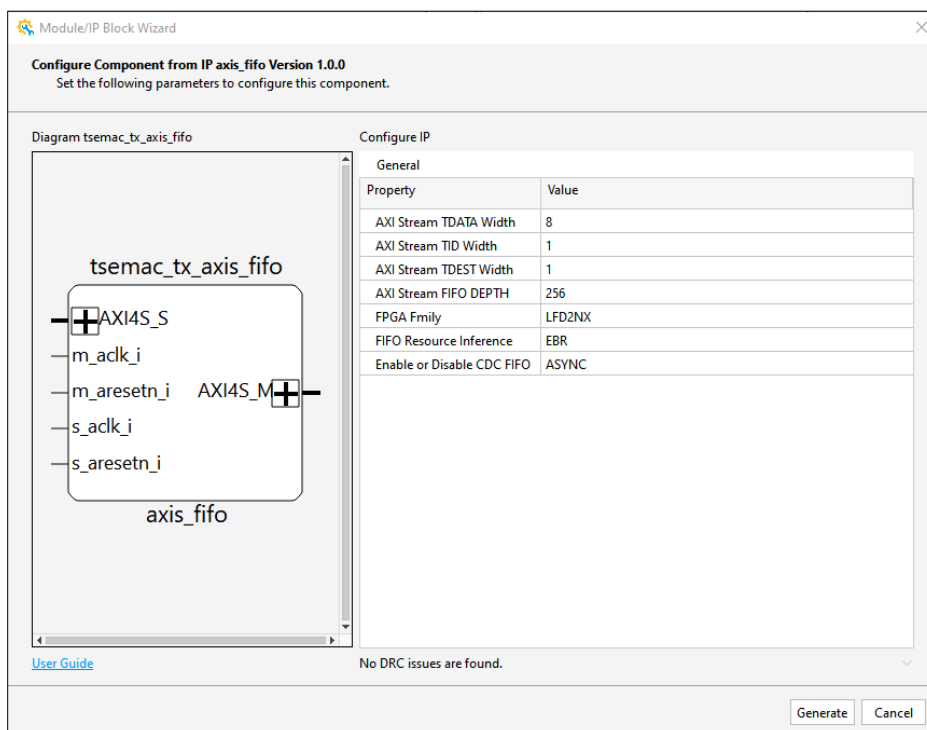


Figure 2.34. AXIS FIFO Module Configuration – General for tsemac_tx_axis_fifo

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 Interfaces on SGDMA Controller IP
Advanced High-performance Bus-Lite	AHB-L	Used for Data Interfaces on RISC-V and System Memory
Advanced Peripheral Interface	APB	Used as Control Interface for low-speed IP
Serial Peripheral Interface	SPI	Used for communication with external SPI Flash
100 Mbps Ethernet	RGMII	Used for communication with external Ethernet PHY
Management Data Input/Output	MDIO	Used for configuring the external Ethernet PHY
Universal Asynchronous Receiver/Transmitter	UART	Used for CPU printouts
General Purpose I/O	GPIO	Used for LEDs to indicate the RGMII link speed

2.7. SoC Memory/Address Map

Table 2.6. GHRD Address Map

Base Address	End Address	Size	Subordinate
0x8000_0000	0x8000_FFFF	64 kB	System Memory
0xC000_0000	0xC000_0FFF	4 kB	UART APB
0xC000_1000	0xC000_1FFF	4 kB	GPIO APB
0xC000_2000	0xC0002_FFF	4 kB	Multi-Boot Config APB
0xC000_3000	0xC000_3FFF	4 kB	SGDMA Controller APB
0xC000_4000	0xC000_7FFF	16 kB	TSE MAC APB
0xC000_8000	0xC000_8FFF	4 kB	Watchdog Timer APB
0xC000_D000	0xC3FF_FFFF	—	Reserved
0xC400_0000	0xC400_0FFF	4 kB	Octal SPI Controller
0xC400_1000	0xF1FF_FFFF	—	Reserved
CPU Local Memory			
0xFFFF_0000	0xFFFF_07FF	2 kB	Timer

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 2.35 shows the approximate GHRD resource utilization and Table 2.8 shows the total LUT4, PFU register, I/O buffer, and EBR resource utilization for LFD2NX-40 device.

impl_1	LUT4	Logic	istributed RAM	Ripple Logic	PFU Registers	IO Registers	IO Buffers	DSP MULT	EBR	Large RAM
▼ soc_golden_gsrdr	13212(2)	10432(2)	210(0)	2570(0)	9523(86)	3(0)	34(24)	6(0)	27(0)	1(0)
▶ tsemac_tx_axis_fifo_inst	88(0)	54(0)	0(0)	34(0)	80(0)	0(0)	0(0)	0(0)	1(0)	0(0)
▶ tsemac_rx_axis_fifo_inst	88(0)	54(0)	0(0)	34(0)	80(0)	0(0)	0(0)	0(0)	1(0)	0(0)
▶ system_uart_inst	243(0)	197(0)	0(0)	46(0)	146(0)	1(0)	0(0)	0(0)	0(0)	0(0)
▶ system_ospifc_inst	1527(0)	1165(0)	120(0)	242(0)	1190(0)	0(0)	4(0)	0(0)	2(0)	0(0)
▶ system_ic_inst	648(0)	618(0)	0(0)	30(0)	208(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ system_clk_pll_inst	1(0)	1(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ system_boot_mem_inst	760(0)	726(0)	0(0)	34(0)	423(0)	0(0)	0(0)	0(0)	0(0)	1(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)
▶ sgdma_mm_axi4_to_ahbl_bridge_inst	531(0)	433(0)	54(0)	44(0)	236(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ sgdma_bd_axi4_to_ahbl_bridge_inst	534(0)	454(0)	36(0)	44(0)	233(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ riscv_cpu_wdt_inst	165(1)	109(1)	0(0)	56(0)	133(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ riscv_cpu_inst	3628(0)	2912(0)	0(0)	716(0)	1906(0)	0(0)	0(0)	6(0)	2(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)
▶ ospi_ahbl_to_axi4_bridge_inst	74(0)	74(0)	0(0)	0(0)	114(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ mac_tx_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_rx_clk_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)
▶ irq_s6_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ irq_s5_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ irq_s4_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ irq_s3_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ irq_s2_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ irq_s1_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ irq_s0_cdc_inst	0(0)	0(0)	0(0)	0(0)	2(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ eth_tsemac_inst	2296(0)	1660(0)	0(0)	636(0)	2304(0)	0(0)	0(0)	0(0)	4(0)	0(0)
▶ eth_sgdmact_inst	2139(0)	1499(0)	0(0)	640(0)	1953(0)	0(0)	0(0)	0(0)	17(0)	0(0)
▶ eth_pll_inst	1(0)	1(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ dual_boot_fpga_config_inst	56(0)	56(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)
▶ apb_ic_inst	110(0)	110(0)	0(0)	0(0)	7(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ apb_gpio_inst	43(0)	43(0)	0(0)	0(0)	29(0)	2(0)	2(0)	0(0)	0(0)	0(0)
▶ ahbl_to_apb_bridge_inst	244(0)	244(0)	0(0)	0(0)	263(0)	0(0)	0(0)	0(0)	0(0)	0(0)

Figure 2.35. LFD2NX-40 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)	13212	40.96%
PFU Register	9523	29.52%
I/O Buffers	34	18.37%
EBR	27	32.14%
LRAM	1	50%

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
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 D25 and D24 on board to indicate RGMII link speed. gpio_o[1:0] (D25:D24) 00, 10 and 11: Not defined 01: 100 Mbps. D25 = ON, D24 = OFF
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
RGMII			
rgmii_rxd_i	Input	4	RGMII receives data from Ethernet PHY
rgmii_rxctl_i	Input	1	RGMII receives control from Ethernet PHY
rgmii_rxc_i	Input	1	RGMII receives clock from Ethernet PHY
rgmii_txd_o	Output	4	RGMII transmits data to Ethernet PHY
rgmii_txctl_i	Output	1	RGMII transmits control to Ethernet PHY
rgmii_txc_i	Output	1	RGMII transmits clock to Ethernet PHY
rgmii_phy_rstn_o	Output	1	Ethernet PHY resetn
MDIO			
rgmii_mdc	Output	1	MDIO Clock
rgmii_mdio	Input/Output	1	MDIO Data
Multi-Boot			
config_active_o	Output	1	Connects to LED D18 to check Multi-Boot Configuration block status 0: Configuration is in progress. LED = ON 1: Configuration is done. LED = OFF
LEDs			
system_pll_lock_o	Output	1	Connect to LED D19 to check System PLL lock status 0: PLL is locked. LED = ON 1: PLL is unlocked. LED = OFF
eth_pll_lock_o	Output	1	Connect to LED D20 to check Ethernet PLL lock status 0: PLL is locked. LED = ON 1: PLL is unlocked. LED = OFF

4. Software Components

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

4.1. Primary and Golden Bootloader

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

- Configures the GPIO and UART IP
- Calculates the CRC value on the entire application software copied into System Memory 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 Dual-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 bare metal with the following functions:

- Executed by RISC-V MC CPU from System Memory where it initializes the BSP
- Initialize the TSE IP by setting up the 100 Mbps Ethernet link speed, MAC address and full duplex mode based on the selected configuration.
- Configure external Ethernet PHY through MDIO interface.
- Configure the SGDMA Controller MM2S (Memory-Mapped to Streaming) interface for transferring the ethernet packets to TSE MAC and set up S2MM (Streaming to Memory Mapped) interface to receive ethernet packets from the TSE MAC.

5. Theory of Operation

The GSRD system comprises two SoCs (System on Chip) design and corresponding software stacks, called as Primary and Golden images. Each SoC is linked with its respective bootloader and bare-metal application software. These SoCs are built using Lattice Propel Builder, and Lattice Radiant software tools. 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 bare-metal application software are stored in the external SPI Flash (Micron).

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 bare-metal application software from external SPI Flash into the System Memory 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 bare-metal application software execution.

5.1. Boot-Up Sequence

This section describes the RISC-V MC CPU boot up sequence in detail that configures IP drivers and 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, and Octal SPI Controller.
- The bootloader fetches the respective Primary application software through the Octal SPI Controller into System Memory.
- The application software is stored in the second half of System Memory. After the application software is loaded, the RISC-V CPU calculates the CRC of the application code in System Memory.
- The CRC of the application software in System Memory is calculated and validated against the reference CRC embedded in SPI flash.
- If the calculated CRC matches, the RISC-V CPU jumps to bare-metal application software execution in System Memory.
- 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 SPI Flash.
- Octal SPI Controller performs a self-diagnostic check for read, write, and erase operations. Refer to [Appendix C. Using the Octal SPI Controller for Read, Write, and Erase Self-Diagnostic Check \(Bare-metal Application Software\)](#) for more information.
- The Bare-metal application software initializes the SGDMA Controller and TSE IP and uses the MDIO interface of TSE IP to configure external Ethernet PHY for Ethernet connection.
- The bare-metal application software initiates Ethernet packets transmission to and reception from the host machine.

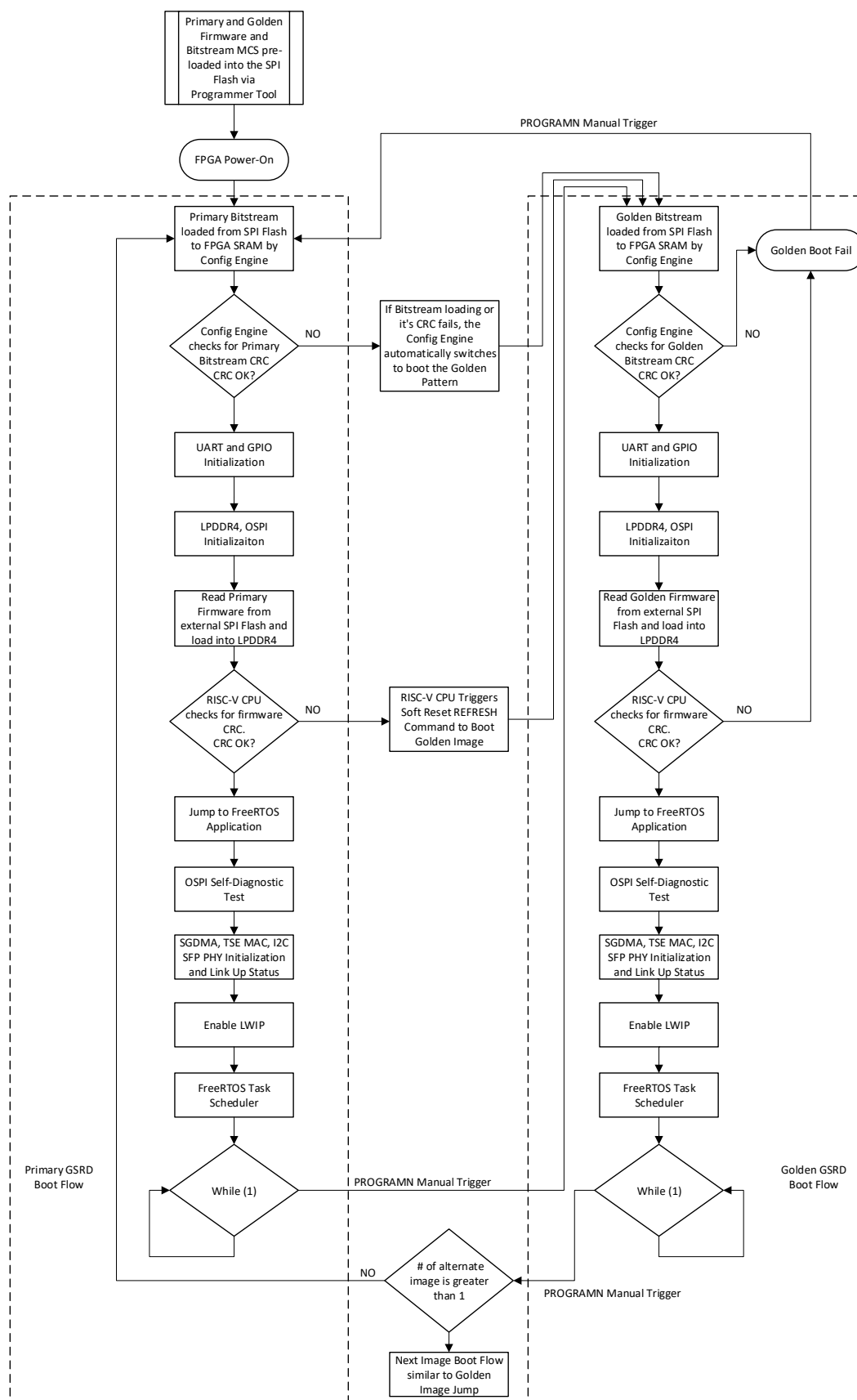


Figure 5.1. GSRD Boot-Up Sequence

5.2. Data Movement

This section describes the RISC-V MC CPU bare-metal application software execution in detail.

- Upon the execution of the correct Primary or Golden application software from System Memory, 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 System Memory.
- When data is written into the System Memory, 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, RISC-V goes back to the main loop to execute the next task.

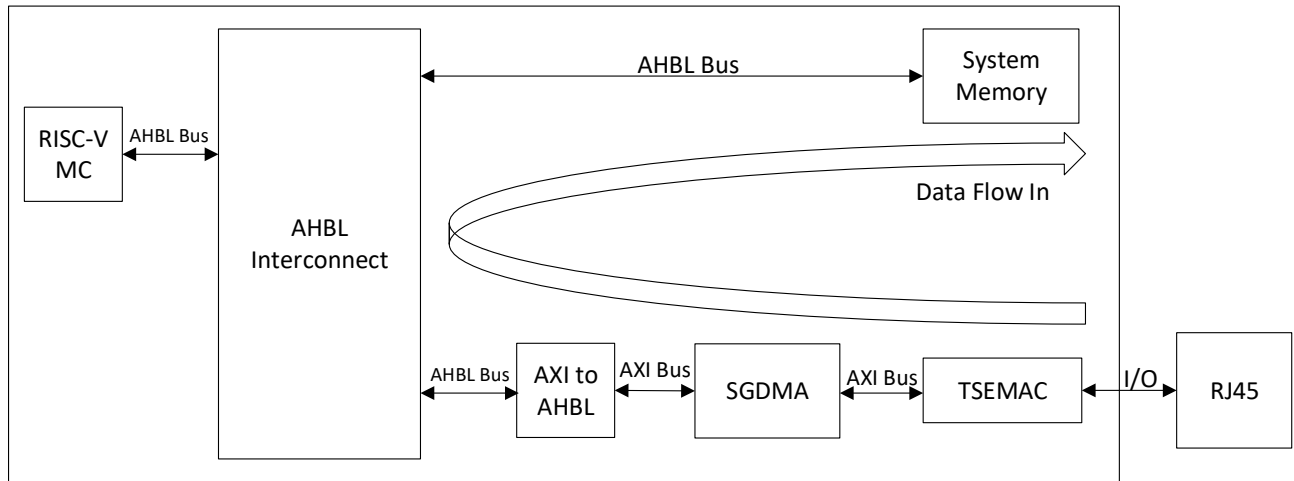


Figure 5.2. Ethernet Data RX Flow

- For outgoing data, data is fetched from a location inside System Memory by RISC-V CPU. 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 its tasks in the main loop based on the loaded software execution.

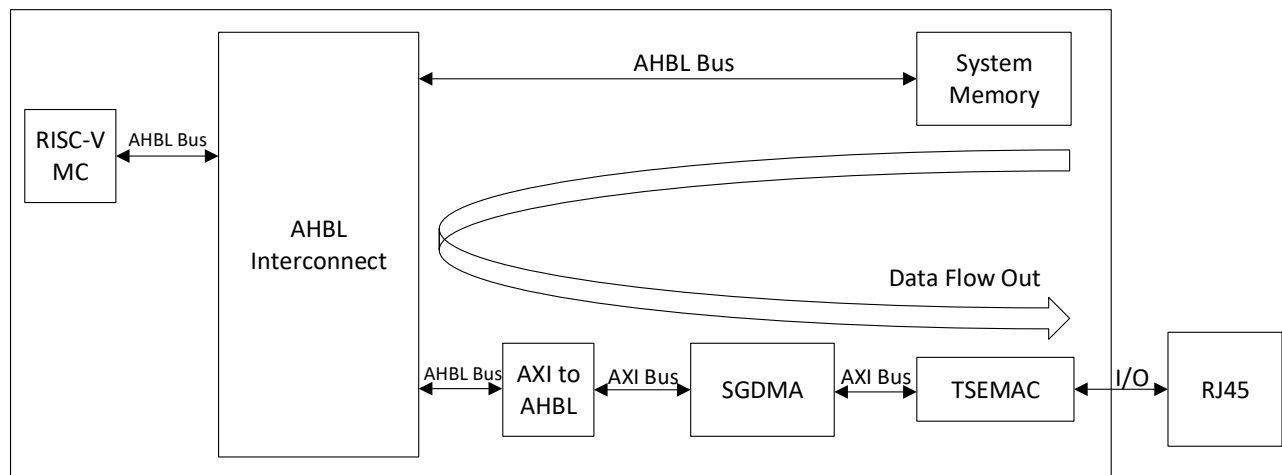


Figure 5.3. Ethernet Data TX Flow

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.

- Download the design package from the Lattice Semiconductor website. Go to *Design File* in the [GHRD/GSRD Demonstration](#), download the *Certus-NX Golden System Reference Design and Demo V1.0 – 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](#).

[Table 6.1](#) shows the bitstreams and software image binaries for programming the FPGA.

Table 6.1. Executable Files for Micron Flash

File Description	File Name	Starting Address in SPI Flash
Primary Software with CRC	c_primary_appcrc.bin	0x008A 0000
Primary FPGA Bitstream	soc_primary_gsr_d_impl_1.bit	0x0000 0000
Golden Software with CRC	c_golden_appcrc.bin	0x0080 0000
Golden FPGA Bitstream	soc_golden_gsr_d_impl_1.bit	0x0000 0000
Multi-Boot MCS File (Golden + Primary Bitstream)	multiboot_system.mcs	0x0000 0000

6.2. Setting Up the Hardware

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

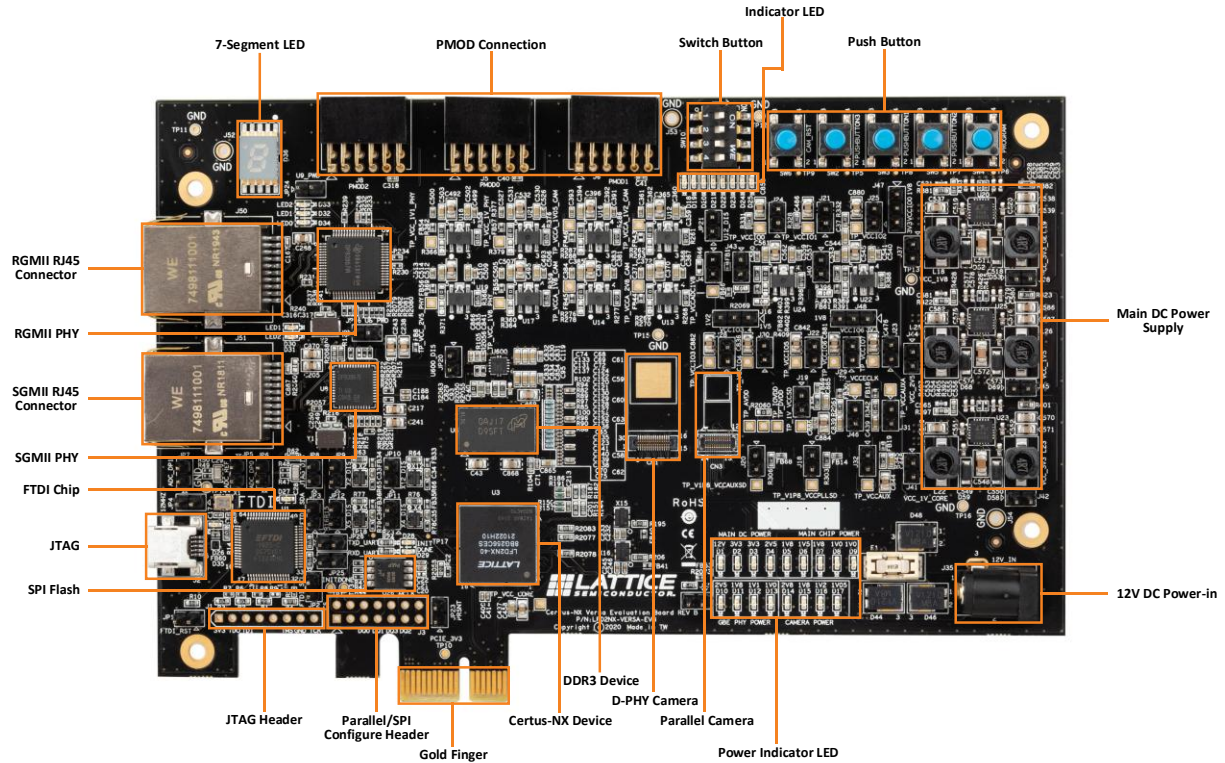


Figure 6.1. Certus-NX Versa Evaluation Board

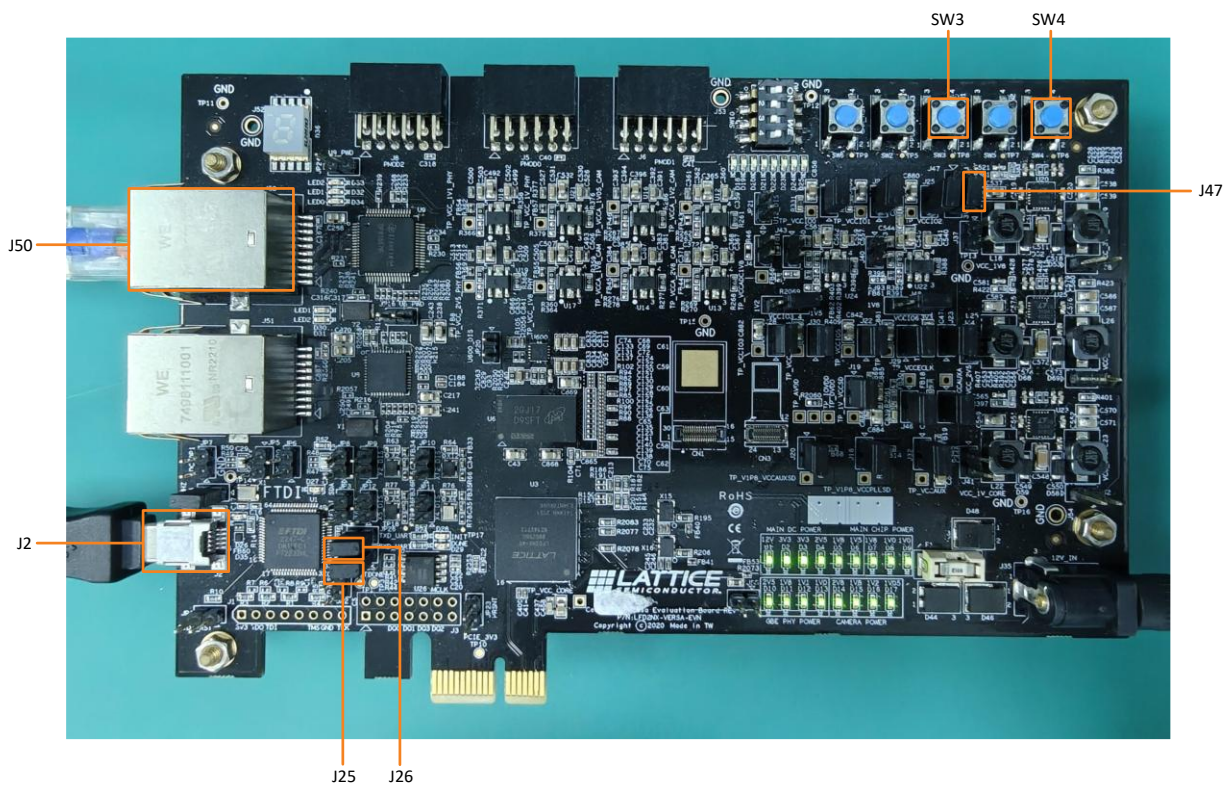


Figure 6.2. Connections, Jumpers and Buttons needed for Demonstration

To set up the hardware, perform the following:

1. Connect the 12 V power adapter to J35 DC input supply jack.
2. Connect the Mini-USB Type-A cable from PC to J2.
3. Connect jumpers on Pin 1 and Pin 2 on both JP25 and JP26 switches to enable UART.
4. Connect jumper on Pin 2 and Pin 3 on J47 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 [Certus-NX Versa Evaluation Board User Guide \(FPGA-EB-02032\)](#).
6. Connect the RJ45 cable from the host PC cable to J50 Ethernet port.
7. Turn on the power supply.

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:

1. Connect the Lattice Certus-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](#).

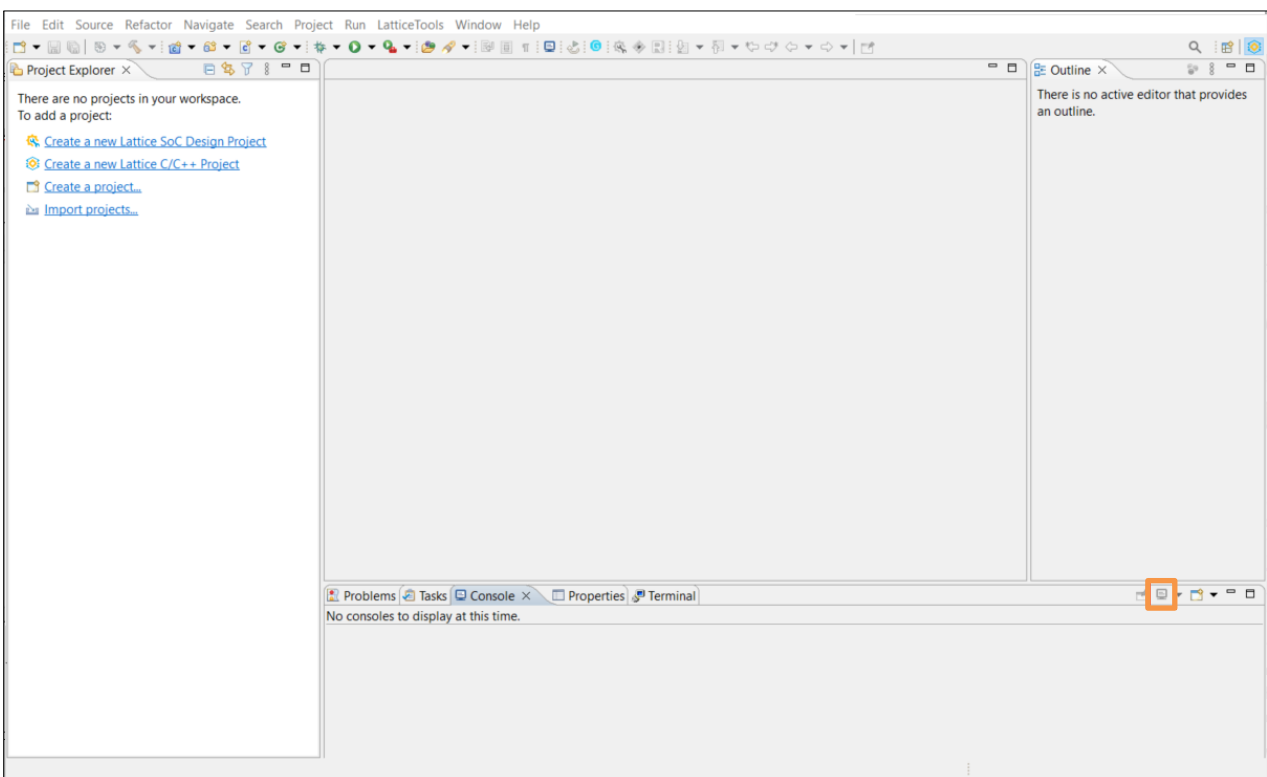


Figure 6.3. UART Terminal Icon on Propel SDK Window

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 the Device Manager. The COM port number can be different. If a USB port does not work, try a different USB port.

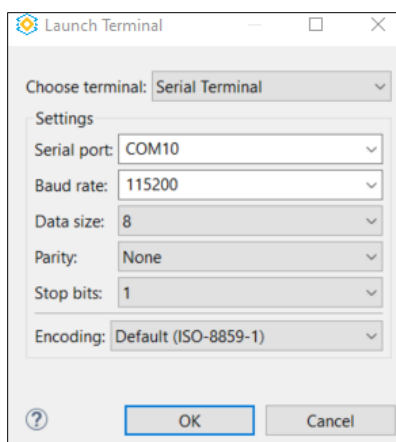


Figure 6.4. UART Launch Terminal Window

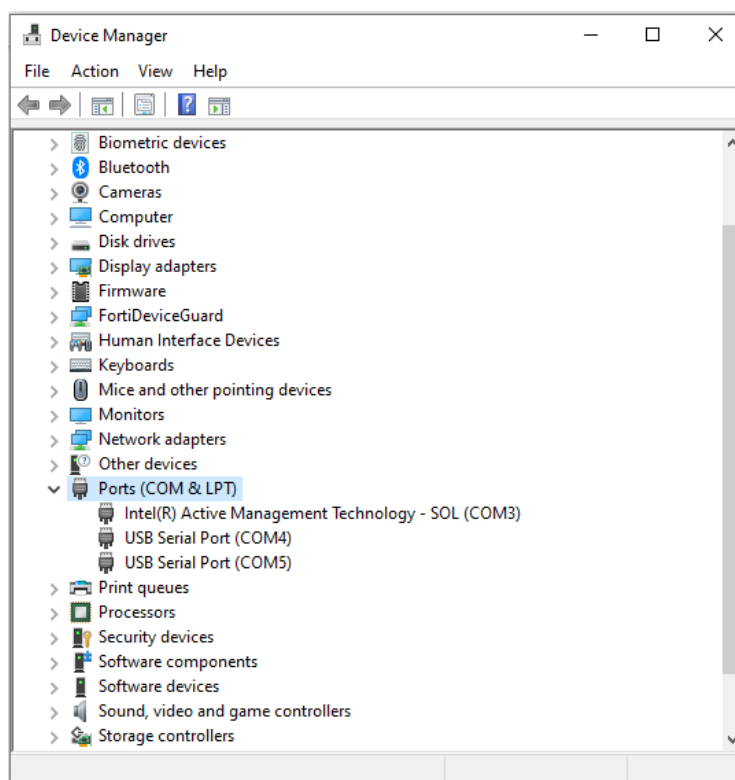


Figure 6.5. Device Manager Window on PC

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

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

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

1. Connect the Certus-NX Versa Evaluation Board to a PC using USB cable as per the hardware setup mentioned in [Setting Up the Hardware](#) section. Make sure the jumpers at JP25 and JP26 are installed properly.
2. Power-on the board.

- 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.

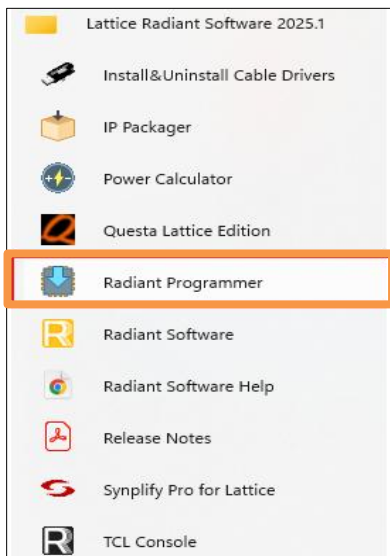


Figure 6.6. Launch Radiant Programmer from Windows Start

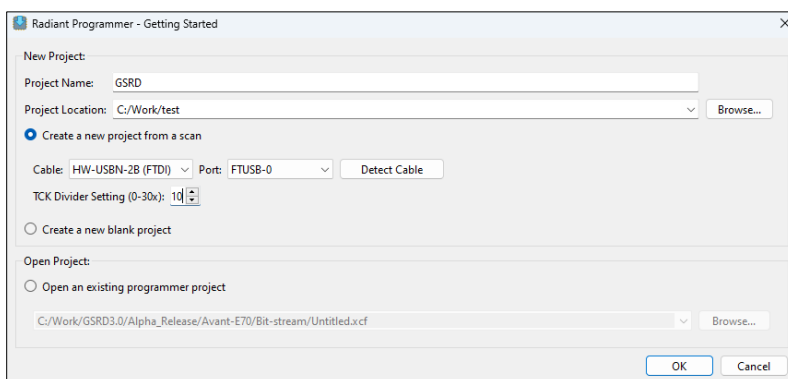


Figure 6.7. Radiant Programmer Start Window

- Click **OK**.

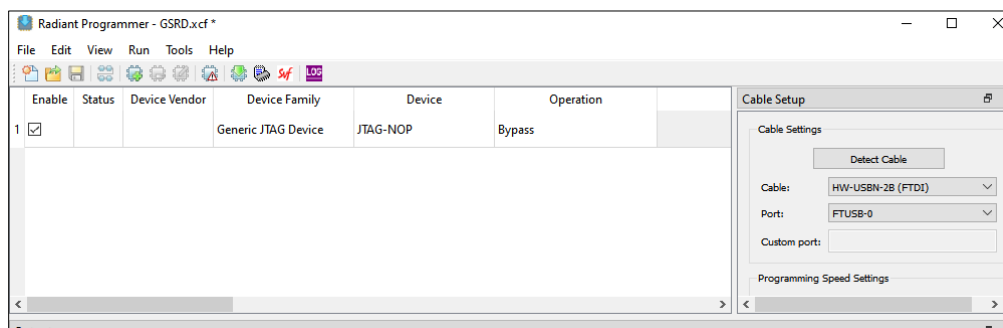


Figure 6.8. Radiant Programmer. xcf Window

- If the **Device Family** shows as **Generic JTAG Device**, click **Scan Device** as shown in Figure 6.9 to update the **Device Family** information automatically.

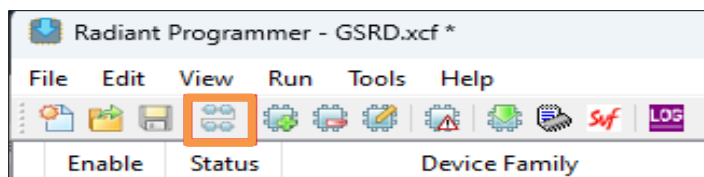


Figure 6.9. Scan Device Icon on Radiant Programmer

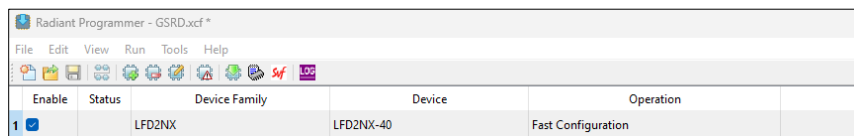


Figure 6.10. Select Device for Programming

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

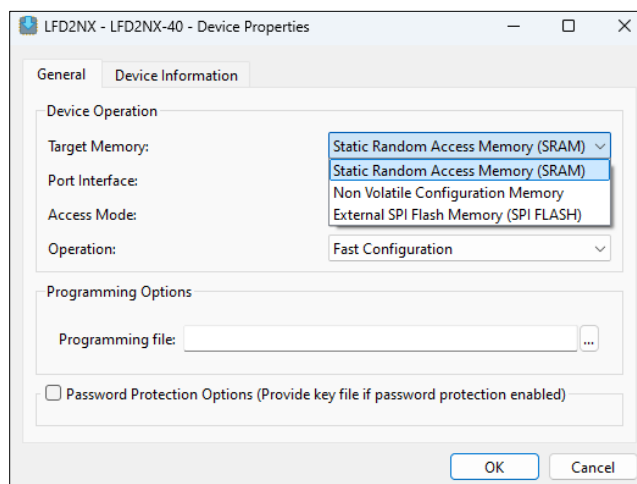


Figure 6.11. Select the Target Memory for Programming – SPI Flash

- Before programming, you need to erase the entire SPI Flash Memory by applying the settings as shown in [Figure 6.12](#).

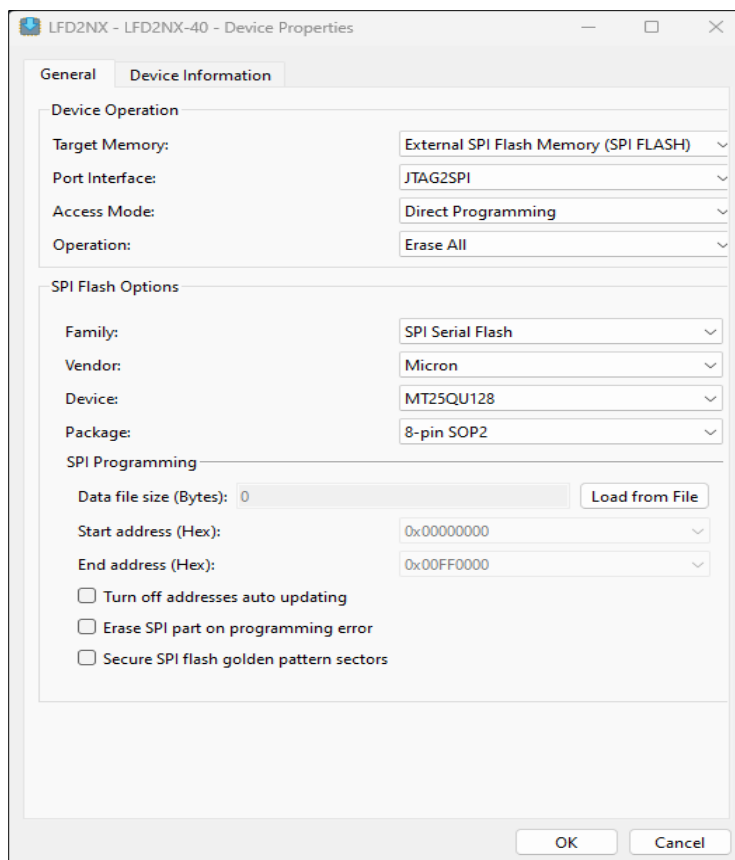


Figure 6.12. Device Properties to Erase the Micron SPI Flash

8. 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.

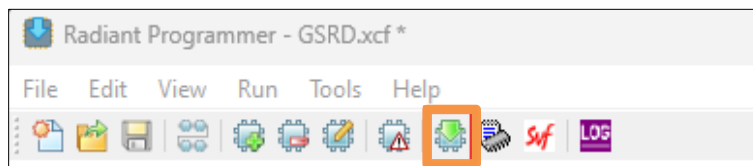


Figure 6.13. Program Button to Program the SPI Flash

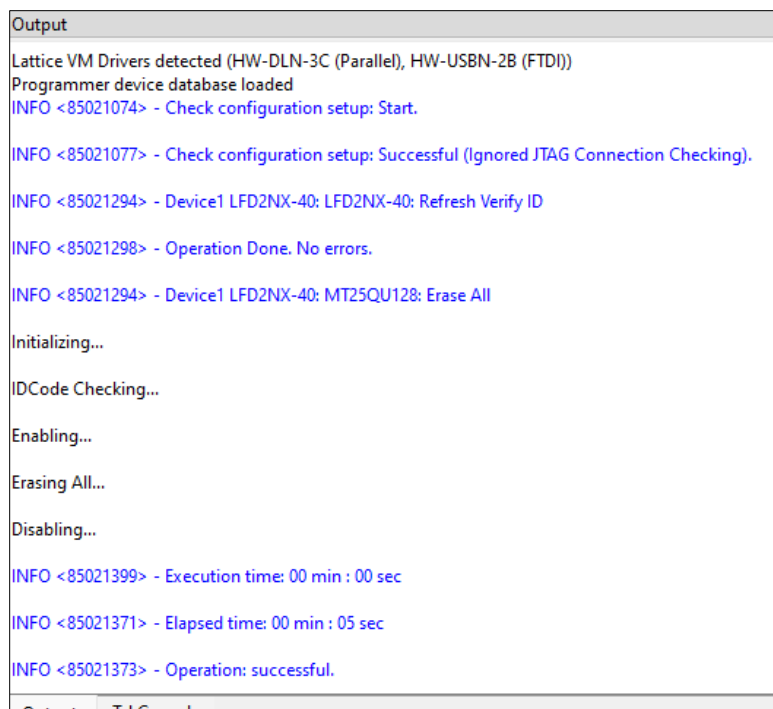


Figure 6.14. Output After Erase All

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

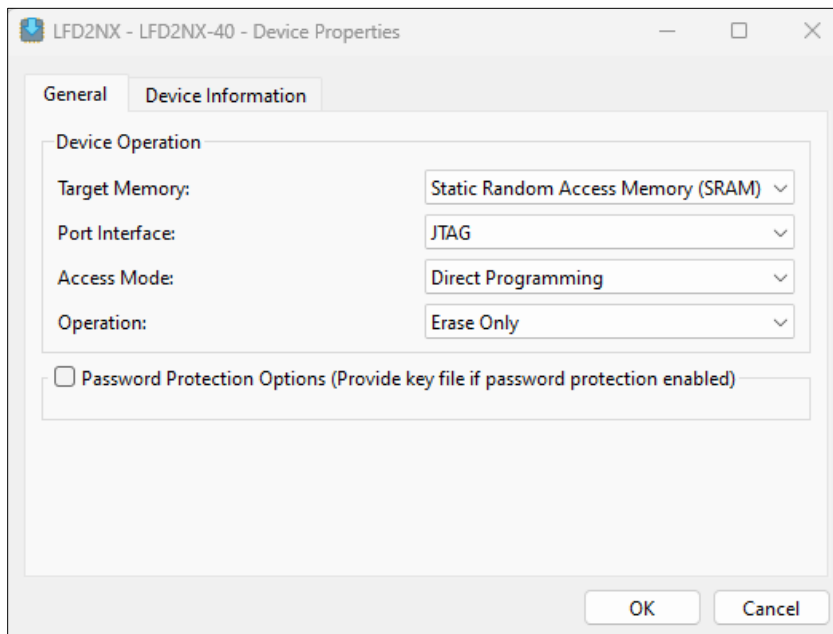


Figure 6.15. Erase Only Operation for SRAM Programming

11. To program the **c_golden_apprcr.bin** file into the SPI Flash, Follow the settings as shown in [Figure 6.16](#).

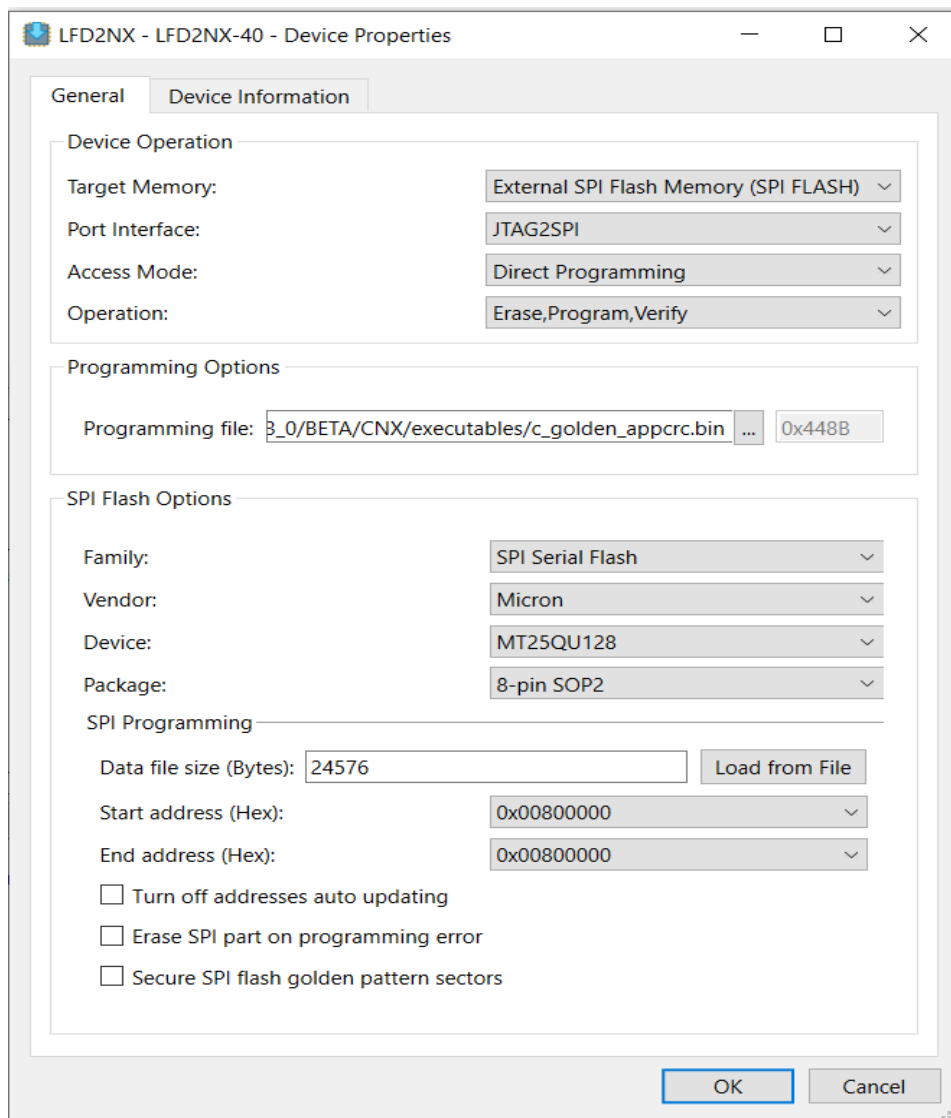


Figure 6.16. Device Properties to Program the Micron SPI Flash

12. Click **OK**.
13. Use TCK Divider Setting (0-30x) to **10**.

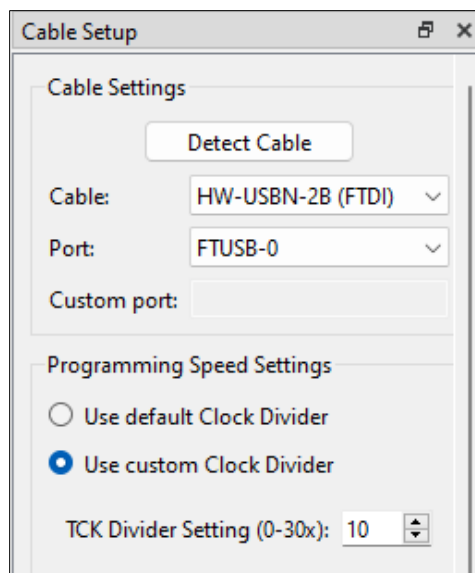


Figure 6.17. Cable Settings for Device Programming

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



Figure 6.18. Radiant Programmer Console Output after Programming the SPI Flash

15. Power cycle the Certus-NX Versa Board.
16. To program the FPGA Golden GSRD bitstream, double-click on the **Operation** tab to update the selections as shown in Figure 6.19. Make sure to provide the path to the .bit file location on your local machine where you have unzipped the executables.

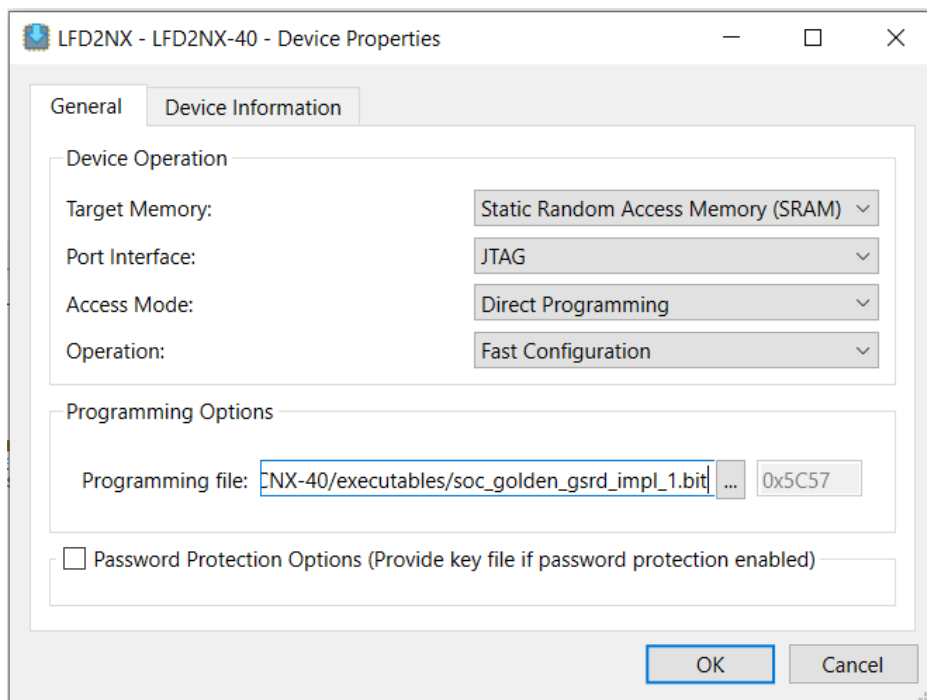


Figure 6.19. Device Properties to Program the FPGA Bitstream in SRAM

17. 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.

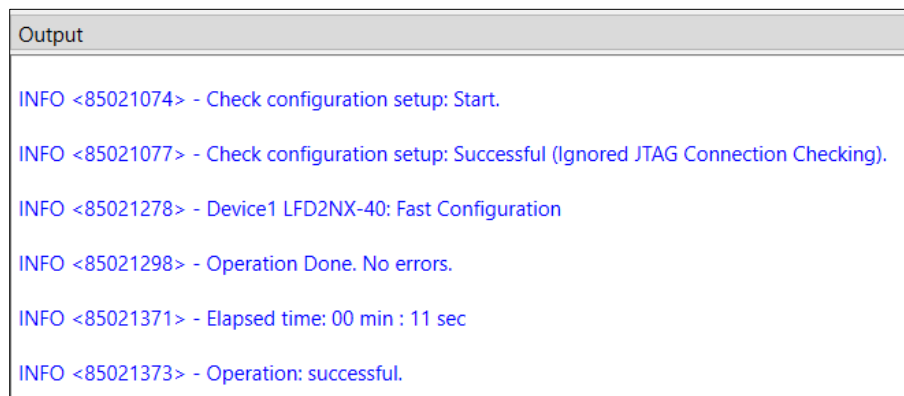


Figure 6.20. Radiant Programmer Console Output after Bitstream is Programmed

18. Set up the UART terminal as mentioned in [Setting Up the UART Terminal](#) section.
19. Press **SW3-Reset** button on the board as highlighted in [Figure 6.21](#).

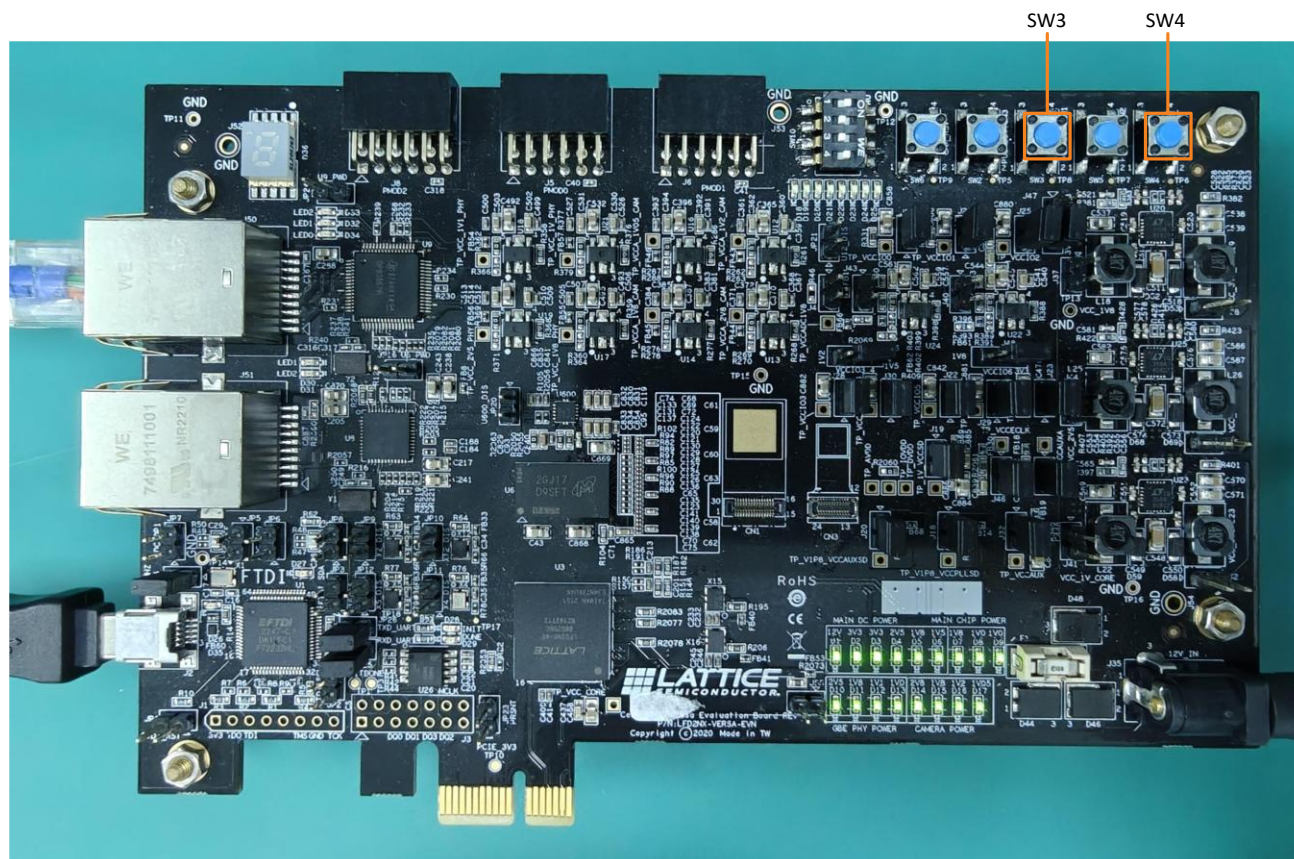


Figure 6.21.-SW3 Reset Button and SW4 PROGRAMN Button

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

- The console must appear with messages as shown in [Figure 6.23](#) and [Figure 6.24](#).
- Hardware LED status as shown in [Figure 6.22](#):
 - D18: Multi-boot configuration status. LED = OFF: Configuration is done. LED = ON: Configuration is in progress.
 - D19: System PLL lock status. LED = OFF: PLL is unlocked. LED = ON: PLL is locked.
 - D20: Ethernet PLL lock status. LED = OFF: PLL is unlocked. LED = ON: PLL is locked.
 - D25 and D24: RGMII link speed. D25 = ON, D24 = OFF: 100 Mbps. Other combinations of LED indication are undefined.

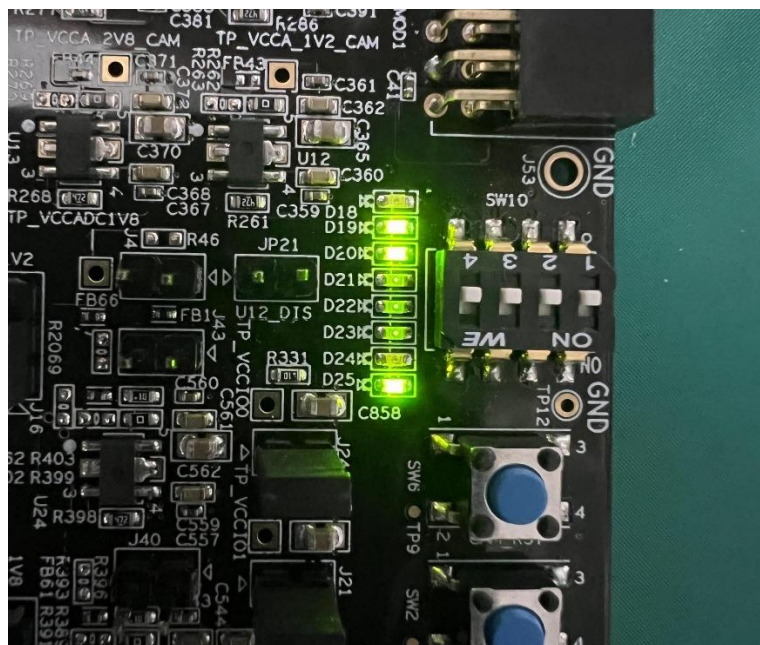


Figure 6.22. LED Status

```
*****
***          GSRD Golden Bootloader LFD2NX-40          ***
*****
Initializing OSPI Controller...
OSPI Init Done
Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 2001
Calculated Firmware CRC value: 2001
CRC matches successfully !!
Jumping to FreeRTOS application ...
```

Figure 6.23. Golden GSRD - Output on UART Terminal for Bootloader and Bare-metal Application Software Starts

```

*****
***          GSRD Golden App LFD2NX-40          ***
*****
Initializing OSPI Controller...
Done
[test_erase4k_prog_read_random] Passed !
PHY Initialization:
bmcrr: 0x00002100
TSEMAC LINK SPEED: 100Mbps
auto_negot_adv: 0x00000101
thousand_base_tx_config: 0x00000000
config_reg3: 0x00000801
phy_status: 0x00006b02
rgmii_ctrl: 0x00000053
rgmii_delay_ctrl: 0x00000077
bmcrr config: 0x00002100
model_number: 0x2000a231
After set - rgmii_ctrl: 0x000000d3
PHY Initialization Complete.
PHY link up.

[Task] [print_rx_data_task] Data Received: 60
 1 80 c2 0 0 e fc 5c ee b7 7e 7d 88 cc 2 7
 4 fc 5c ee b7 7e 7d 4 7 3 fc 5c ee b7 7e 7d
 6 2 e 11 fe 9 0 12 f 1 3 0 1 0 0 fe
 7 0 12 bb 1 0 1 1 0 0 0 0
TX/RX PKT.

```

Figure 6.24. Golden GSRD - Output on UART Terminal for Bare-metal Application Software Running

21. Follow the same steps 5 to step 19 to load the Primary GSRD Software and Bitstream. Refer to [Executables](#) section for the folder and file names.
22. The results of running the Primary GSRD are as shown in [Figure 6.25](#) and [Figure 6.26](#).

```

*****
***          GSRD Primary Bootloader LFD2NX-40          ***
*****
Initializing OSPI Controller...
OSPI Init Done
Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 69e4
Calculated Firmware CRC value: 69e4
CRC matches successfully !!
Jumping to FreeRTOS application ...

```

Figure 6.25. Primary GSRD Bootloader- Output on UART Terminal


```

*****
***          GSRD Primary App LFD2NX-40          ***
*****
Initializing OSPI Controller...
Done
[test_erase4k_prog_read_random] Passed !
PHY Initialization:
bmcr: 0x00002100
TSEMAC LINK SPEED: 100Mbps
auto_nego_adv: 0x00000101
thousand_base_tx_config: 0x00000000
config_reg3: 0x00000801
phy_status: 0x00006b02
rgmii_ctrl: 0x00000053
rgmii_delay_ctrl: 0x00000077
bmcr config: 0x00002100
model_number: 0x2000a231
After set - rgmii_ctrl: 0x000000d3
PHY Initialization Complete.
PHY link up.
TX/RX PKT.
TX/RX PKT.
TX/RX PKT.
TX/RX PKT.
TX/RX PKT.
TX/RX PKT.

[Task] [print_rx_data_task] Data Received: 60
1 80 c2 0 0 e fc 5c ee b7 7e 7d 88 cc 2 7
4 fc 5c ee b7 7e 7d 4 7 3 fc 5c ee b7 7e 7d
6 2 e 11 fe 9 0 12 f 1 3 0 1 0 0 fe
7 0 12 bb 1 0 1 1 0 0 0 0

```

Figure 6.26. Primary GSRD Bare-metal Application Software– Output on UART Terminal

6.5. 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 6 from the [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.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 6.27](#).

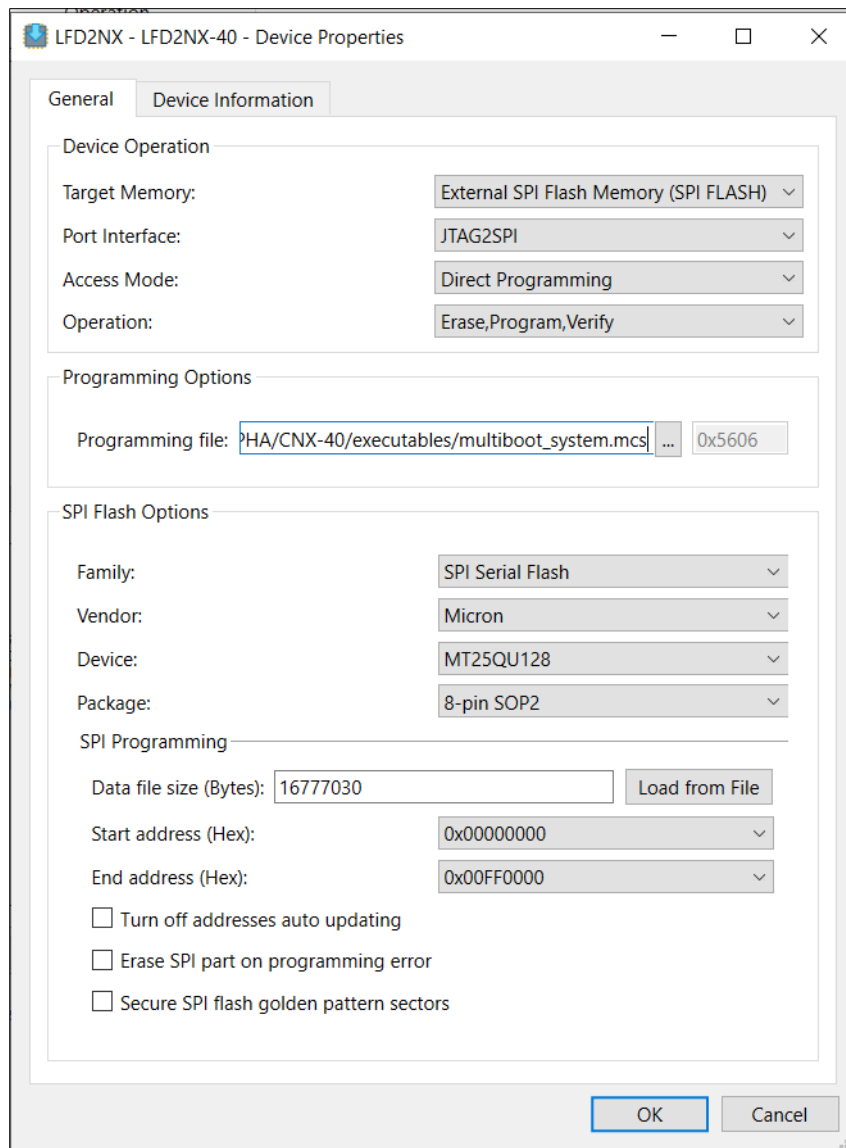


Figure 6.27. Device Properties Window to Setup MCS Programming File

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 1 to 2 minutes.
5. Do not power-cycle the board yet.
6. Program both the **c_primary_appcrc.bin** and **c_golden_appcrc.bin** files as mentioned in the [Programming Standalone Golden or Primary GSRD Bitstream and Application Software](#) section. Make sure to confirm that Starting Address (Hex) for both the binaries as per [Executables](#) section.
7. Once both binaries are programmed, switch off the board.
8. Switch on the board.
9. Set up the UART terminal as mentioned in the [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 6.28](#).
 - It loads the Primary GSRD project.
 - If you press the **SW3** button again, it loads the same Primary GSRD project.

```
*****
***      GSRD Primary Bootloader LFD2NX-40      ***
*****
Initializing OSPI Controller...
OSPI Init Done

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 222b
Calculated Firmware CRC value: 222b

CRC matches successfully !!

Jumping to FreeRTOS application ...
```

Figure 6.28. UART Terminal Output after Power-Cycling Board with MCS and Binaries Programmed

12. For the manual multi-boot, press **SW4 (PROGRAMN)** button mentioned in the [Setting Up the Hardware](#) section on the board.
13. The results on UART terminal are displayed as shown in [Figure 6.29](#). It switches to Golden GSRD project and stays there unless the **SW4 PROGRAMN** button is pressed.

```
*****
***      GSRD Golden Bootloader LFD2NX-40      ***
*****
Initializing OSPI Controller...
OSPI Init Done

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: 3947
Calculated Firmware CRC value: 3947

CRC matches successfully !!

Jumping to FreeRTOS application ...

*****
***      GSRD Golden App LFD2NX-40      ***
*****
Initializing OSPI Controller...
Done
[test_erase4k_prog_read_random] Passed !

PHY Initialization:
bmcr: 0x00002100
TSEMAC LINK SPEED: 100Mbps
auto_nego_adv: 0x00000101
thousand_base_tx_config: 0x00000000
config_reg3: 0x00000801
phy_status: 0x00006b02
rgmii_ctrl: 0x00000053
rgmii_delay_ctrl: 0x00000077
bmcr config: 0x00002100
model_number: 0x2000a231
After set - rgmii ctrl: 0x000000d3
```

Figure 6.29. Switches to Golden GSRD Upon SW4 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_gsrdsimpl_1.bit soc_golden_gsrdsimpl_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 bare-metal application software binary files using Lattice Propel SDK and verifying the primary or golden image and SoC design are built correctly	c_primary_appcrc.bin c_golden_appcrc.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 the Propel SDK application.

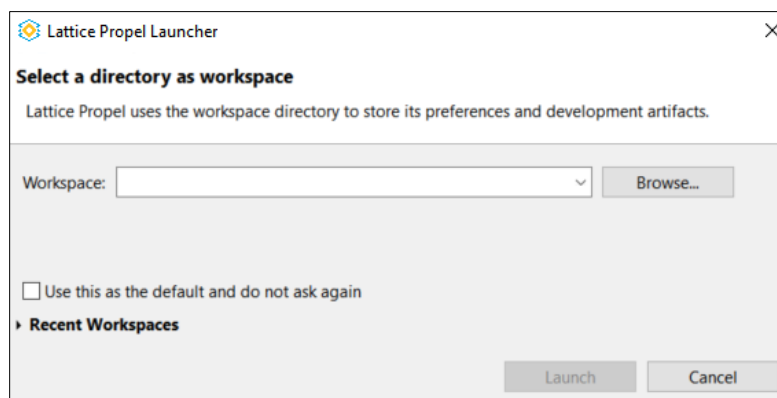


Figure 7.1. Propel SDK Launcher

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.

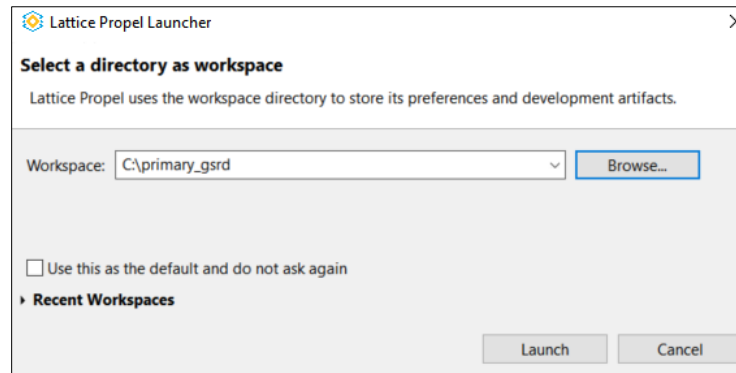


Figure 7.2. Provide Name for the Workspace Directory

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

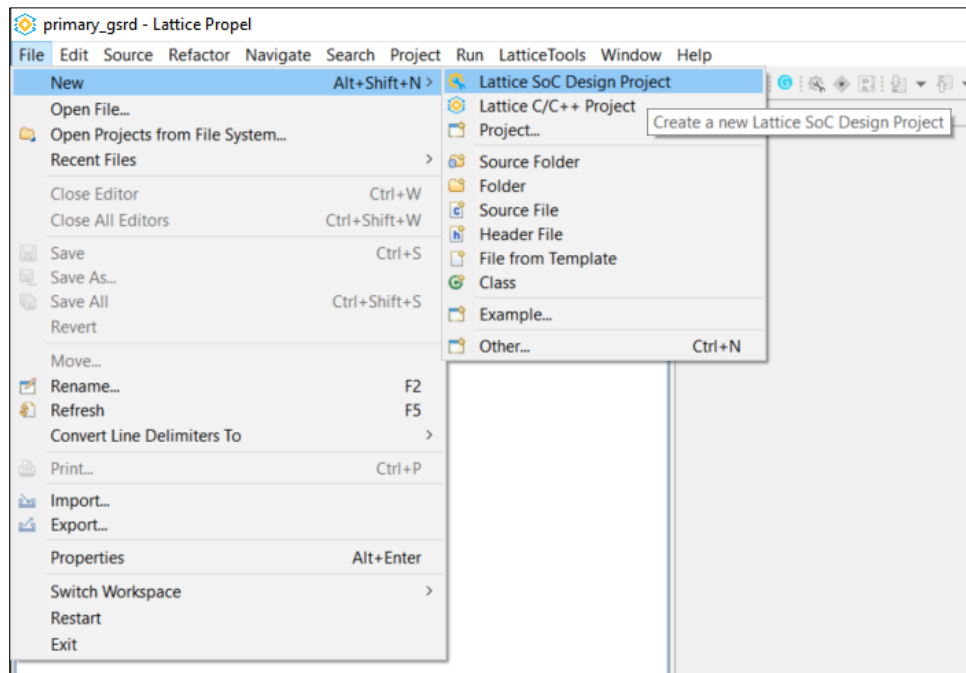


Figure 7.3. Creating Lattice SoC Design Project

5. On the SoC Project window:
 - a. Enter a name for your SoC project.
 - b. Enable **Board** option.
 - c. From **Board Select** section, select **Certus-NX Versa Evaluation Board**.
 - d. From **Processor** section, select **RISC-V MC**.
 - e. From **Template Design** section, select **GHRD SoC Project LFD2NX** as shown in Figure 7.4.

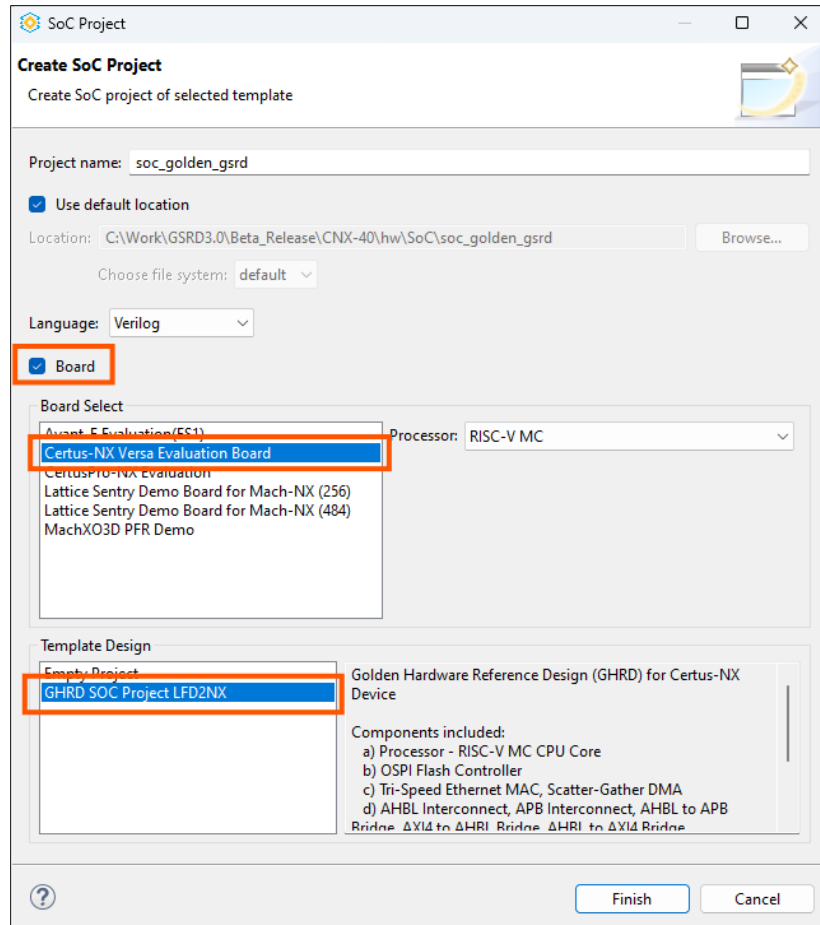


Figure 7.4. SoC Project Window

- 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.

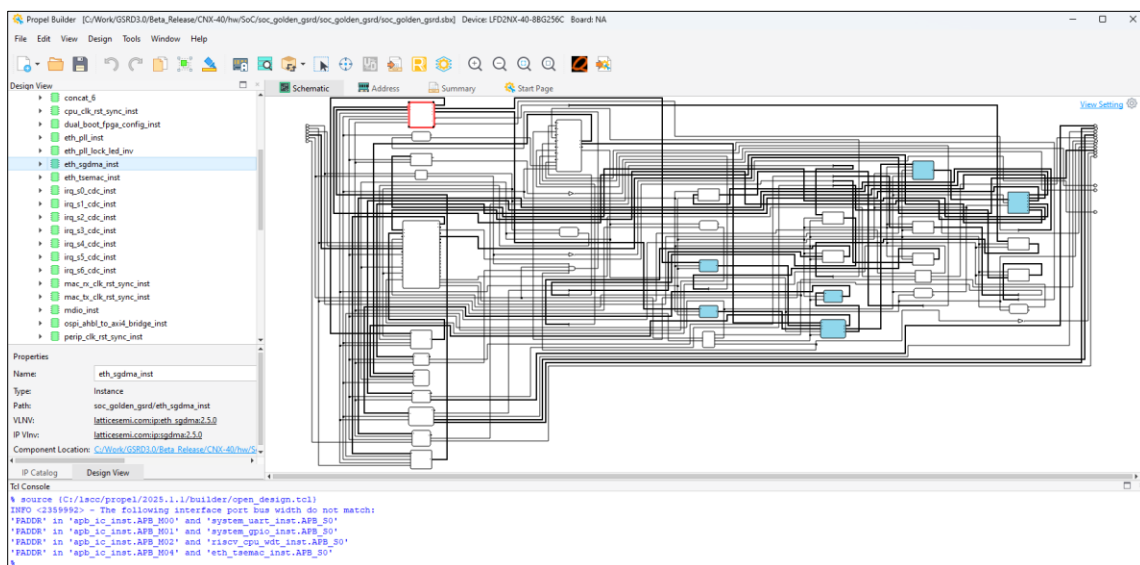


Figure 7.5. Launched SoC in Propel Builder

7. 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 should see no error in the **TCL Console** window. The following **INFO** and **WARNINGS** are expected.

```
Tcl Console
$ sbp_design drc
INFO <2359136> - Start: sbp_design drc.
INFO <2359992> - Dangling inputs are set to default value (ARUSER=0,AWUSER=0,WUSER=0) in sgdma_bd_axi4_to_ahbl_bridge_inst.AXI4_S,sgdma_mm_axi4_to_ahbl_bridge_inst.AXI4_S
INFO <2359992> - Dangling inputs are set to default value (TDEST=0,TID=0) in tsemac_rx_axis_fifo_inst.AXI4S_S
WARNING <2359991> - The bus interface port eth_sgdma_inst.MM2S_IRQ/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port eth_sgdma_inst.S2MM_IRQ/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port eth_tsemac_inst.INTR/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_wdt_inst.INTR/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S1/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S2/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S4/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S5/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S3/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S6/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port riscv_cpu_inst.IRQ_S0/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port eth_tsemac_inst.MDIO/mdc is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port system_gpio_inst.INTR/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port system_opspi_fc_inst.INTR/IRQ is not connected as a part of the interface connection.
WARNING <2359991> - The bus interface port system_uart_inst.INT_M0/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

8. Click on **Design > Generate** or click on the icon below as highlighted in screenshot.



Figure 7.8. Generate in Propel Builder

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

```
Tcl Console
$ sbp_design pge sge -i (C:\Work\GSRD3.0\Beta_Release\CNX-40\hw\SoC\soc_golden_gsrdsoc_golden_gsrdsoc_golden_gsrdsoc.sbx) -o (C:\Work\GSRD3.0\Beta_Release\CNX-40\hw\SoC\soc_golden_gsrdsoc_golden_gsrdsoc\...) --nc
WARNING <2359991> - (PGE SoCDesign) eth_sgdma_inst_MM2S_IRQ remains unconnected
WARNING <2359991> - (PGE SoCDesign) eth_sgdma_inst_S2MM_IRQ remains unconnected
WARNING <2359991> - (PGE SoCDesign) eth_tsemac_inst_INTR remains unconnected
WARNING <2359991> - (PGE SoCDesign) eth_tsemac_inst_MDIO remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_TIMER_IRQ_M0 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S0 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S1 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S2 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S3 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S4 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S5 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_inst_IRQ_S6 remains unconnected
WARNING <2359991> - (PGE SoCDesign) riscv_cpu_wdt_inst_INTR remains unconnected
WARNING <2359991> - (PGE SoCDesign) system_gpio_inst_INTR remains unconnected
WARNING <2359991> - (PGE SoCDesign) system_opspi_fc_inst_INTR remains unconnected
WARNING <2359991> - (PGE SoCDesign) system_uart_inst_INT_M0 remains unconnected
INFO <2359992> - (PGE FileExistence) No available driver for fpga_config
INFO <2359992> - (PGE FileExistence) No available driver for watchdog_timer
$
```

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 bare-metal application projects in the next sections.

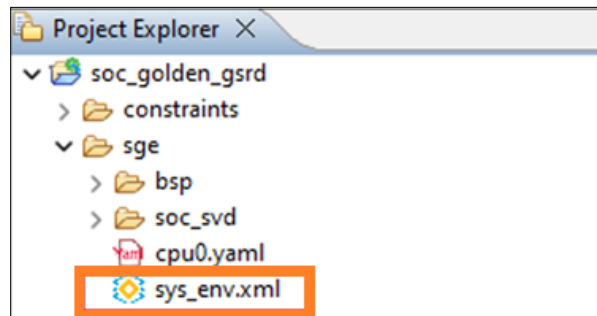


Figure 7.10. `sys_env.xml` File Created

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](#).

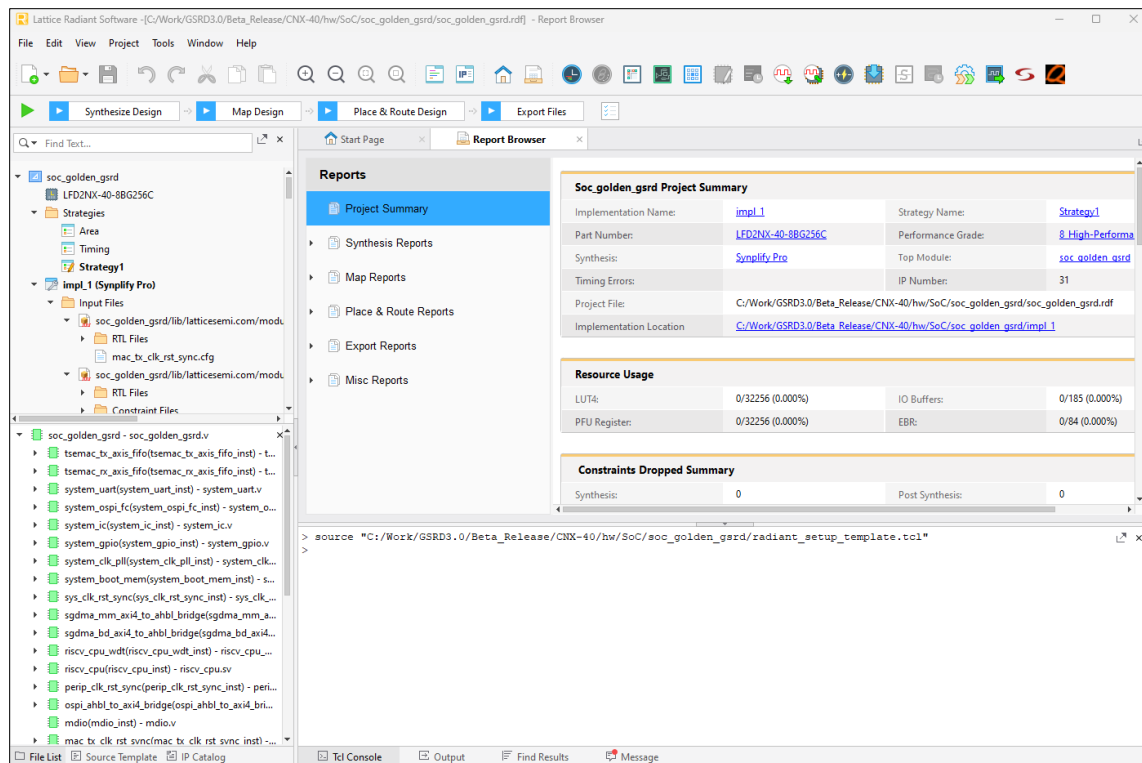


Figure 7.12. Lattice Radiant Window

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

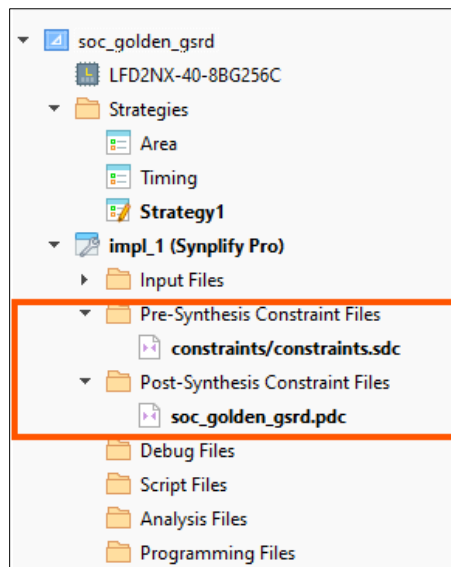


Figure 7.13. Lattice Radiant Window

4. The **Place and Route** strategy is used for the GSRD testing.

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

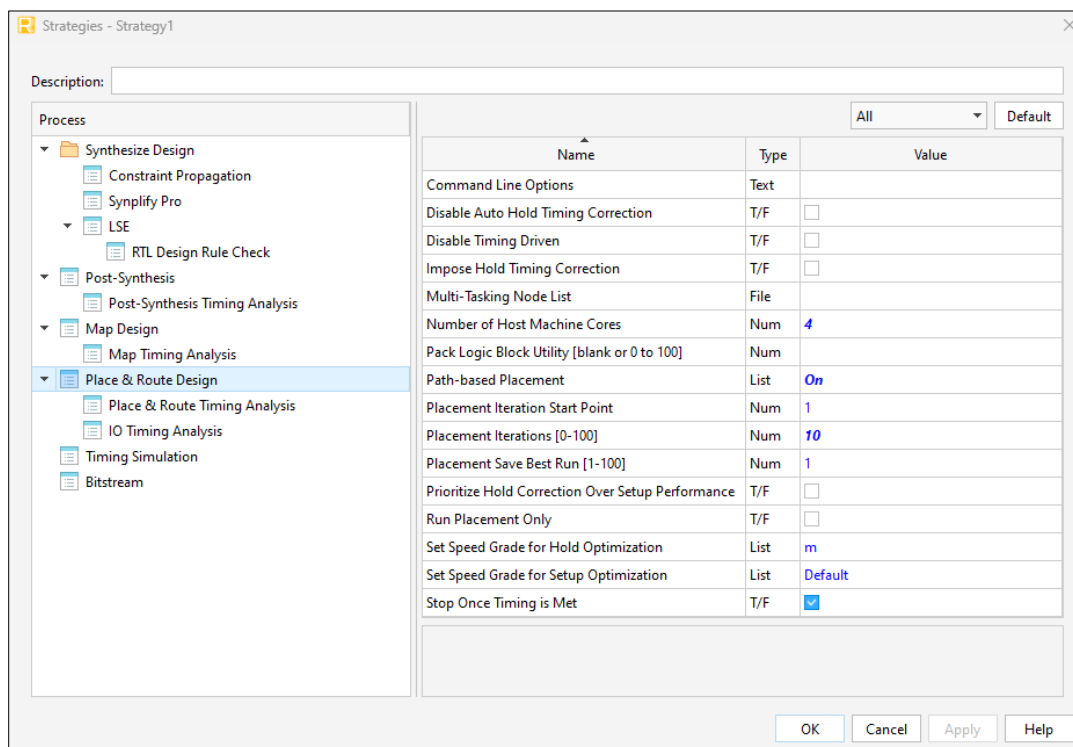


Figure 7.14. Strategy Used for Certus-NX GSRD Testing

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

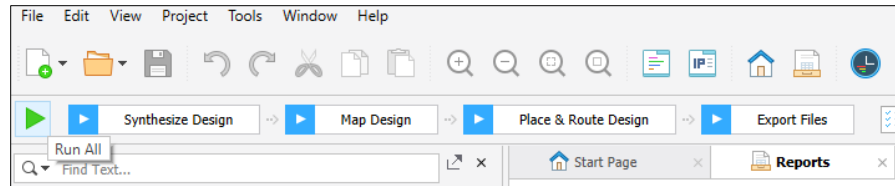


Figure 7.15. Generating the Bit File

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_primary_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 forces the 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 the ten placement results and select the best timing result.

7.3. Building the Hello World Program Using the 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 Propel SDK application.
3. Proceed to the Propel SDK window. Click on **File > New > Lattice C/C++ Project**.

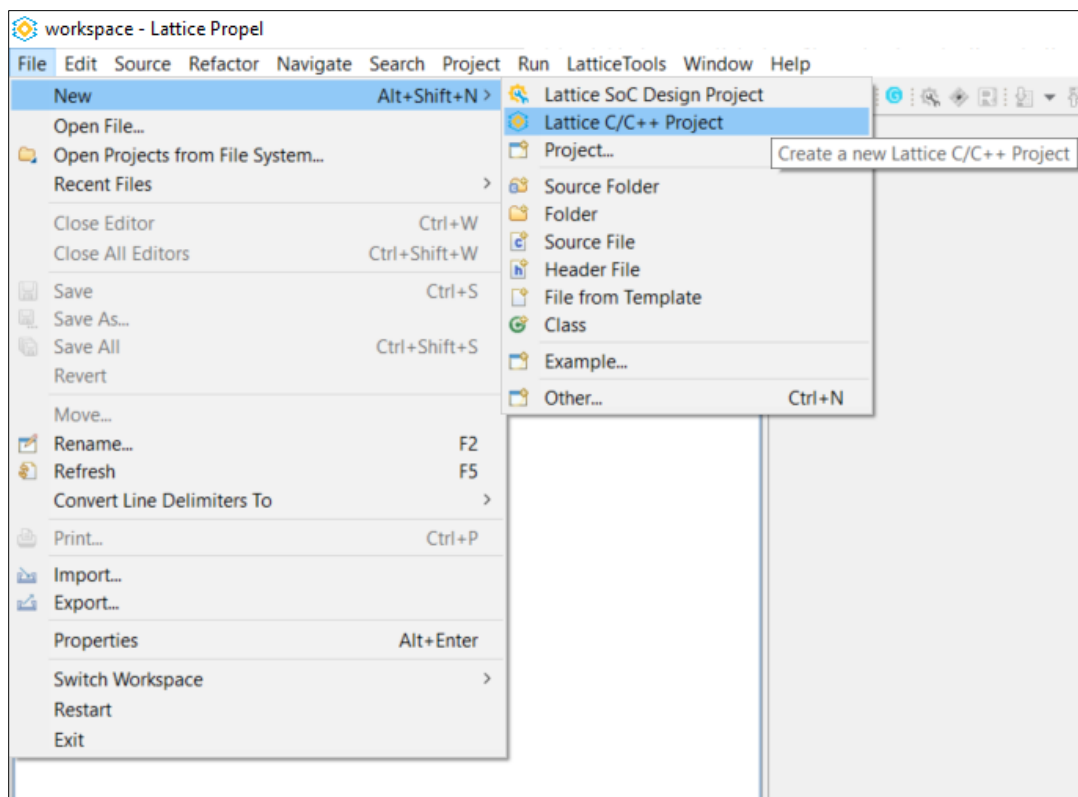


Figure 7.17. Creating Lattice C/C++ Project for Hello World

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:\Certus-NX\soc_gsrdsn\cnx\sgs\sys_env.xml Browse...

Select processor core to create C/C++ Project

Core selected: riscv_cpu_inst

Project type: C

System information

Device Family	CPU Name	Instance Name
LFD2NX	riscv_mc	riscv_cpu_inst

Select Example Application

GSRD CNX Application	Example-HelloWorld-blink-uart 1.Led blink. The corresponding SoC project should include gpio instance. 2.HelloWorld
GSRD CNX Bootloader	
Hello World Project	
Mtimer Project	
Hardware Interrupt Project	

Project name: riscv_mc_helloworld

☒ Use default location

Location: C:\Certus-NX\riscv_mc_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**.

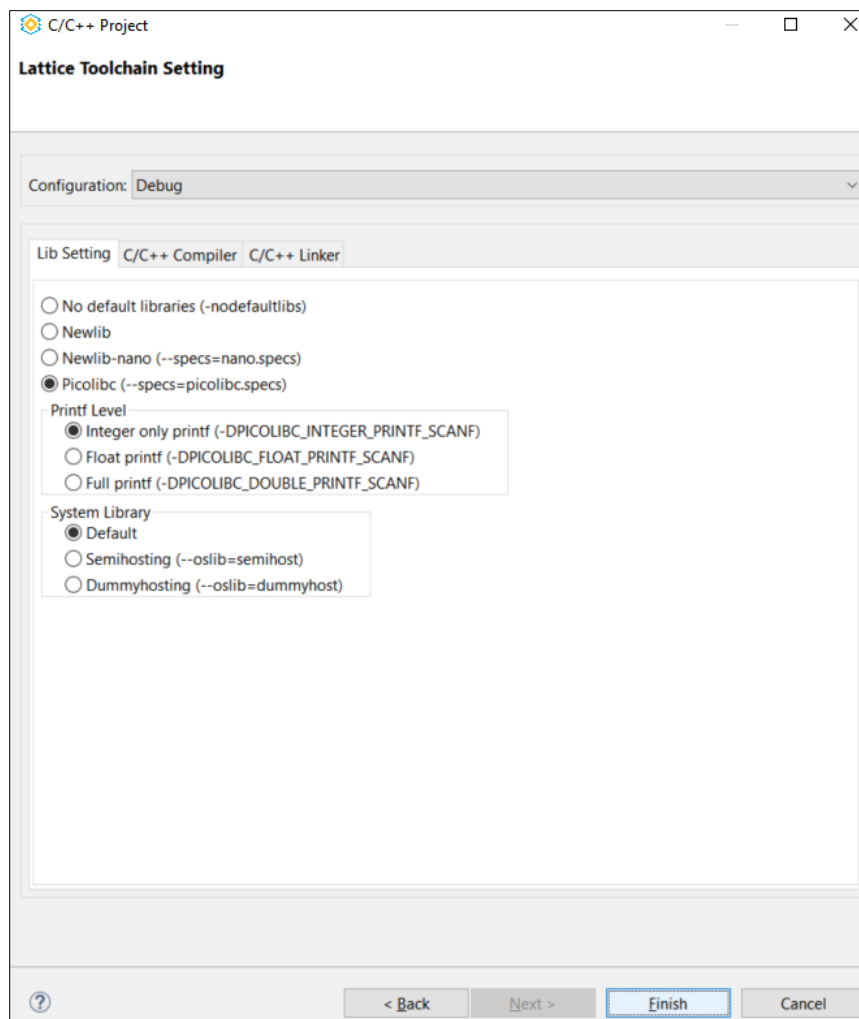


Figure 7.19. C/C++ Lattice Toolchain Setting

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

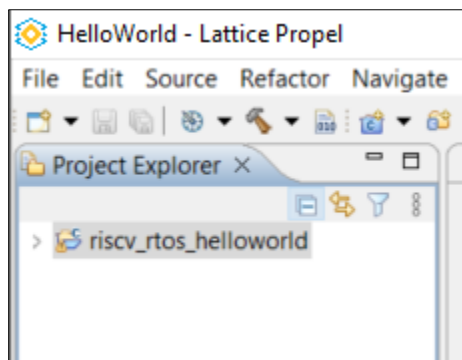


Figure 7.20. Hello World C Project Created

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

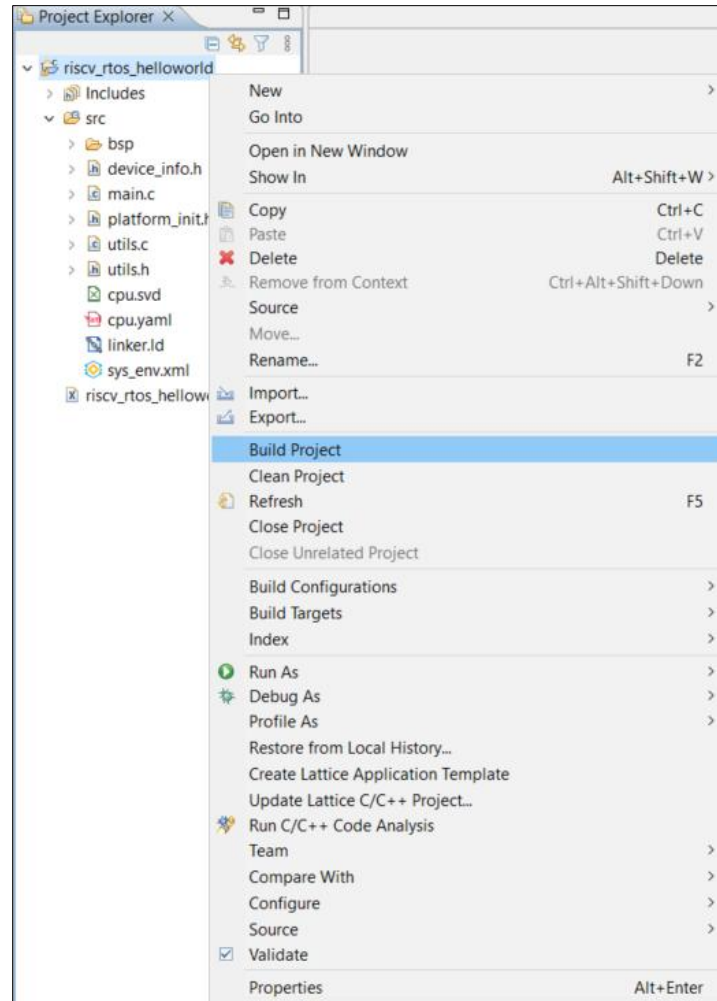


Figure 7.21. Build Hello World Project

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

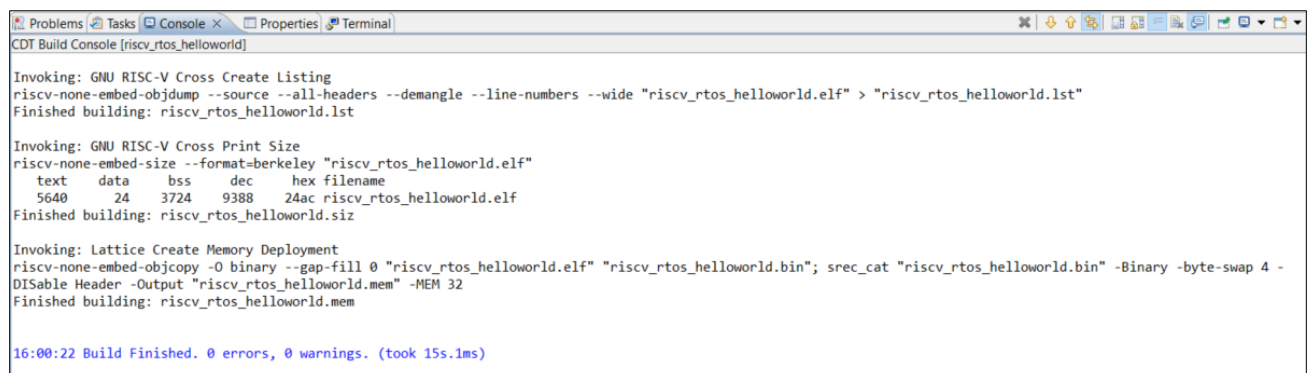


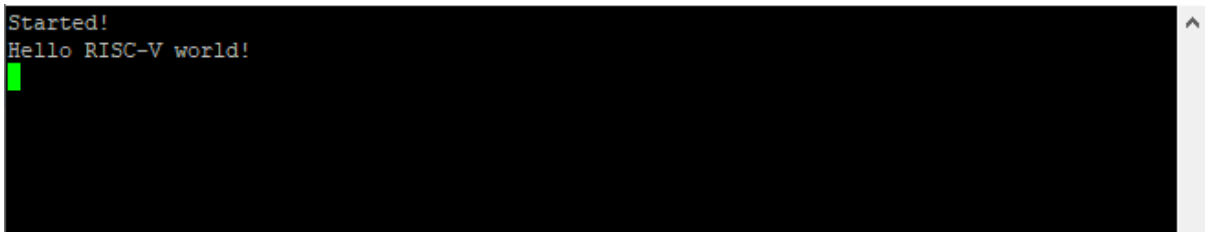
Figure 7.22. Hello World, Project Build Console Output

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 the 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.

You must see the *Hello World* message as shown in [Figure 7.23](#)

12. Figure 7.23.



```
Started!
Hello RISC-V world!
```

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 of the [Lattice Propel 2025.1 SDK User Guide \(FPGA-UG-02234\)](#).

7.4. Building the Bare-metal Bootloader using 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**.

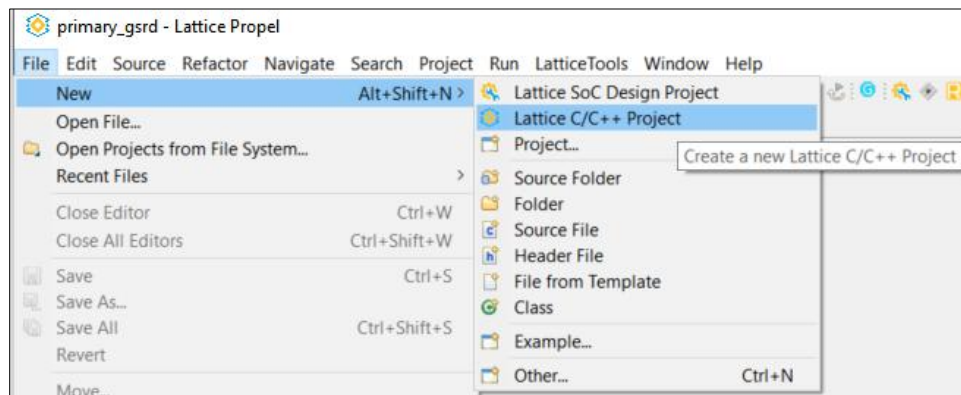
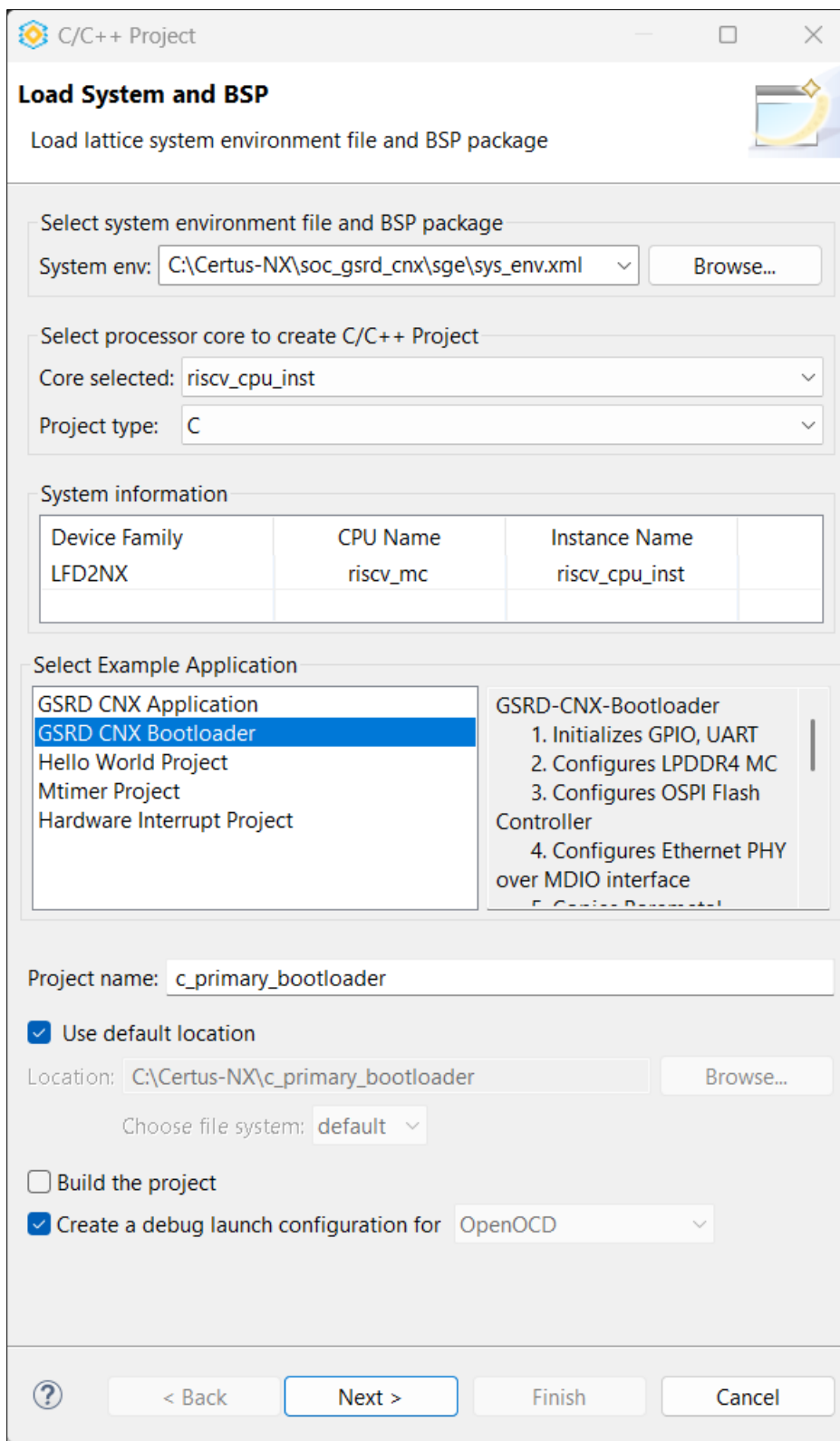


Figure 7.24. Creating Lattice C/C++ Project for Bootloader

3. In C/C++ Project window, it automatically selects the generated `sys_env.xml` file. In the *Select Example Application* section, select the **GSRD CNX 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:\Certus-NX\soc_gsrdsn\sgs\sys_env.xml Browse...

Select processor core to create C/C++ Project

Core selected: riscv_cpu_inst

Project type: C

System information

Device Family	CPU Name	Instance Name
LFD2NX	riscv_mc	riscv_cpu_inst

Select Example Application

GSRD CNX Application
GSRD CNX Bootloader
Hello World Project
Mtimer Project
Hardware Interrupt Project

GSRD-CN-Bootloader
1. Initializes GPIO, UART
2. Configures LPDDR4 MC
3. Configures OSPI Flash
Controller
4. Configures Ethernet PHY
over MDIO interface
5. Configures...

Project name: c_primary_bootloader

☒ Use default location

Location: C:\Certus-NX\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](#). Click **Finish**.

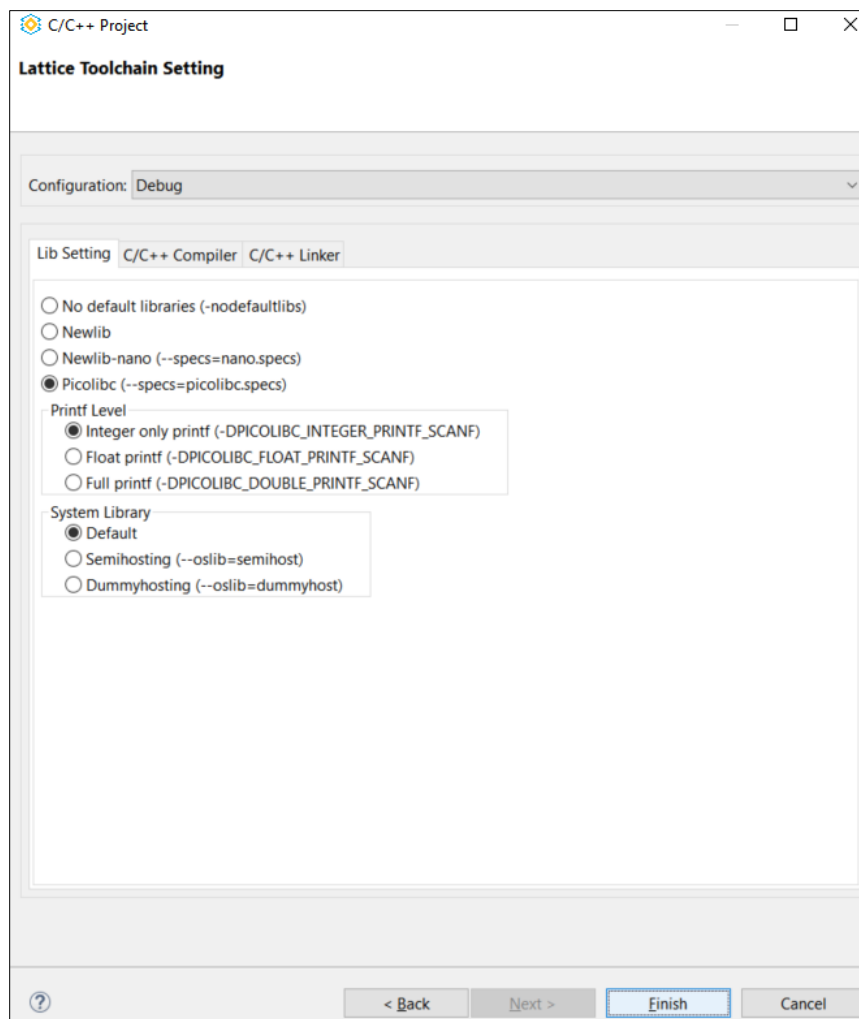


Figure 7.26. C/C++ Lattice Toolchain Setting

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

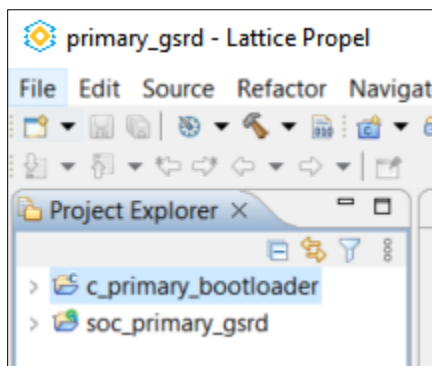


Figure 7.27. Bootloader C Project Created

- 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 bare-metal application software image in the SPI flash. Refer to [Appendix B. Using Different SPI Flash Manufacturer in GSRD Bare-metal Bootloader](#) to modify the Octal SPI 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`

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

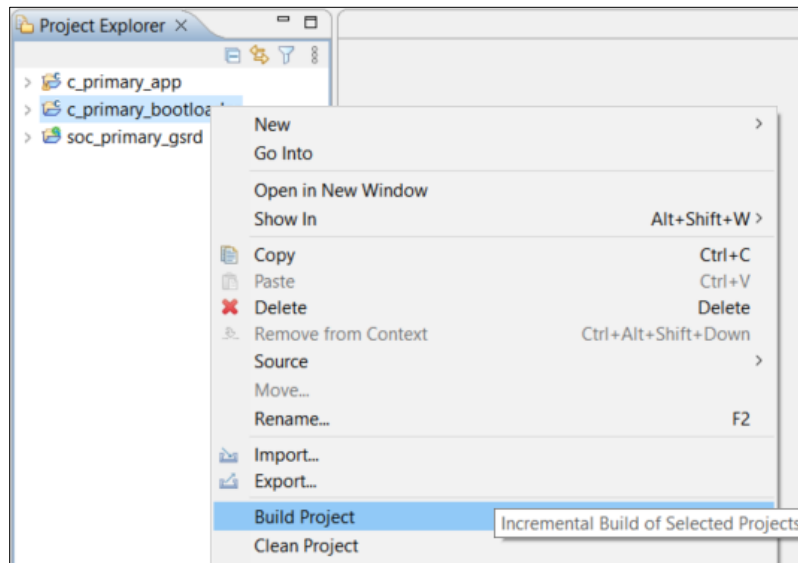


Figure 7.29. Build Bootloader Project

- 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=rv32imc -mabi=ilp32 -msmall-data-limit=8 -mno-save-restore -O2 -fmessage-length=0 -fsigned-char -ffunction-sections -fdata-sections -T
"C:/lsc/propel/2025.1/workspace/cnx/c_primary_bootloader/src/linker.ld" -nostartfiles -Wl,-Map,"c_primary_bootloader.map" --specs=picolibc.specs -DPICOLIBC_INTEGER_PRINTF_SCANF -o
"c_primary_bootloader.elf" ./src/bsp/driver/uart/uart.o ./src/bsp/driver/tse_mac/ethernet.o ./src/bsp/driver/sgdma/sgdma.o ./src/bsp/driver/riscv_mc/cache.o ./src/bsp/driver/riscv_mc/crt0.o
./src/bsp/driver/riscv_mc/debug.o ./src/bsp/driver/riscv_mc/exit.o ./src/bsp/driver/riscv_mc/interrupt.o ./src/bsp/driver/riscv_mc/iob.o ./src/bsp/driver/riscv_mc/pic.o
./src/bsp/driver/riscv_mc/reg_access.o ./src/bsp/driver/riscv_mc/timer.o ./src/bsp/driver/riscv_mc/util.o ./src/bsp/driver/octal_spi_controller/octal_spi_controller.o ./src/bsp/driver/gpio/gpio.o
./src/bsp/reg_test.o ./src/crc16.o ./src/debuglib.o ./src/main.o ./src/utlis.o
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"
Invoking: GNU RISC-V Cross Print Size
riscv-none-embed-size --format=berkeley "c_primary_bootloader.elf"
Invoking: Lattice Create Memory Deployment
riscv-none-embed-objcopy -O binary --gap-fill 0 "c_primary_bootloader.elf" "c_primary_bootloader.bin"; src_cat "c_primary_bootloader.bin" -Binary -byte-swap 4 -DISable Header -Output
"c_primary_bootloader.mem" -MEM 32
text data bss dec hex filename
12580 28 3320 15928 3e38 c_primary_bootloader.elf
Finished building: c_primary_bootloader.siz
Finished building: c_primary_bootloader.lst
Finished building: c_primary_bootloader.mem

09:43:58 Build Finished. 0 errors, 5 warnings. (took 8s.885ms)
```

Figure 7.30. Bootloader Build Project Console Output

- 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 the System Memory in Propel Builder SoC design. This can be done through ECO editor or directly from the Propel Builder System Memory user interface.

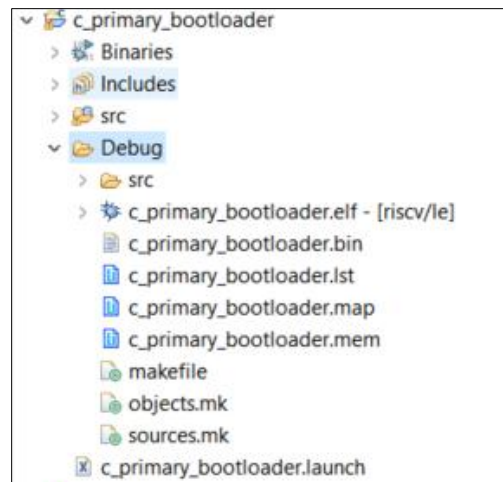


Figure 7.31. Bootloader Binary Created

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

```

*****
***          GSRD Primary Bootloader LFD2NX-40          ***
*****
Initializing OSPI Controller...
OSPI Init Done

Reading and verifying firmware CRC value ...
Embedded LPDDR CRC value: c7bc
Calculated Firmware CRC value: ffff

ERROR: CRC Mis-matched!!!

```

Figure 7.32. Bootloader Boots Up without Bare-metal 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 step 1 to 11 to create the golden bootloader project.

7.5. Building the Bare-metal Application Software using Lattice Propel SDK (Primary and Golden)

To build the bare-metal application software using Propel SDK, perform the following:

1. Similarly, create the bare-metal application software C Project. Go to **File > New > Lattice C/C++ Project**.

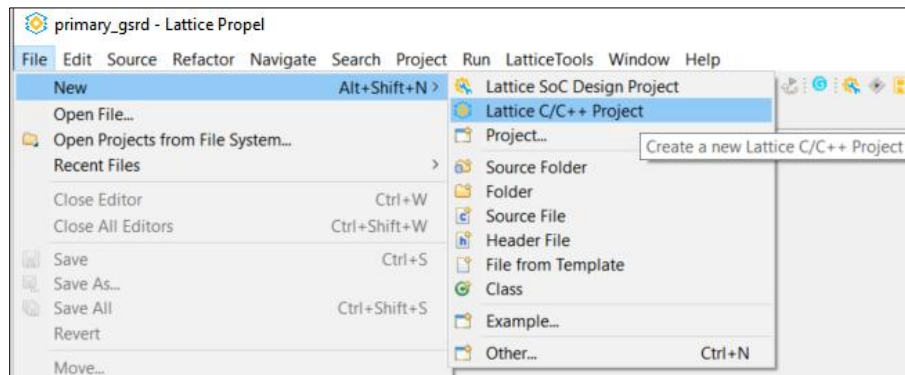


Figure 7.34. Creating C/C++ Project for bare-metal application software

2. In the *Select Example Application*, select the **GSRD CNX BM APP** project. Provide the project name as shown in Figure 7.36.

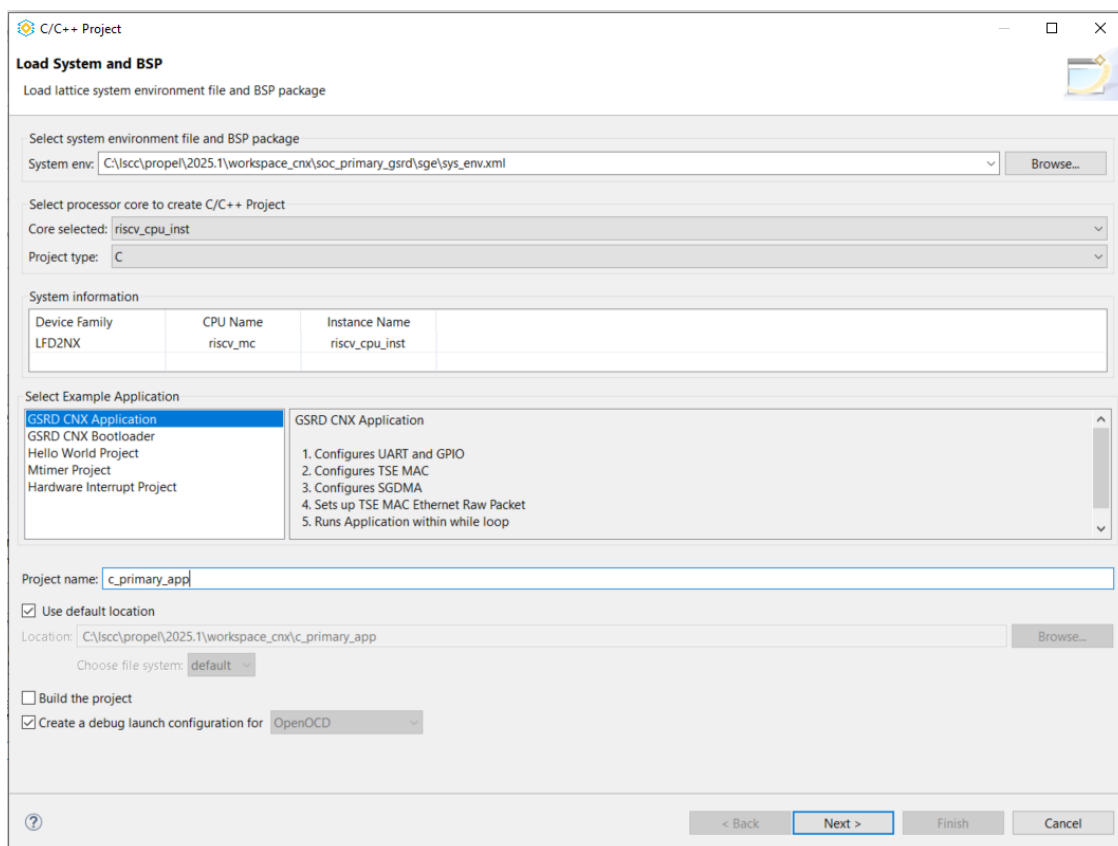


Figure 7.35. FreeRTOS C/C++ Selection

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.

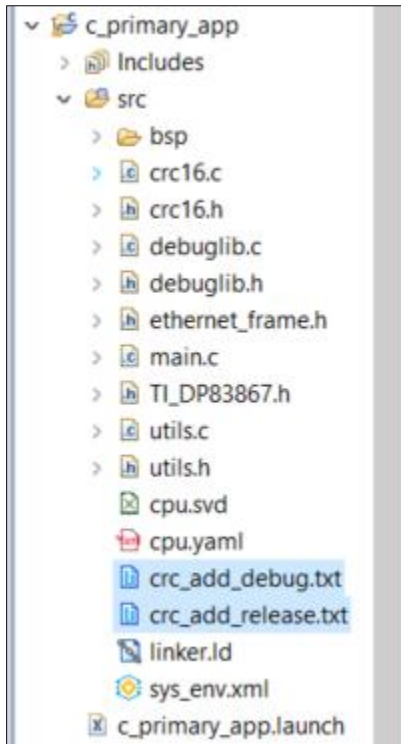


Figure 7.36. Bare-metal Application Software Project Created

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

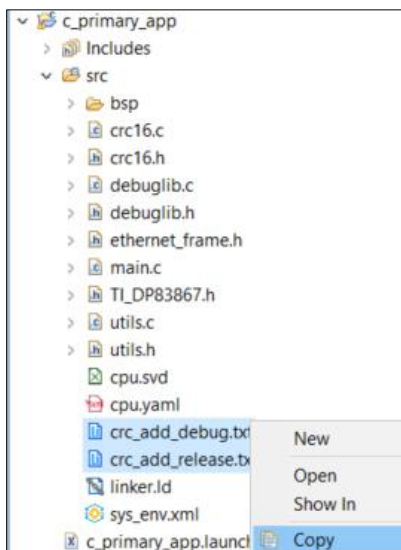


Figure 7.37. Copy the CRC and Add Files

5. Paste it in your main `c_primary_app` project as shown in Figure 7.38.

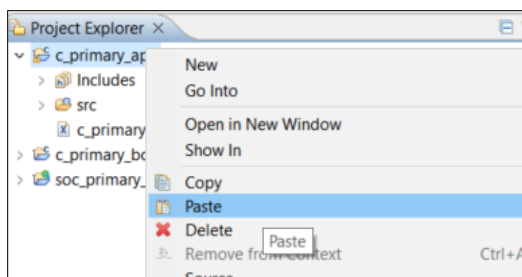


Figure 7.38. Paste in Bare-metal Application C Project

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

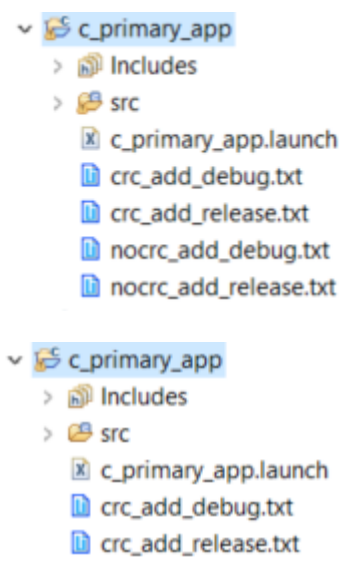


Figure 7.39. Copied Text Files

7. 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 bare-metal application software 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`

8. 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 0x000 0x6000          # fill code area with 0xff
7-crop 0x000 0x5ffe              # just keep code area for CRC calculation below
8-CRC16_Big_Endian 0x5ffe -CCITT  # calculate big endian CCITT CRC16 at given address.
9-crop 0x5ffe 0x6000            # keep the CRC itself
10
11#second: add application file
12..\Debug\c_primary_app.bin -Binary
13-fill 0xFF 0x000 0x5ffe          # 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

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

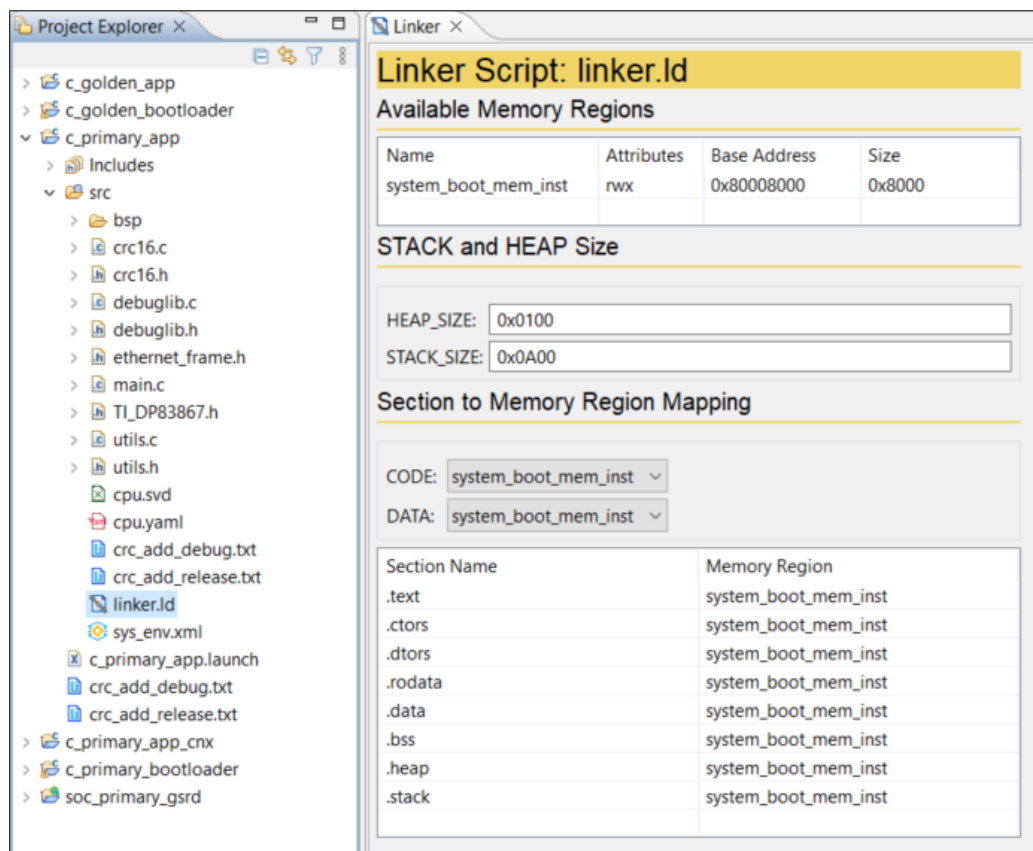


Figure 7.42. Open Linker.Ld File

10. Update the MEMORY org address to **0x80008000** for bare-metal application software to run from System Memory.

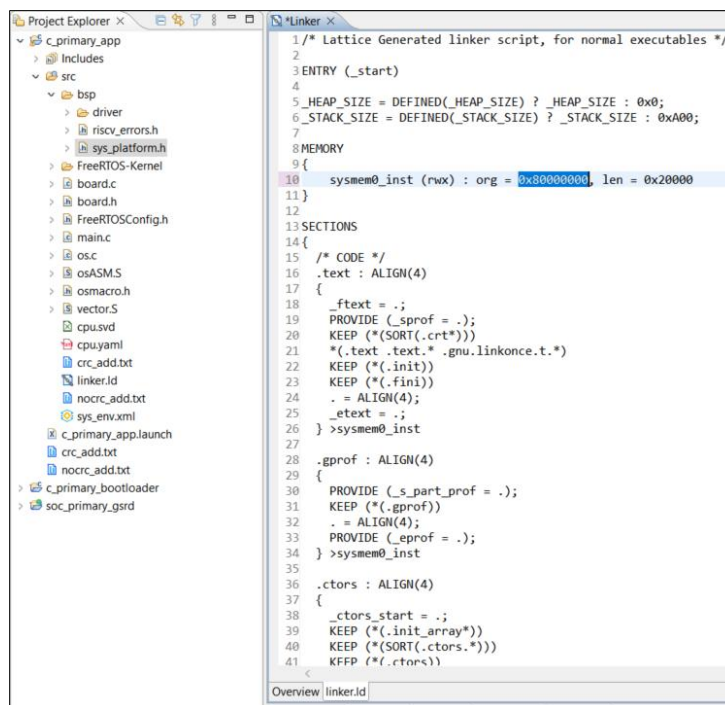


Figure 7.43. Update Linker.Id File

11. Press **Ctrl + s** to save the change as shown in Figure 7.44.

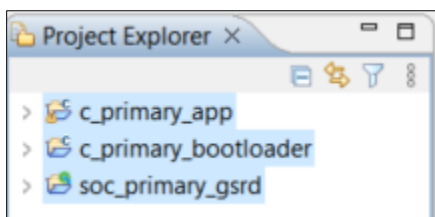


Figure 7.44. Workspace

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

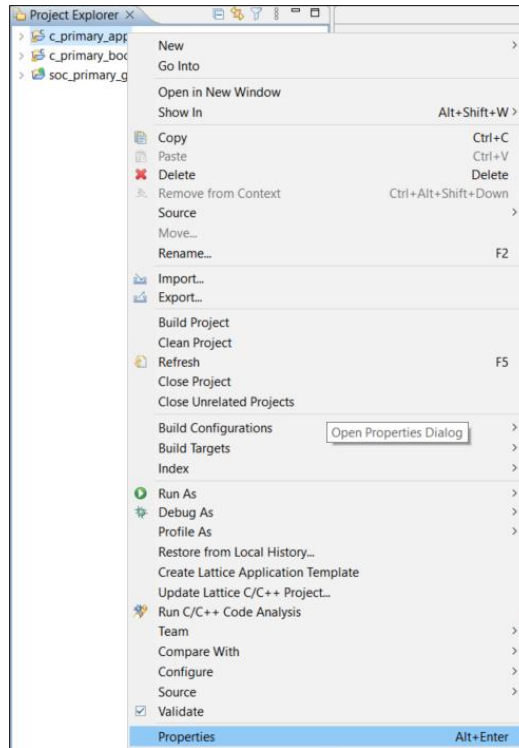


Figure 7.45. Properties

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** and select **Release**. Click **Set Active**, then click **OK**.

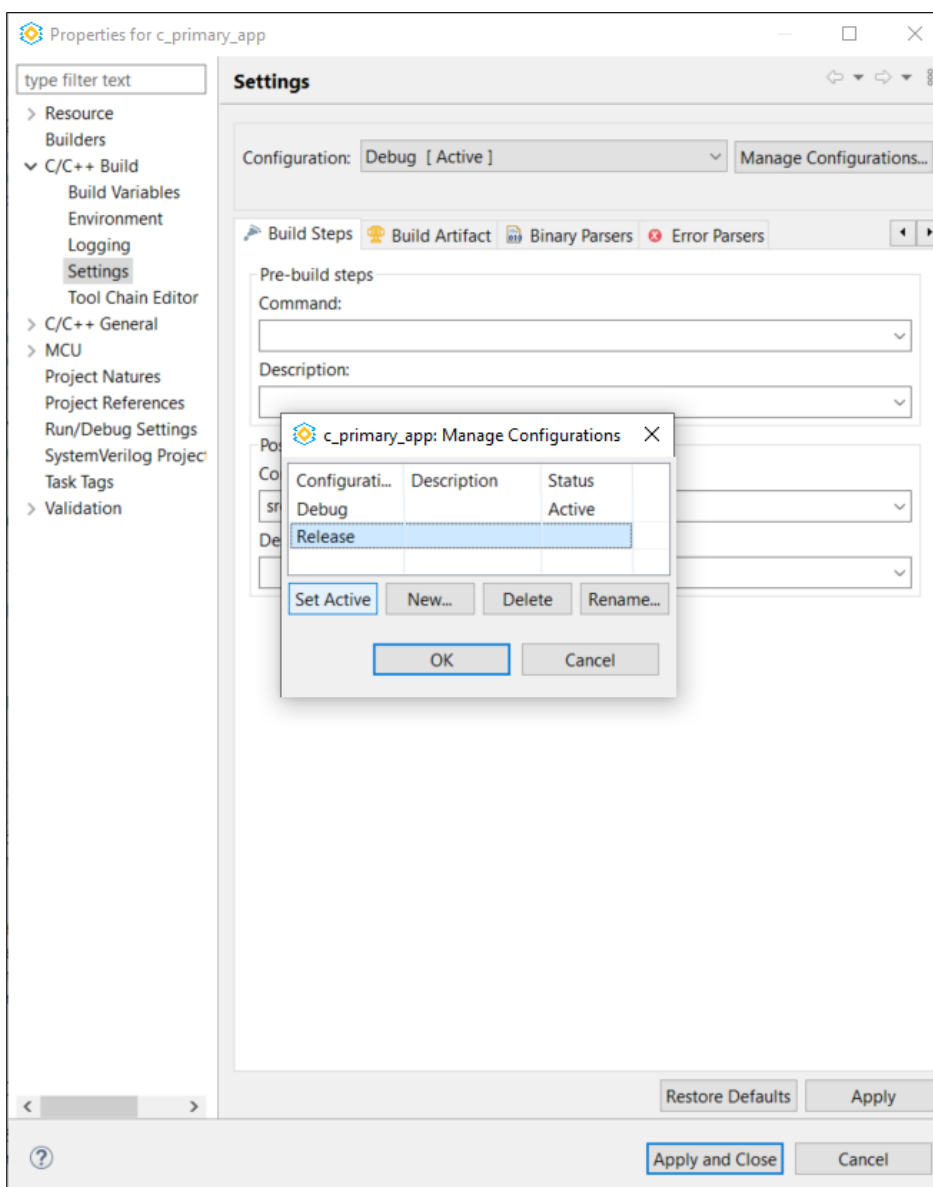


Figure 7.46. Set Release as Active Configuration

- 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"
```

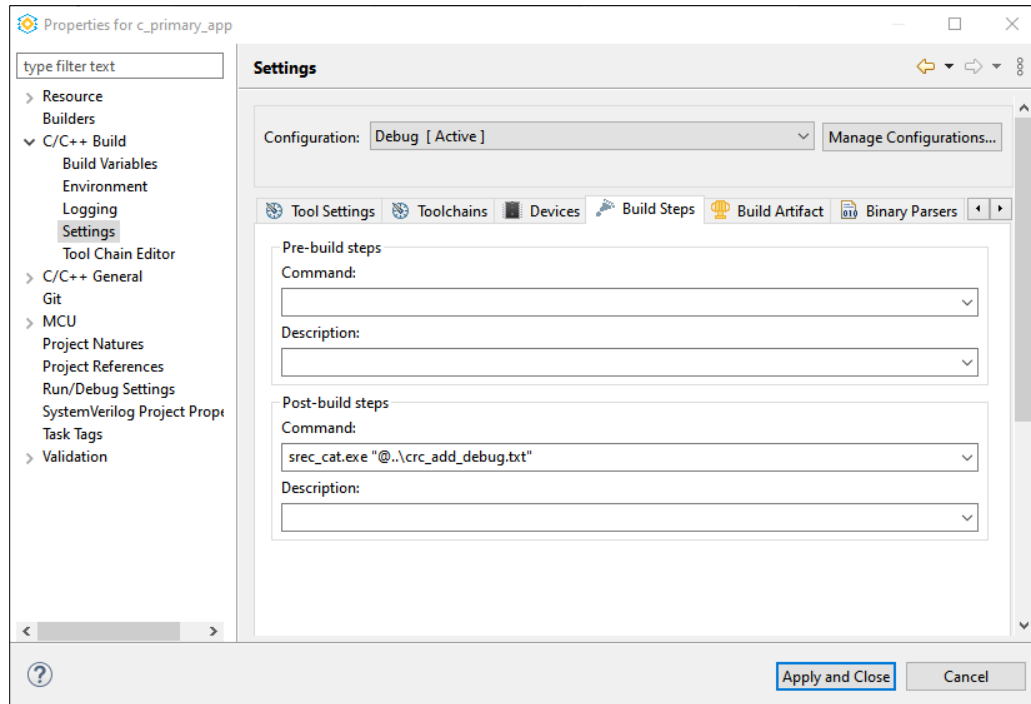


Figure 7.47. Adding Post Build Step for Bare-metal Application CRC Binary Append (Windows)

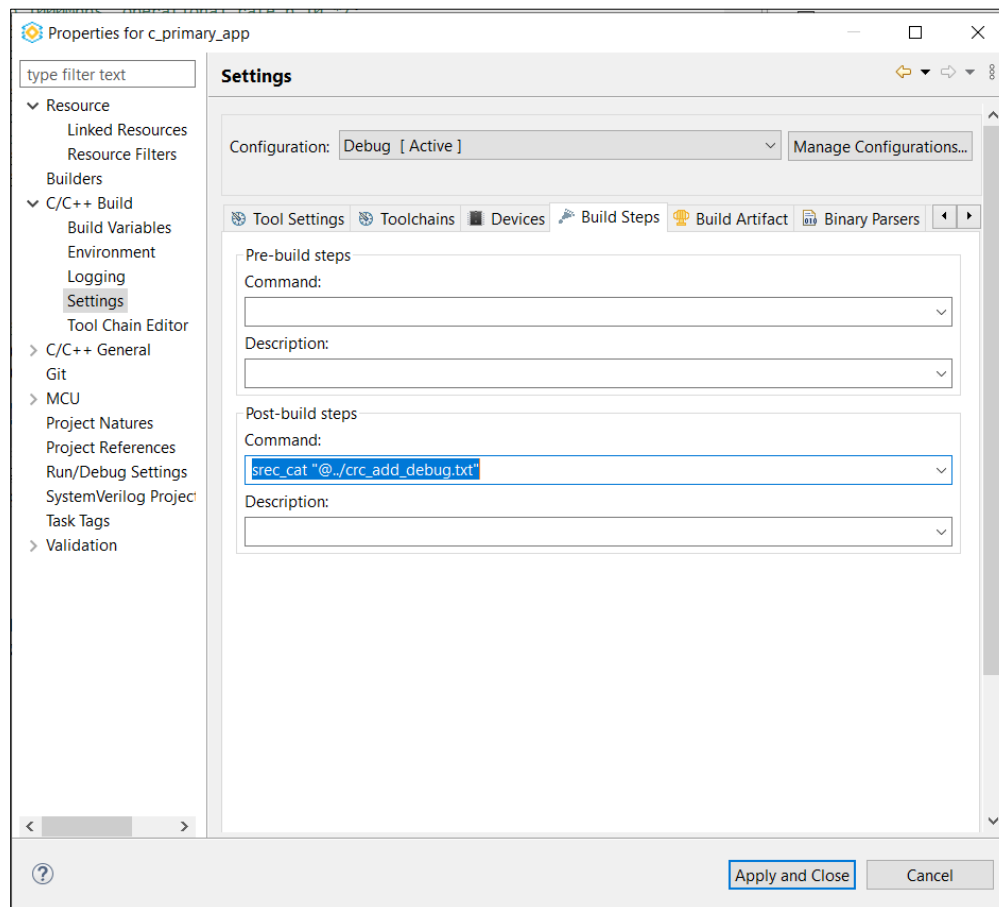


Figure 7.48. Adding Post Build Step for Bare-metal Application CRC Binary Append (Linux)

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

```
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_golden_app.bin -Binary
6-fill 0xFF 0x000 0x6000          # fill code area with 0xff
7-crop 0x000 0x5ffe              # just keep code area for CRC calculation below
8-CRC16_Big_Endian 0x5ffe -CCITT  # calculate big endian CCITT CRC16 at given address.
9-crop 0x5ffe 0x6000            # keep the CRC itself
10
11#second: add application file
12..\Debug\c_golden_app.bin -Binary
13-fill 0xFF 0x000 0x5ffe        # fill code area with 0xff
14
15# generate a Binary file
16-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**.

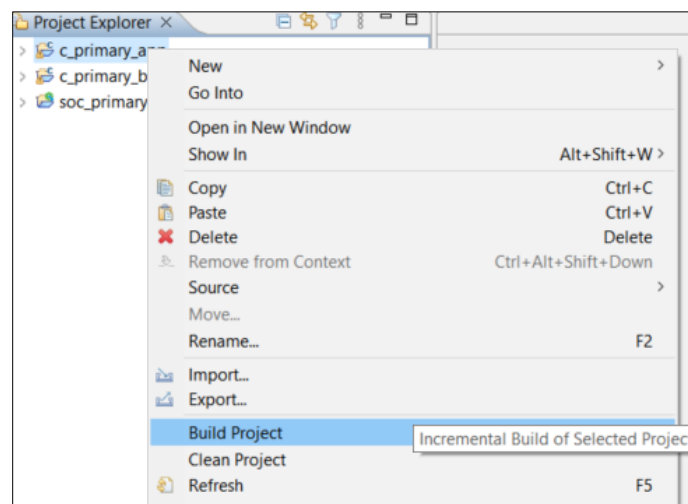


Figure 7.50. Build c_primary_app C/C++ Project

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

Note: The warnings can be ignored.

```
Building target: c_primary_app.elf
Invoking: GNU RISC-V Cross C Linker
riscv-none-embed-gcc -march=rv32imc -mabi=ilp32 -msmall-data-limit=8 -mno-save-restore -O0 -fmessage-length=0 -fsigned-char -ffunction-sections -fdata-sections -T "C:/lsc/propel/2025.1/workspace/cnx/c_primary_
Finished building target: c_primary_app.elf

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

Invoking: GNU RISC-V Cross Print Size
riscv-none-embed-size --format=berkeley "c_primary_app.elf"
   text  data  bss  dec  hex filename
18184   124  14292  32600  7f58 c_primary_app.elf
Finished building: c_primary_app.size

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

srec_cat.exe "@..\c_crc_add_debug.txt"

10:15:33 Build Finished. 0 errors, 14 warnings. (took 19s.997ms)
```

Figure 7.51. Bare-metal Application 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](#).

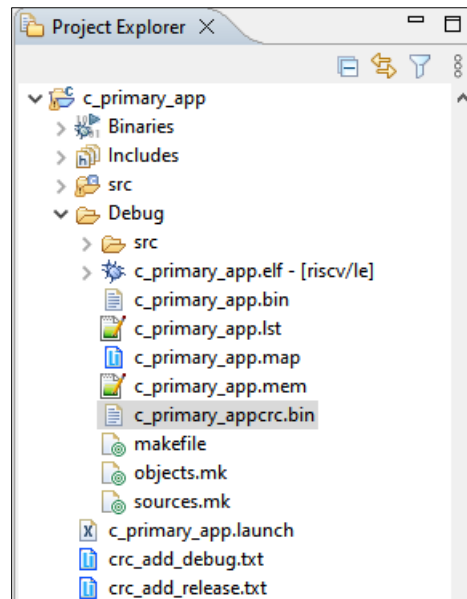


Figure 7.52. Bare-metal App Binaries Created with CRC

21. For programming into the flash and confirming both primary bootloader and bare-metal application project 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 step 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](#).

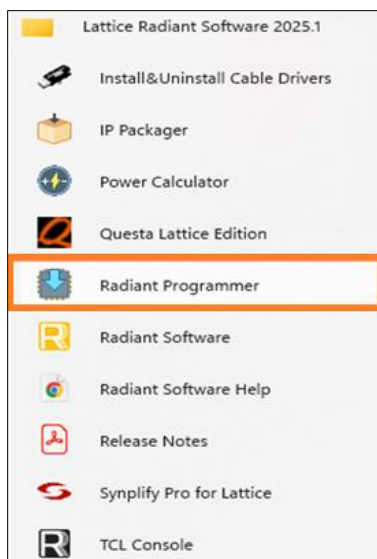


Figure 7.54. Launch Radiant Programmer from Windows Start

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

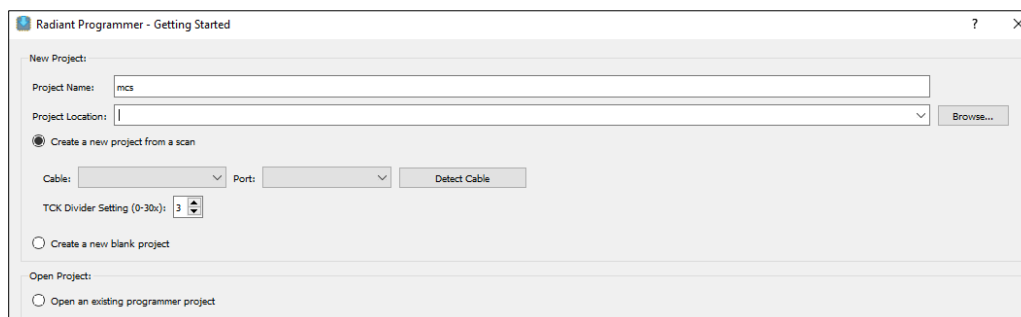


Figure 7.55. Radiant Programmer Getting Started Window

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

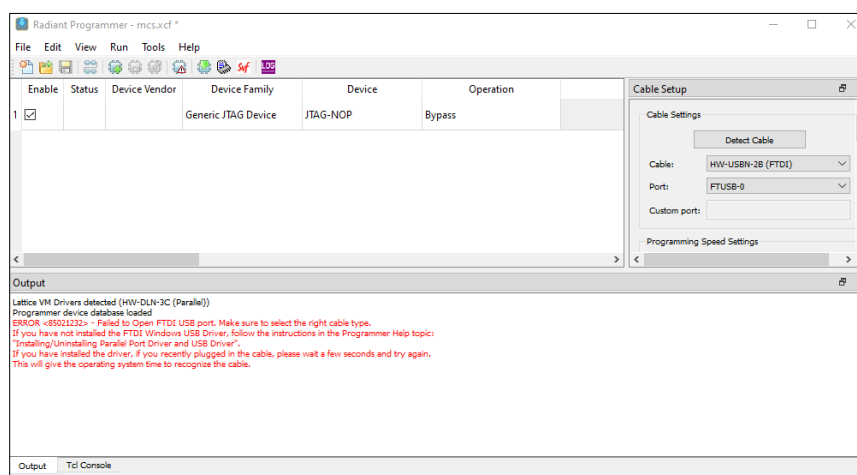


Figure 7.56. Error if No Board is Connected

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

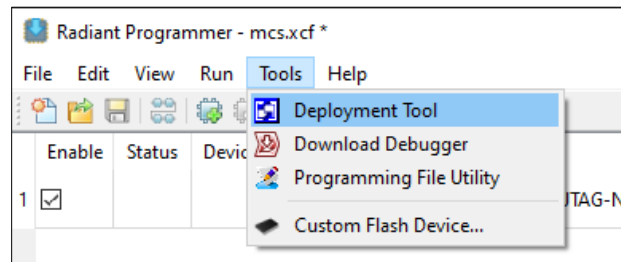


Figure 7.57. Open Deployment Tool from Radiant Programmer

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

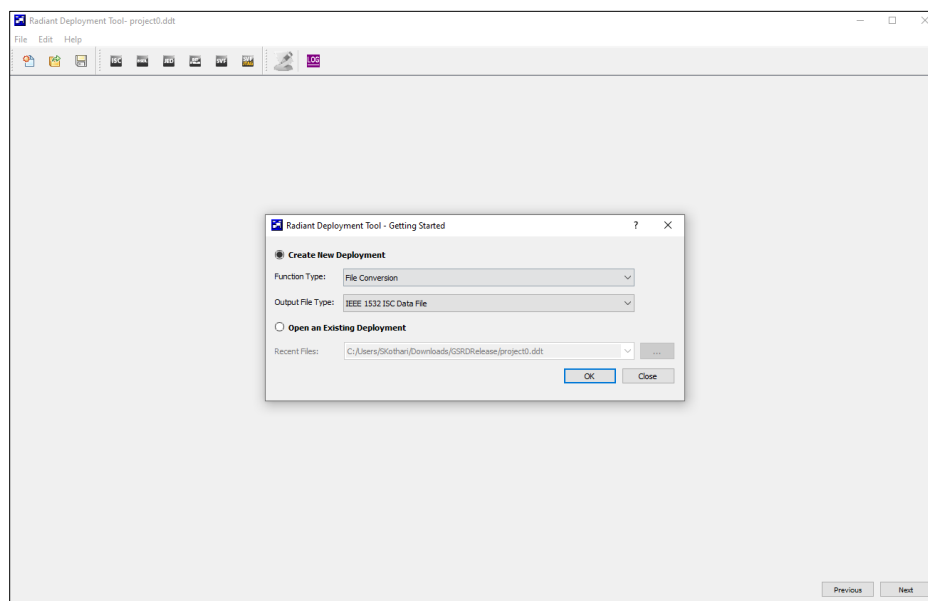


Figure 7.58. Deployment Tool Start Window

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

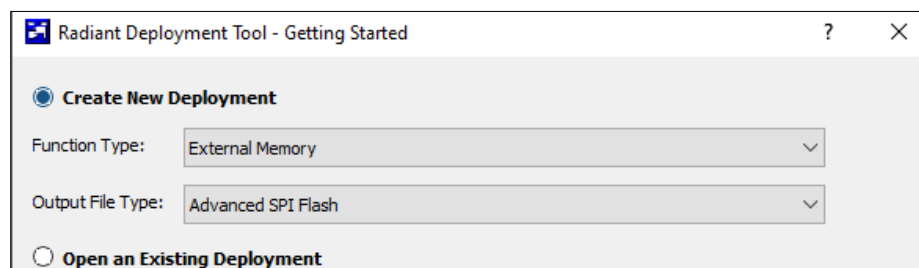


Figure 7.59. Options for Creating New Deployment

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**.

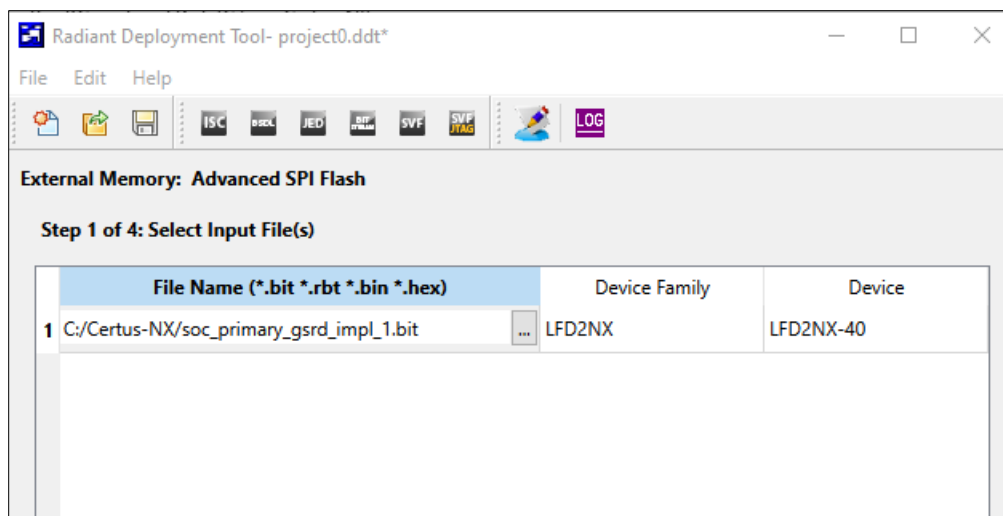


Figure 7.60. External Memory Step 1 of 4: Select Input Files

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 **128**.
 - b. Select **Quad I/O SPI Flash Read** from the SPI Flash Read Mode dropdown menu.

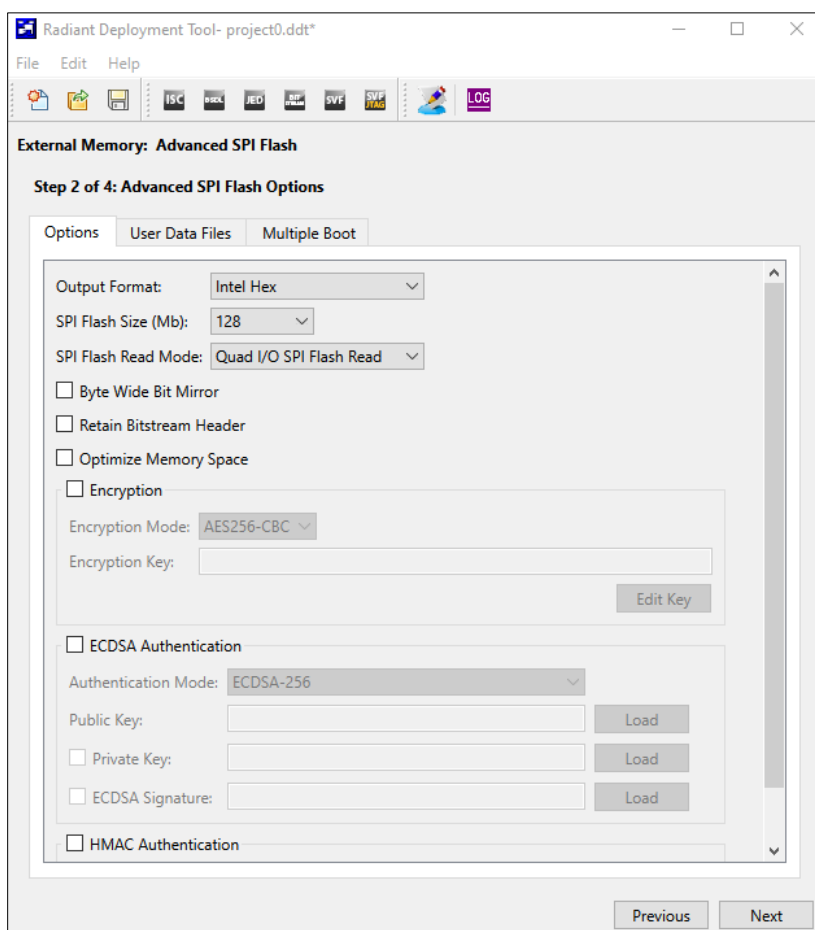


Figure 7.61. External Memory Step 2 of 4: Select Options

- c. No changes needed in the **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_gsrdr_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 the next PROGRAMN/REFRESH or soft reset.
 - v. Click **Next**.

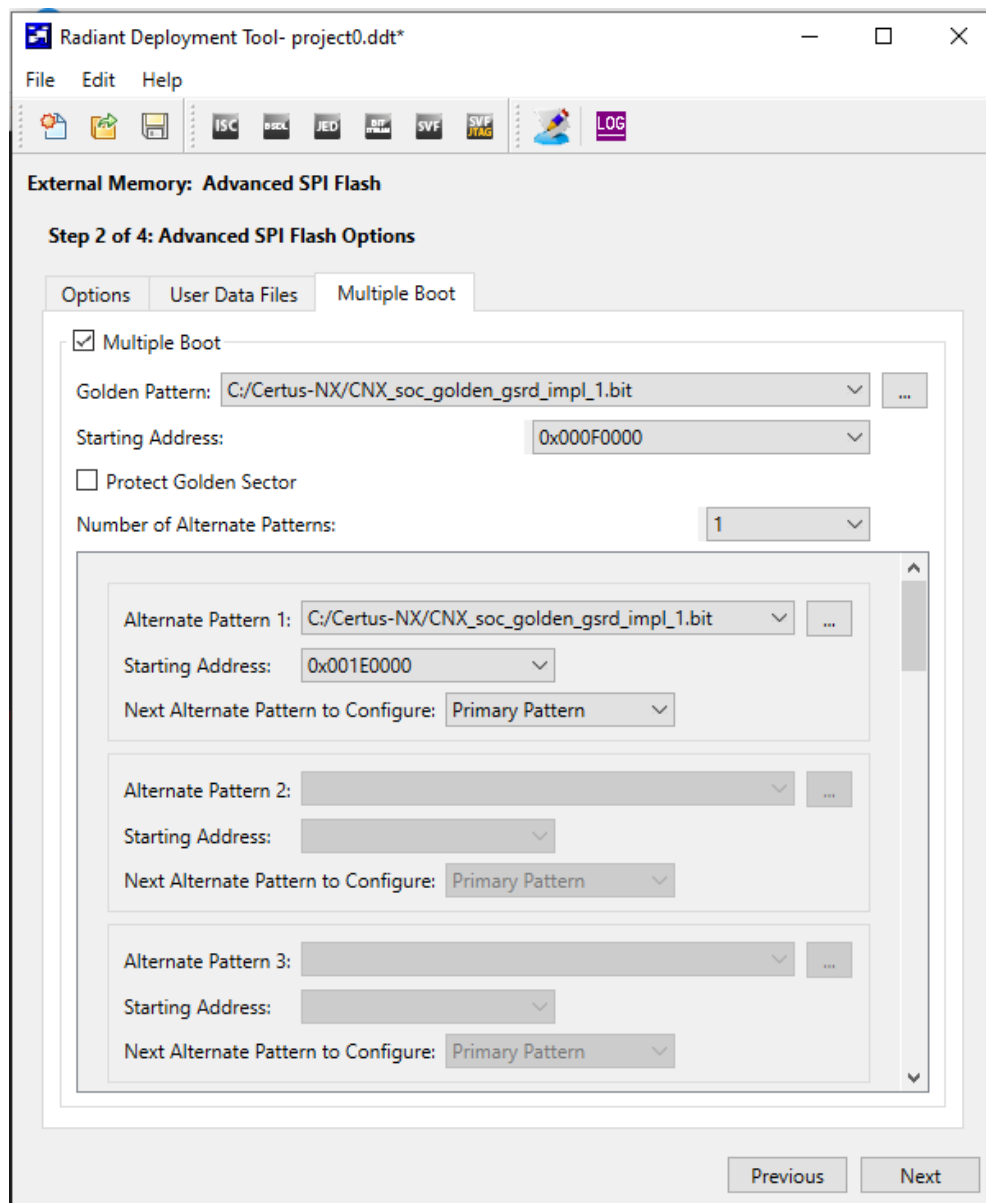


Figure 7.62. External Memory Step 2 of 4: Multiple Boot

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.

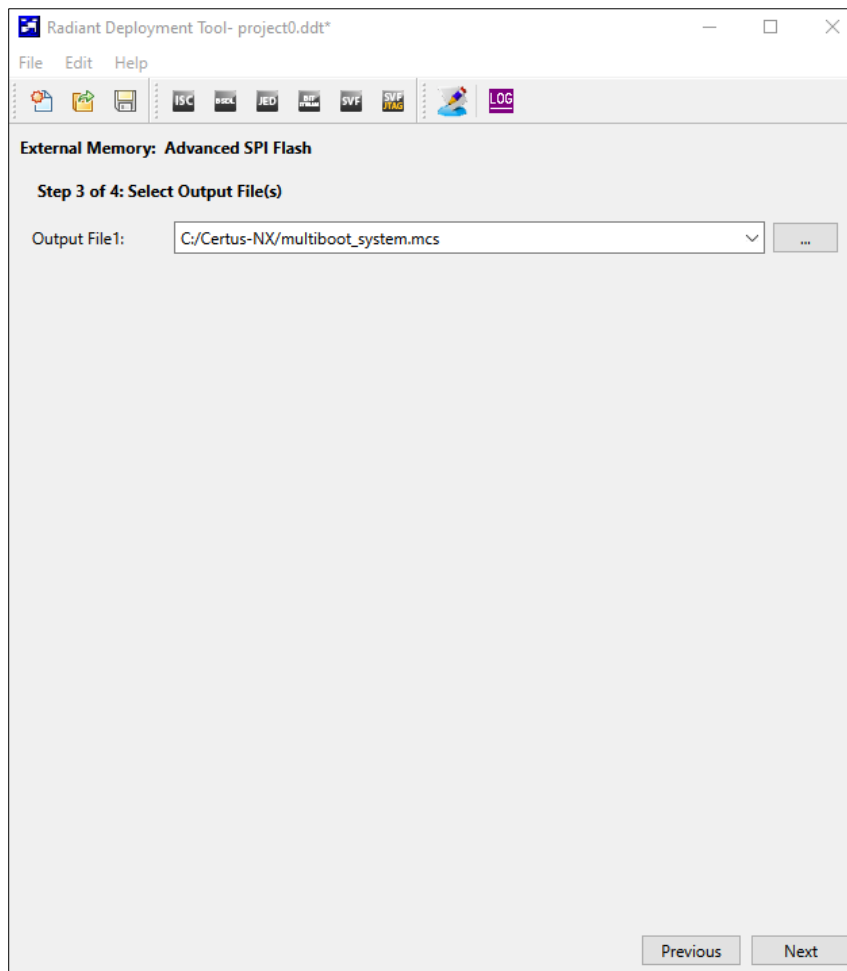


Figure 7.63. External Memory Step 3 of 4: Select Output File(s)

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

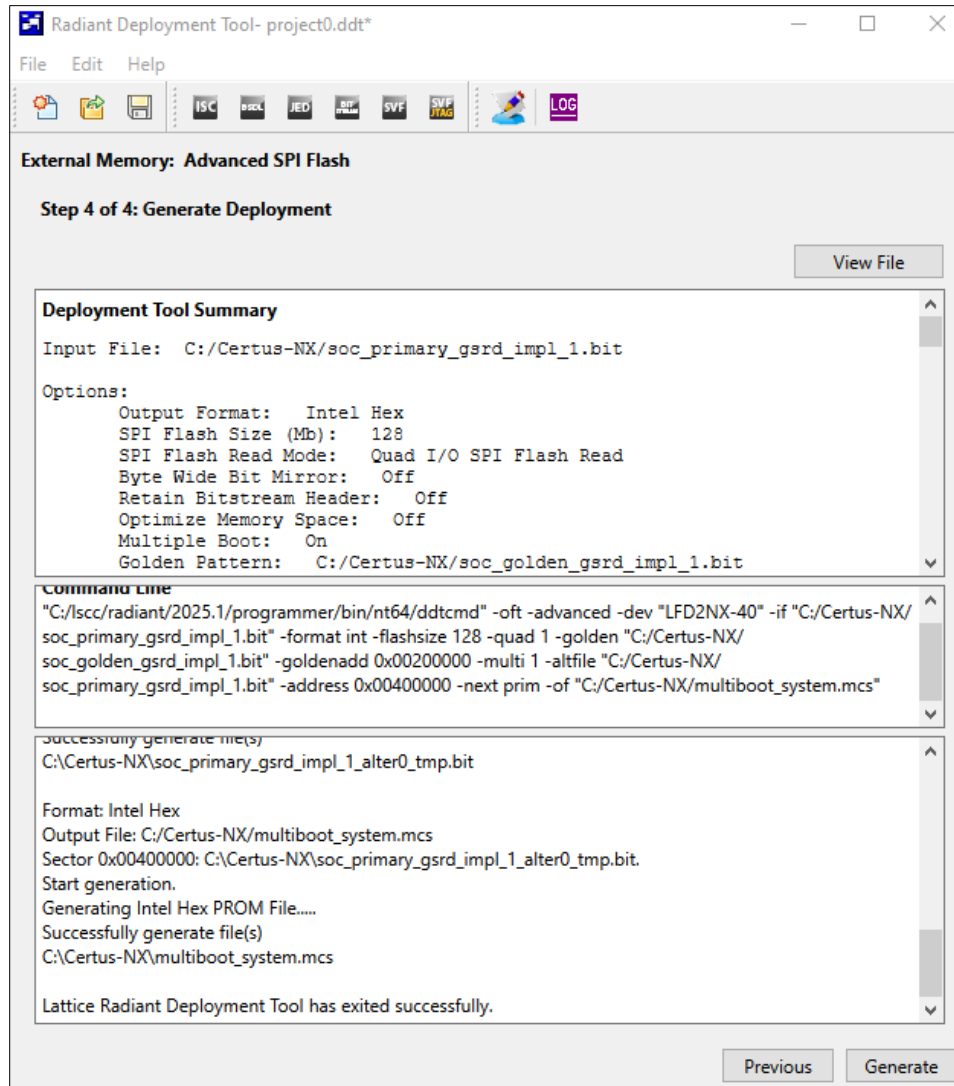


Figure 7.64. External Memory Step 4 of 4: General Development

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](#) section.

8. Customizing the GSRD

This section describes the customization that can be applied in the 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](#).

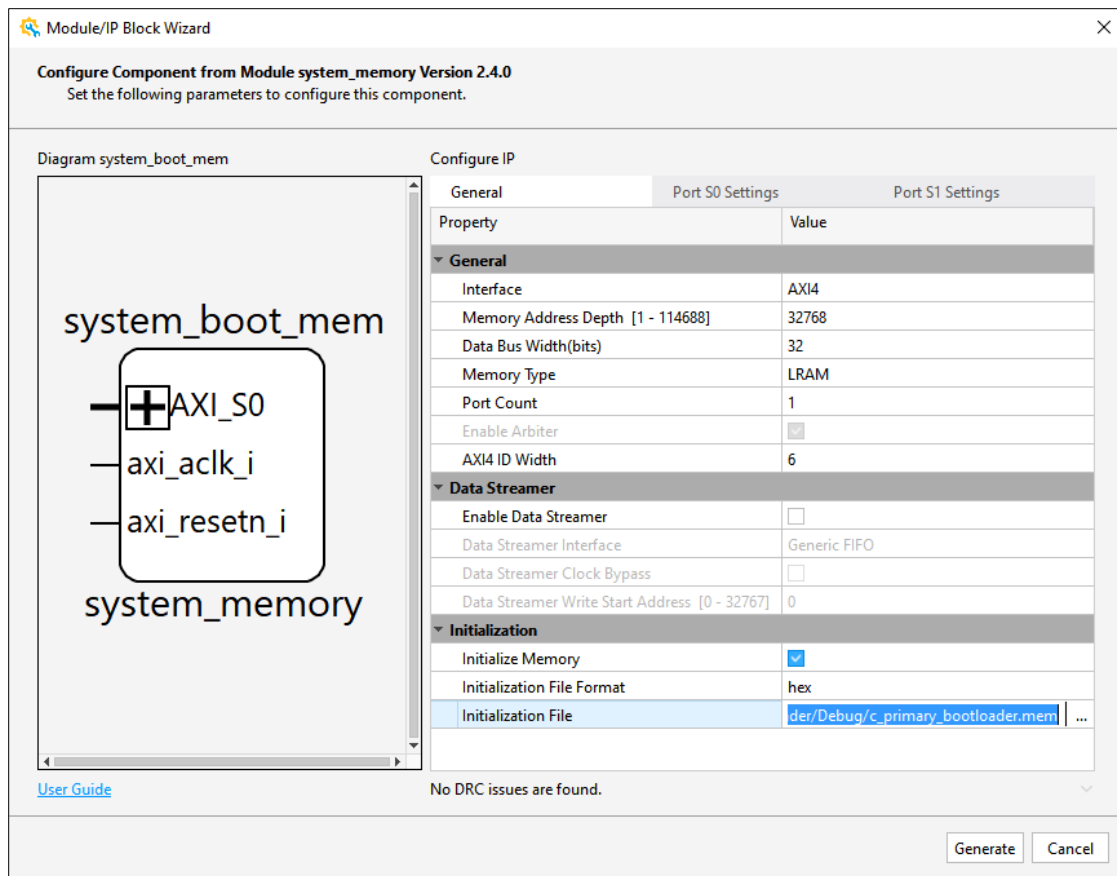


Figure 8.1. Bootloader File Updated in the System Memory

- 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.

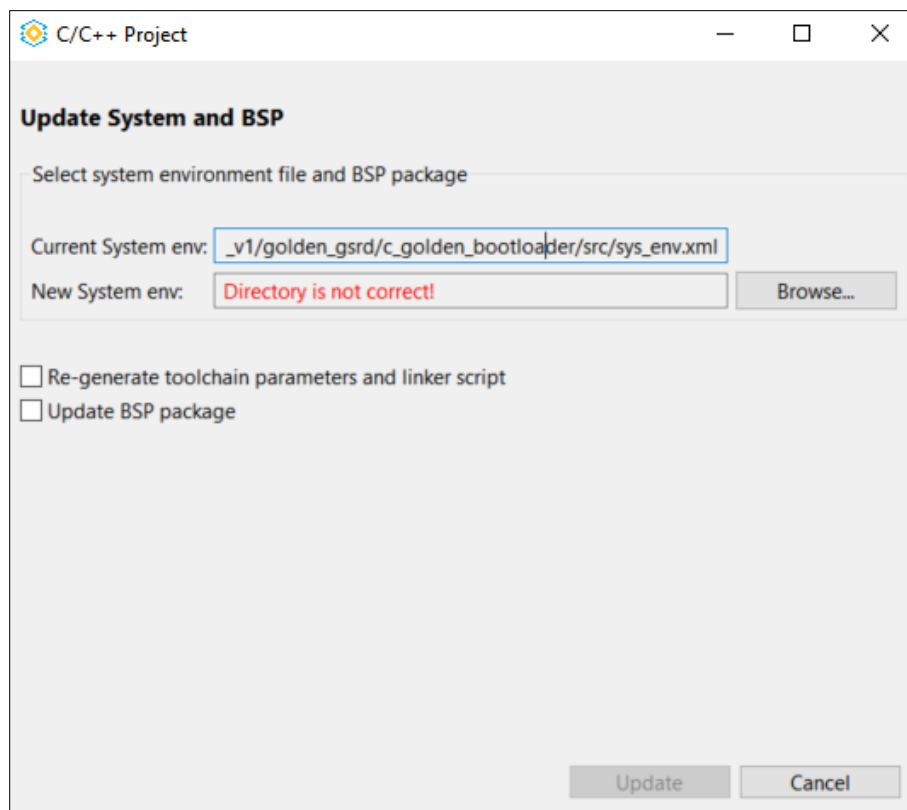


Figure 8.2. New System env Error

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. Using the 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 the .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.

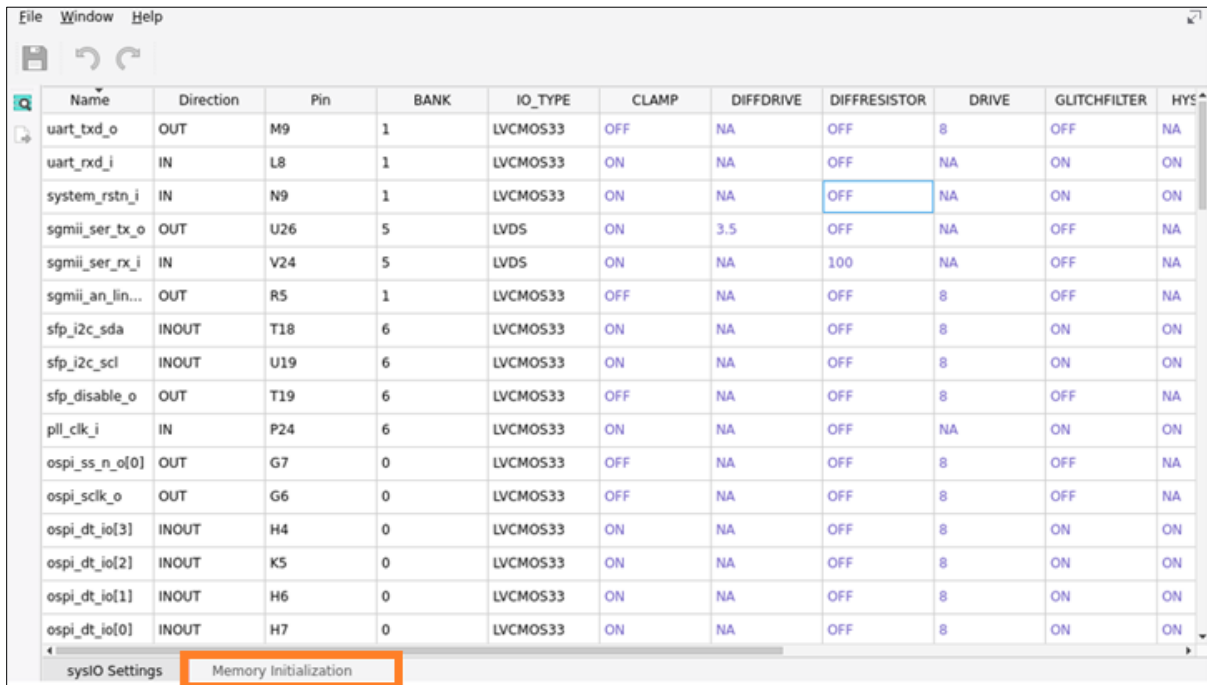
The following example demonstrates how to change/update the .mem file dynamically.

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



Figure 8.3. ECO Editor Icon in Radiant Software

- Click on **Memory Initialization** tab.

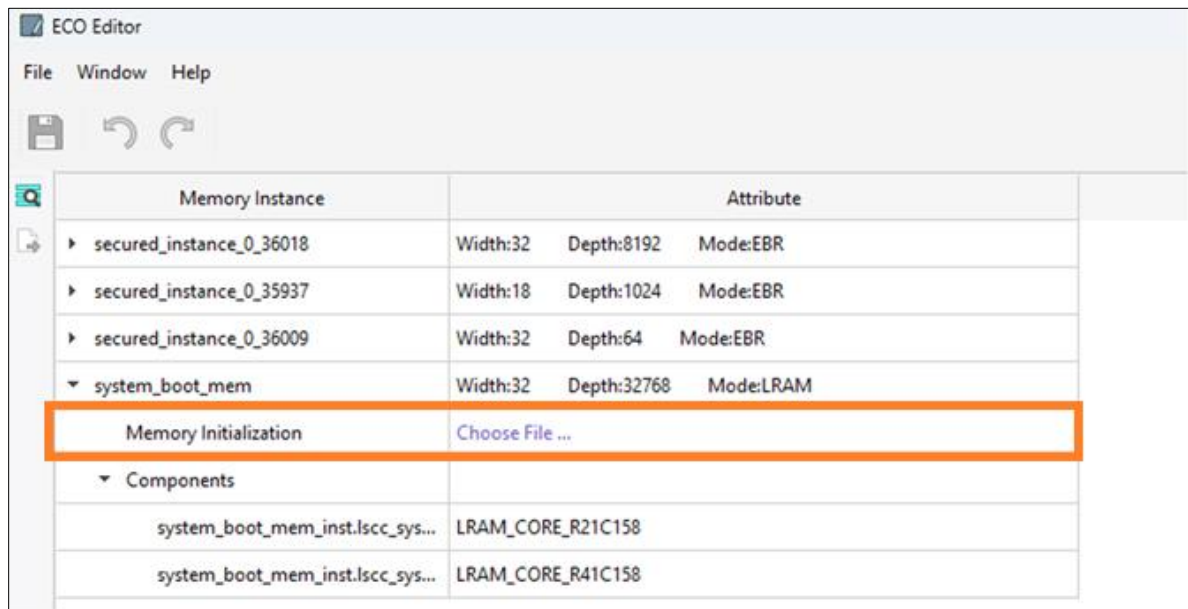


Name	Direction	Pin	BANK	IO_TYPE	CLAMP	DIFFDRIVE	DIFFRESISTOR	DRIVE	GLITCHFILTER	HYS
uart_txd_o	OUT	M9	1	LVC MOS33	OFF	NA	OFF	8	OFF	NA
uart_rxd_i	IN	L8	1	LVC MOS33	ON	NA	OFF	NA	ON	ON
system_rstn_i	IN	N9	1	LVC MOS33	ON	NA	OFF	NA	ON	ON
sgmii_ser_tx_o	OUT	U26	5	LVDS	ON	3.5	OFF	NA	OFF	NA
sgmii_ser_rx_i	IN	V24	5	LVDS	ON	NA	100	NA	OFF	NA
sgmii_an_lin...	OUT	R5	1	LVC MOS33	OFF	NA	OFF	8	OFF	NA
sfp_i2c_sda	INOUT	T18	6	LVC MOS33	ON	NA	OFF	8	ON	ON
sfp_i2c_scl	INOUT	U19	6	LVC MOS33	ON	NA	OFF	8	ON	ON
sfp_disable_o	OUT	T19	6	LVC MOS33	OFF	NA	OFF	8	OFF	NA
pll_clk_i	IN	P24	6	LVC MOS33	ON	NA	OFF	NA	ON	ON
ospi_ss_n_o[0]	OUT	G7	0	LVC MOS33	OFF	NA	OFF	8	OFF	NA
ospi_sclk_o	OUT	G6	0	LVC MOS33	OFF	NA	OFF	8	OFF	NA
ospi_dt_io[3]	INOUT	H4	0	LVC MOS33	ON	NA	OFF	8	ON	ON
ospi_dt_io[2]	INOUT	K5	0	LVC MOS33	ON	NA	OFF	8	ON	ON
ospi_dt_io[1]	INOUT	H6	0	LVC MOS33	ON	NA	OFF	8	ON	ON
ospi_dt_io[0]	INOUT	H7	0	LVC MOS33	ON	NA	OFF	8	ON	ON

sysI/O Settings **Memory Initialization**

Figure 8.4. ECO Editor sysI/O 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 8.5. ECO Editor Memory Initialization Tab

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

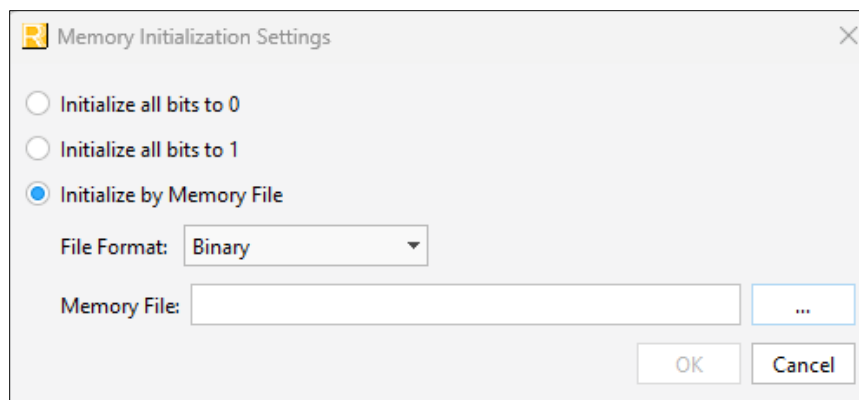


Figure 8.6. Select bootloader in Memory Initialization Settings

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

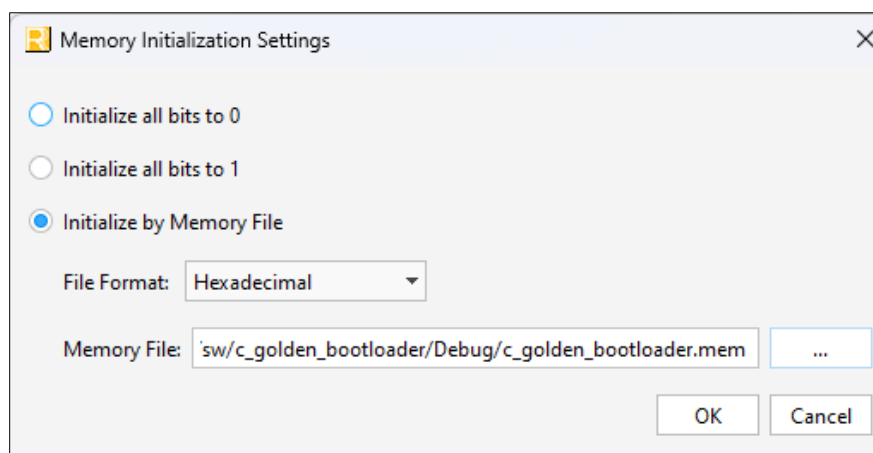


Figure 8.7. Change File Format in Memory Initialization Settings

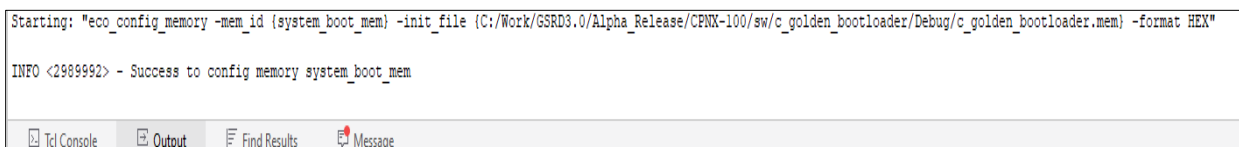


Figure 8.8. Updated System Memory Content

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

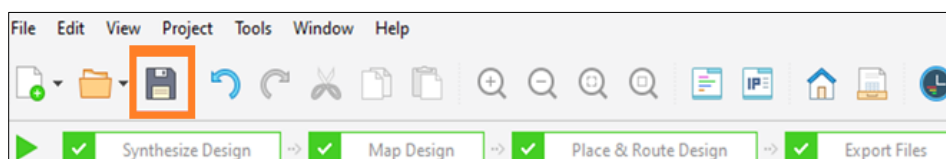


Figure 8.9. Save Icon

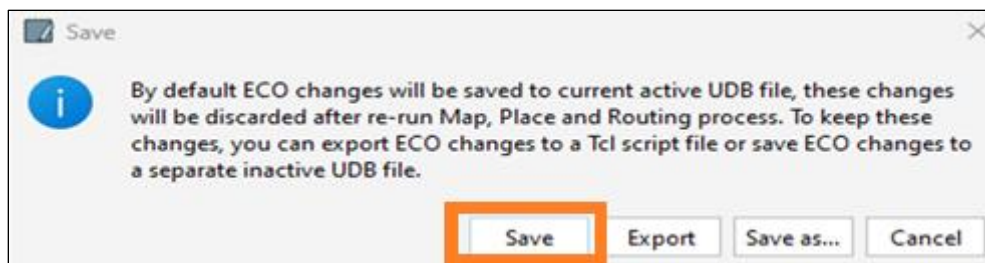


Figure 8.10. Save ECO Changes

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

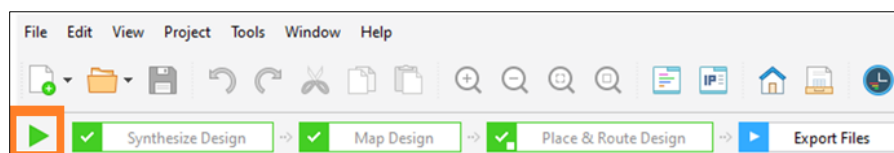


Figure 8.11. Re-run Partial Radiant Flow

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

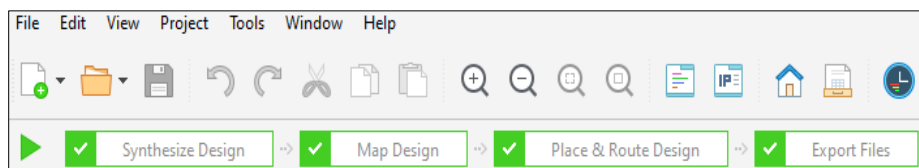


Figure 8.12. Bitstream Generation Flow Completed Successfully



Figure 8.13. 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 A. Enabling Verbosity Level in Software](#) to enable software verbosity level for debugging purposes.

Appendix A. 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 A.1. Debuglib Verbose Levels

Debug Level	Description
<code>DEBUG_INIT</code>	Used for all initialization printout. Mandatory. Enabled for both Debug and Release build.
<code>DEBUG_INFO</code>	Used for additional information printout for debugging purposes. Enabled for only Debug build.
<code>DEBUG_ERROR</code>	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**.

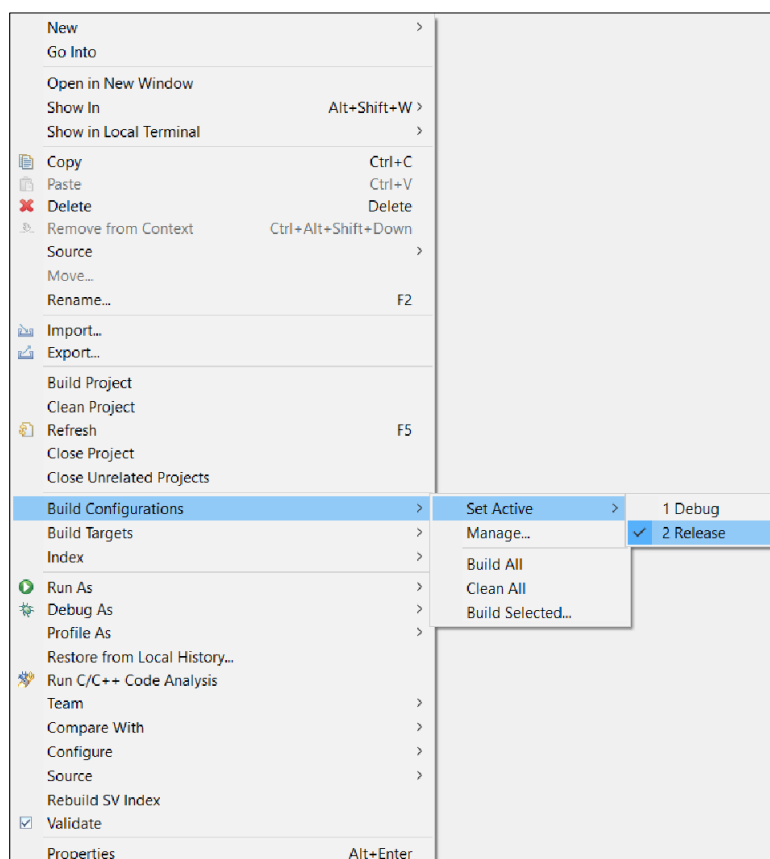


Figure A.1. Set Release Build Configurations

- 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 **LSSC_RELEASE_BUILD** > **Apply** and **Close**.

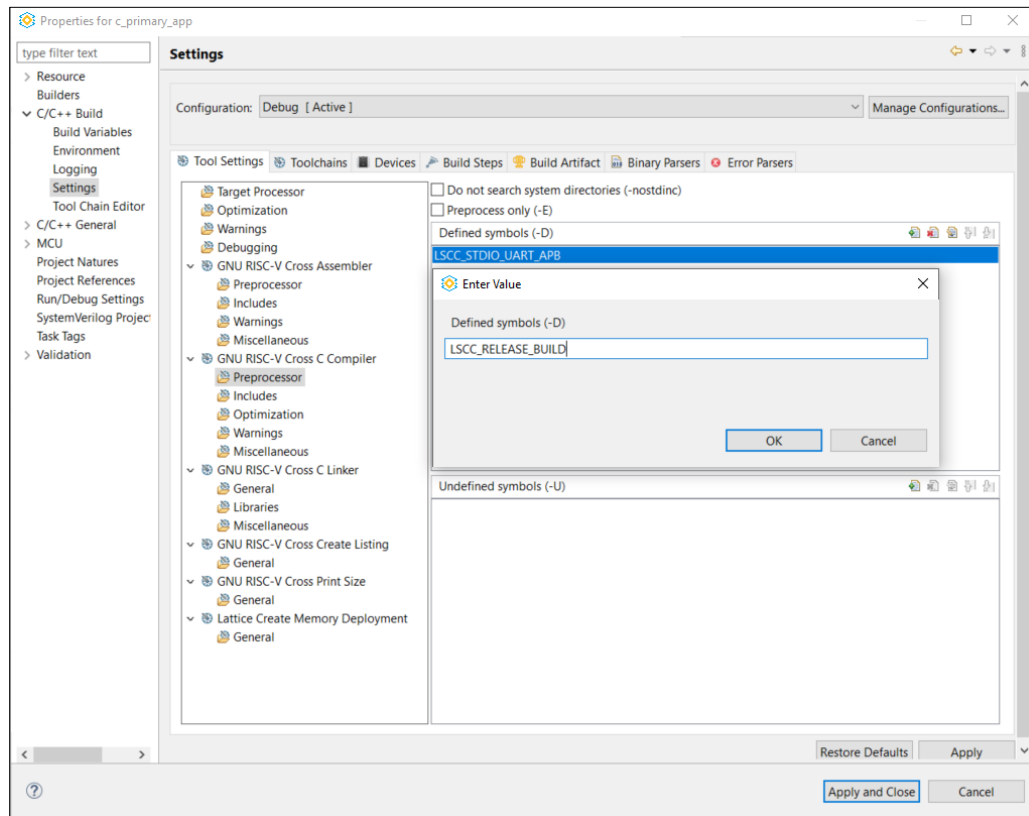


Figure A.2. Add LSSC_RELEASE_BUILD Defined Symbols for Release Build Output

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

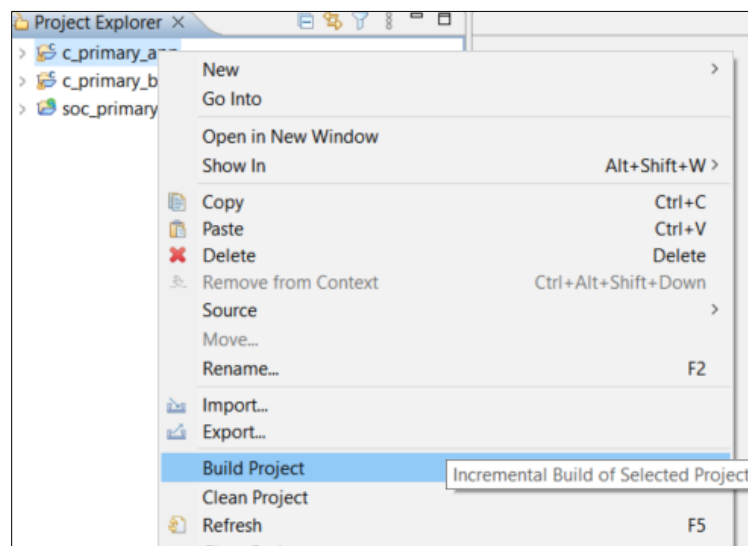


Figure A.3. Build c_primary_app C/C++ Project

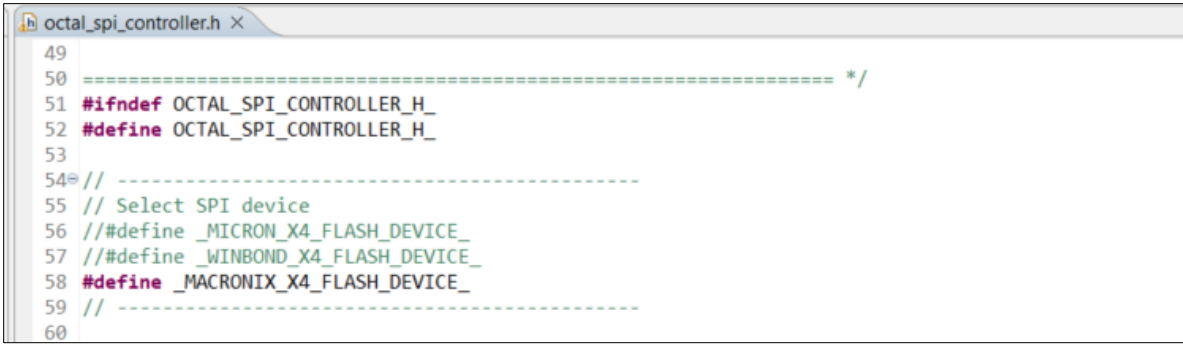
- These actions enable verbosity levels **DEBUG_ERROR** and **DEBUG_INIT** in the Release Build.

Appendix B. 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 B.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 B.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 Lattice Propel SDK \(Primary and Golden\)](#) and [Building the Bare-metal Application Software using Lattice Propel SDK \(Primary and Golden\)](#) section to update the bootloader and bare-metal application software.

Appendix C. Using the Octal SPI Controller for Read, Write, and Erase Self-Diagnostic Check (Bare-metal Application Software)

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

1. The Octal SPI Controller first erases the 4kB in the SPI flash at the offset 0x2000000.
2. Then, it performs a write of a fixed pattern into the 4kB 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 bare-metal 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 C.1. Octal SPI Controller Read, Write and Erase Check-In Bare-metal 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 = beefdebc, obs_data = ffffffff
exp_addr = 1d554, obs_addr = 1e5c0
[test_erase4k_prog_read_random] Failed !
```

Figure C.2. Self-Diagnostic Check Failed

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

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

pmp entry0: mode=0x01, perm=0x07, addr=0x00007c7f(*4)=0x0001f1fc, locked=0
pmp entry1: mode=0x01, perm=0x00, addr=0x00007c80(*4)=0x0001f200, locked=1
pmp entry2: mode=0x01, perm=0x00, addr=0x00007c83(*4)=0x0001f20c, locked=0
pmp entry3: mode=0x01, perm=0x07, addr=0x3fffffff(*4)=0xffffffffc, locked=0
#####
lwip_tcpip_init
Starting lwIP, local interface IP is 192.168.1.4
PHY Initialization:
```

Figure C.3. Self-Diagnostic Check Passed

Appendix D. 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 below).

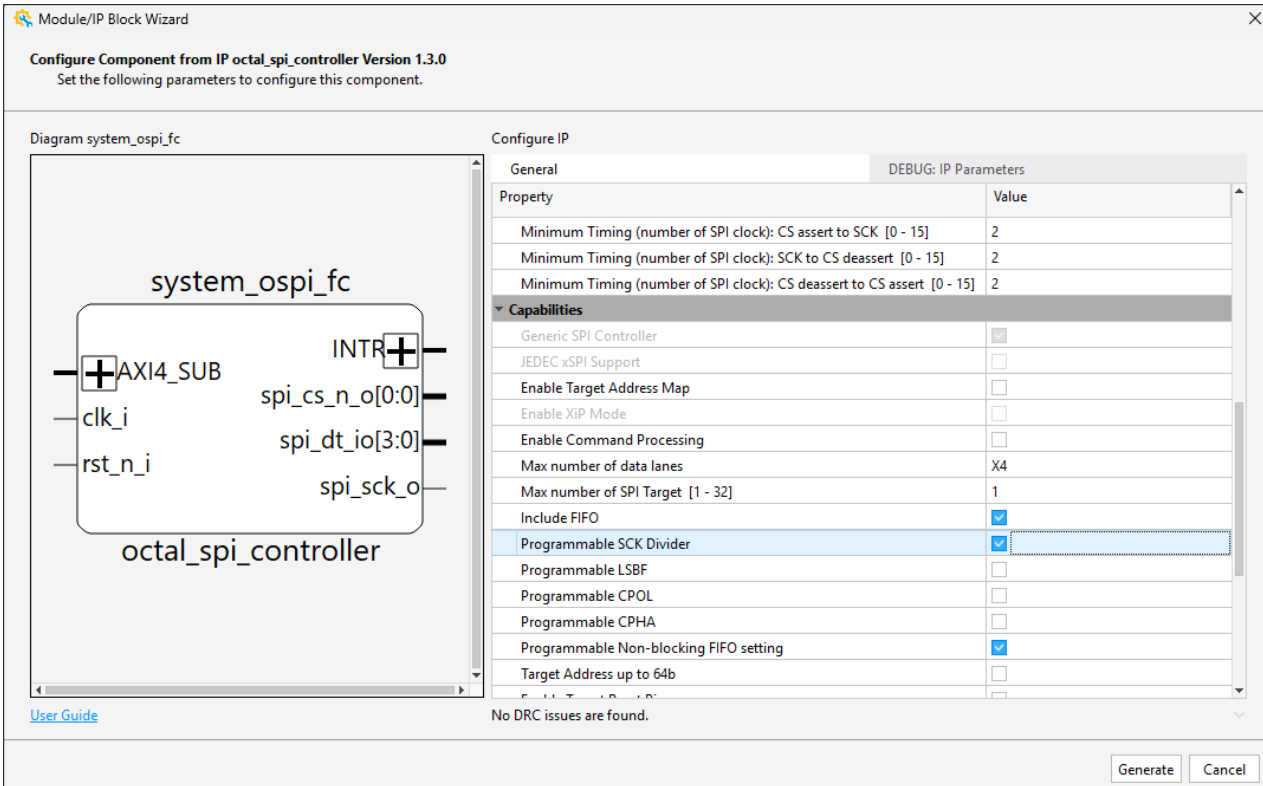


Figure D.1. Programmable SCK Divider Option in the IP User Interface

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 / $(2 \times 2^{[1]}) = 50$ 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 D.2. Octal SPI Controller `sck_rate` Configuration

References

- [Certus-NX Versa Evaluation Board User Guide \(FPGA-EB-02032\)](#)
- [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 \(FPGA-AN-02059\)](#)
- [Octal SPI Controller IP Core User Guide \(FPGA-IPUG-02248\)](#)
- [RISC-V MC CPU IP User Guide \(FPGA-IPUG-02278\)](#)
- [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