Golden System Reference Design User Guide

Reference Design

FPGA-RD-02283-1.1

May 2024

Disclaimers

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

Inclusive Language

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

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AHBL	Advanced High-performance Bus-Lite
AI	Artificial Intelligence
API	Application Programming Interface
BLDC	Brushless DC
CCU	CNN Co-Processor Unit
CLINT	Core Local Interrupter
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DDR	Double Data Rate
DMA	Direct Memory Access
FIFO	First-In-First-Out
GHRD	Golden Hardware Reference Design
GSRD	Golden System Reference Design
ISR	Interrupt Service Routines
LMMI	Lattice Memory Mapped Interface
LPDDR4	Low Power Double Data Rate Generation 4
ML	Machine Learning
MPMC	Multi-Port Memory Controller
PLIC	Platform-Level Interrupt
QSPI	Quad Serial Peripheral Interface
RISC-V	Reduced Instruction Set Computer-V
RTL	Register-Transfer Level
SoC	System on Chip
TSEMAC	Tri-Speed Ethernet Media Access Controller
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Data gram Protocol

1. Introduction

The Golden Hardware Reference Design (GHRD) is a hardware reference design that you can use as a base starting point to create your own FPGA hardware design. It includes key hardware building blocks like RISC-V, flash controller, multiport memory controller, Scatter-Gather Direct Memory Access (SGDMA), Tri-Speed Ethernet Media Access Controller (TSEMAC), UART, and GPIO. All these building blocks are connected with an industry standard interconnect, such as AXI4 for high speed blocks and AHBL for low speed blocks.

The Golden System Reference Design (GSRD) is a comprehensive embedded system created by incorporating the GHRD with the necessary device drivers. GSRD runs on RISC-V RX that supports the FreeRTOS open-source operating system. GSRD enables a bootloader that checks for firmware consistency by performing CRC checks while enabling key IP components using compatible FreeRTOS device drivers.

The GSRD is capable of establishing the integrity of binaries and execute the appropriate program. There are two FPGA images – Primary Image and Golden Image. If the Primary Image integrity check fails, then the system switches to the Golden Image.

1.1. GHRD Components

You can easily build a GHRD using the Lattice Propel™ builder environment. The key features of the GHRD are as follows:

- RISC-V RX soft CPU
- Multi-Port Memory Controller (LPDDR4 memory controller support)
- Quad Serial Peripheral Interface (QSPI) flash controller and on-chip memory
- TSEMAC
- SGDMA data transfer IP
- UART
- General purpose I/O (GPIO)
- FPGA configuration for multi-boot
- SFP configuration
- AXI4 interconnect
- APB interconnect

Note: For more details on the components listed above, refer to their respective user guides as listed in the References section.

1.2. GSRD Components

The GSRD includes the following components:

- GHRD
- Reference RISC-V software including:
  - Bootloader
  - FreeRTOS device drivers for all IPs
  - FreeRTOS sample applications

1.3. Features Overview

Key features of the GSRD SoC include:

- The GSRD is targeted for the CertusPro™-NX FPGA
- Supports RISC-RX, MPMC, SGDMA, LPDDR4, and QSPI controller over the AXI interface
- Bootloader, Primary image, and Golden image
- The Golden image acts as a baseline version of the system
- The Primary image is an updated version of the system
- Bootloader firmware supports CRC checking and switching between the Primary image and Golden image
- FPGA image CRC and booting are performed by the configuration engine of the FPGA
- The firmware has the option to manually boot the FPGA image based on CRC check
- The firmware copies from the QSPI flash to LPDDR4 before execution
- LPDDR4 supports 32-bit data interface and 400/533 MHz DDR frequency
- The GSRD meets 110 MHz CPU frequency and 1 Gbps Ethernet throughput

2. Design Overview

The system hardware (SoC) supports RISC-V RX, MPMC, SGDMA, and QSPI controller over the AXI interface. It includes the bootloader, Primary image, and Golden image.

2.1. GSRD Architecture

The GSRD architecture is shown in Figure 2.1.

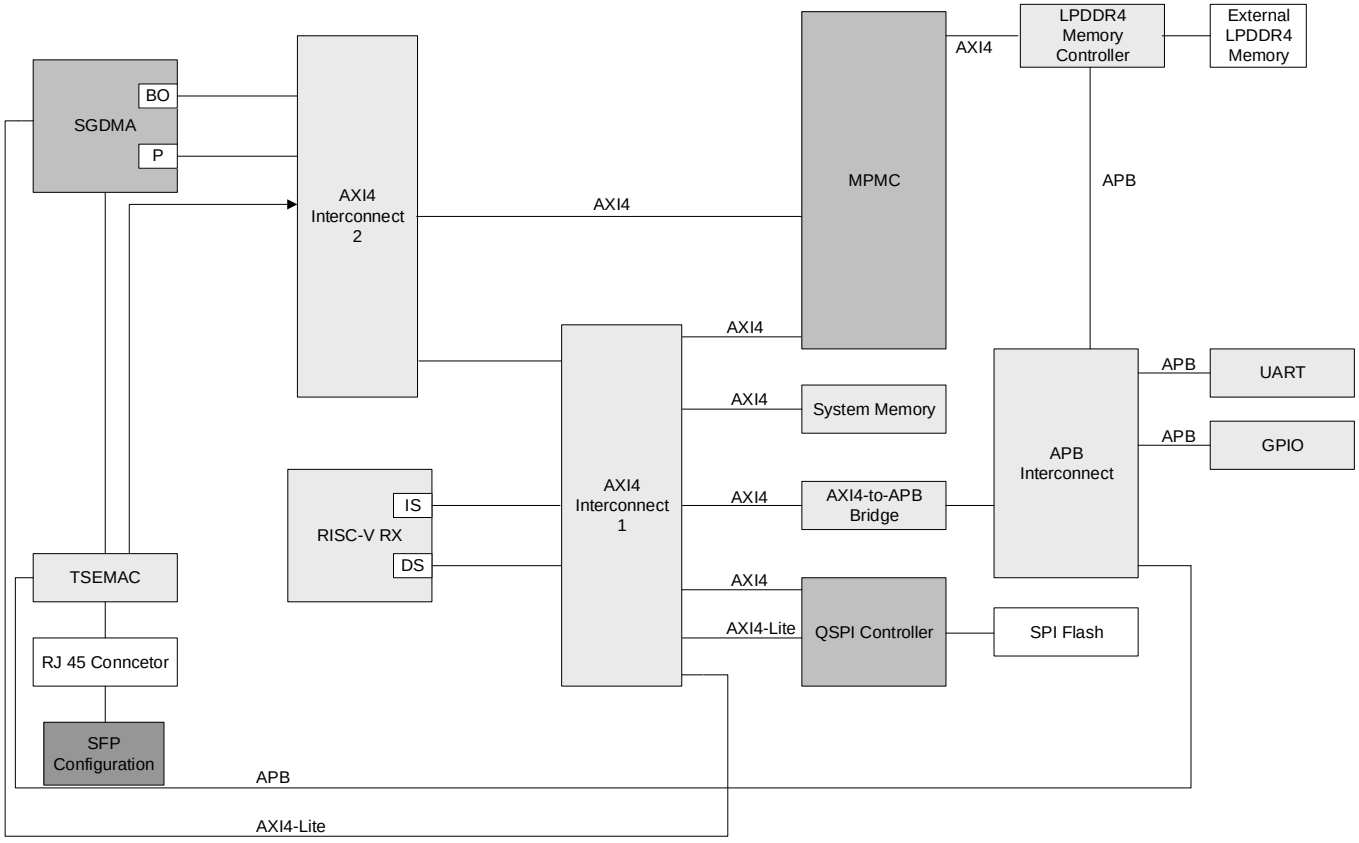


Figure 2.1. GSRD Architecture

The GSRD architecture includes a RISC-V RX processor, various IP modules, and two AXI4 interconnects.

2.2. IP Modules

The GSRD architecture shown in Figure 2.1 includes the following IP modules:

TSEMAC

The TSEMAC IP core supports the ability to transmit and receive data between standard interfaces, such as APB, AHB-Lite or AXI4-Lite, and an Ethernet network. The main function of TSEMAC IP is to ensure that the Media Access rules specified in the 802.3 IEEE standard are met while transmitting a frame of data over Ethernet. On the receiving side, the TSEMAC extracts different components of a frame and transfers them to higher applications through the FIFO interface.

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

LPDDR4 Memory Controller

The Lattice Semiconductor LPDDR4 Memory Controller for Nexus Devices provides a turnkey solution consisting of a controller, DDR PHY, and associated clocking and training logic to interface with LPDDR4 SDRAM. The IP core is implemented in System Verilog HDL using the Lattice Radiant™ software integrated with the Lattice Synthesis Engine (LSE) and Synplify Pro® synthesis tools.

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

QSPI Controller

The Quad Serial Peripheral Interface (QSPI) is a four-tri-state data line serial interface that is commonly used to program, erase, and read SPI Flash memories. QSPI enhances the throughput of a standard SPI by four times since four bits are transferred every clock cycle. The Lattice QSPI Flash Controller IP core supports the SPI, DSPI, and QSPI protocols. The design is implemented in Verilog HDL.

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

MPMC

The AXI4 Multi Port Bridge for Memory Controller (MPMC) IP connects multiple external managers to a single memory controller. This IP runs only on AXI4, and supports separate clock domains between each external managers and the subordinate.

For more information about the IP core, refer to AXI4 Multi Port Bridge for Memory Controller Module (FPGA-IPUG-02246). For register map information, refer to SGMII and Gb Ethernet PCS IP Core - Lattice Radiant Software (FPGA-IPUG-02077).

SGMII

The SGMII/Gb Ethernet PCS IP core converts GMII frames into 8-bit code groups in both transmit and receive directions and performs auto-negotiation with a link partner as described in the Cisco SGMII and IEEE 802.3z specifications. The SGMII IP is a connection bus for MACs and PHYs and is often used in bridging applications and/or PHY implementations. It is widely used as an interface for a discrete Ethernet PHY chip.

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

SGDMA

The Scatter Gather Direct Memory Access Controller (SGDMA) IP core provides access to the main memory independent of the processor. It offloads processor intervention. The processor initiates transfer to SGDMAC and receives an interrupt on completion of the transfer by the DMA Engine. The Lattice SGDMAC IP core implements a configurable, AXI Lite-compliant DMA controller with scatter-gather capability.

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

For API details, refer to SGDMA Driver API Reference (FPGA-TN-02340).

UART

The Lattice Semiconductor UART (Universal Asynchronous Receiver/Transmitter) IP Core is designed for use in serial communication, supporting the RS-232. The UART IP Core has many characteristics similar to those of the NS16450 UART.

The UART IP Core performs two main functions:

- Serial-to-parallel conversion on data characters received from an external UART device; and
- Parallel-to-serial conversion on data characters received from the Host located in the FPGA.

For more information about the IP core including register map information, refer to UART IP Core - Lattice Propel Builder (FPGA-IPUG-02105).

FPGA Configuration Module

The Nexus configuration logic provides an LMMI interface to allow user logic residing inside the FPGA fabric to access the device configuration (CFG) functionalities. You need to instantiate the CONFIG_LMMIA and OSC primitives. The FPGA configuration module executes the LSC_REFRESH command to toggle the PROGRAMN pin to automatically switch to an alternate pattern (Golden pattern).

For more details, refer to *Appendix D. Configuration Access from User Logic* of sysCONFIG User Guide for Nexus Platform (FPGA-TN-02099).

SFP Configuration module

The SFP configuration module includes an LMMI module and an I^2C controller module. The I^2C controller drives the LMMI to perform SFP configuration. It generates the SFB Disable signal and uses the Link OK signal of the SGMII IP and the SFB Absent signal.

2.3. AXI4 Interconnects

The RISC-V RX soft processor is connected to most of the peripheral blocks via AXI4 interconnect 1 as shown in Figure 2.1.

There are two AXI4 interconnects:

- AXI4 interconnect 1 connects the MPMC block, system memory, QSPI controller, TSEMAC, and other low speed peripherals (UART/GPIO) to RISC-V and SGDMA.
- AXI4 interconnect 2 connects the MPMC block and AXI4 interconnect 1 to the SGDMA block. The purpose of this interconnect is to improve the system performance by storing the data coming from TSEMAC directly into the LPDDR4 memory.

2.4. Memory Map

Table 2.1 shows the GSRD memory map.

Table 2.1. GSRD Memory Map

Base Address	End Address	Range (Bytes in Hex)	Size (kB)	Block
00000000	0003FFFF	40000	256	CPU Data Ram
00300000	0037FFFF	80000	512	SPI Flash Controller
10000000	10000FFF	1000	4	GPIO
10001000	10004FFF	4000	16	TSE MAC
10090000	10090FFF	1000	4	UART
10092000	10092FFF	1000	4	LPDDR4 APB
10093000	10093FFF	1000	4	SGDMA
1009B000	1009BFFF	1000	4	FPGA Config APB
10110000	5010FFFF	40000000	1048576	LPDDR4 AXI
F2000000	F20FFFFF	100000	1024	CLINT (CPU)
FC000000	FC3FFFFF	400000	4096	PLIC (CPU)
F0000400	FFFFFFFF	FFFFC00	262143	Reserved (CPU)

3. Firmware Components

The GSRD is enabled by RISC-V core based FreeRTOS and Lattice FPGA IP modules. Lattice developed BSP drivers in C language act as intermediaries, facilitating communication between the hardware elements on the FPGA and FreeRTOS software. During boot up, these drivers initialize and configure FPGA peripherals to establish effective coordination with the RISC-V processor. For details on the boot flow, refer to the GSRD Demo User Guide (FPGA-UG-02205).

3.1. Primary Bootloader

The Primary bootloader initializes the BSP and OS. Address, speed mode, MAC upper and MAC lower is passed to the TSE MAC handler. It passes the TSE core object to initialize the Ethernet IP and set the Ethernet MAC address. It initializes the LPDDR and QSPI. After initialization, it reads data from the flash memory with the QSPI flash controller when FIFO is disabled using the QSPI flash read fifo disable API. It calculates the CRC value with the *crc16_ccit()* API and compares the value with the calculated *crc_fw* value. Failure of the CRC check on the Primary firmware triggers reconfiguration of the FPGA with the Golden image.

The terminal prints the following statement when the primary bootloader is loaded on the device:

```
"---------------Primary Image Bootloader----------"
```

The following sections provide details of the APIs.

- unsigned short crc16_ccitt(const void *buf, int len, unsigned short start)
  This API calculates the CRC value.

Table 3.1. CRC Value API

In/Out	Parameter	Description	Returns
In	buf	Pointer to the buffer to store the CRC value.	Calculated CRC value
In	len	Denotes the number of blocks of data.	
In	start	Start value of CRC16	

  - The following is an example API used in the GSRD primary bootloader:

```
crc16_ccitt((void*)LPDDR_APPLICATION_MEMORY_START_ADDR, LPDDR_APPLICATION_MEMORY_END_ADDR-
LPDDR_APPLICATION_MEMORY_START_ADDR, CRC16_START_VAL);
```

    - Start address of LPDDR is passed as first argument
    - Length of total memory is calculated and passed as second argument
    - The start value of CRC16 is passed as third argument
- unsigned int qspi_flash_cntl_init(struct qspi_flash_cntl_instance_t *this_qspi_flash_cntl, unsigned int base_addr)
  This API initializes the base address of the QSPI flash controller.

Table 3.2. QSPI Flash Controller Base Address

In/Out	Parameter	Description	Returns
In	this_qspi_flash_cntl	Pointer to the qspi_flash_cntl_instance_t structure	0: success 1: failure
In	base_addr	Base address that is assigned to the controller.	

  - Figure 3.1 shows the API flow diagram.

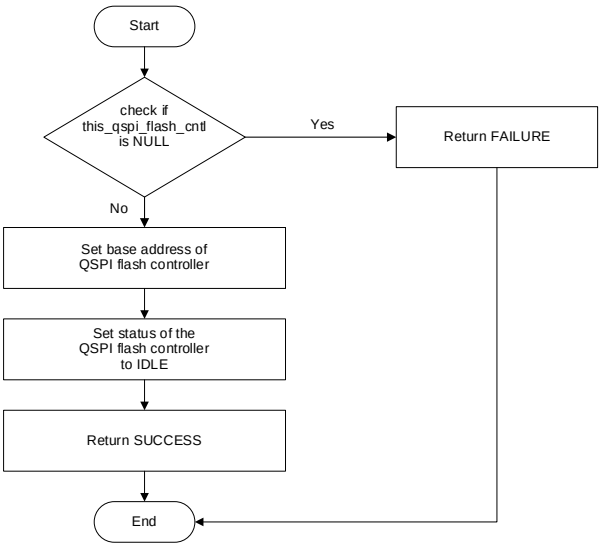


Figure 3.1. qspi_flash_cntl_init()

- The following shows an example of the API used in the GSRD primary bootloader where the QSPI handle and base address of controller are passed as first and second arguments, respectively.

```
qspi_flash_cntl_init(&qspi_core,
QSPI0_INST_QSPI_FLASH_CONTROLLER_MEM_MAP_AXI4LITE_BASE_ADDR)
```

- void qspi_flash_cntl_read_fifo_dis(qspi_params_t *this_qspi_pkt, unsigned int *buffer)
  This API reads data from the flash memory from a given flash address using Data Packet.

Table 3.3. SPI Flash Memory Read Data API

In/Out	Parameter	Description	Returns
In	this_qspi_pkt	Pointer to the qspi_params_t structure.	None
In	buffer	Buffer where the data is stored after reading from the Data Packet register.	

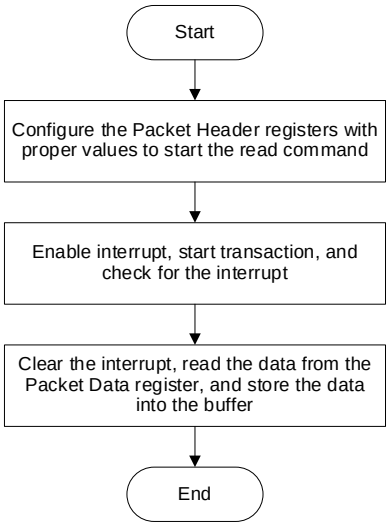


Figure 3.2. qspi_flash_cntl_read_fifo_dis()

- The following shows an example of the API used in the primary bootloader where qspi_params_t handle and the address of the LPDDR memory are passed to the API:

```
qspi_flash_cntl_read_fifo_dis(&pkt_param(unsigned int*)LPDDR_APPLICATION_MEMORY_START_ADDR)
```

3.1.1. Primary Bootloader Flow

Figure 3.3 shows the initialization of BSP, OS, LPDDR, and QSPI.

The required parameters are passed to the TSE MAC handler. Data is read from the flash memory with the QSPI Flash controller. CRC is calculated and the calculate the crc and either the Primary image of the Golden image is loaded onto LPDDR based on the calculated CRC.

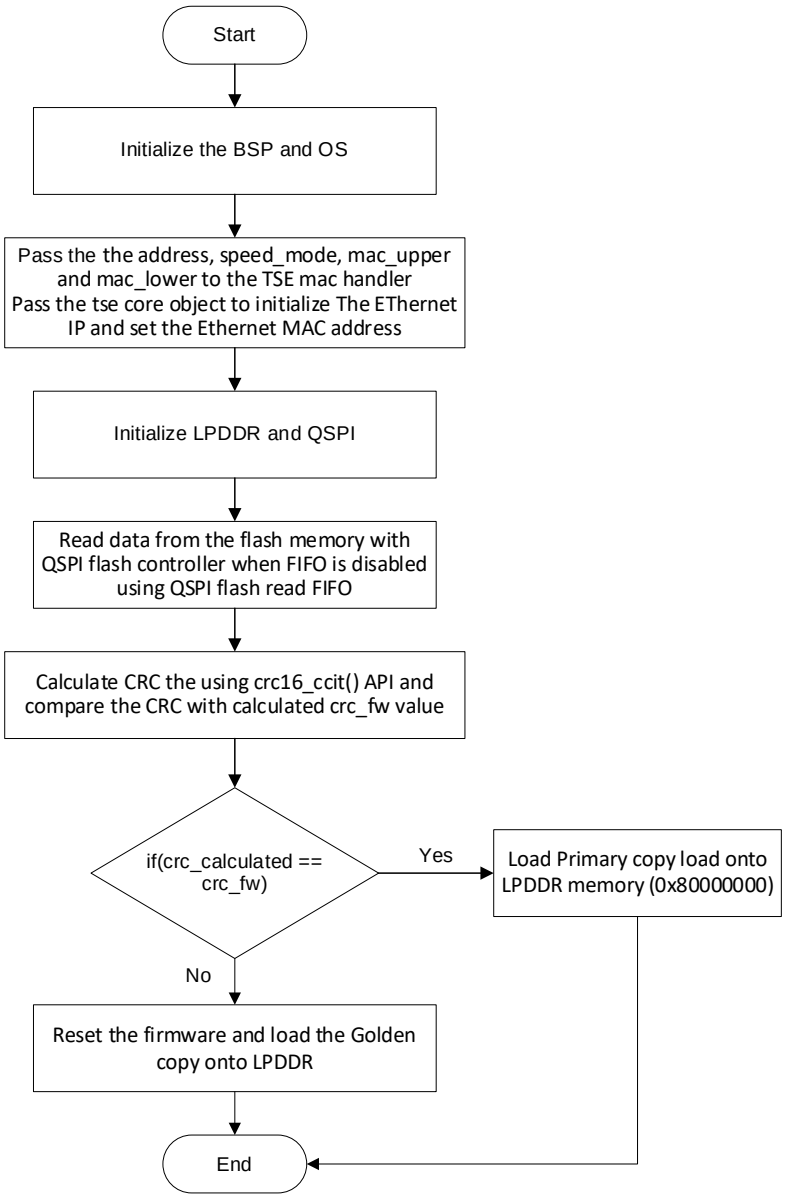


Figure 3.3. Primary Bootloader Flow Diagram

3.2. Primary Application

The Primary Application initializes the BSP, OS, and TSE. It waits for Ethernet packets and stores the packets into the sgdma_core structure. After that, it Initializes the single and multi descriptor instances by passing arguments such as structure handle, *sgdma_base_addr*, and the size of buffer to the *sgdma_init()* API. It calls the *s2mm_buf_desc_dma()* API to configure single and multiple buffer descriptors to update the registers and trigger the DMA. It reads the SGDMA received packet stored in the SGDMA structure buffer. When data is received, it gets the s2mm BD status transfer length error, cmpl, subordinate error, and descriptor error information using the *get_s2m_bd_status()* API.

The terminal prints the following statement when the primary bootloader is loaded on the device:

```
“---------------Primary FW Image ----------"

“FreeRTOS v10.0.1 on RISC-V"
```

The following sections provide details of the APIs.

- unsigned int s2mm_buf_desc_dma(sgdma_instance_t *this_sgdma)
  This function configures single and multiple buffer descriptors to update the registers and trigger the DMA. Example API used in the GSRD primary application:

```
s2mm_buf_desc_dma(&sgdma_core)
```

Table 3.4. SGDMA Pointer

In/Out	Parameter	Description	Returns
In	this_sgdma	Information of the base address of s2mm buffer, buffer length, and num_of_desc specifies the single or multiple buffer descriptor value.	0: success 1: failure

- unsigned int get_s2mm_bd_status(int idx)
  - This function returns the s2mm single and multi-buffer descriptor status fields using a specific index, which specifies the number of the particular buffer descriptor.
  - Example API used in the GSRD primary application, where 0 is the index of the s2mm buffer descriptor:

```
get_s2mm_bd_status(0)
```

Table 3.5. Buffer Descriptor Index

In/Out	Parameter	Description	Returns
In	idx	Index that specifies the particular s2mm buffer descriptor number in multiple buffer descriptor.	Descriptor status field

- void sgdma_init(sgdma_instance_t *this_sgdma, unsigned int base_addr, unsigned int num_of_desc)
  - This function initializes the single and multi descriptor instances.
  - Example API used in the GSRD primary application:

```
sgdma_init(&sgdma_core, SGDMA_BASE_ADDR, BUF_SIZE)
```

Table 3.6. Initialize Descriptor

In/Out	Parameter	Description	Returns
In	this_sgdma	• Initializes with the IP base address and number of descriptor values • Holds the mm2s and s2mm BD addresses.	None
In	base_addr	Specifies the SGDMA IP base address.	
In	num_of_desc	Specifies the single or multi-buffer descriptor value.	

3.2.1. Primary Application Flow

Figure 3.4 shows the flow to initialize the BSP, how to get the Ethernet packet and initialize the single and multiple descriptor instances, and read the SGDMA received packet.

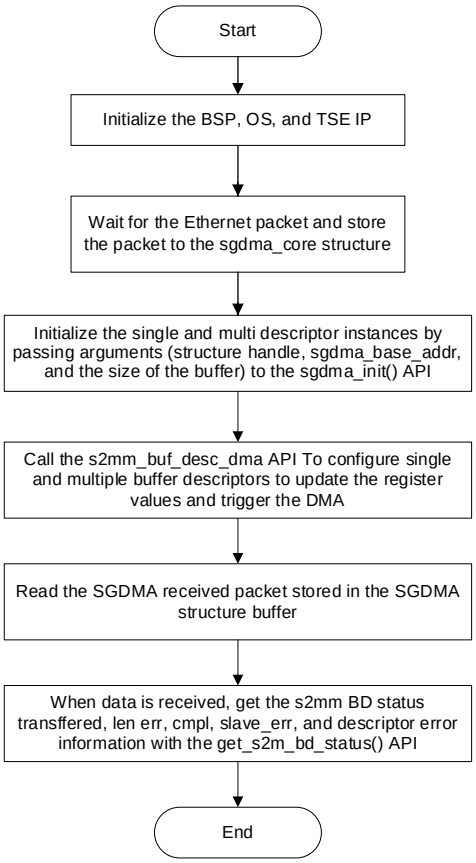


Figure 3.4. Primary Application Flow Diagram

3.3. Golden Bootloader

The Golden Bootloader is the same as the Primary bootloader. Different print statements on the serial terminal indicate which bootloader is loaded. If the Primary Bootloader and Primary Application fail to boot, the device reboots itself and boots up with the Golden Bootloader.

The following statement is shown on the serial terminal:

```
“---------------Golden Image Bootloader----------"
```

For more details, refer to the GSRD bootflow diagram in the GSRD Demo User Guide (FPGA-UG-02205).

3.4. Golden Application

The Golden Application is the same as the Primary Application. Different print statements on the serial terminal indicate which application is loaded.

The following statement is shown on the serial terminal:

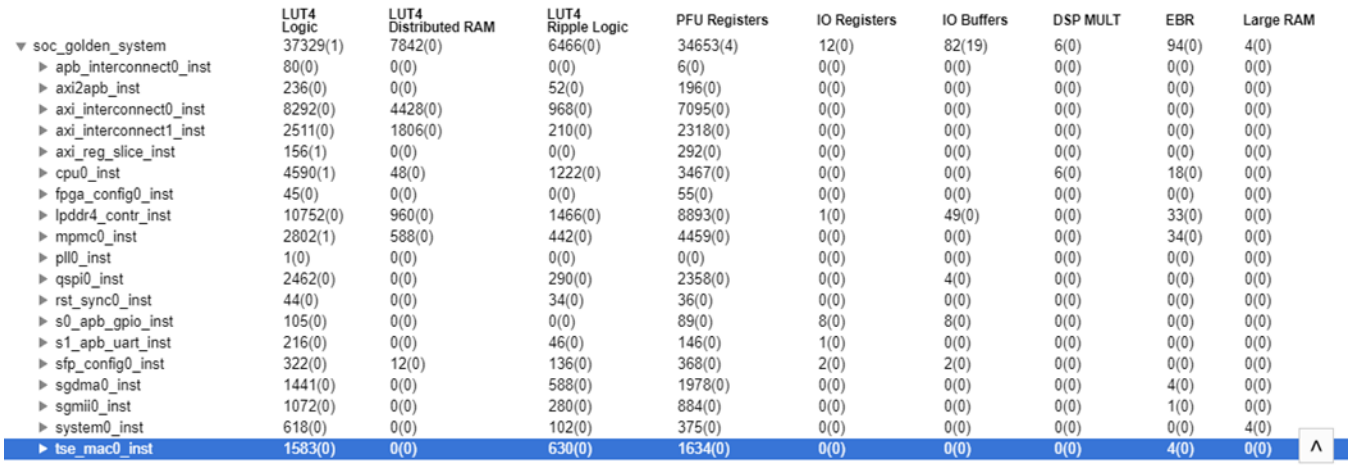
```
“---------------Golden FW Image ----------"

“FreeRTOS v10.0.1 on RISC-V."
```

For more details, refer to the GSRD bootflow diagram in the GSRD Demo User Guide (FPGA-UG-02205).

4. Resource Utilization

Figure 4.1 shows the GSRD resource utilization and Table 4.1 shows the total LUT4, PFU register, I/O buffer, and EBR resource utilization.



	LUT4 Logic	LUT4 Distributed RAM	LUT4 Ripple Logic	PFU Registers	IO Registers	IO Buffers	DSP MULT	EBR	Large RAM
▼ soc_golden_system	37329(1)	7842(0)	6466(0)	34653(4)	12(0)	82(19)	6(0)	94(0)	4(0)
▶ apb_interconnect0_inst	80(0)	0(0)	0(0)	6(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ axi2apb_inst	236(0)	0(0)	52(0)	196(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ axi_interconnect0_inst	8292(0)	4428(0)	968(0)	7095(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ axi_interconnect1_inst	2511(0)	1806(0)	210(0)	2318(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ axi_reg_slice_inst	156(1)	0(0)	0(0)	292(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ cpu0_inst	4590(1)	48(0)	1222(0)	3467(0)	0(0)	0(0)	6(0)	18(0)	0(0)
▶ fpga_config0_inst	45(0)	0(0)	0(0)	55(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ lpddr4_contr_inst	10752(0)	960(0)	1466(0)	8893(0)	1(0)	49(0)	0(0)	33(0)	0(0)
▶ mpmc0_inst	2802(1)	588(0)	442(0)	4459(0)	0(0)	0(0)	0(0)	34(0)	0(0)
▶ pll0_inst	1(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ qspi0_inst	2462(0)	0(0)	290(0)	2358(0)	0(0)	4(0)	0(0)	0(0)	0(0)
▶ rst_sync0_inst	44(0)	0(0)	34(0)	36(0)	0(0)	0(0)	0(0)	0(0)	0(0)
▶ s0_apb_gpio_inst	105(0)	0(0)	0(0)	89(0)	8(0)	8(0)	0(0)	0(0)	0(0)
▶ s1_apb_uart_inst	216(0)	0(0)	46(0)	146(0)	1(0)	0(0)	0(0)	0(0)	0(0)
▶ sfp_config0_inst	322(0)	12(0)	136(0)	368(0)	2(0)	2(0)	0(0)	0(0)	0(0)
▶ sgdma0_inst	1441(0)	0(0)	588(0)	1978(0)	0(0)	0(0)	0(0)	4(0)	0(0)
▶ sgmii0_inst	1072(0)	0(0)	280(0)	884(0)	0(0)	0(0)	0(0)	1(0)	0(0)
▶ system0_inst	618(0)	0(0)	102(0)	375(0)	0(0)	0(0)	0(0)	0(0)	4(0)
▶ tse_mac0_inst	1583(0)	0(0)	630(0)	1634(0)	0(0)	0(0)	0(0)	4(0)	0(0)

Figure 4.1. GSRD Resource Utilization

Table 4.1. GSRD Total Resource Utilization

Resource	Usage
LUT4	51637
PFU Register	34653
I/O Buffers	82
EBR	94

Appendix A. GSRD Bitstream and Binary Generation

This section describes the steps for generating the bitstream and binary for the GSRD reference design.

A.1. Primary GSRD

A.1.1. Generating the Binary

1. Open the Lattice Propel 2023.2 application.
2. To select the workspace, browse to the template location or where your project is located by clicking on the **Browse** button as shown in Figure A.1 and click on **Launch** to launch the workspace.

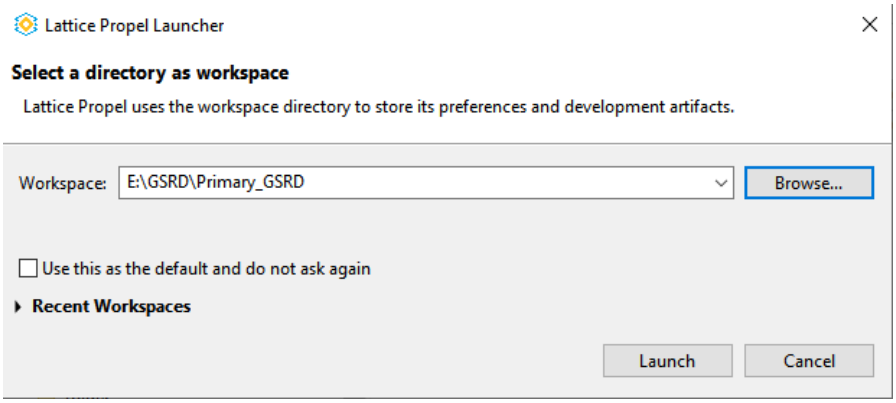


Figure A.1. Select Directory

3. Click **Import Projects** or go to **Import** from **File** to import the firmware project template.

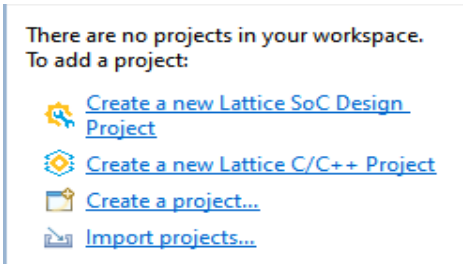


Figure A.2. Import Project

4. Select **Existing Projects into Workspace** from the **General** list and click **Next** as shown in Figure A.3.

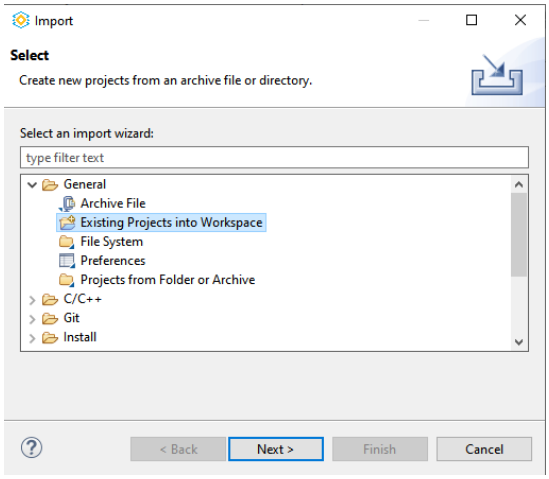


Figure A.3. Existing Project into Workspace

5. Select the root directory and browse to the template location from the Propel Builder tool in your patch installation directory.

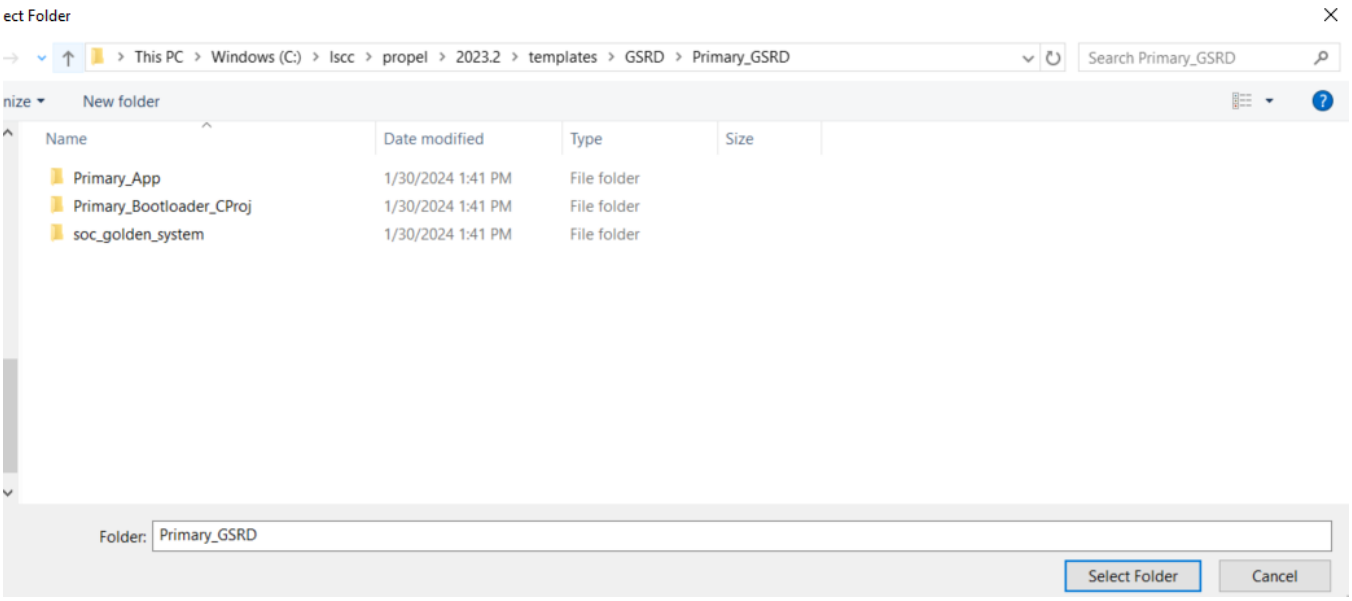


Figure A.4. Primary GSRD Folder

6. Select the project *Primary_GSRD* as shown in Figure A.5.
7. Click **Finish**.

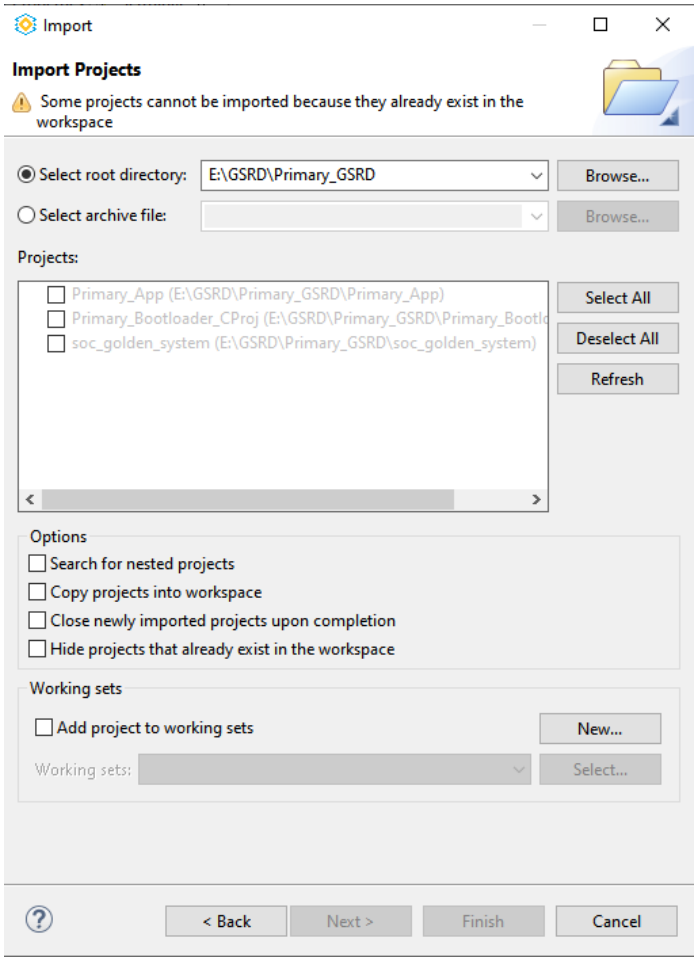


Figure A.5. Import Project

A.1.1.1. Primary_Bootloader_Cproj

1. Right-click on the firmware project folder *Primary_Bootloader_Cproj* and select the option as shown in Figure A.6 to clean the project before building.

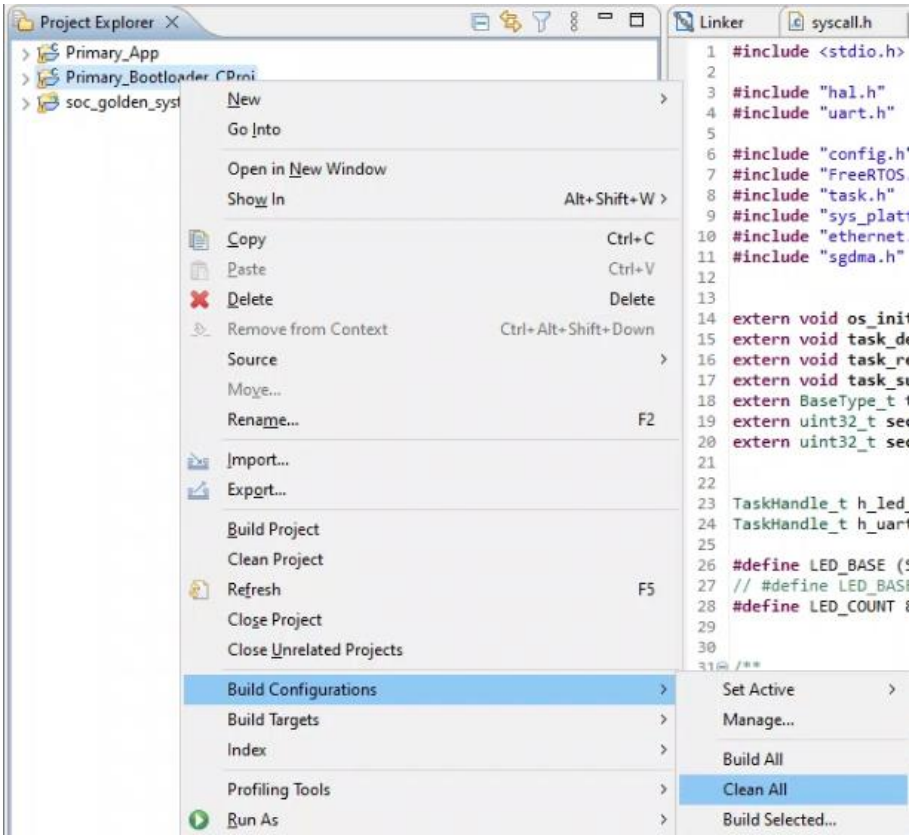


Figure A.6. Clean All Configurations

2. After selecting the option as shown in Figure A.6, wait for the process to complete. After completion, a message appears in the console as shown in Figure A.7.

Figure A.7. Console

3. After cleaning, right-click on *Primary_Bootloader_Cproj* and select the option as shown in Figure A.8 to build the project.

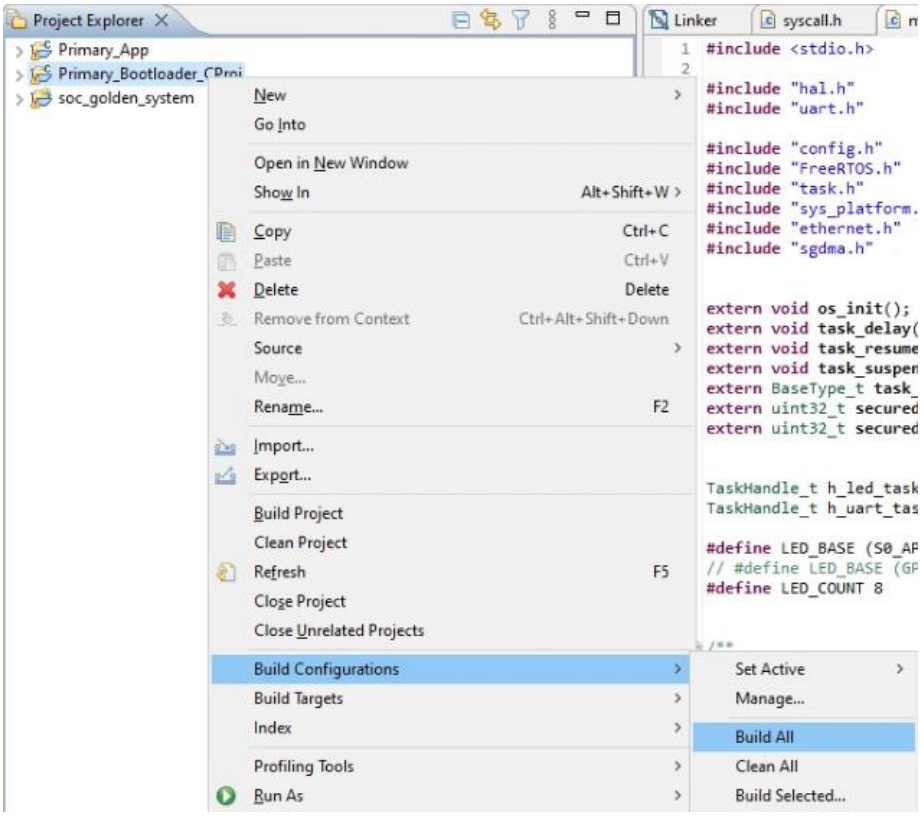


Figure A.8. Build All

4. Wait for the process to complete to 100%. After completion, a message appears in the console as shown in Figure A.9.

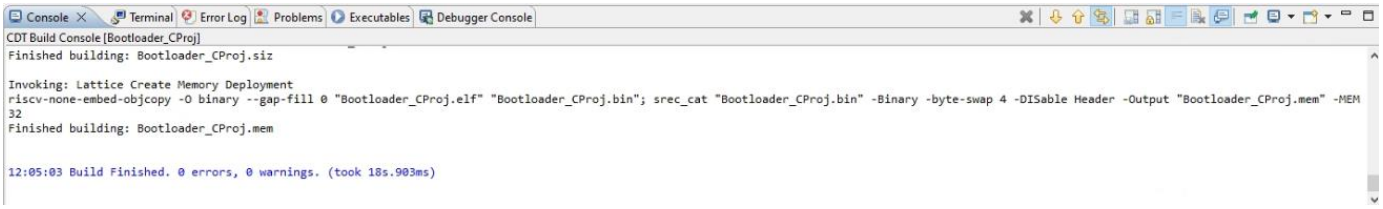


Figure A.9. Completing Process

5. Locate the binary file and *.mem* file in the following path: *Primary_GSRD**Primary_Bootloader_Cproj**Debug*.

A.1.1.2. Primary_App

1. Right-click on the firmware project folder *Primary_App* and select the option as shown in Figure A.10 to clean the project before building.

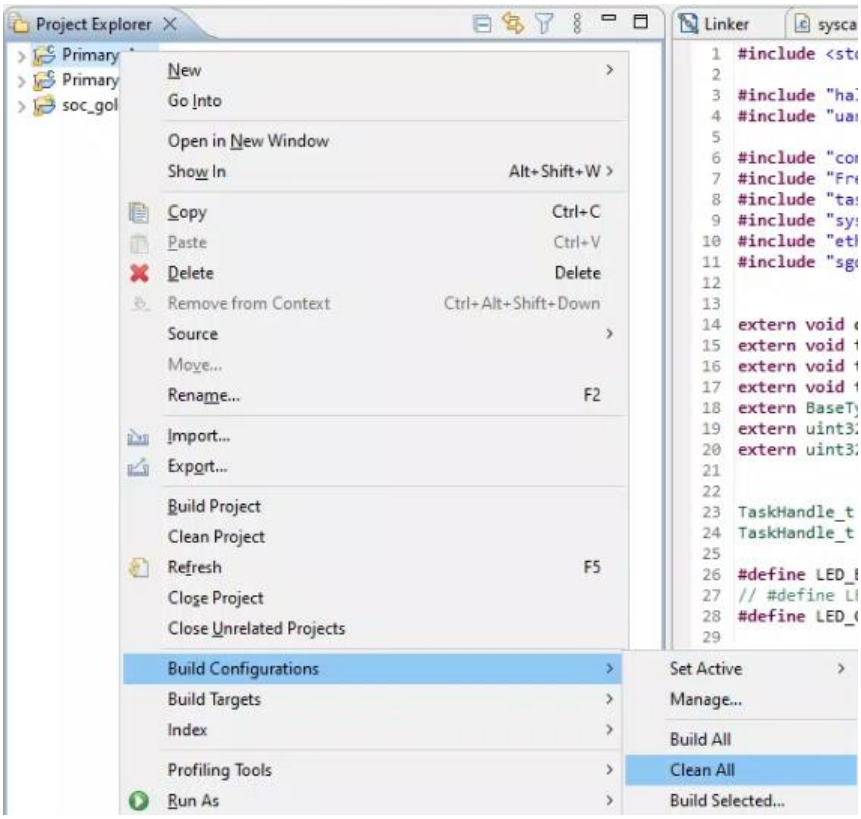


Figure A.10. Clean All Configurations

2. Wait for the process to complete to 100%. After completion, a message appears on the console as shown in Figure A.11.

Figure A.11. Console

3. After cleaning, right click on *Primary_App* and select the option as shown in Figure A.12 to build the project.

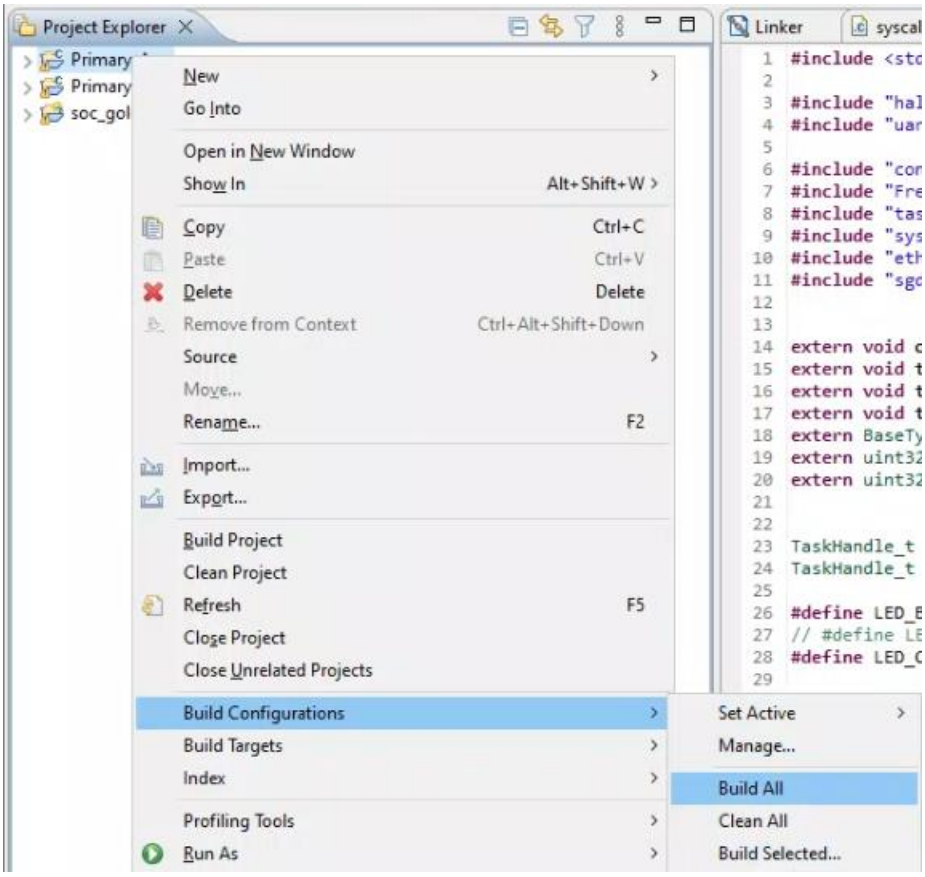


Figure A.12. Build All

4. Wait for the process to complete to 100%. After completion, a message appears in the console as shown below.

Figure A.13. Completing Process

5. Locate the binary crc file in the following path: *Primary_GSRD**Primary_App**Debug*.

Notes:

- *Primary_Bootloder_Proj* is used for mem file initialization in Propel Builder.
- *Primary_AppCrc* is used for binary file.

A.1.2. Generating the Bit File

1. Open the Propel Builder 2023.2 tool.
2. Open the *soc_golden_system* project *sbx* file from: *\Primary_GSRD\soc_golden_system\soc_golden_system\soc_golden_system.sbx*.
   Note: Ensure there is no space in the folder name.
3. Find *system0_inst*.
4. Double-click on *system0_inst*.
5. A pop-up appears on the screen as shown in Figure A.14.
6. Initialize data memory with the generated Primary_Bootloader_Cproj.mem file in \Primary_GSRD\Primary_Bootloader_Cproj\Debug.

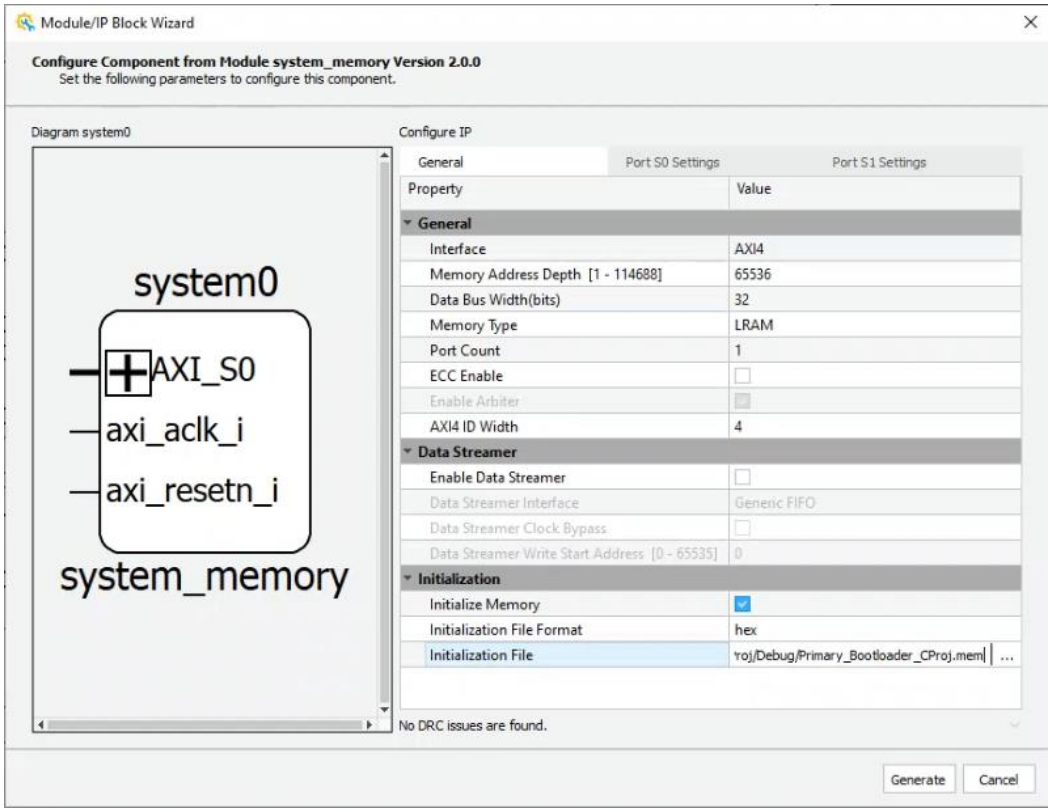


Figure A.14. System Initialization File

7. Click **Generate.** After that, click **Finish**.
8. Click the **Validate** button as shown in Figure A.15.

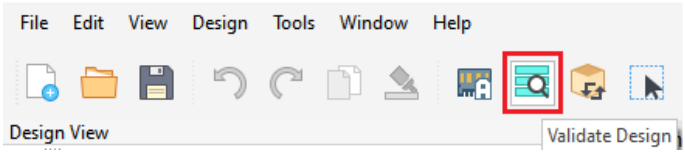


Figure A.15. Validate Button

9. Click the **Generate SGE** button as shown in Figure A.16.

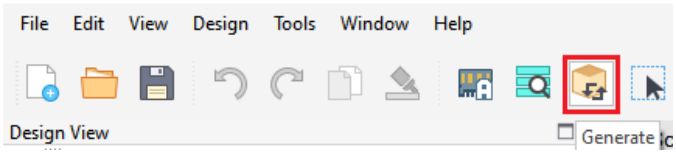


Figure A.16. Generate SGE Button

10. Open the the **Radiant** tool from the Propel Builder interface as shown in Figure A.17.

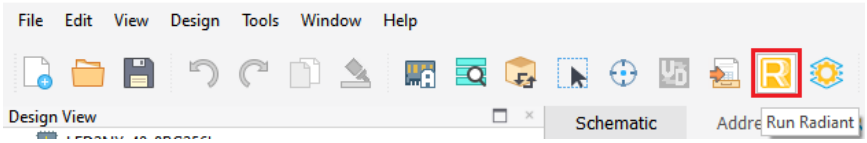


Figure A.17. Radiant Tool Button

11. Click **Run All** to generate a *bit* file. Wait for the *bit* file generation and check the output logs.

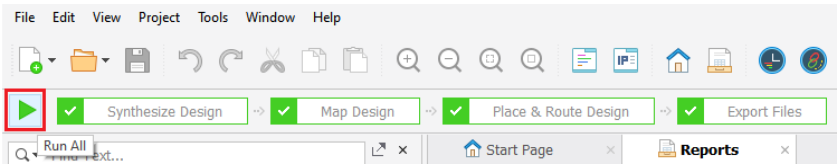


Figure A.18. Run All Button

12. Locate the bit stream file in the following path: *Primary_GSRD\soc_golden_system\impl_1*.

    Note: To programme the flash of GSRD, refer to *Appendix A* of the GSRD Demo User Guide (FPGA-UG-2205) .

A.2. Golden GSRD

A.2.1. Generating the Binary File

1. Open the Propel 2023.2 application.
2. To select the workspace, browse to the template location or where your project is located. Select *Golden_GSRD* by clicking on the **Browse** button as shown in Figure A.19.
3. Click on the **Launch** button to launch the workspace.

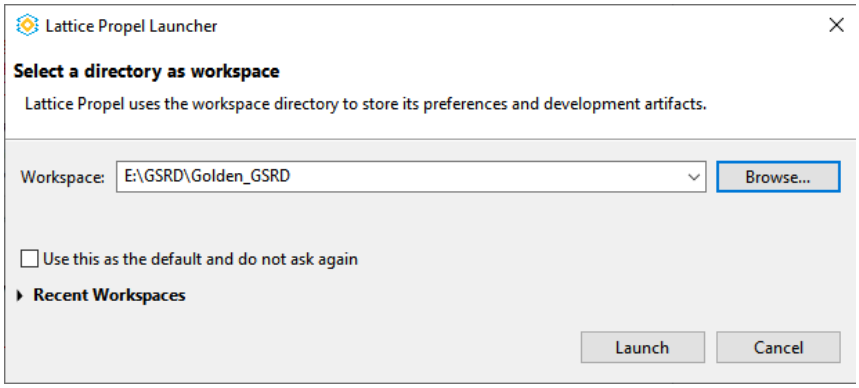


Figure A.19. Select Directory

4. Click **Import projects** or go to **File > Import** to import the firmware project template.

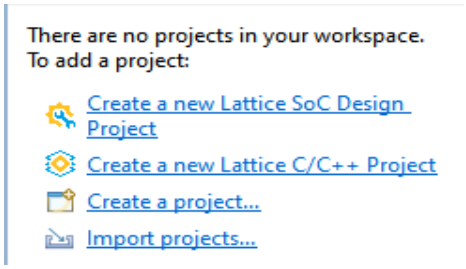


Figure A.20. Import Project

5. Select **Existing Project in Workspace** from the **General** dropdown list and click on **Next** as shown in Figure A.21.

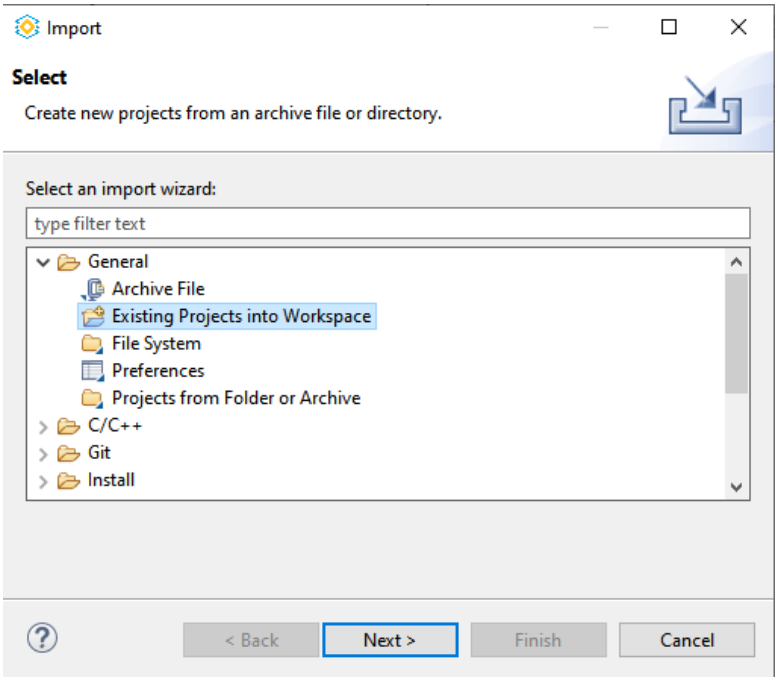


Figure A.21. Select Import Wizard

6. Select the root directory and browse to the template location from the Propel Builder tool in your patch installation directory.
7. Select the *Golden_GSRD* project as shown below.

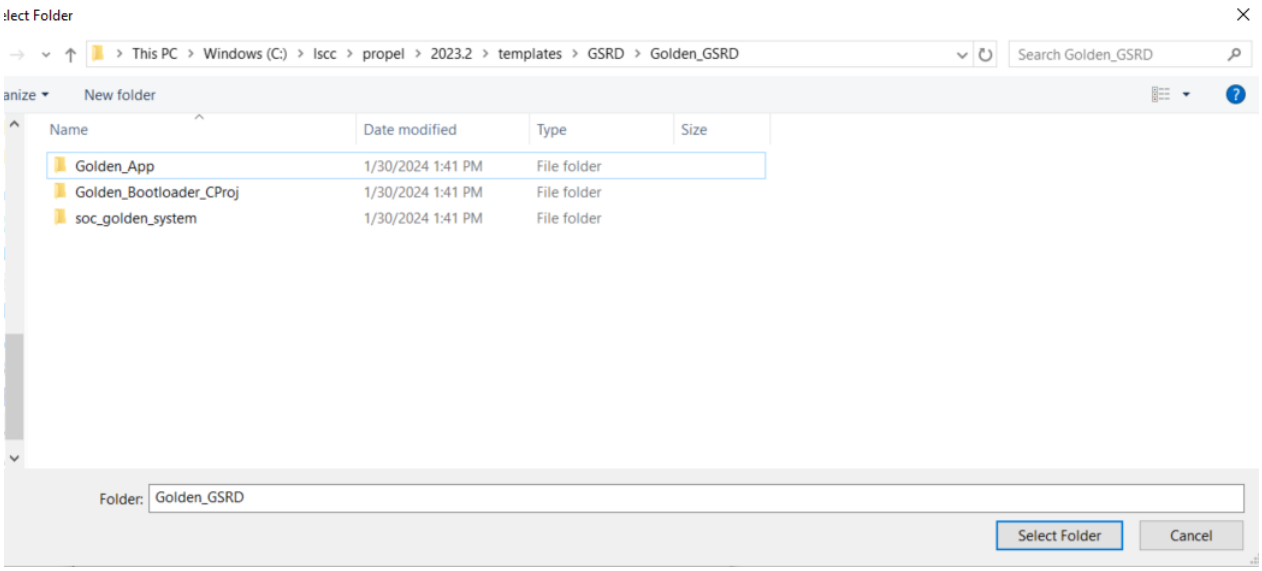


Figure A.22. Golden GSRD Folder

8. Click **Finish**.

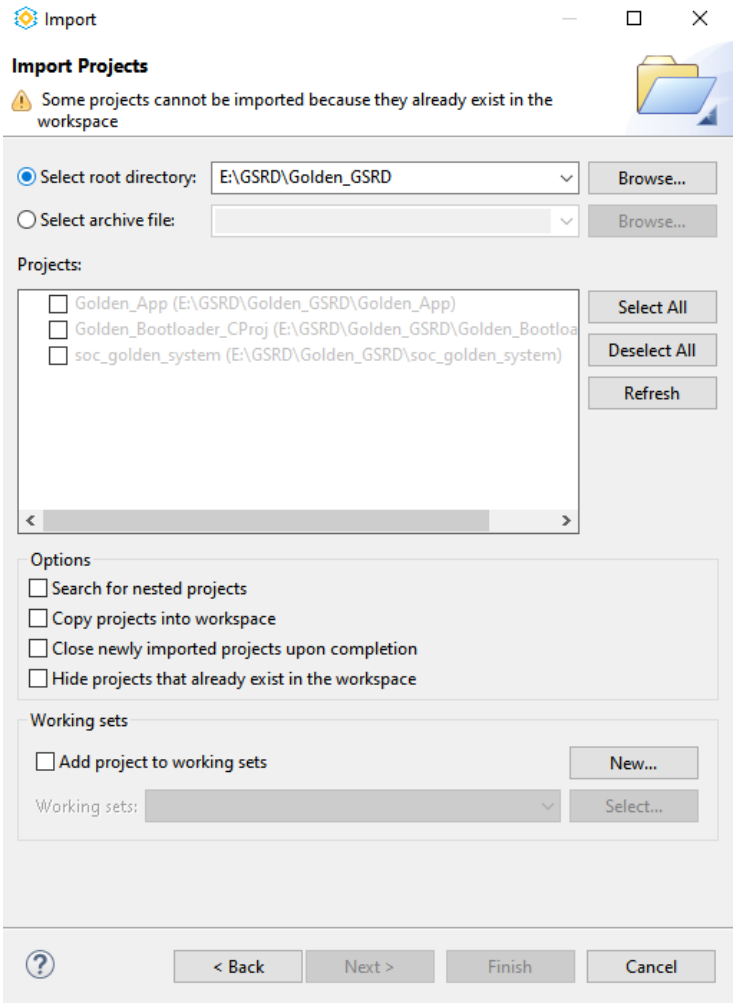


Figure A.23. Import Project

A.2.1.1 Golden_Bootloader_Cproj

1. Right-click on the firmware project folder: *Golden_Bootloader_Cproj* and select the **Clean All** option as shown in Figure A.23 to clean the project before building.

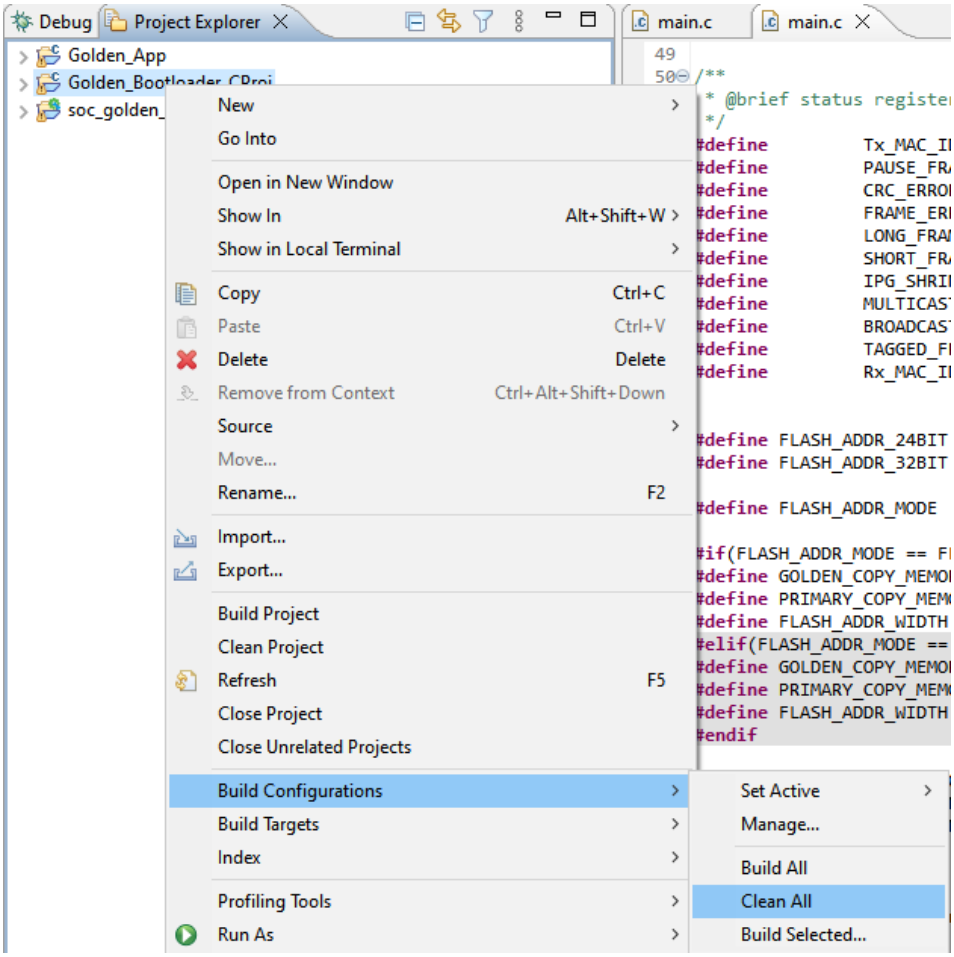


Figure A.24. Clean All Configurations

2. View the console and wait for the process to complete. After completion, a message is printed in the console as shown in Figure A.25.

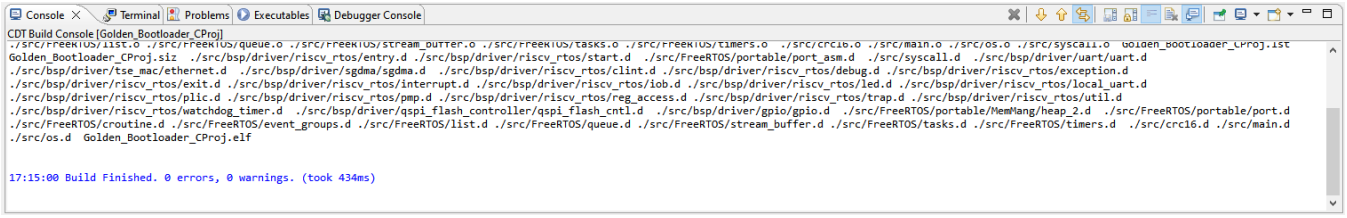


Figure A.25. Clean Complete

3. After cleaning, right click on *Golden_Bootloader_Cproj* and select the **Build All** option as shown in Figure A.26 to build the project.

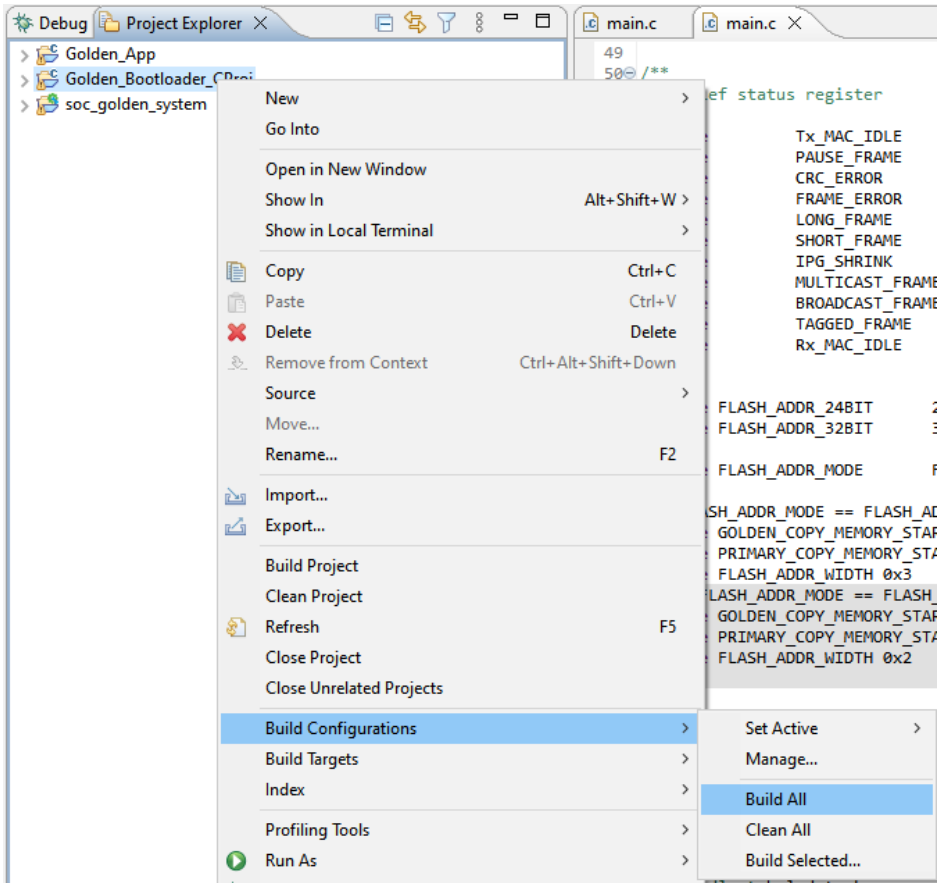


Figure A.26. Build All

4. Wait for the process to complete. After completion, a message is printed in the console as shown in Figure A.27.

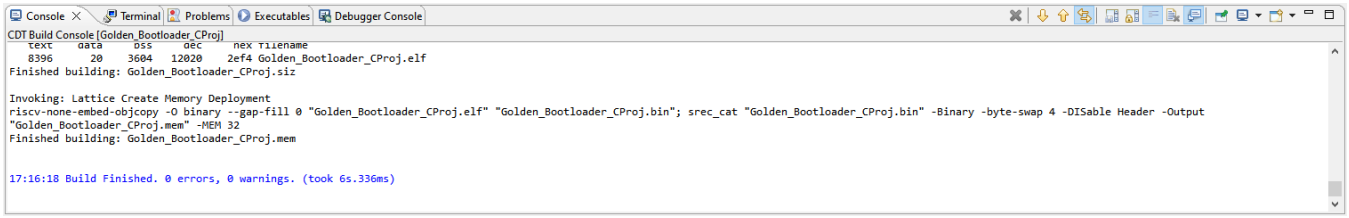


Figure A.27. Build Complete

5. To locate the binary file and *.mem* file, go to the following path: *\Golden_GSRD\Primary_Bootloader_Cproj\Debug: Primary_Bootloader_Cproj.mem*.

A.2.1.2. Golden_App

1. Right-click on the firmware project folder **Golden_App** and select the **Clean All** option as shown in Figure A.28 to clean the project before building.

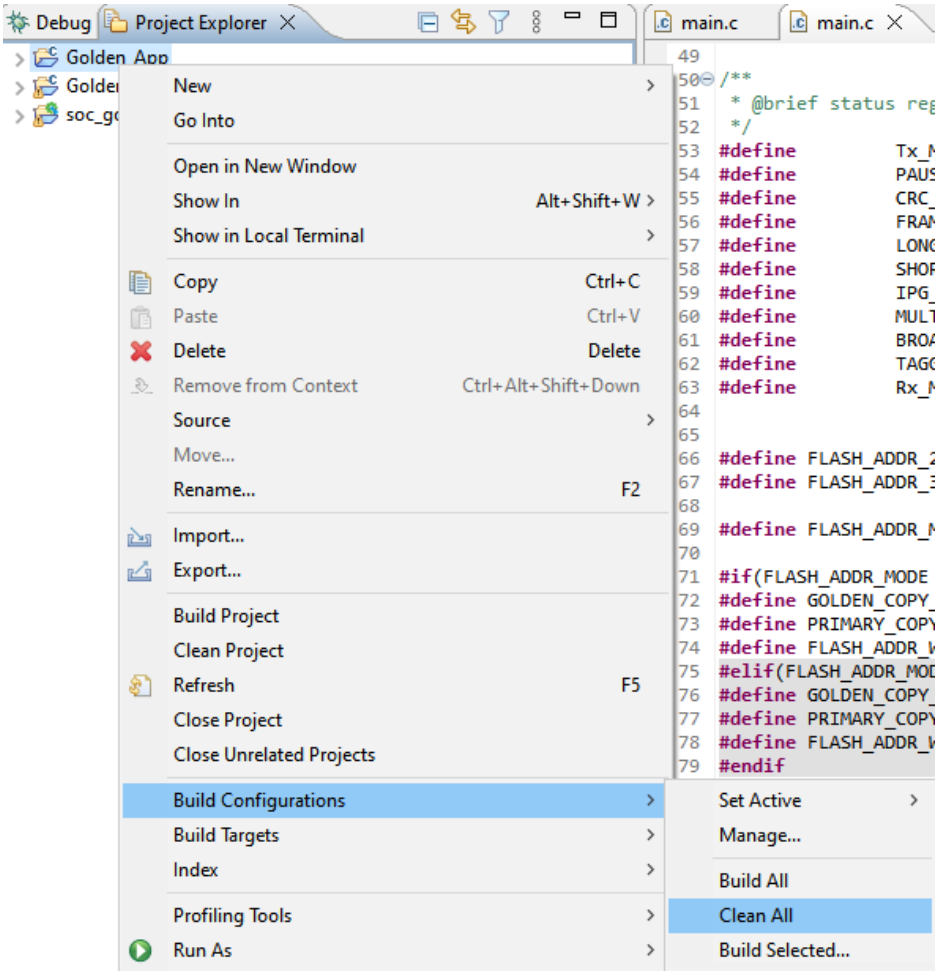


Figure A.28. Clean All Configurations

2. Wait for the process to complete. After completion, a message is printed in the console as shown in Figure A.29.

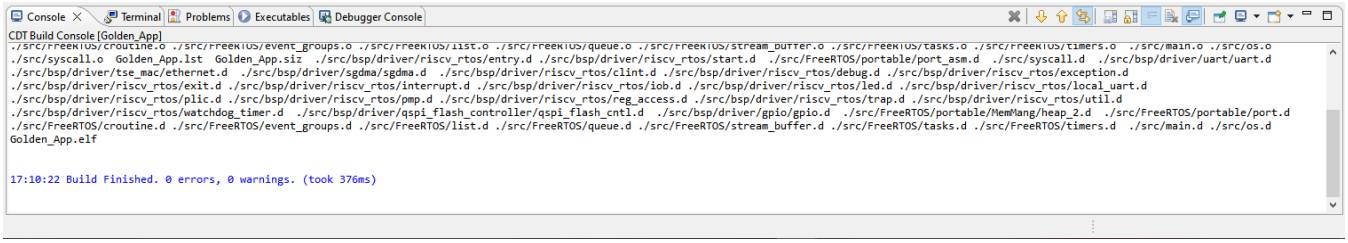


Figure A.29. Clean Complete

3. After cleaning, right-click on **Golden_App** and select the **Build All** option as shown in Figure A.30 to build the project.

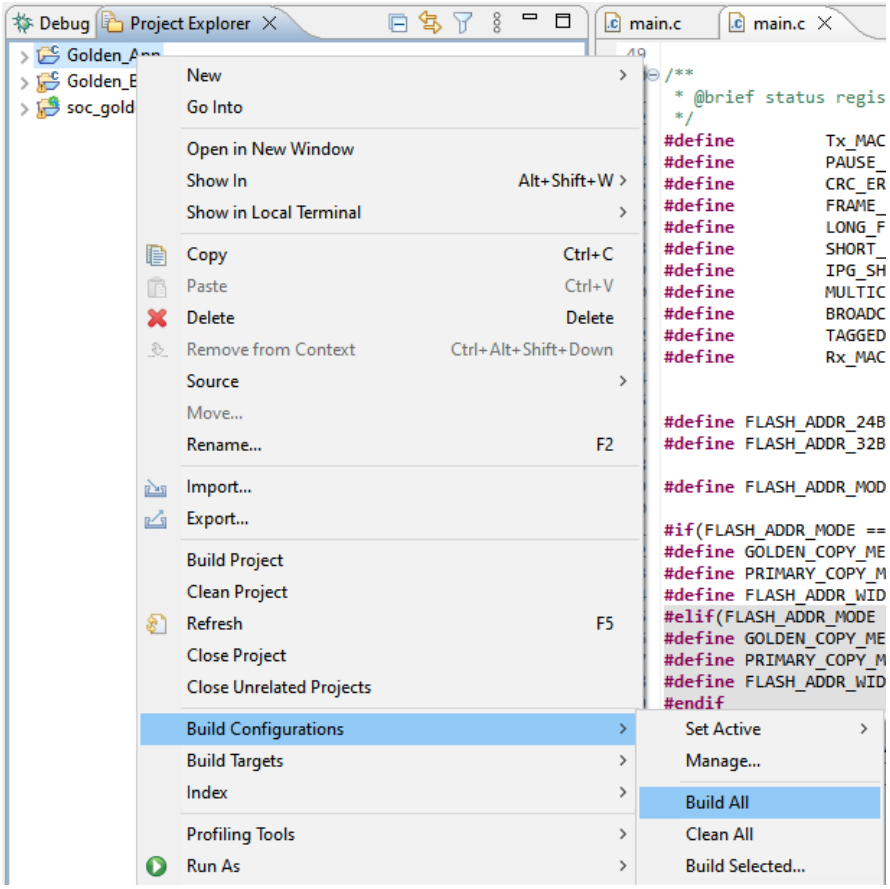


Figure A.30. Build All

4. Wait for the process to complete. After completion, a message is printed in the console as shown in Figure A.31.

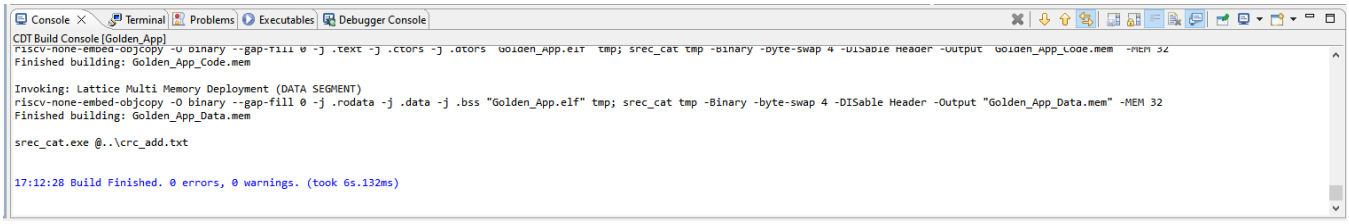


Figure A.31. Build Complete

5. To locate the binary crc file, go to: \Golden_GSRD\Golden_App\Debug: Golden_AppCrc.bin.

Notes:

- *Golden_Bootloder_Proj* is use for *mem* file initialization in the Propel Builder application.
- *Golden_AppCrc* is used for binary file.

A.2.2. Generating the Bitstream

1. Open the Propel Builder 2023.2 tool.
2. Open the *soc_golden_system* project sbx file from *\Golden_GSRD\soc_golden_system\soc_golden_system\soc_golden_system.sbx* to open the design. Ensure that there is no space in the folder name.
3. Locate and double-click on *system0_inst*.
4. A pop-up window appears on the screen as shown in Figure A.32.
5. Initialize data memory with the generated Golden_Bootloader_Cproj.mem file in \Golden_GSRD \Golden_Bootloader_Cproj\Debug.
6. Click the **Generate** button and click **Finish** after generation.

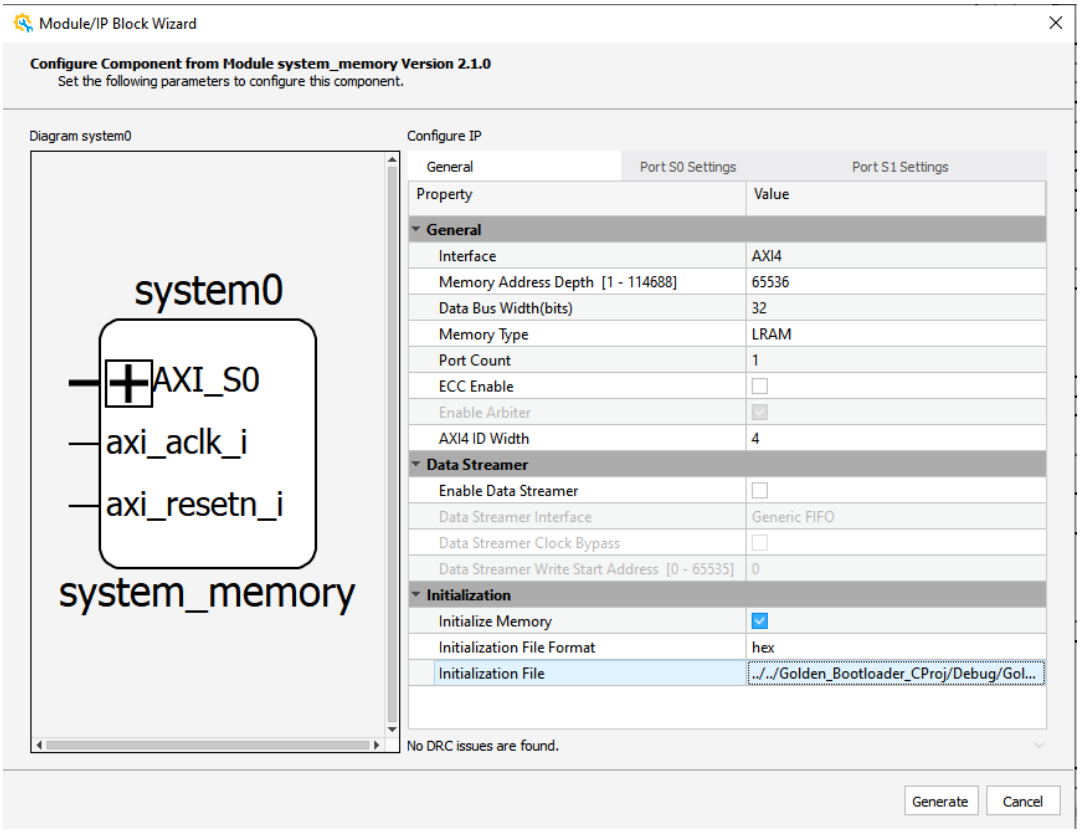


Figure A.32. System Initialization File

7. Click the **Validate** button.

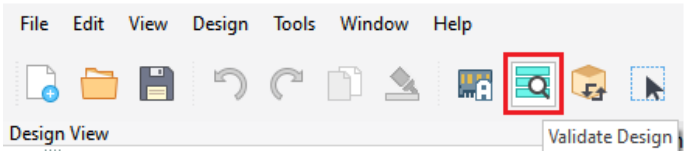


Figure A.33. Validate Button

8. Click the **Generate SGE** button.

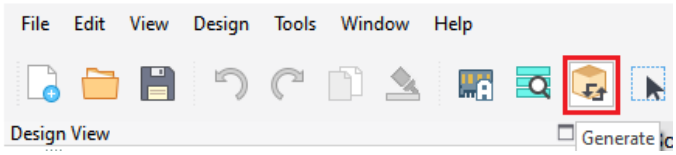


Figure A.34. Generate SGE Button

9. Open the Radiant tool from the Propel Builder interface.

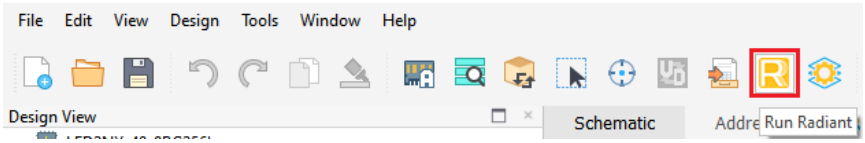


Figure A.35. Radiant Tool Button

10. Click **Run All** to generate a bit file. Wait for the bit file generation and check the output logs.

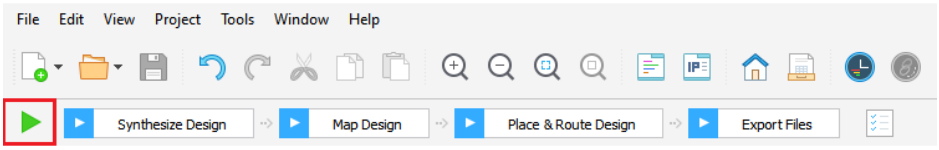


Figure A.36. Run All Button

11. Locate the bit stream file in the following path: *\Golden_GSRD\soc_golden_system\soc_golden_system\impl_1*.
Note: To program the flash of GSRD, refer to *Appendix A* of the GSRD Demo User Guide (FPGA-UG-2205).

Appendix B. Creating the MCS File

The following steps describes the procedure for generating a Multi-Boot PROM hex file using the Radiant Deployment tool.

1. In the Lattice Radiant Programmer, go to **Tools > Deployment Tool**.

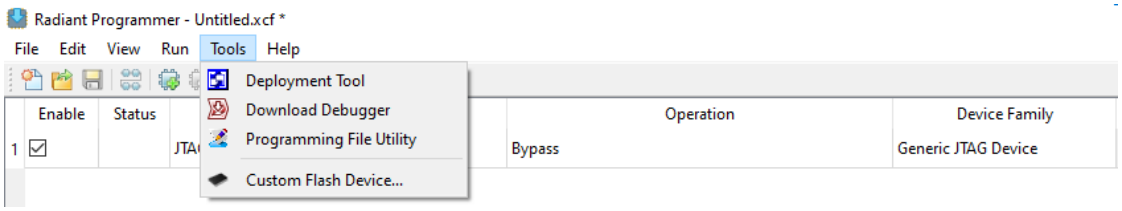


Figure B.1. Deployment Tool

2. Under **Function Type**, select **External Memory** and **Output File Type**, select **Advanced SPI Flash** as shown in Figure B.2.
3. Click **OK**.

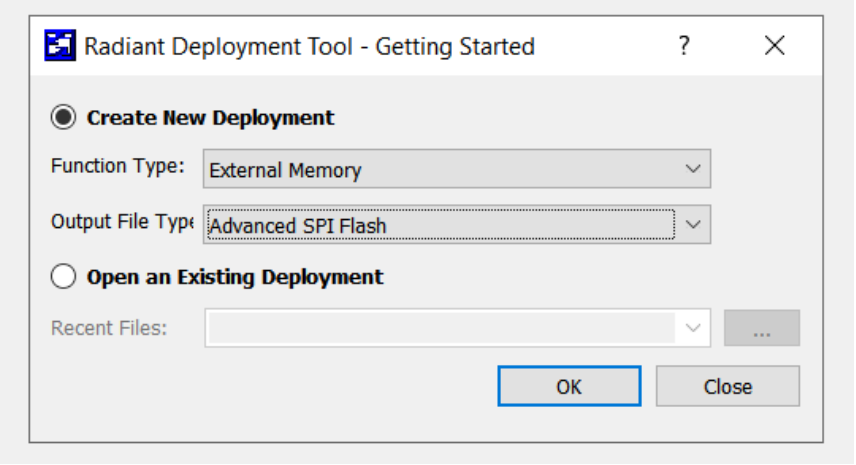


Figure B.2. Create New Deployment

4. In the **Step 1 of 4: Select Input File(s)** window:
   a. Click the **File Name** field to browse to and select the primary bitstream file to create the PROM hex file.
   b. The **Device Family** and **Device** fields auto populates based on the bitstream files selected.
   c. Click **Next**.

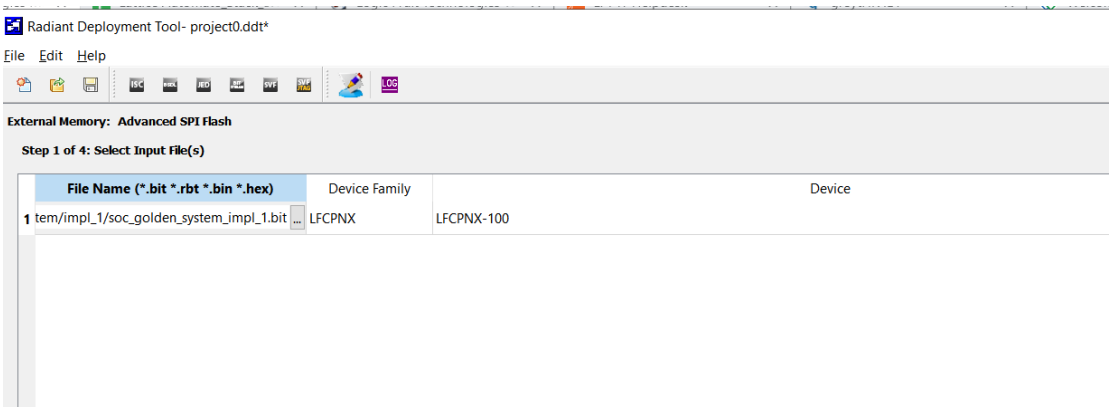


Figure B.3. Select Input File Window

5. In the **Step 2 of 4: Advanced SPI Flash Options** window:
   a. Go to the **Multiple Boot** tab.
   b. Select the **Multi Boot** option.
   c. Under **Golden Pattern**, browse and select the Golden pattern bitstream. The **Starting Address** of the Golden pattern is automatically assigned. You change it by clicking on the drop down menu.
   d. Under the **Number of Alternate Patterns** section, select the number of patterns to include from the drop down menu.
   e. In the **Alternate Pattern 1** field, click on the browse button to select the primary pattern bitstream. The **Starting Address** of the primary pattern is automatically assigned. You can change it by clicking on drop down menu.
   f. The address of next Alternate pattern to configure field is automatically populated. This is the pattern that is loaded during the next PROGRAMN/REFRESH event. You can change the pattern by clicking on the drop down menu.
   g. Click **Next**.

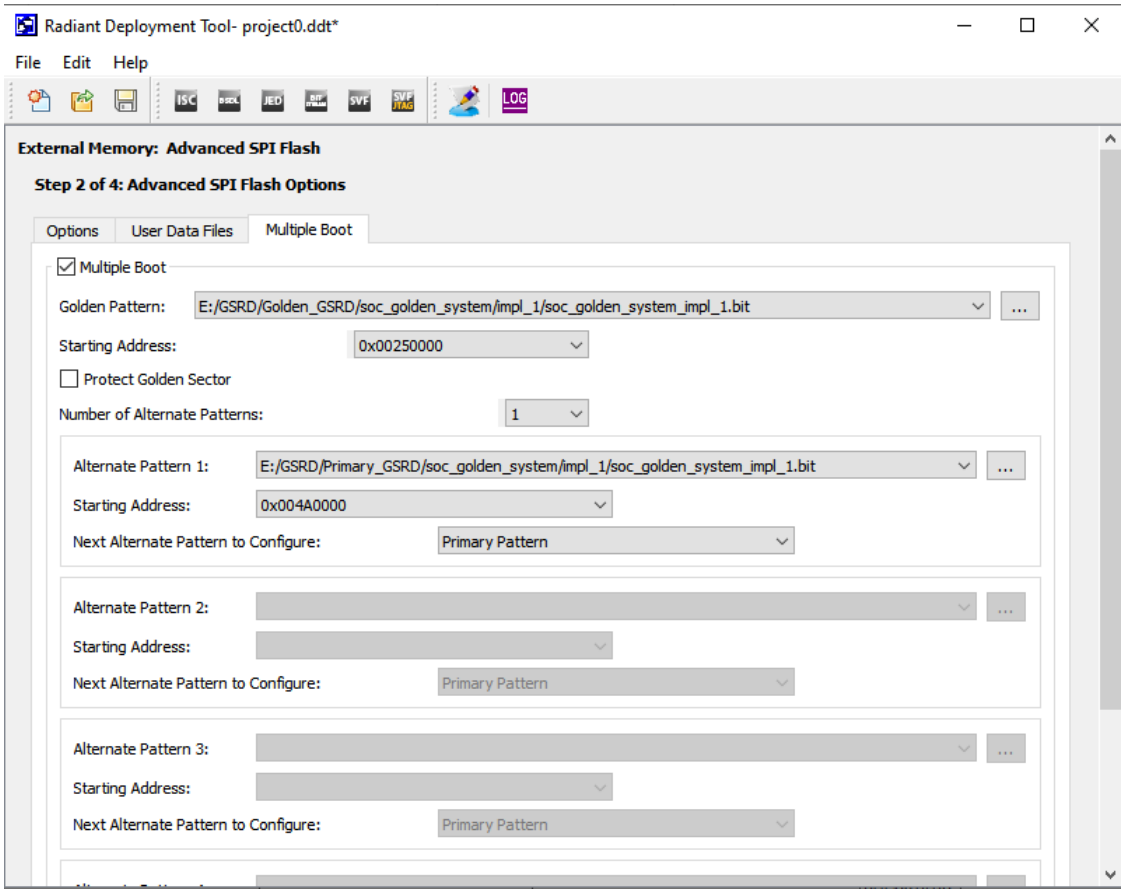


Figure B.4. Advanced SPI Flash Options –Multiple Boot Tab Window

Note: The starting address of the Golden pattern must be greater than the size of the Primary pattern and the starting address of Alternate {attern 1 should be greater than the starting address + size of golden pattern.

6. In the **Step 3 of 4: Select Output File(s)** window:
   a. Specify the name of the output PROM hex file in **Output File 1**.
   b. Click **Next**.

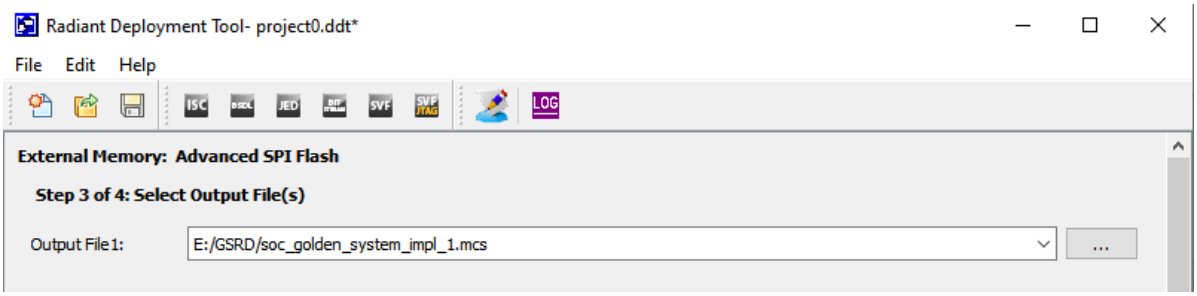


Figure B.5. Select Output File Window

7. In the **Step 4 of 4: Generate Deployment window**:
   a. Review the summary information.
   b. If everything is correct, click the **Generate** button.
   c. The generate deployment pane indicates the PROM file is successfully generated.
   d. Save the deployment setting by selecting **File > Save**.
   e. To exit, go to **File > Exit**.

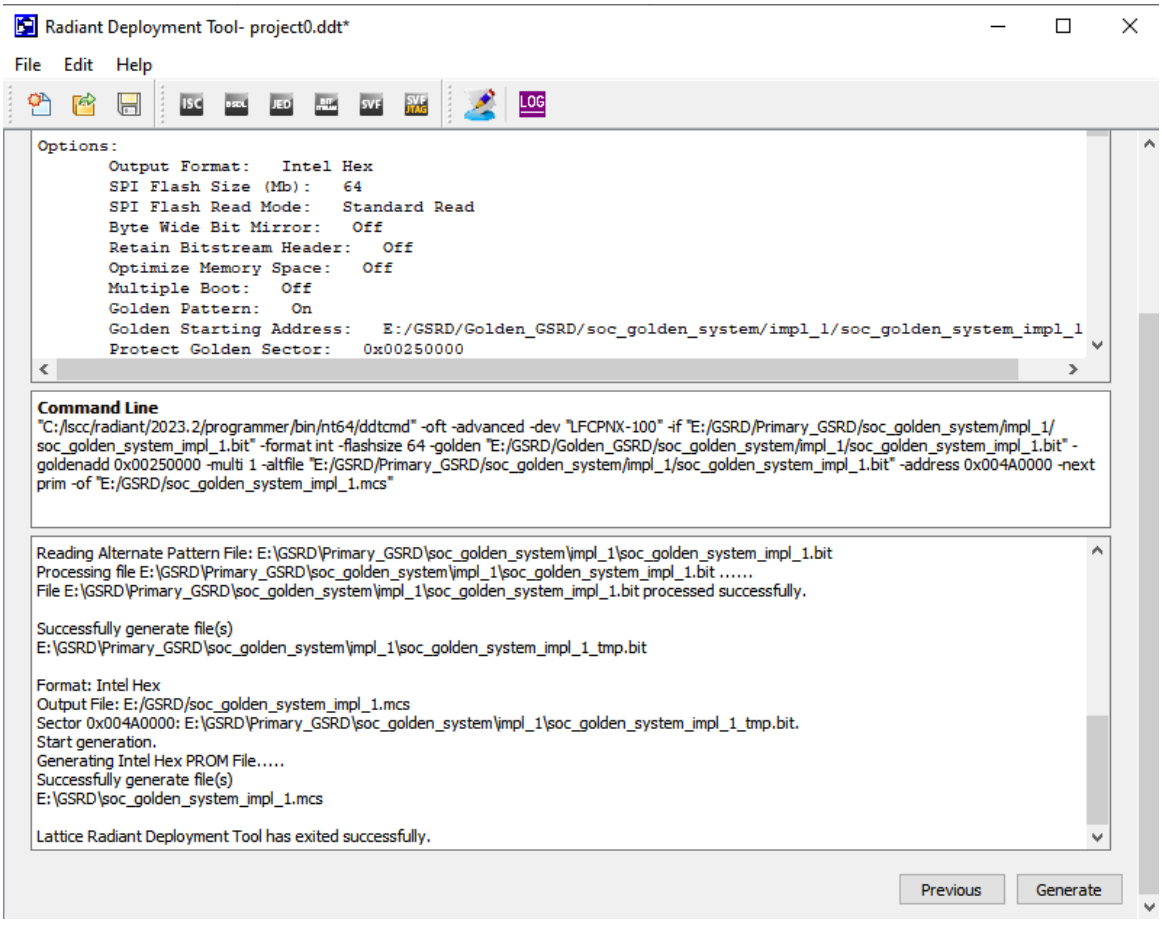


Figure B.6. Generate Deployment Window

8. You can program the *.mcs* file in the external flash using the Radiant Programmer.

References

- GSRD Demo User Guide (FPGA-UG-02205)
- SGMII and Gb Ethernet PCS IP Core - Lattice Radiant Software (FPGA-IPUG-02077)
- Tri-Speed Ethernet MAC IP Core - Lattice Radiant Software (FPGA-IPUG-02084)
- LPDDR4 Memory Controller for Nexus Devices (FPGA-IPUG-02127)
- QSPI Flash Controller IP Core (FPGA-IPUG-02248)
- AXI4 Multi Port Bridge for Memory Controller Module (FPGA-IPUG-02246)
- SGDMA Driver API Reference (FPGA-TN-02340)
- SGDMA Controller IP Core - Lattice Radiant Software (FPGA-IPUG-02131)
- UART IP Core - Lattice Propel Builder (FPGA-IPUG-02105)
- sysCONFIG User Guide for Nexus Platform (FPGA-TN-02099)
- RISC-V RX CPU IP Core (FPGA-IPUG-02241)
- CertusPro-NX web page
- Lattice Solutions IP Cores web page
- Lattice Radiant Software FPGA 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, refer to the Lattice Answer Database at www.latticesemi.com/ Support/AnswerDatabase.

Revision History

Revision 1.1, May 2024

Section	Change Summary
All	Corrected the document ID for the GSRD Demo User Guide across the document.
Acronyms in This Document	Added CLINT, LMMI, and PLIC definitions.
Introduction	Corrected References link in GHRD Components section.
Design Overview	Corrected the following typos: • DM to *SGDMA* in Design Overview section. • SDMAC to *SGDMA* in SGDMA section. • FGPA to *FPGA* in Table 2.1. GSRD Memory Map.
Firmware Components	Added link to FreeRTOS.
References	Added doc reference to RISC-V RX CPU user guide.

Revision 1.0, January 2024

Section	Change Summary
All	Initial release.

www.latticesemi.com