CrossLinkU-NX USB3 Vision

Reference Design

FPGA-RD-02319-1.1

February 2026

Disclaimers

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

Inclusive Language

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

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviations	Definition
AXI4	Advanced Extensible Interface 4
AHB	Advanced High-Performance Bus
AHB-Lite	Advanced High-Performance Bus – Lite
APB	Advanced Peripheral Bus
B2P	Byte to Pixel
CCM	Color Correction Matrix
CPU	Central Processing Unit
CSI-2	Camera Serial Interface 2
DMA	Direct Memory Access
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPS	Frame Per Second
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
IEBM	In Endpoint Buffer Manager
IEDS	In Endpoint Data Source
I2C	Inter-Integrated Circuit
INT	Interrupt
IP	Intellectual Property
IRQ	Interrupt Request
ISP	Image Signal Processing
LED	Light Emitting Diode
LMMI	Lattice Memory Mapped Interface
MC	Microcontroller
MIPI	Mobile Industry Processor Interface
PC	Personal Computer
PLL	Phase-Locked Loop
RAM	Random Access Memory
RGB	Red Green Blue
RISC-V	Reduced Instruction Set Computer – Five
RTL	Register Transfer Level
SCL	Serial Clock Line
SDA	Serial Data Line
SPI	Serial Peripheral Interface
TDP	True Dual Port
TRB	Transfer Request Block
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
U3V	USB3 Vision

1. Introduction

The Lattice Semiconductor CrossLinkU™-NX USB3 Vision (U3V) reference design provides you a template for video streaming from camera sensor over USB using the USB hard IP in a CrossLinkU-NX device. The U3V is an interface standard for industrial cameras. This design implements the standard defined in the U3V specifications.

1.1. Quick Facts

Download the reference design files from the U3V reference design web page.

Table 1.1. Summary of the Reference Design

General	Target Devices	LIFCL-33U
	Source Code Format	C code, RTL
Simulation	Functional Simulation	Not supported
	Timing Simulation	Not supported
	Hardware Validation	Fully validated
Software Requirements	Software Tool and Version	• Lattice Propel™ SDK 2024.2 • Lattice Propel Builder 2024.2 • Lattice Radiant™ Software version 2024.2 • Lattice Radiant Programmer version 2024.2
	Propel Software Soft IP Version	• RISC-V RX version 2.5.0 • Tightly Coupled Memory version 1.5.0 • AXI4 to AHB-Lite Bridge version 1.3.0 • AHB-Lite Interconnect version 1.3.2 • AHB-Lite to APB Bridge version 1.1.2 • AHB-Lite Feedthrough version 1.0.0 • APB Interconnect version 1.2.1 • I2C Controller version 2.2.0 • UART version 1.4.0 • Octal SPI Controller version 1.0.0 • GPIO version 1.7.0 Custom IP: • IEBM 2.2.1.0 • AHB-Lite to Memory Bridge version 1.0.0 • AHB-Lite to AXI-Lite Bridge version 1.0.0 • AHB-Lite to LMMI Bridge version 1.0.0
	Radiant Software Soft IP Version	• Byte-to-Pixel Converter version 1.7.0 • Color Space Converter version 2.3.0 • Debayer version 1.2.2 • CSI-2/DSI D-PHY Receiver version 1.9.0 • FIFO_DC version 2.4.0
Hardware Requirements	Board	LIFCL-33U-EVN Evaluation Board REV-B
	Camera Sensor	Raspberry PI Camera Module V2
	Cables	• USB 3.0 Superspeed Data cable (USB C to USB C cable, or USB C to USB A cable) • USB A to Micro USB cable

1.2. Features

Key features of the CrossLinkU-NX USB3 Vision reference design include:

- USB3 Vision specifications, which include device identification, device control, and data streaming.
- GenICam-compliant XML file.
- Video data streaming at USB 3 SuperSpeed at 3.4 Gbps.
- Multiple video formats and resolutions:
  - RAW8, RAW10, Mono8, Mono10, and YCbCr422.
  - 1280x720 (at 45 FPS), 1920x1080 (at 30 FPS) and 3280x2160 (at 14 FPS).
- Works with Pleora eBUS Player to control the imaging source parameters and view the video stream.
- RISC-V processor system with running Zephyr real-time operating system (RTOS).
- Soft Mobile Industry Processor Interface (MIPI) D-PHY and Camera Serial Interface (CSI-2) for image sensor aggregation. The Lattice Semiconductor D-PHY Receiver IP converts CSI-2 data to 8-bit data.
- The Lattice Semiconductor Byte-to-Pixel Converter IP converts CSI-2 standard-based video payload packets from the D-PHY Receiver module output to pixel format.
- The Lattice Semiconductor Debayer IP converts raw image data into an RGB image.
- Hard USB IP for USB video class to stream the video from the image sensor to a PC through USB Type C connector.
- Primarily targets the Raspberry PI Camera Module 2. For more information, refer to the Raspberry Pi Camera Module 2 page on the Raspberry Pi website.

1.3. Naming Conventions

1.3.1. Nomenclature

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

1.3.2. Signal Names

- *_n* are active low signals, asserted when value is logic 0.
- *_i* are input signals.
- *_o* are output signals.

2. Directory Structure and Files

This reference design source code release is controlled. Only the demo bitstream and binary files are released on the public web page. Contact Lattice Sales or Technical Support to obtain the complete source code.

The following figure shows the directory structure for this reference design with source code.

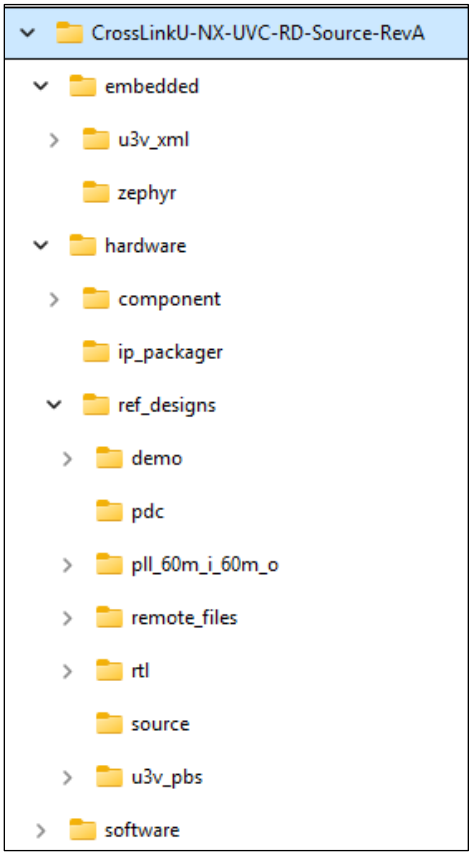

Figure 2.1. Directory Structure

The following table shows the list of files included in the reference design package.

Table 2.1. File List

Attribute	Description
embedded\u3v_xml (1)	Contains the XML file used for USB3 Vision camera.
embedded\zephyr (1)	Contains the Zephyr source code in patch format.
hardware\component (1)	Contains the custom IP and RTL used in this reference design.
hardware\ref_design (1)	Contains the complete Radiant software and Propel Builder software project files.
hardware\ref_design/demo (2)	Contains the design demo file.
software\utilities (2)	Contains the Windows® application for performing SPI flash operation in this reference design.

Notes:

1. Files that are only available in the source code-controlled release.
2. Files are available in both controlled and public releases.

3. Functional Description

The block diagram of the CrossLinkU-NX USB3 Vision reference design is shown in the figure below.

Refer to the Design Components section for the details of each component in this block diagram.

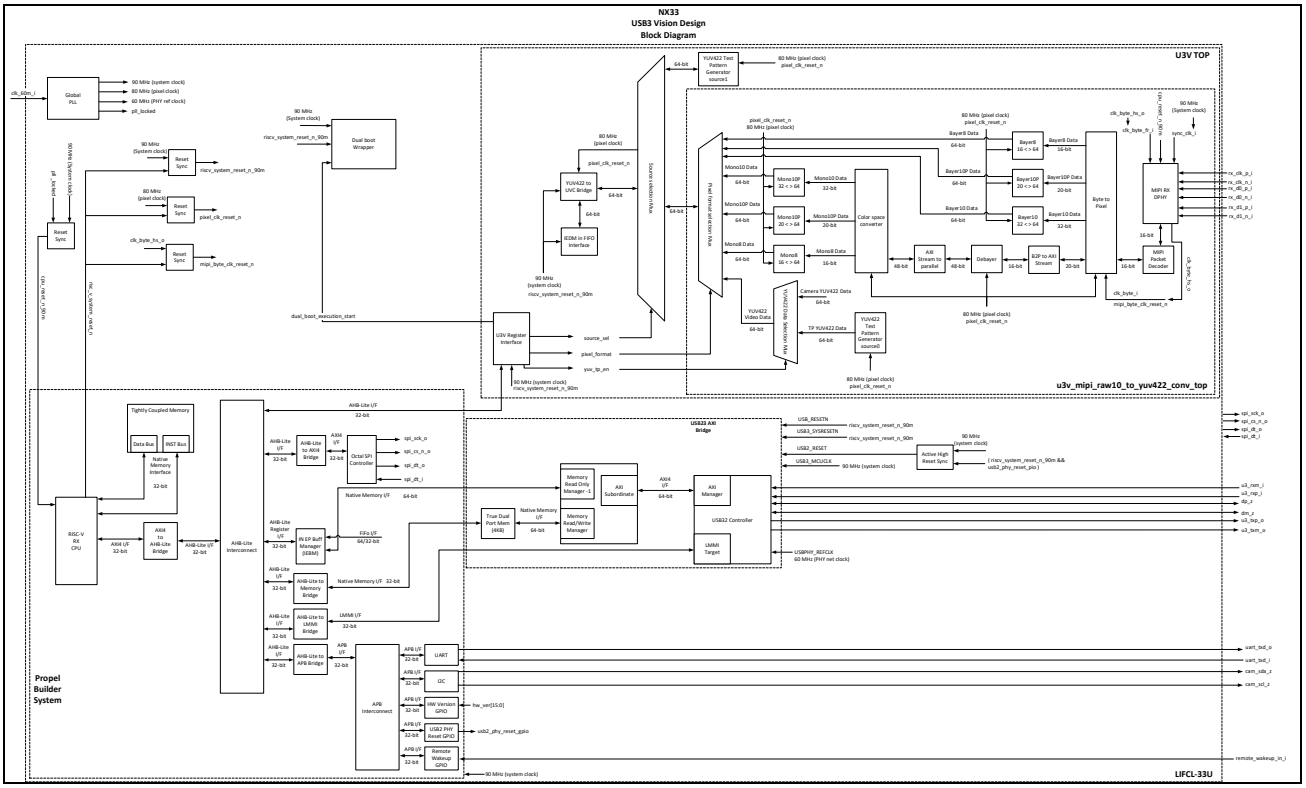

Figure 3.1. Reference Design Block Diagram

3.1. Design Components

This section describes all the blocks used in the CrossLinkU-NX USB3 Vision reference design.

3.1.1. PLL

The phase-locked loop (PLL) IP generates three clocks output with different frequencies, which are used by various blocks in the design. The USB23 PHY operates at 60 MHz, the MIPI RX DPHY and video IP operate at 80 MHz, and RISC-V subsystem operates at 90 MHz.

Figure 3.2, Figure 3.3, and Figure 3.4 show the general and optional ports configuration of the PLL IP.

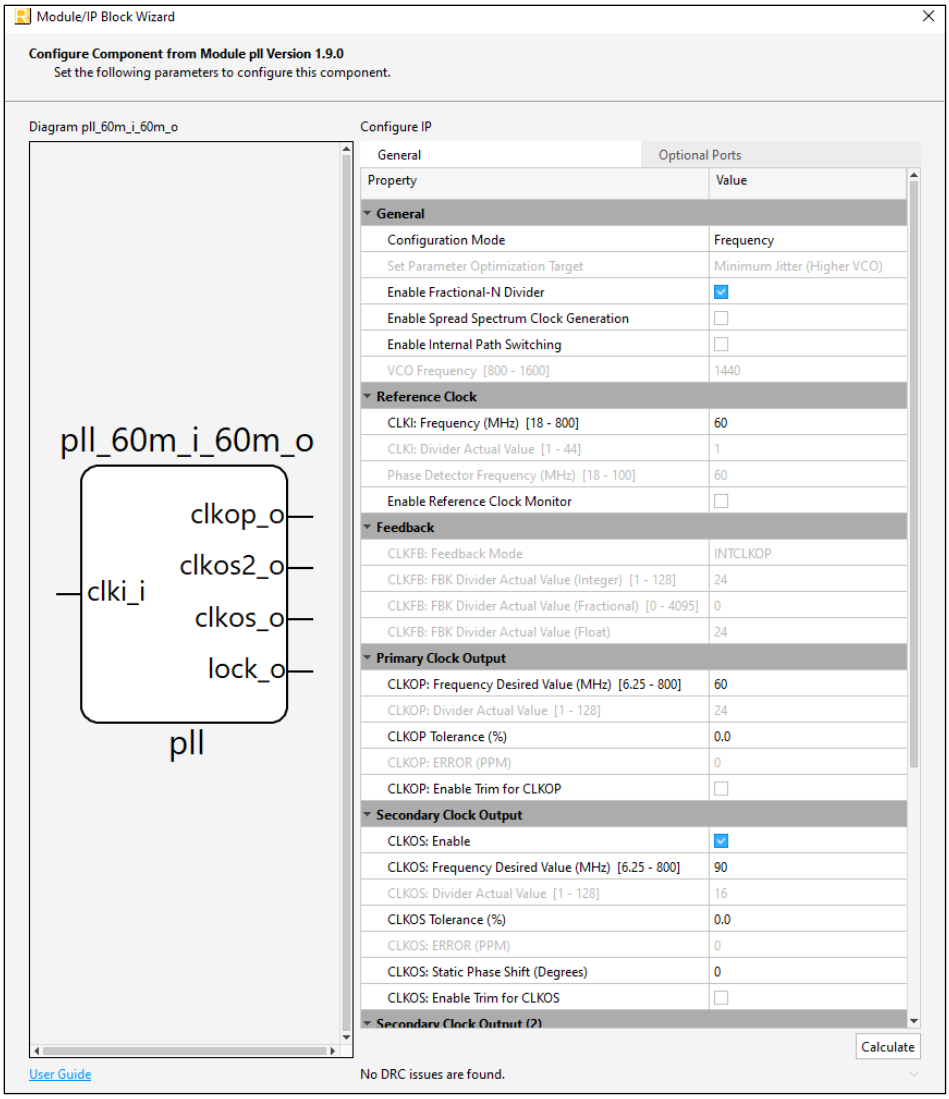

Figure 3.2. PLL Configuration – General, Reference Clock, and Primary Clock

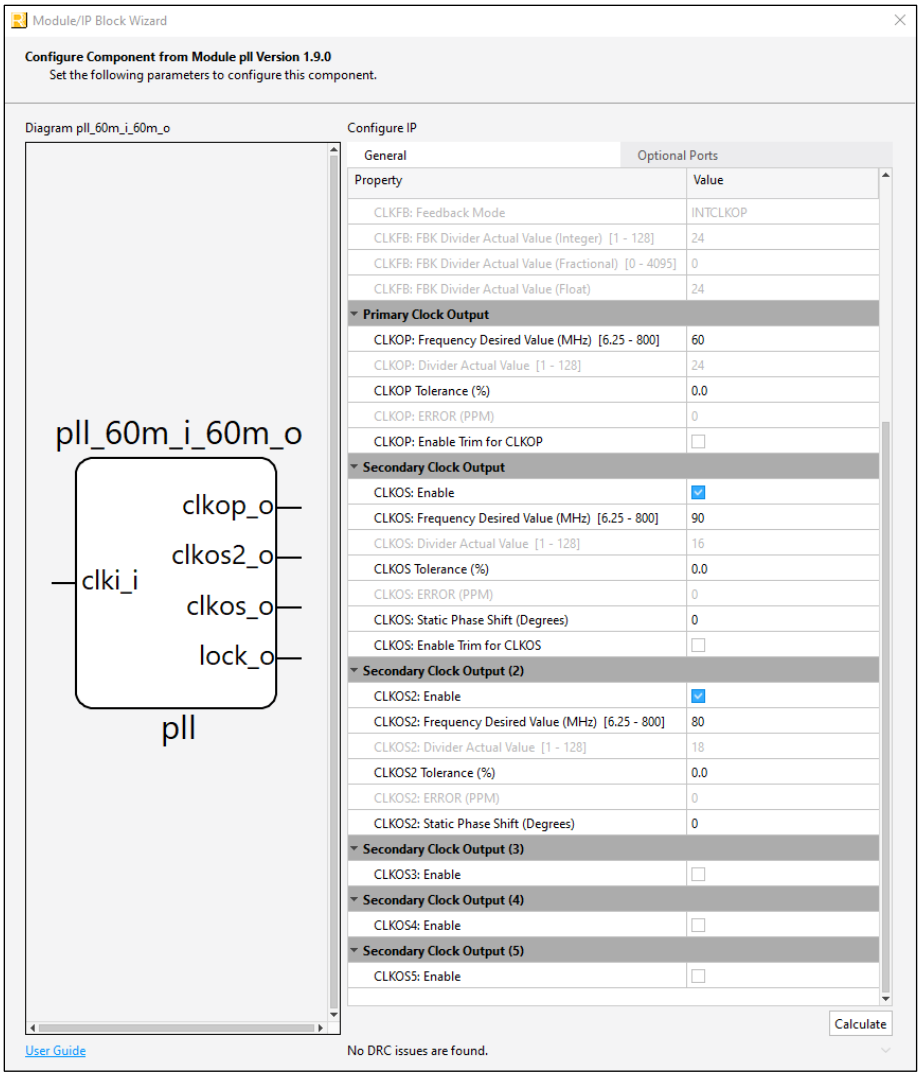

Figure 3.3. PLL Configuration – Secondary Clock

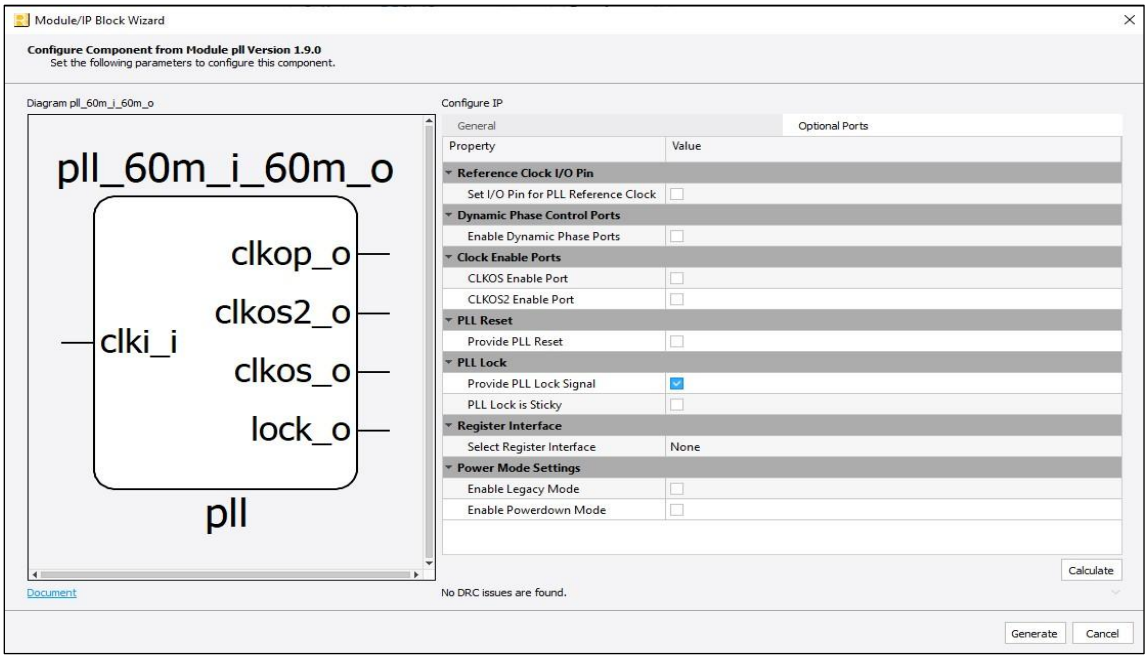

Figure 3.4. PLL Configuration – Optional Ports

3.1.2. USB23 Controller

The USB23 controller is the hardened IP core in CrossLinkU-NX FPGAs. This IP handles the USB2 and USB3 communication between the FPGA and the Host PC.

The AXI4 Manager interface of the USB23 controller is connected to the following two different memories via the USB23 AXI bridge:

- For USB Bulk transfers, the USB23 controller fetches data from IN Endpoint Buffer Manager (IEBM) block. This IEBM memory is not shared with any other manager in the system including the RISC-V processor.
- For other operations, USB23 controller access the shared memory block to communicate with the RISC-V processor.

3.1.3. AHB-Lite to LMMI Bridge

The USB23 controller register interface can be configured through its LMMI interface. The AHB-Lite to LMMI bridge allows the RISC-V processor to access the LMMI interface on the USB23 controller registers.

3.1.4. RISC-V RX Processor

The RISC-V RX processor manages the system including USB controller and endpoint operations, I2C transfers, prints messages via UART processes and others. The processor executes the Zephyr RTOS with driver support for peripherals in the system. The processor handles the external interrupts from USB23 controller and IEBM core. The RISC-V RX processor configuration is shown in Figure 3.5, Figure 3.6, Figure 3.7, and Figure 3.8.

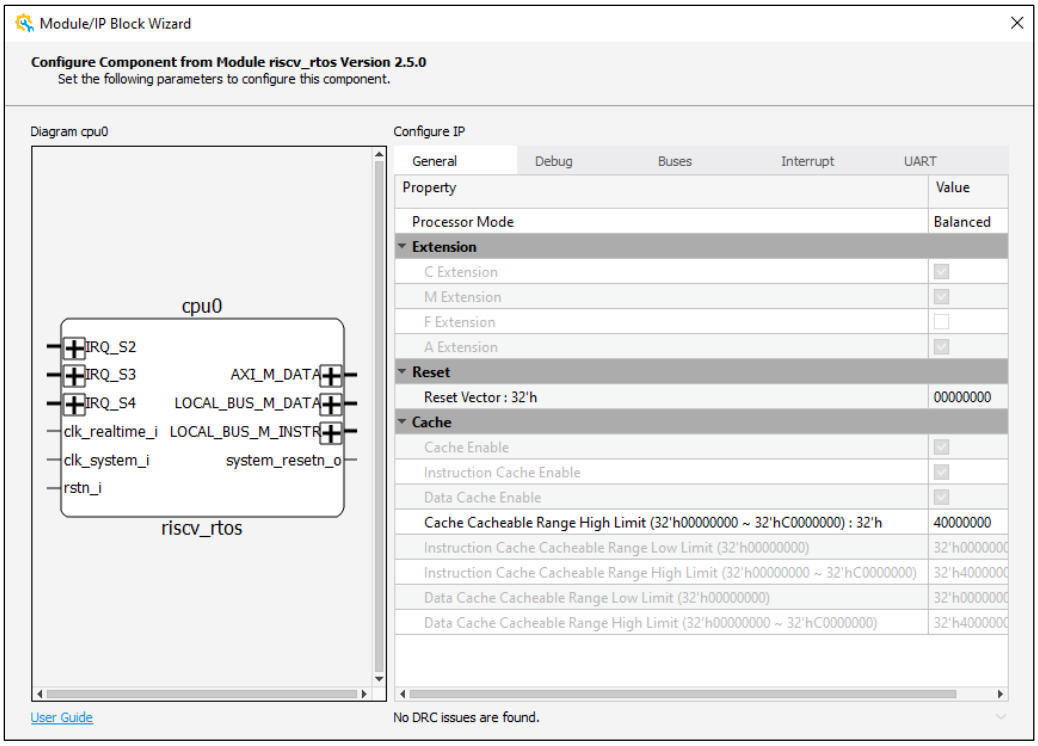

Figure 3.5. RISC-V General Tab Configuration

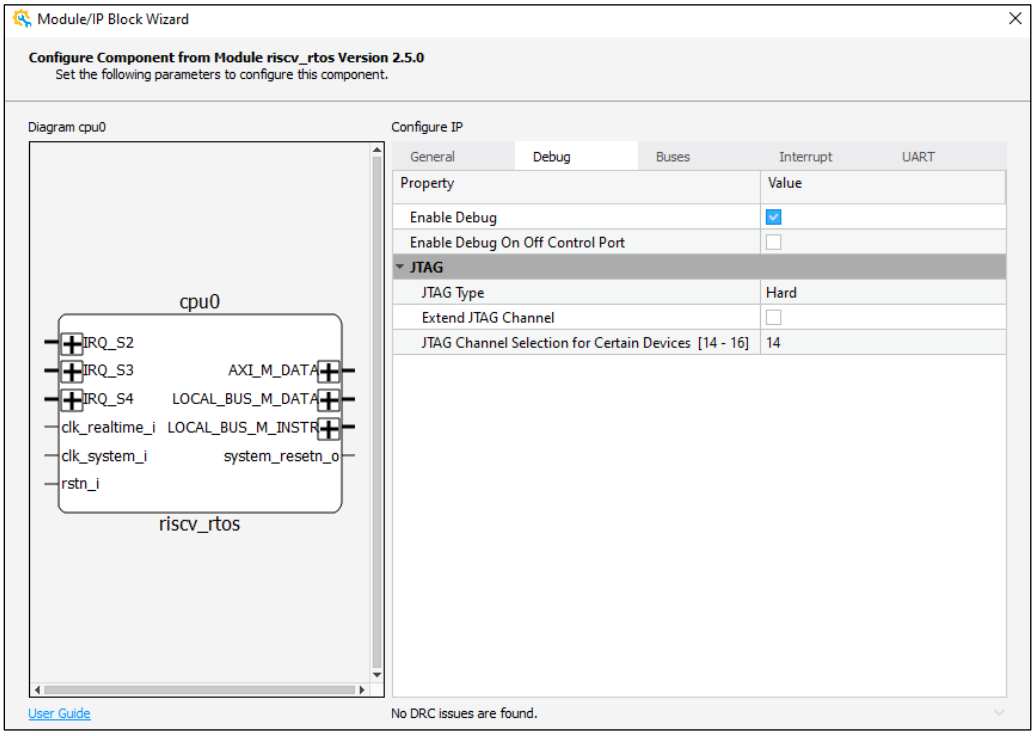

Figure 3.6. RISC-V Debug Tab Configuration

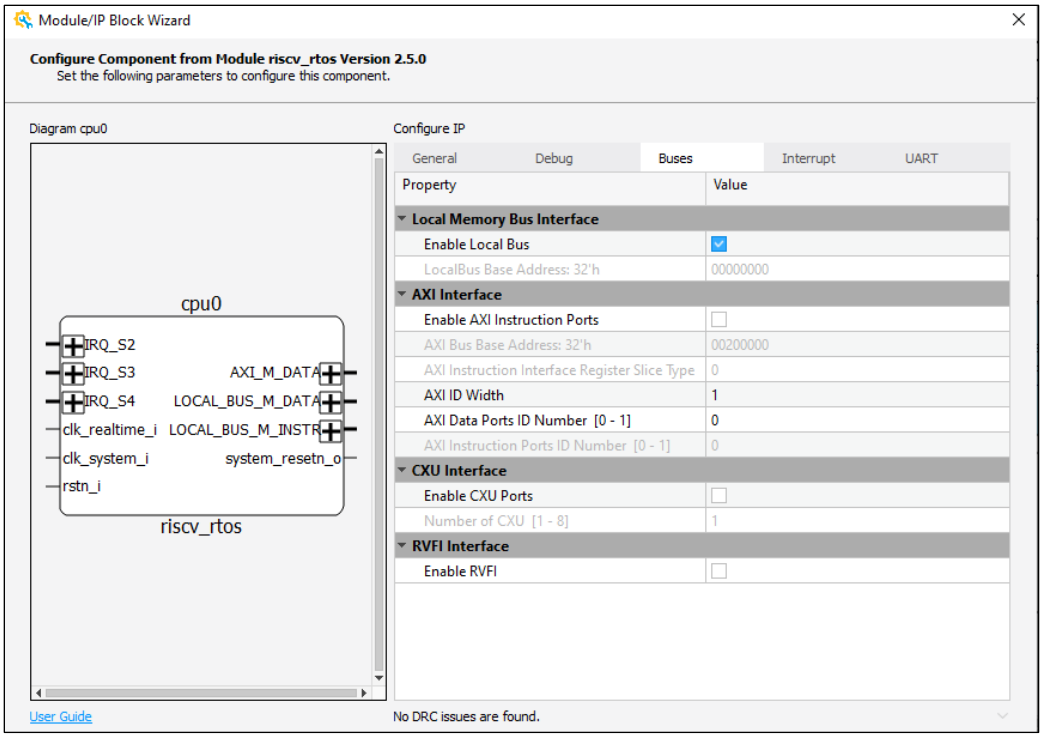

Figure 3.7. RISC-V Buses Tab Configuration

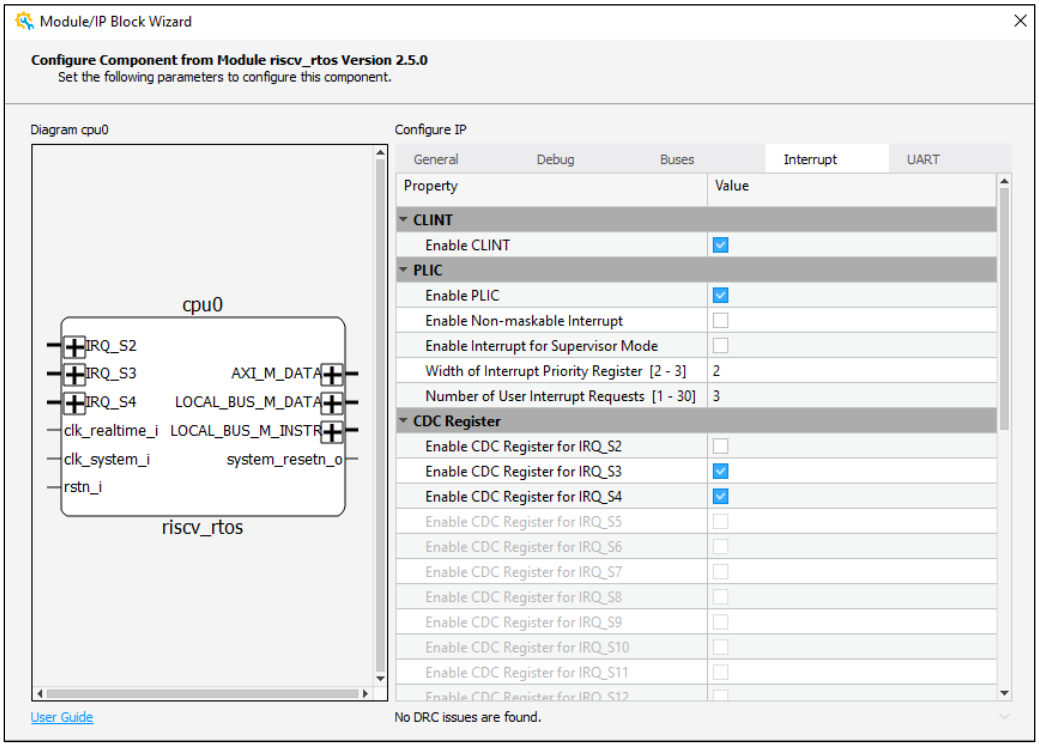

Figure 3.8. RISC-V Interrupt Tab Configuration

3.1.5. UART

The UART IP core performs serial-to-parallel conversion on data characters received from a peripheral UART device and parallel-to-serial conversion on data characters received from the host (RISC-V processor) located inside the FPGA through an APB Interface. The UART IP core configuration is shown in the figure below.

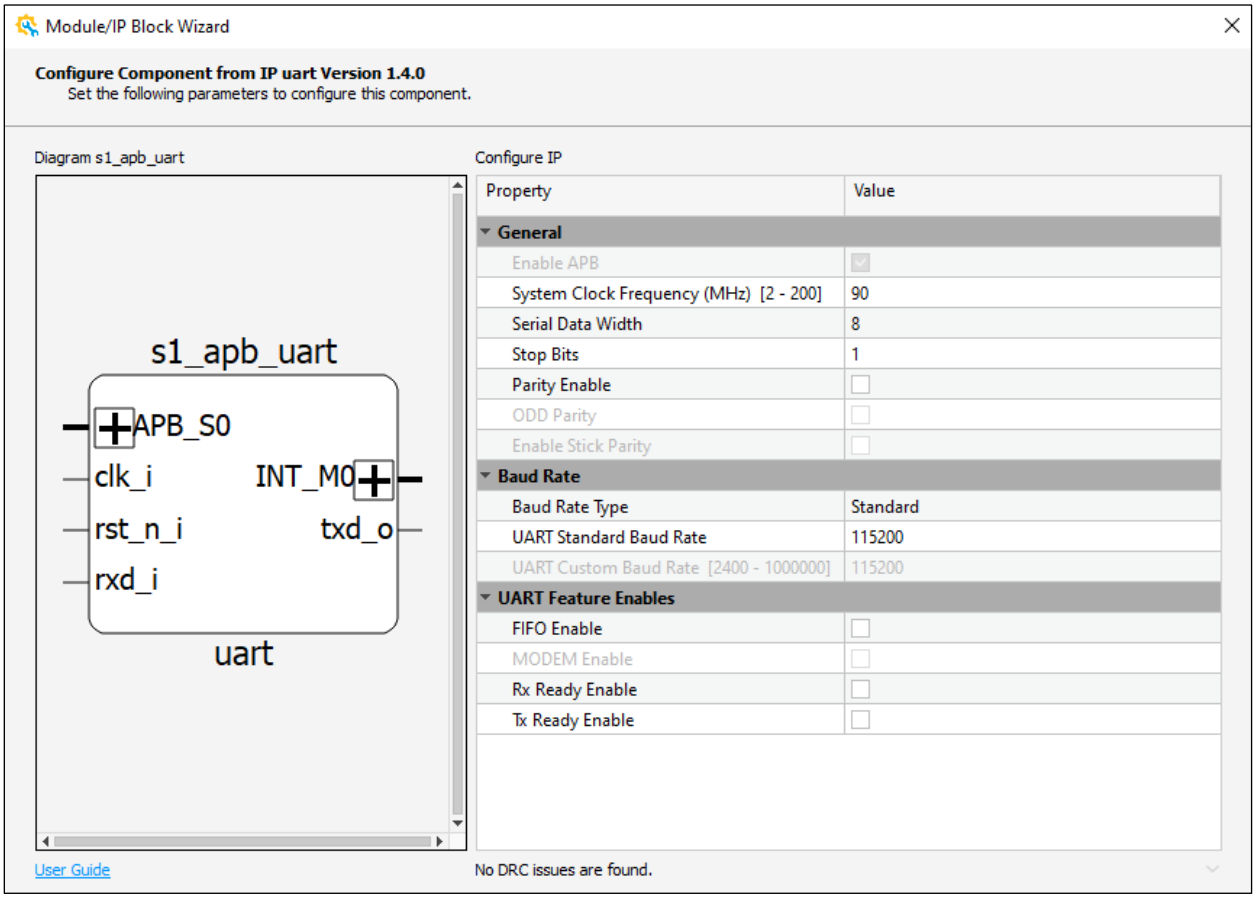

Figure 3.9. UART Configuration

3.1.6. Remote Wakeup GPIO

The Remote Wakeup GPIO input pin is typically used to trigger a wake-up event from a low-power or sleep mode through an external GPIO signal. When the remote_wakeup_in_i pin is activated (For example, button press or signal transition), it generates an interrupt to the RISC-V processor. This interrupt prompts the processor to take the necessary steps to exit the low-power or sleep state and resume normal operation.

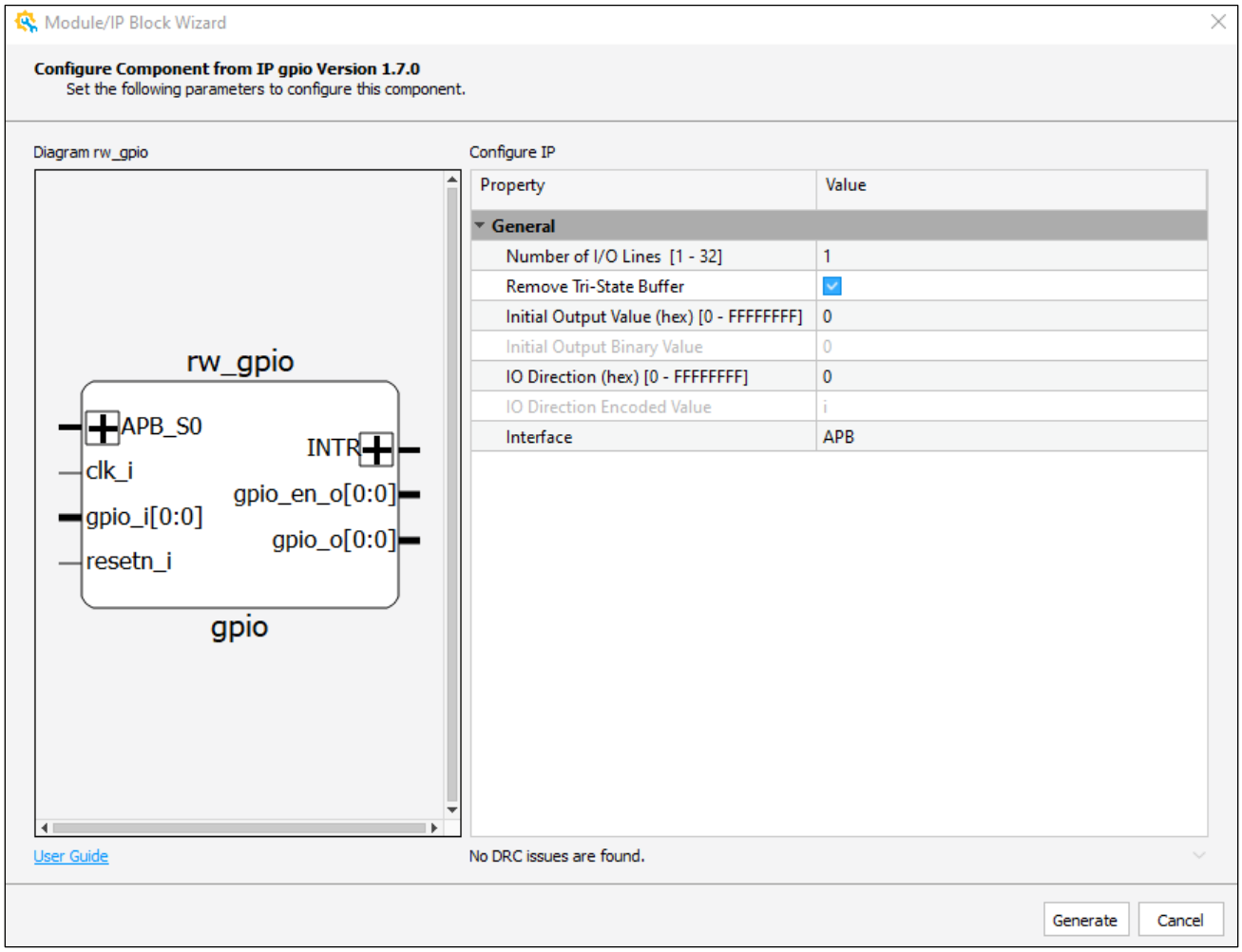

Figure 3.10. Remote Wakeup GPIO

3.1.7. Hardware Version GPIO

The hardware version GPIO tracks the version of the hardware design and the RISC-V firmware reads to know the current hardware version supported by the design. The GPIO input signals are connected to the following local parameters in the design top-level RTL file (*u3v_u23toaxi_ahbl_revBbd_des.sv*) as shown in the figure below.

```
92    // HW Design Major version
93    localparam    FPGA_DES_MAJOR_VERSION = 8'h01;
94
95    // HW Design Minor version
96    localparam    FPGA_DES_MINOR_VERSION = 8'h00;
97
98    // HW_VERSION indicator
99    localparam    FPGA_VERSION_TRACKER = { FPGA_DES_MAJOR_VERSION , FPGA_DES_MINOR_VERSION};
100
```

Figure 3.11. Hardware Version Definition in RTL

The figure below shows the GPIO IP parameters configured in the Propel Builder software project. The GPIO IP is set to has 16-bit input signal that is connected to local parameters as mentioned above.

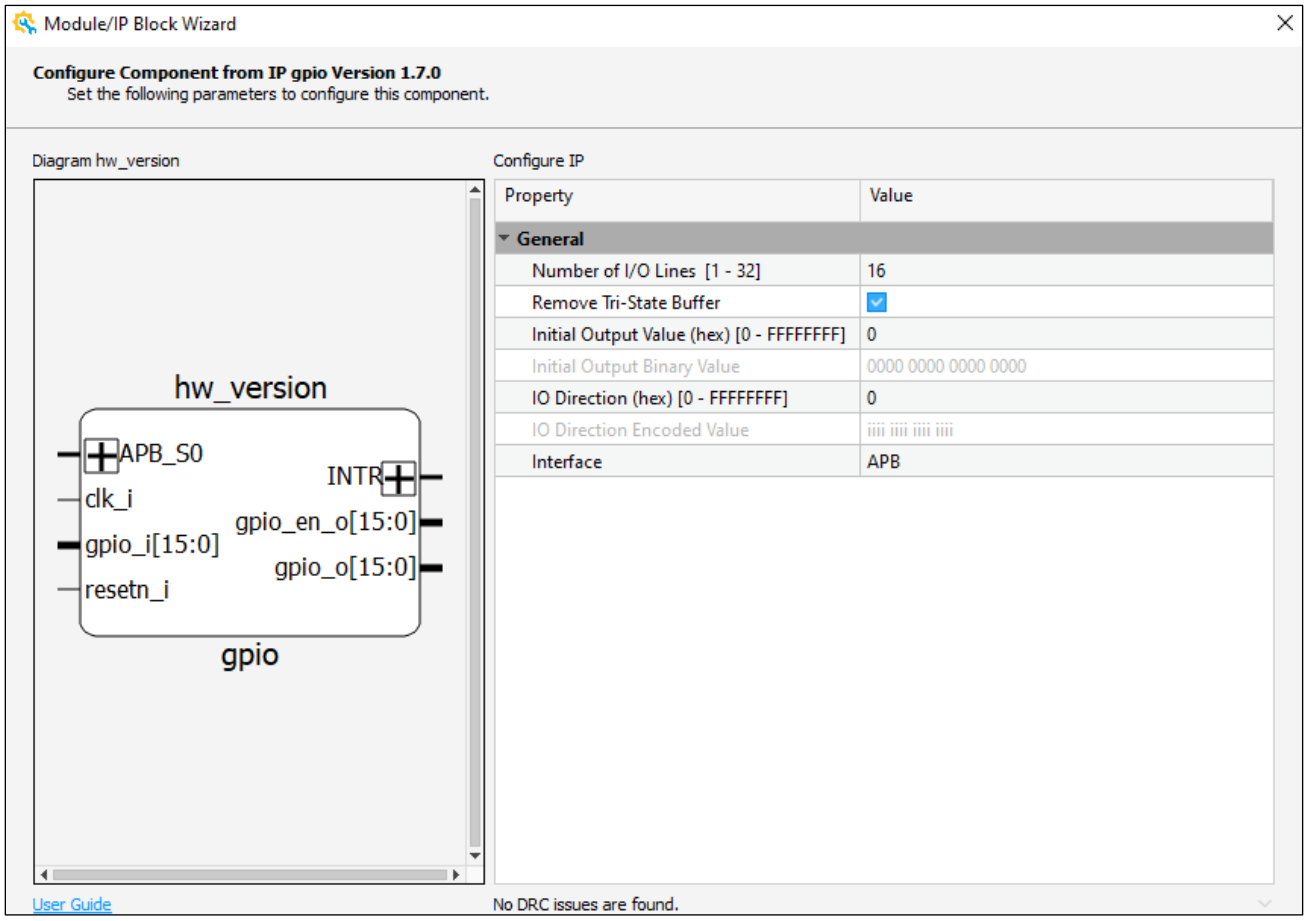

Figure 3.12. Hardware Version GPIO IP

3.1.8. USB2 PHY Reset GPIO

While switching between USB2 and USB3 modes, the current state of the USB2 PHY must be reset. This reset operation is controlled by the RISC-V processor firmware through a GPIO IP where its signal is connected to the USB2 PHY reset.

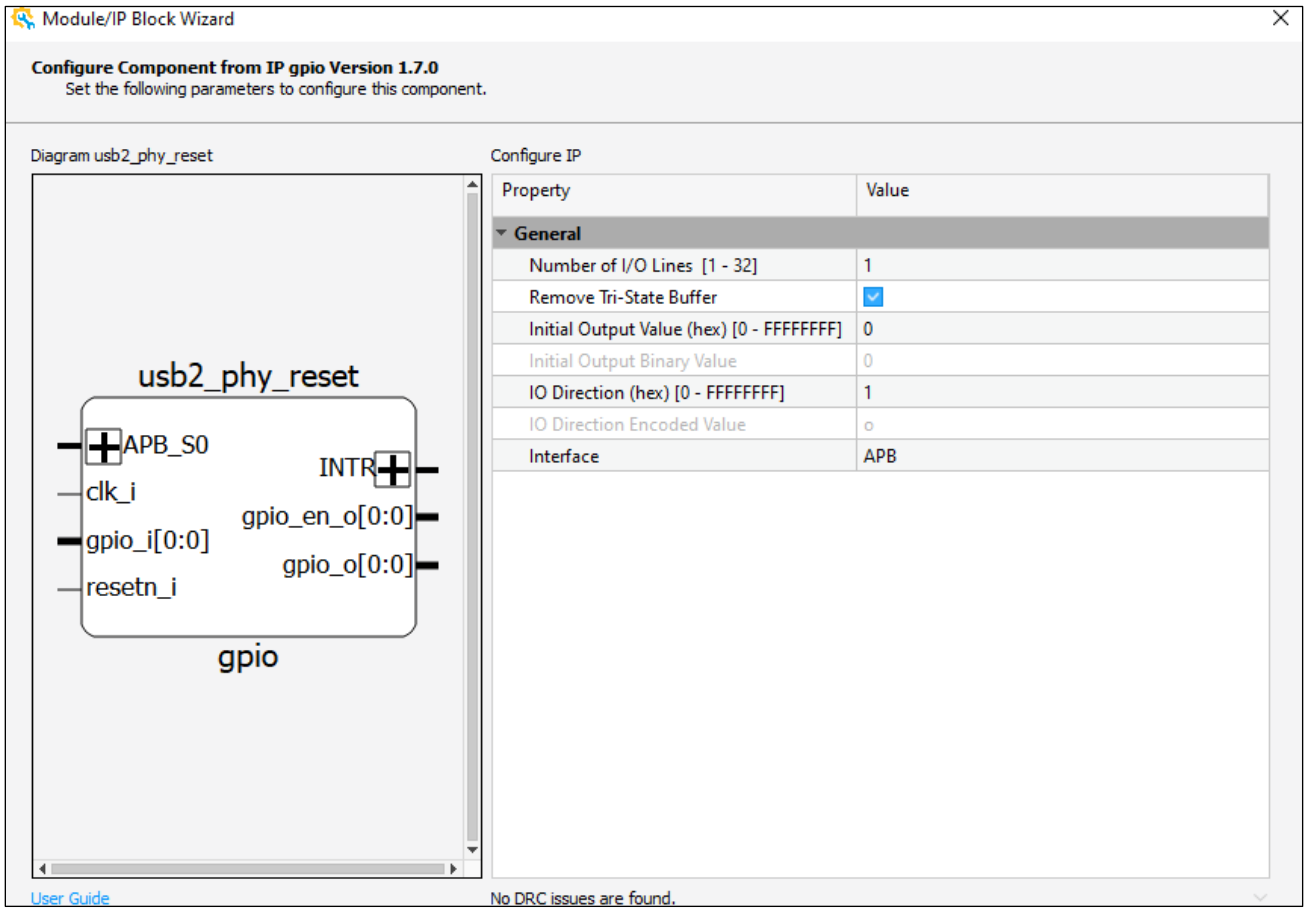

Figure 3.13. USB2 PHY Reset GPIO

3.1.9. Octal SPI Controller

The Octal SPI Controller is used to communicate with high-speed flash memory available on the NX33U board.

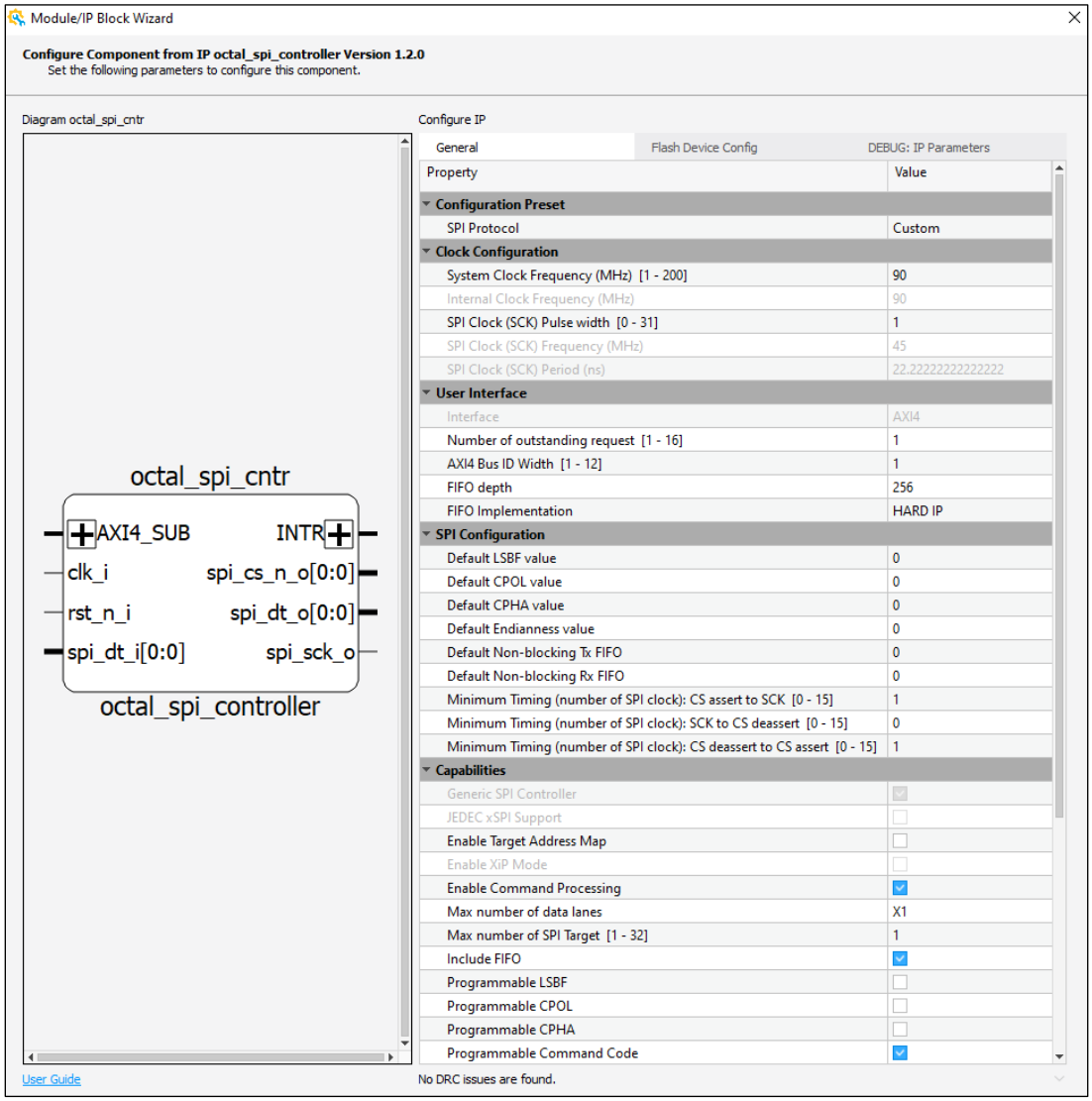

Figure 3.14. Octal SPI Controller

3.1.10. I2C Controller

The I2C Controller IP core is used to communicate with the I2C devices. I2C write and I2C read operations can be performed to the I2C device that is connected to the CrossLinkU-NX FPGA on the board. In this design the RISC-V processor configures the external camera sensor register via the I2C interface using the I2C Controller IP. The I2C Controller IP core configuration is shown in the figure below.

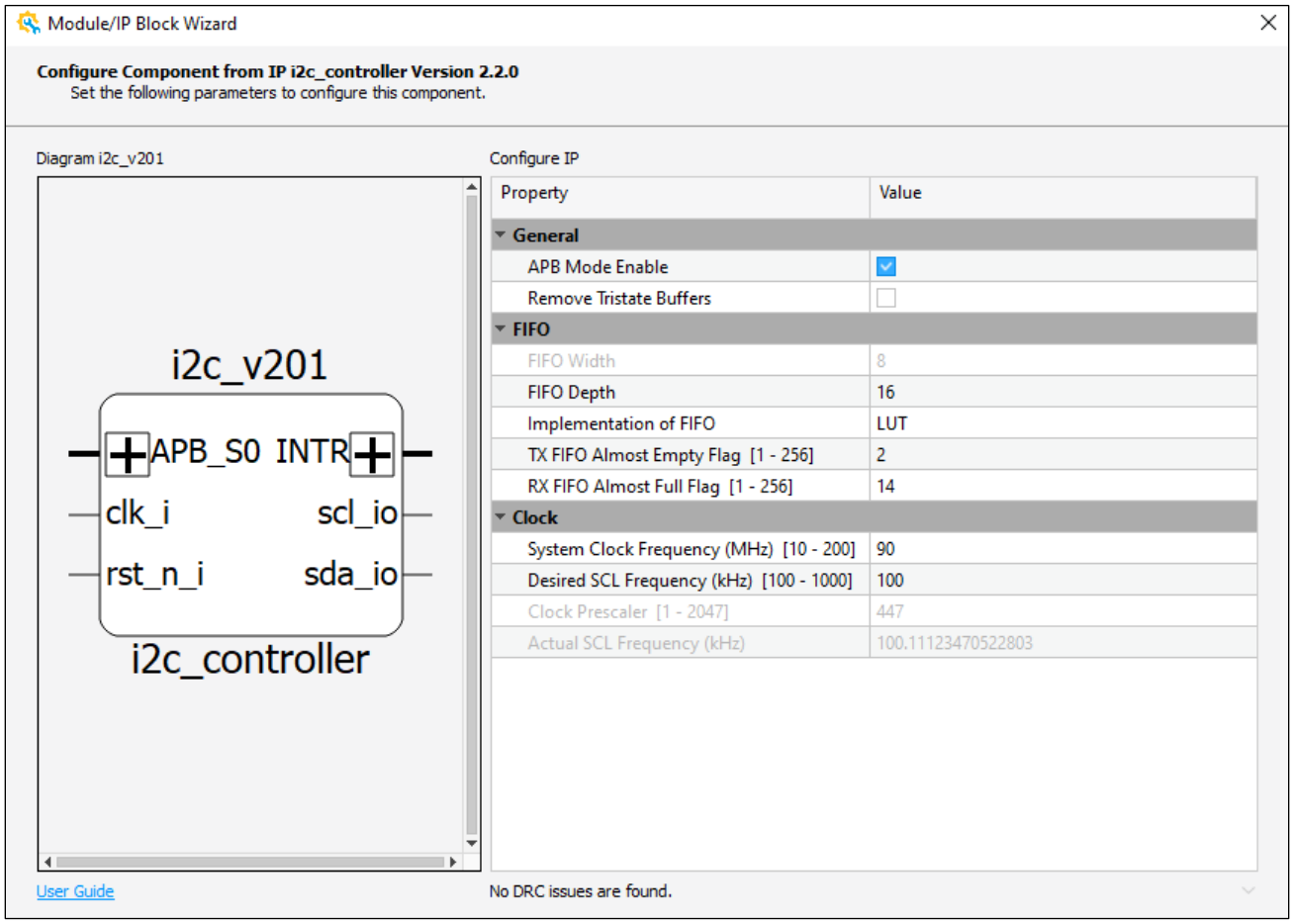

Figure 3.15. Lattice I2C Master Configuration

3.1.11. Tightly Coupled Memory

The Tightly Coupled Memory (TCM) IP core is the main memory used by the RISC-V processor. The RISC-V processor reads the instructions stored in this memory via the local bus memory interface and execute the instructions accordingly. The TCM memory also holds the data required for RISC-V processor execution. This IP core has two local bus interfaces by which the processor can communicate with. The dedicated connection between processor and TCM provides faster reading of instructions and data, and improves the overall system performance. A total of 192 Kb of TCM memory space is available for RISC-V processor usage. The TCM IP core configuration is shown in Figure 3.16, Figure 3.17, and Figure 3.18.

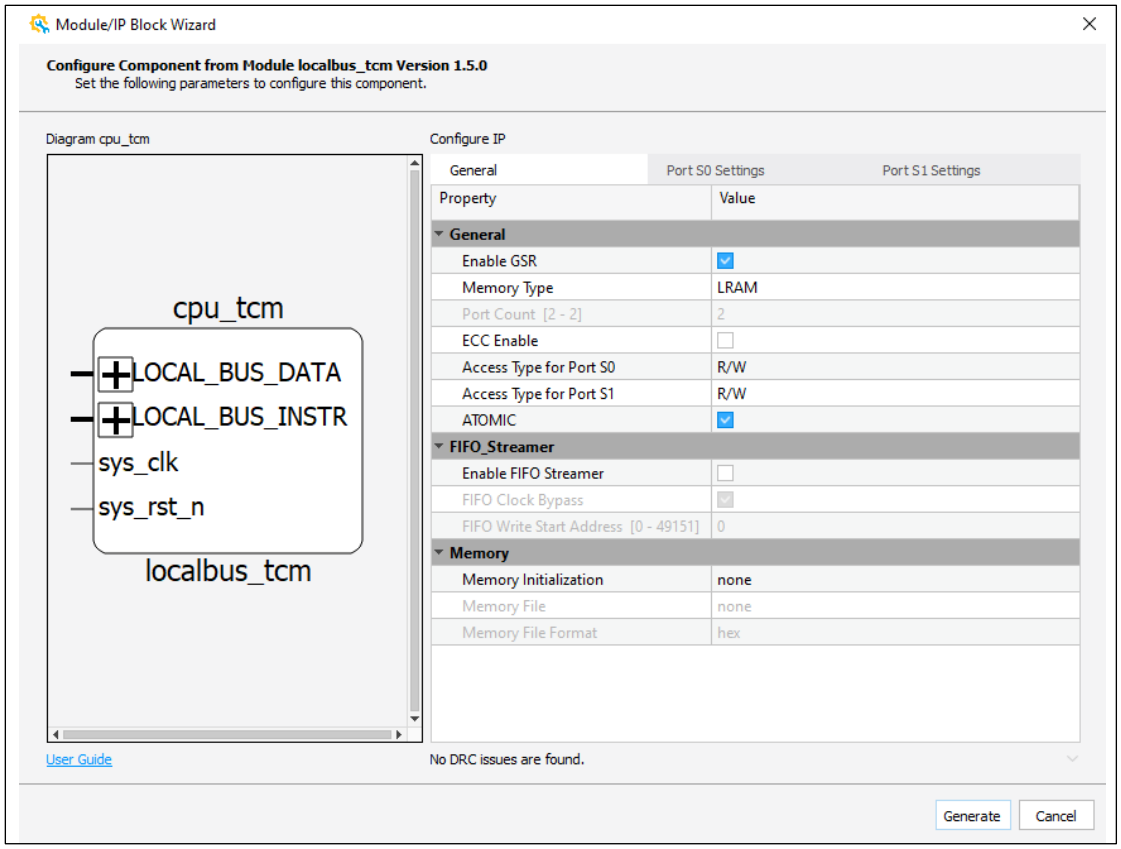

Figure 3.16. TCM General Configuration

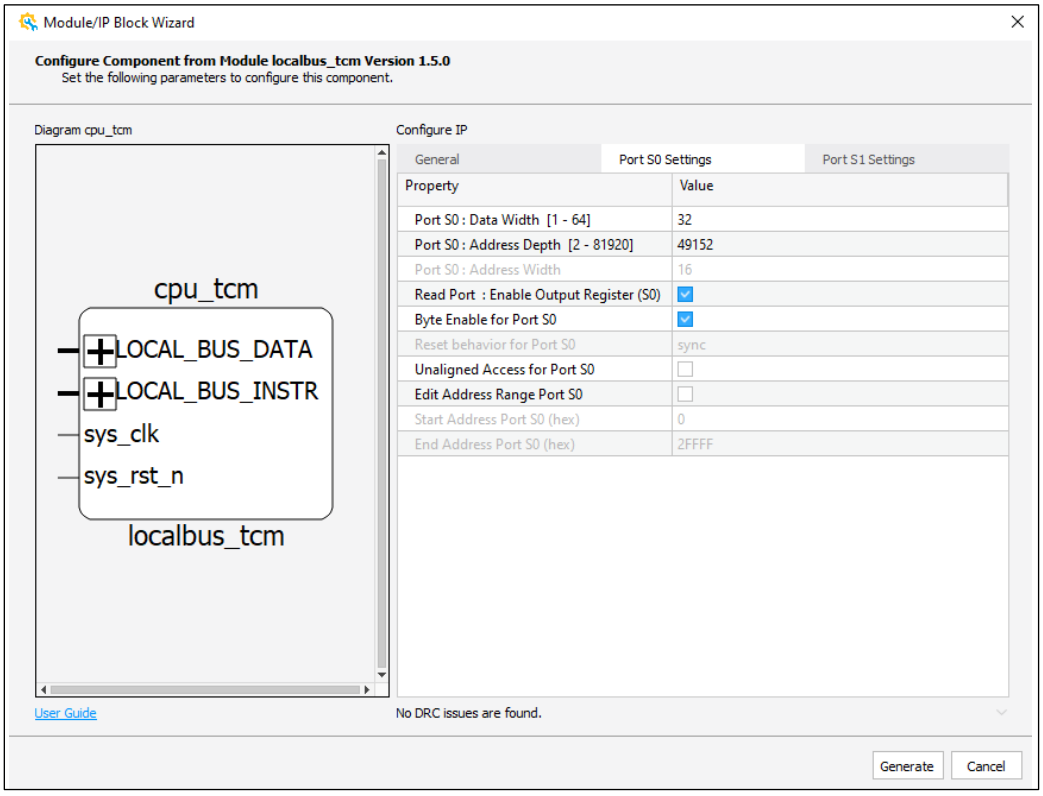

Figure 3.17. TCM Port S0 Configuration

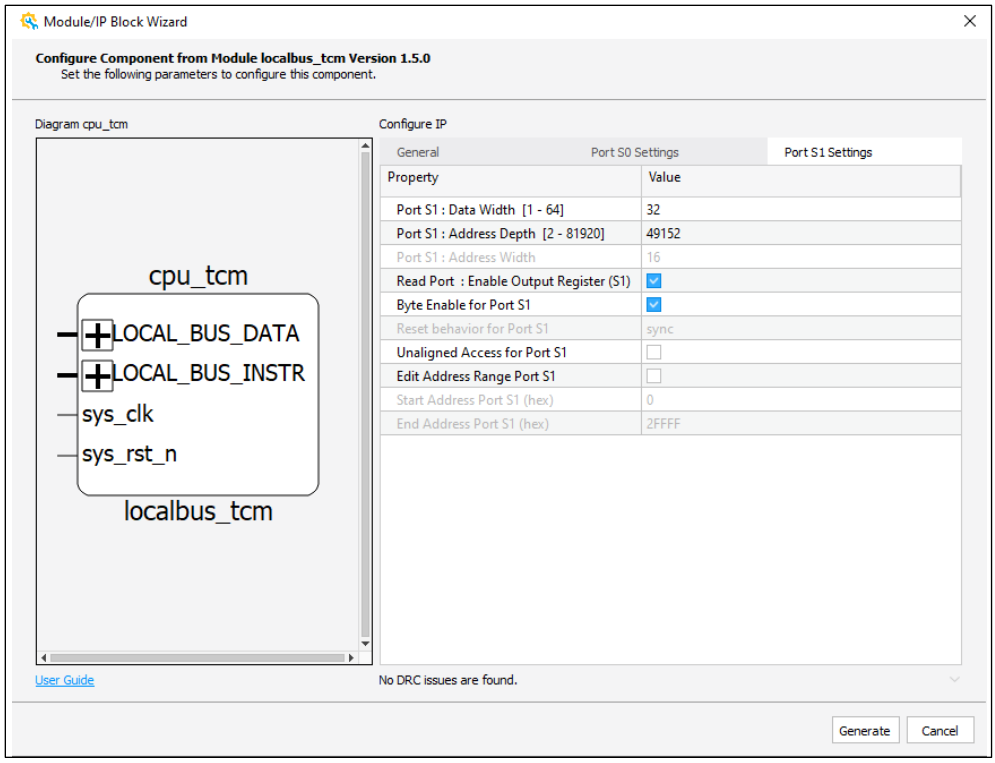

Figure 3.18. TCM Port S1 Configuration

3.1.12. AXI4 to AHB-Lite Bridge

The RISC-V RX AXI4-based data interface is used to access and control various peripherals in the system including UART, I2C controller, USB register, and others. The AHB-Lite interconnect is used in this system to arbitrate the access from the RISC-V processor to multiple peripherals subordinate interfaces. Therefore, you must convert the processor AXI4 interface to AHB-Lite using the AXI4 to AHB-Lite Bridge IP. This IP has configurations shown in the figure below.

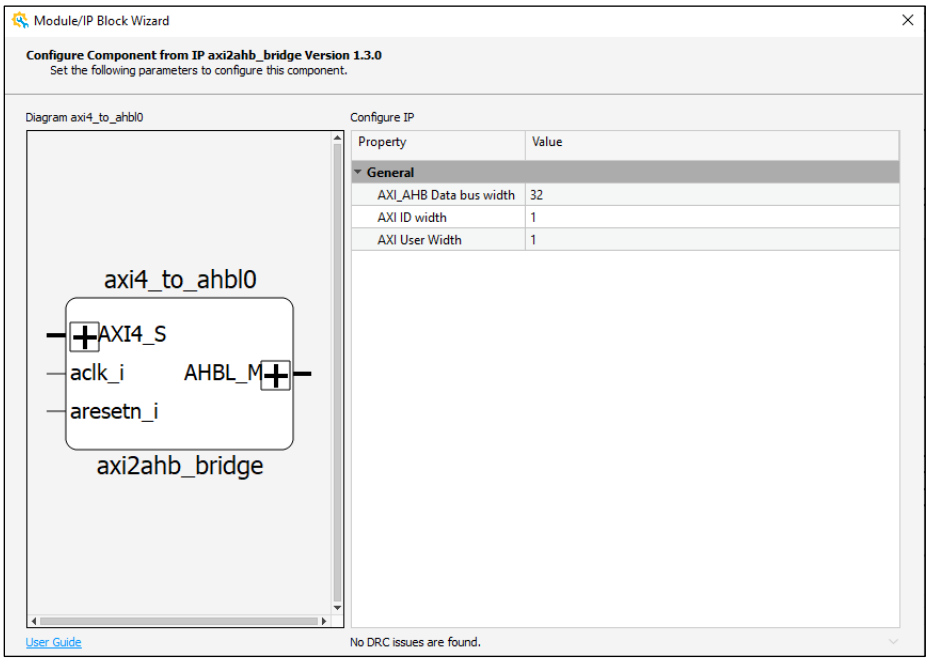

Figure 3.19. AXI4 to AHB-Lite Bridge Configuration

3.1.13. AHBL Interconnect

This IP provides the communication interface between RISC-V RX processor and peripherals (UART, I2C Controller, IEBM, RISCV to TDP RAM bridge and USB AHB-Lite to LMMI bridge) which supports AHB-Lite interface. Figure 3.20, Figure 3.21, Figure 3.22, and Figure 3.23 show the AHB-Lite interconnect configuration.

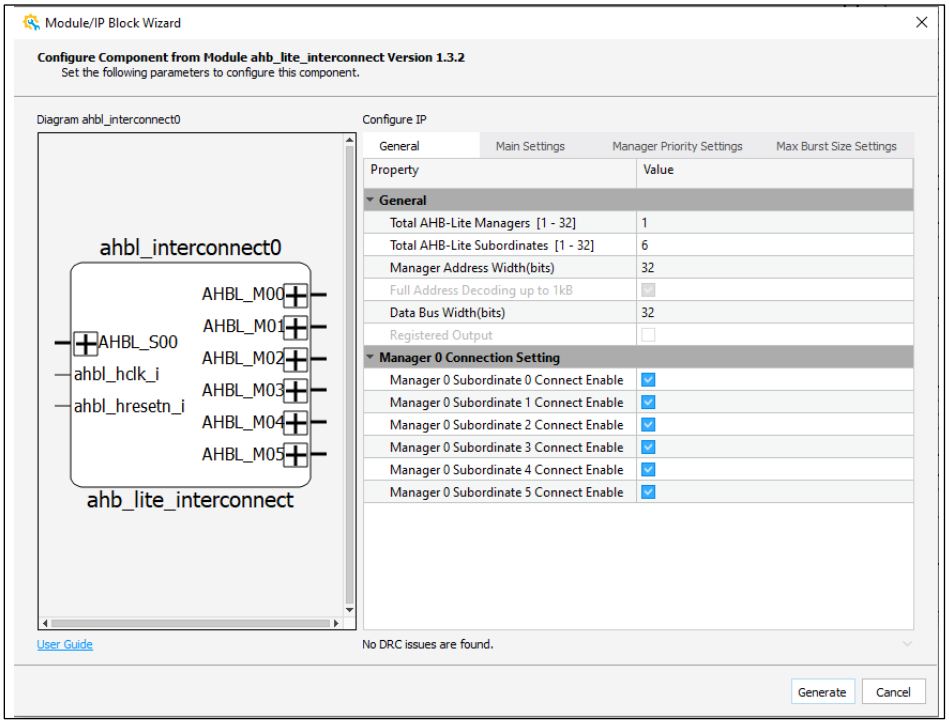

Figure 3.20. AHB-Lite Interconnect General Configuration

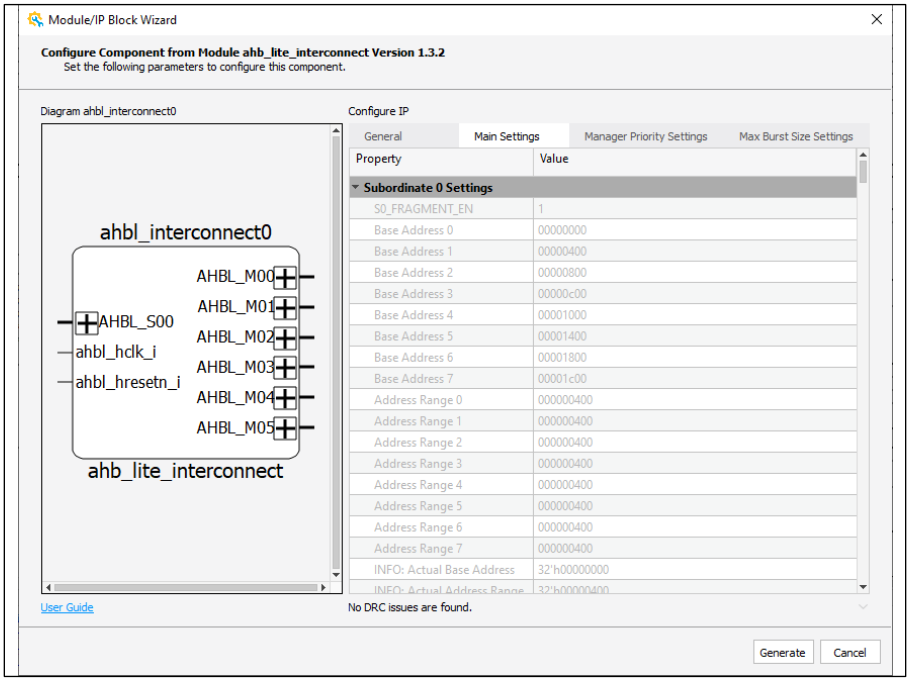

Figure 3.21. AHB-Lite Interconnect Main Configuration

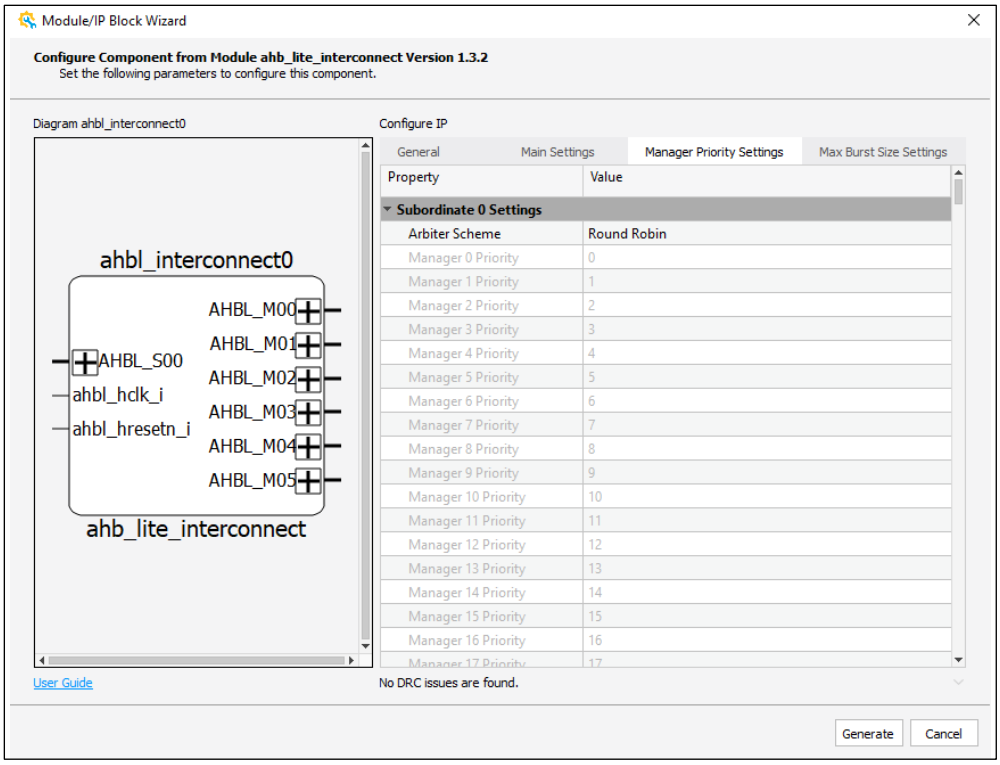

Figure 3.22. AHB-Lite Interconnect Manager Priority Settings

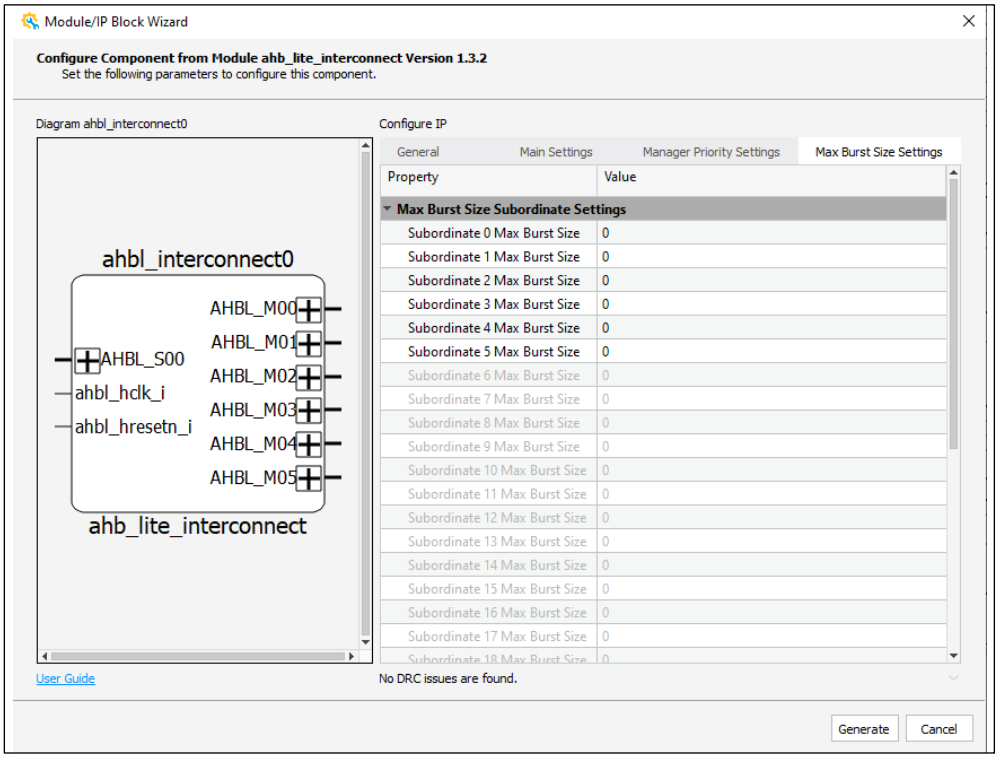

Figure 3.23. AHB-Lite Interconnect Max Burst Settings

3.1.14. AHB-Lite to APB Convertor

The AHB-Lite to APB bridge provides an interface between the AHB-Lite manager and APB subordinate. Read-and-write transfers on the AHB-Lite are converted into equivalent transfers on the APB. This conversion is required to allow the RISC-V RX processor access to APB-based peripherals such as UART and I2C controller. The AHB-Lite to APB Convertor IP core configuration is shown in the figure below.

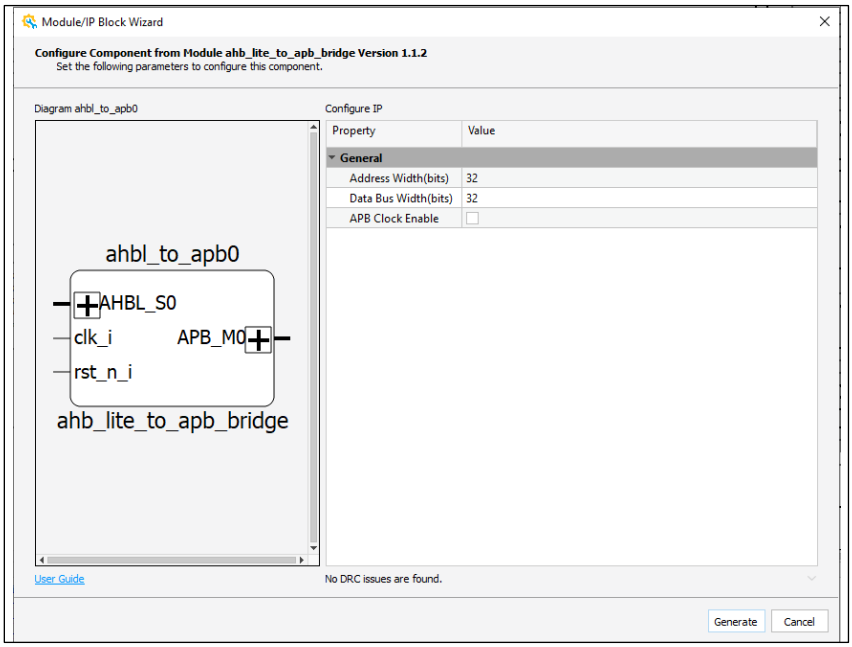

Figure 3.24. AHB-Lite to APB Configuration

3.1.15. APB Interconnect

The APB Interconnect IP arbitrate RISC-V processor manager access to various subordinates in the system such as the UART and I2C Controller that support the APB interface. The APB interconnect configuration is shown in Figure 3.25 and Figure 3.26.

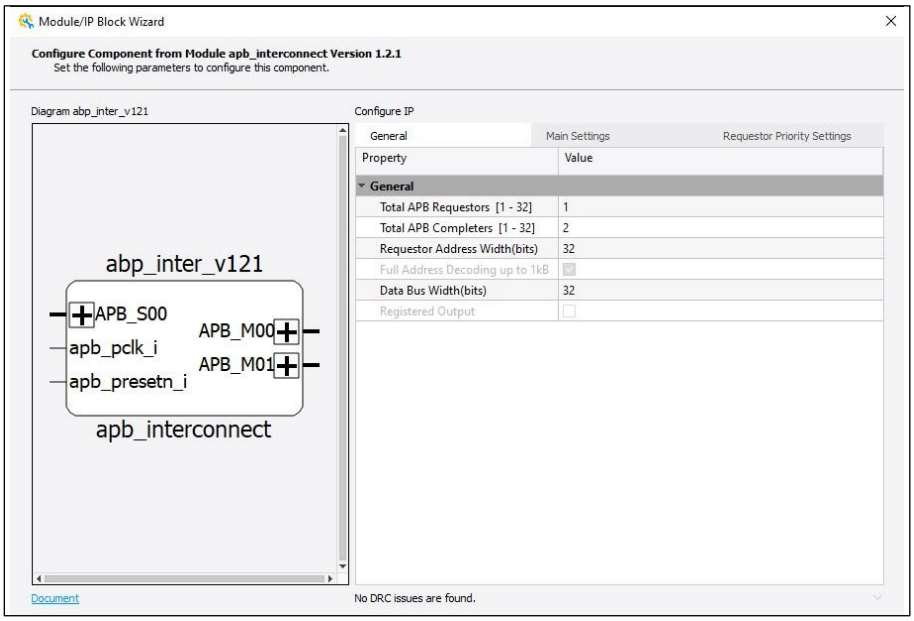

Figure 3.25. APB Interconnect General Configuration

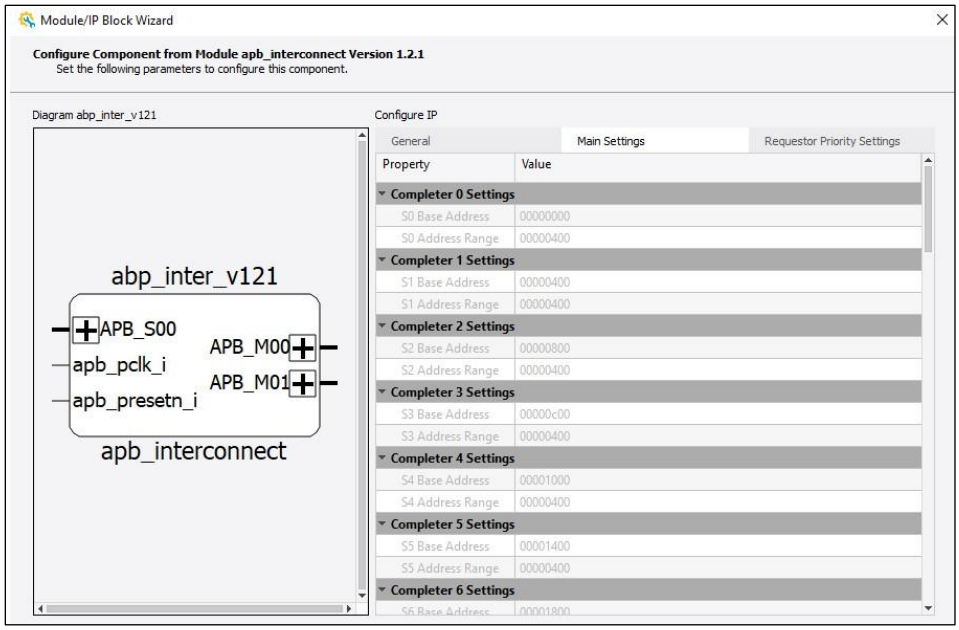

Figure 3.26. APB Interconnect Main Settings Configuration

3.1.16. CSI-2/DSI D-PHY Receiver

The CSI-2/DSI D-PHY Receiver IP core converts the CSI-2 data from the Raspberry Pi MIPI camera to 8-bit data for the subsequent processing in the FPGA. The MIPI camera sensor data enters FPGA in serial form and this IP converts those serial data into parallel by following MIPI CSI-2 protocol. The IP configuration is shown in Figure 3.27, Figure 3.28, and Figure 3.29.

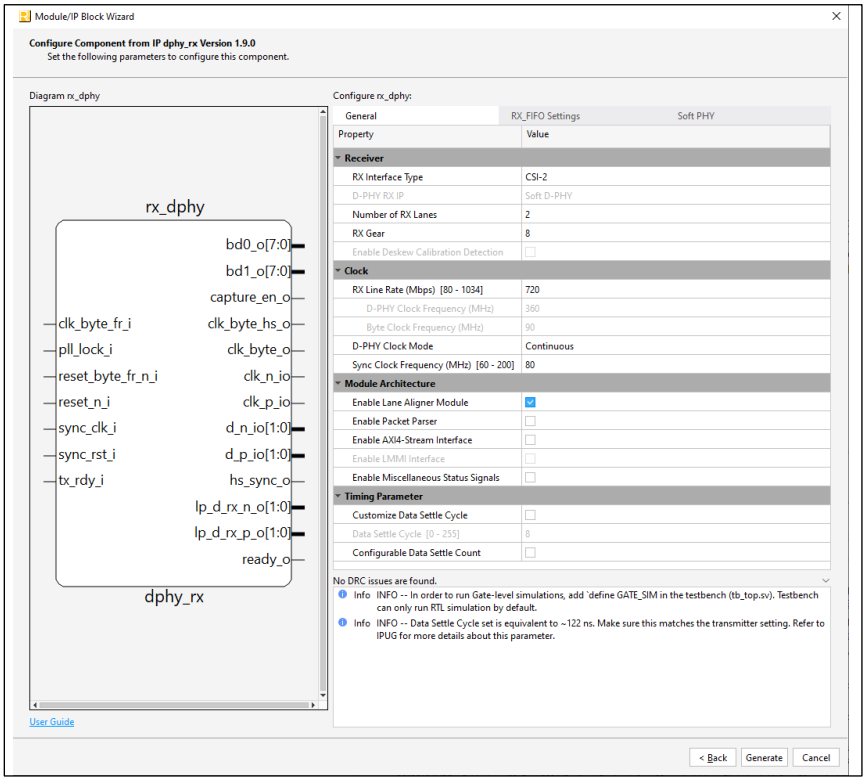

Figure 3.27. RX DPHY General Configuration

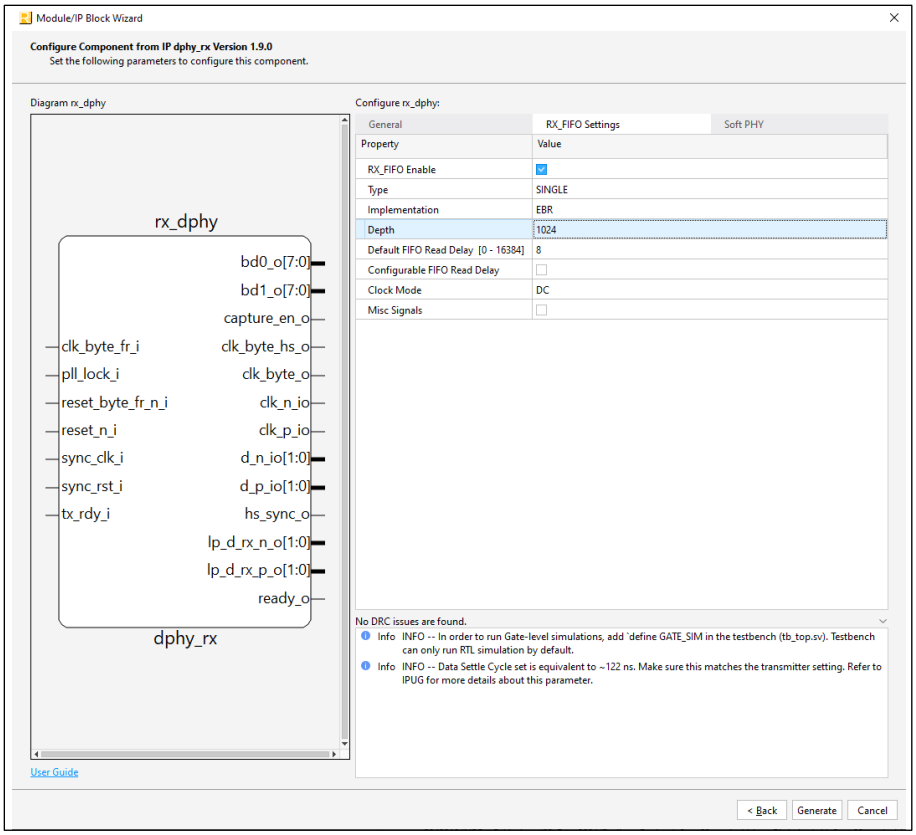

Figure 3.28. RX DPHY RX FIFO Configuration

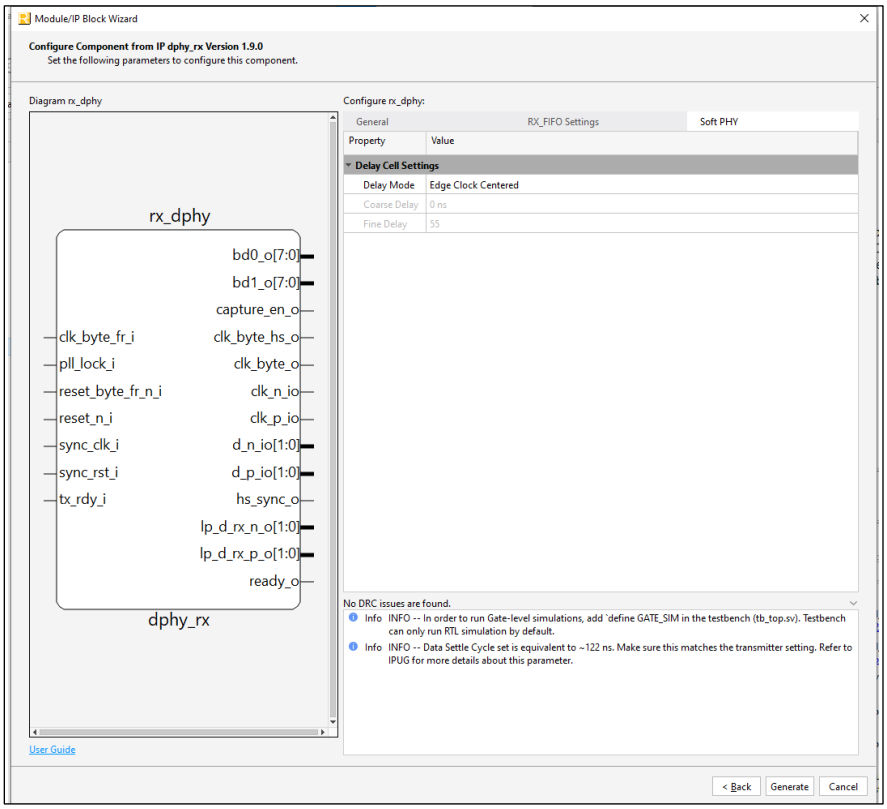

Figure 3.29. RX DPHY Soft PHY Configuration

3.1.17. MIPI Packet Decoder

This module decodes the data packet from the camera sensor that is received via the RX D-PHY interface. These packets include data, control, synchronization, and configuration information. After decoding the packets, the RAW10 data is passed to the Byte-to-Pixel Converter IP core. This module is a custom block that is not part of Lattice IP in the IP Catalog Server.

3.1.18. Byte-to-Pixel Converter IP Core

The Byte-to-Pixel Converter IP core converts the incoming RAW10 data (from MIPI Packet Decoder) into pixel format. It receives raw bytes from a parser and rearranges them to output pixels. This module outputs 2-Pixels (each pixel of 10-bit) in one clock with a valid signal and its configuration is shown in the figure below.

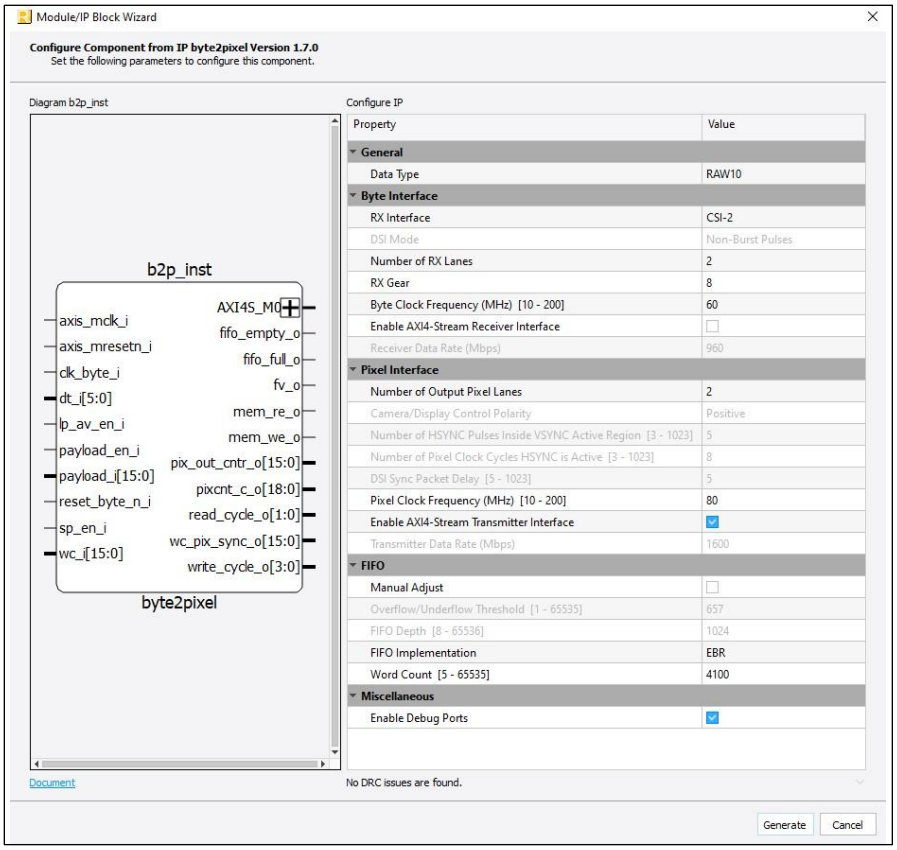

Figure 3.30. Byte-to-Pixel Configuration

3.1.19. Byte-to-Pixel (B2P) to AXI Stream Conversion

This module converts the pixel data according to the AXI Stream protocol to meet Lattice Debayer AXI Stream input requirement. This module act as a bridge between the Lattice Byte-to-Pixel Converter IP core and Lattice Debayer IP core. This module is a custom block that is not part of Lattice IP in the IP Catalog Server.

3.1.20. Debayer IP Core

The camera sensor produces pixel data in Bayer format. The Debayer IP core extracts the red, green, and blue (RGB) component from the Bayer data. The Debayer converts RAW10 image data into an RGB component and its configuration is shown in the figure below.

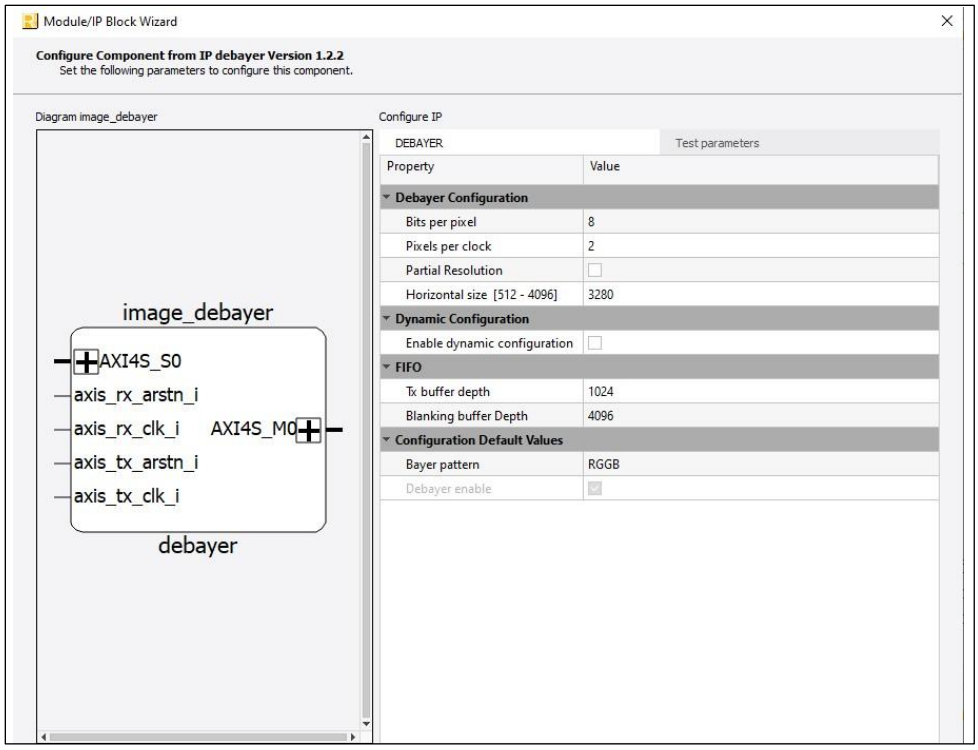

Figure 3.31. Debayer Configuration

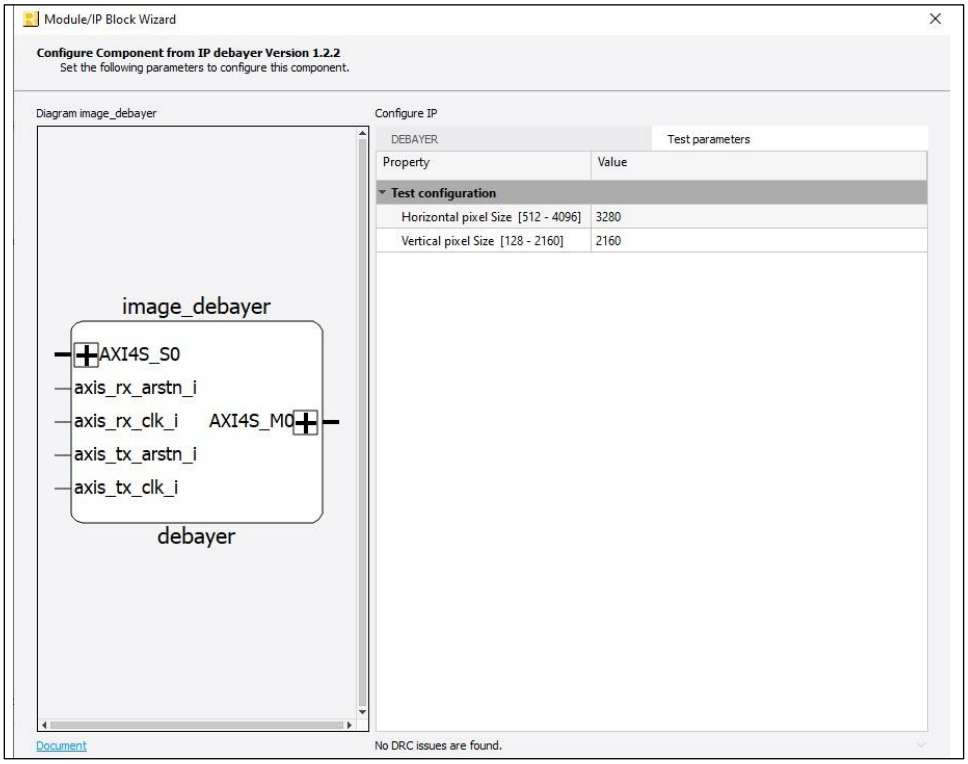

Figure 3.32. Debayer Configuration – Test Configuration

3.1.21. AXI Stream to Parallel Conversion

This module converts the AXI Stream protocol signal to parallel data. In this design, this module acts as a bridge between the Lattice Debayer IP core and Lattice CSC IP core. This module is a custom block that is not part of Lattice IP in the IP Catalog Server.

3.1.22. Pixel Data Conversion

Multiple conversion modules are used (as shown in Figure 3.1) after the Byte-to-Pixel Converter IP to convert the RAW10 pixel data into different formats such as Bayer8, Bayer10 Packed, and Bayer 10. This module is a custom block that is not part of Lattice IP in the IP Catalog Server.

3.1.23. Color Space Converter

This IP block converts RGB data into YUV format and its configuration is shown in Figure 3.33 and Figure 3.34

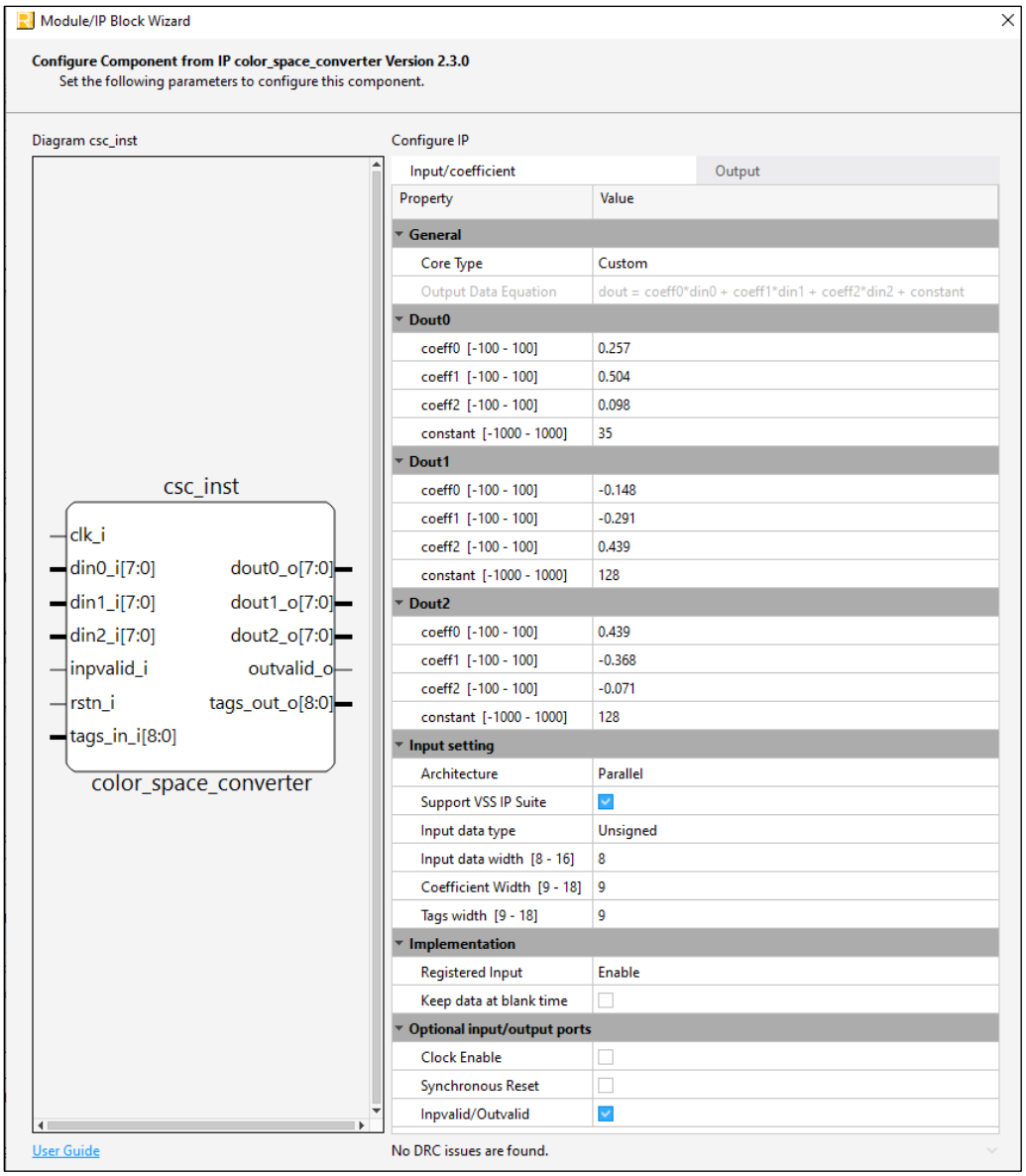

Figure 3.33. Color Space Converter Configuration

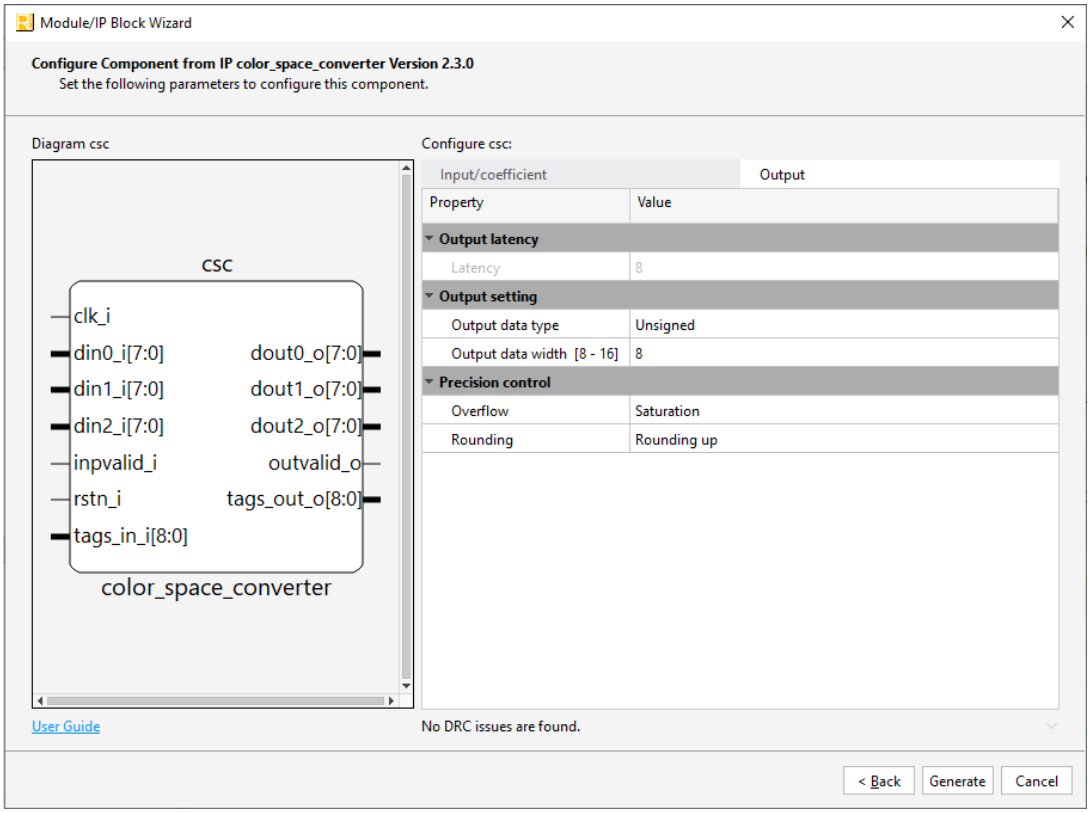

Figure 3.34. Color Space Converter Configuration – Output and Precision Control

3.1.24. YUV422 to UVC Bridge

This module features an internal DC-FIFO called YUV422 FIFO that is designed to store incoming YUV422 data along with associated data indicator codes. The FIFO Write interface works on Pixel Clock and the Read interface works on the IEBM clock. The YUV422 FIFO plays a critical role in managing and buffering video data, ensuring smooth data flow and processing. This module is a custom block that is not part of Lattice IP in the IP Catalog Server.

3.1.25. YUV422 Test Pattern Generator

This block generates the YUY422 solid color test pattern. It allows generating test pattern to test the video to USB path in this design. The video data source selection can be made in the Host PC via the Pleora eBUS Player software, to either selecting the camera sensor (for YUV422 format) or this pattern generator video data. A multiplexer is implemented in the design to facilitate this selection.

3.1.26. IEBM IN FIFO Interface

When the IEBM block grants permission to write the data, the IN FIFO controller module initiates the process. It reads the data from the YUV422 FIFO of the YUV422 to UVC Bridge, where the video data is temporarily stored and write into the IEBM buffers.

3.1.27. In Endpoint Buffer Manager (IEBM)

The IEBM is responsible for managing the IN endpoint buffers for the USB23 Controller. It handles the efficient and orderly flow of video data from the camera sensor or a test pattern generator. The data is written into different buffers sequentially through the First-In-First-Out (FIFO) interface. Once the data is available in these buffers, the USB23 Controller reads it through the AXI interface. This entire process, from managing the FIFO writes to facilitating the AXI reads, is managed by this IEBM module. The IEBM IP configuration is shown in the figure below.

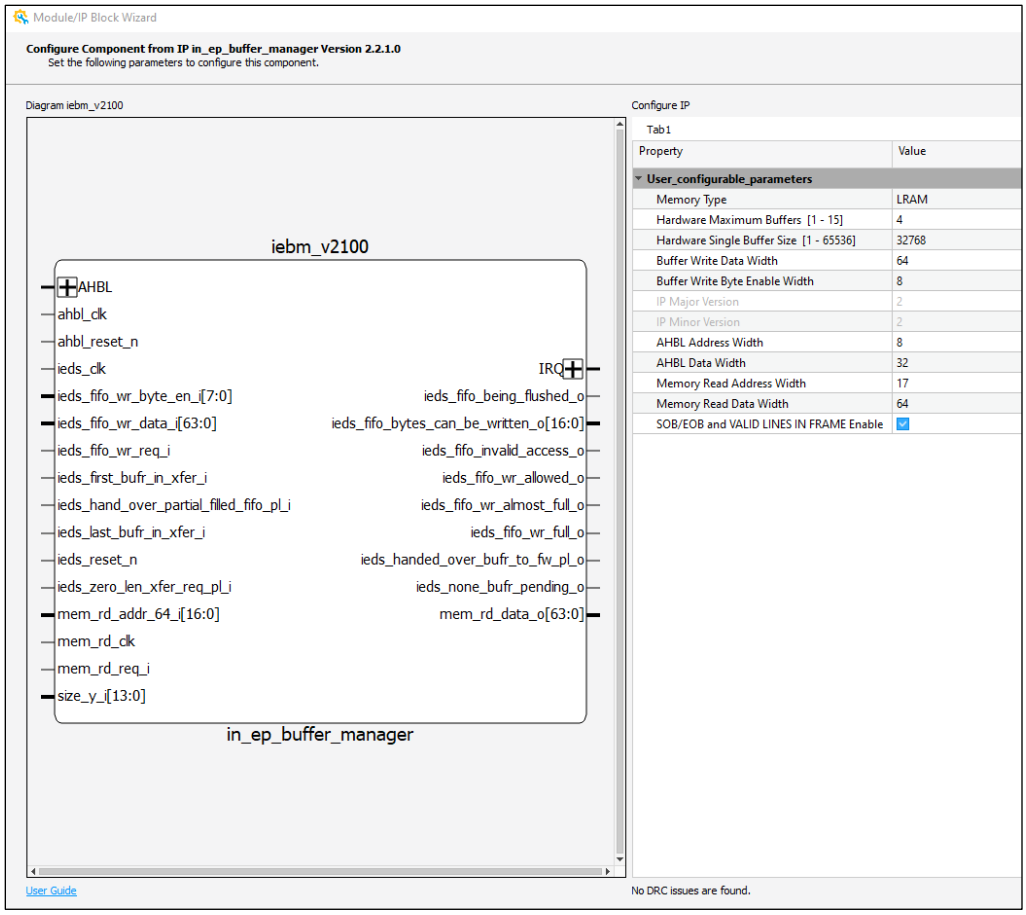

Figure 3.35. IEBM Configuration

The IEMB has an AHB-Lite based interface for accesses to its registers. This interface is connected to the RISC-V RX for control via firmware. The FIFO interface on the IEBM is configured as 64 bit in this design, and it is connected to the IN FIFO block described in the previous section. The FIFO interface is used to supply the video data into the IEBM before it is retrieved by the USB23 Controller to perform IN Endpoint transfer upon request by the Host PC. The USB23 Controller reads data out from the IEBM via the Memory Native interface.

The Appendix B section describes the details of the IEBM IP core.

3.1.28. TDP Memory and AHB-Lite to Memory Bridge

This block helps the RISC-V data manager to access True Dual Port memory that is dedicated for USB operations. This block converts AHB-Lite subordinate signals in to native memory manager interface signals. True Dual Port (TDP) memory is used as communication between the RISC-V firmware and the USB23 Controller for operations such as control data and USB events handling, creating USB3 Vision leader and trailer data and others. The TDP memory has 32-bit and 64-bit data interfaces for RISC-V and USB23 Controller accesses respectively, and these interfaces are not AMBA AHB-Lite based. Hence, the AHB-Lite to Memory Bridge IP is used to allow the RISC-V processor access to the TDP memory. The IP configuration is shown in the figure below.

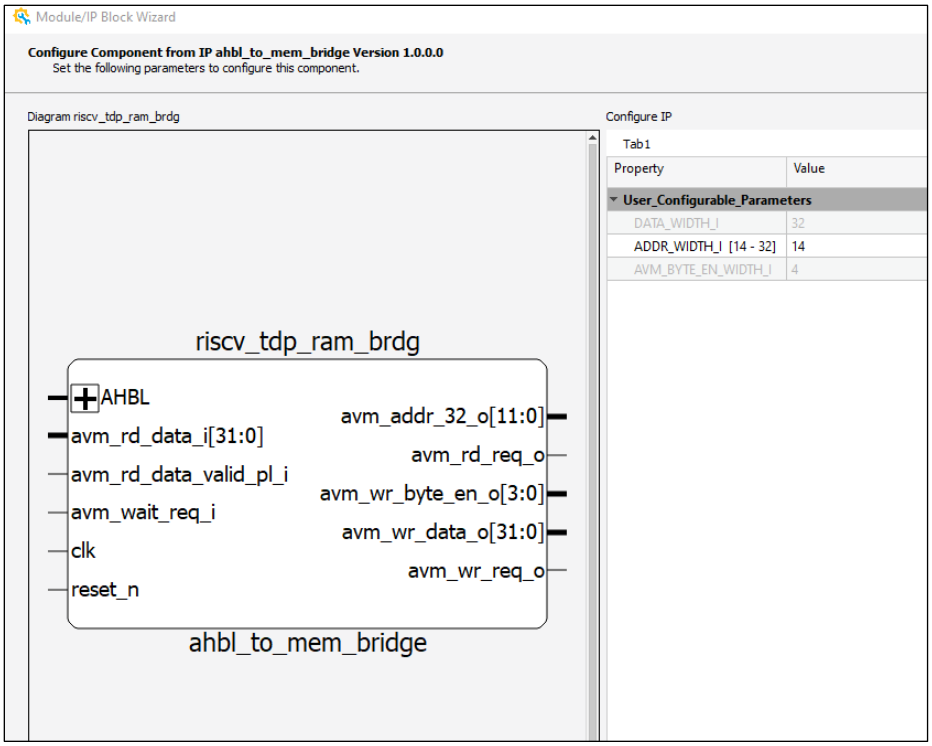

Figure 3.36. AHB-Lite to Memory Bridge Configuration

3.1.29. U3V Register Interface

This is an AHB-Lite register interface module for USB3 Vision registers set. The U3V Register interface stores user configuration during runtime such as the image format, image height and width, test pattern enable, and others. This module is a custom block that is not part of Lattice IP in the IP Catalog Server.

3.1.30. USB23 AXI Manager to Memory Interface Bridge

This section describes the architecture, functionality, and other technical information of the USB23 AXI manager to memory interface bridge. This bridge enables the conversion of AXI4 transfer signals into the native memory interface, which is crucial for performing read-and-write operations with the TDP memory. The USB23 Controller uses this path to access data provided by the RISC-V processor. Furthermore, the bridge offers a secondary memory interface that allows the USB23 controller to retrieve data from the IEBM data RAM.

3.1.30.1. USB23 AXI Manager to Memory Bridge Architecture

The architecture of the USB23 AXI manager to memory bridge is shown in the figure below. The bridge can be configured to have native memory read-only and write-only interfaces. The read-only interface is used to read data from buffer that corresponds to a USB IN Endpoint, while the write-only interface writes data into a buffer that corresponds to a USB OUT Endpoint. In this reference design, only one IN Endpoint is used for streaming the video data, therefore it is configured to have only the read-only interface.

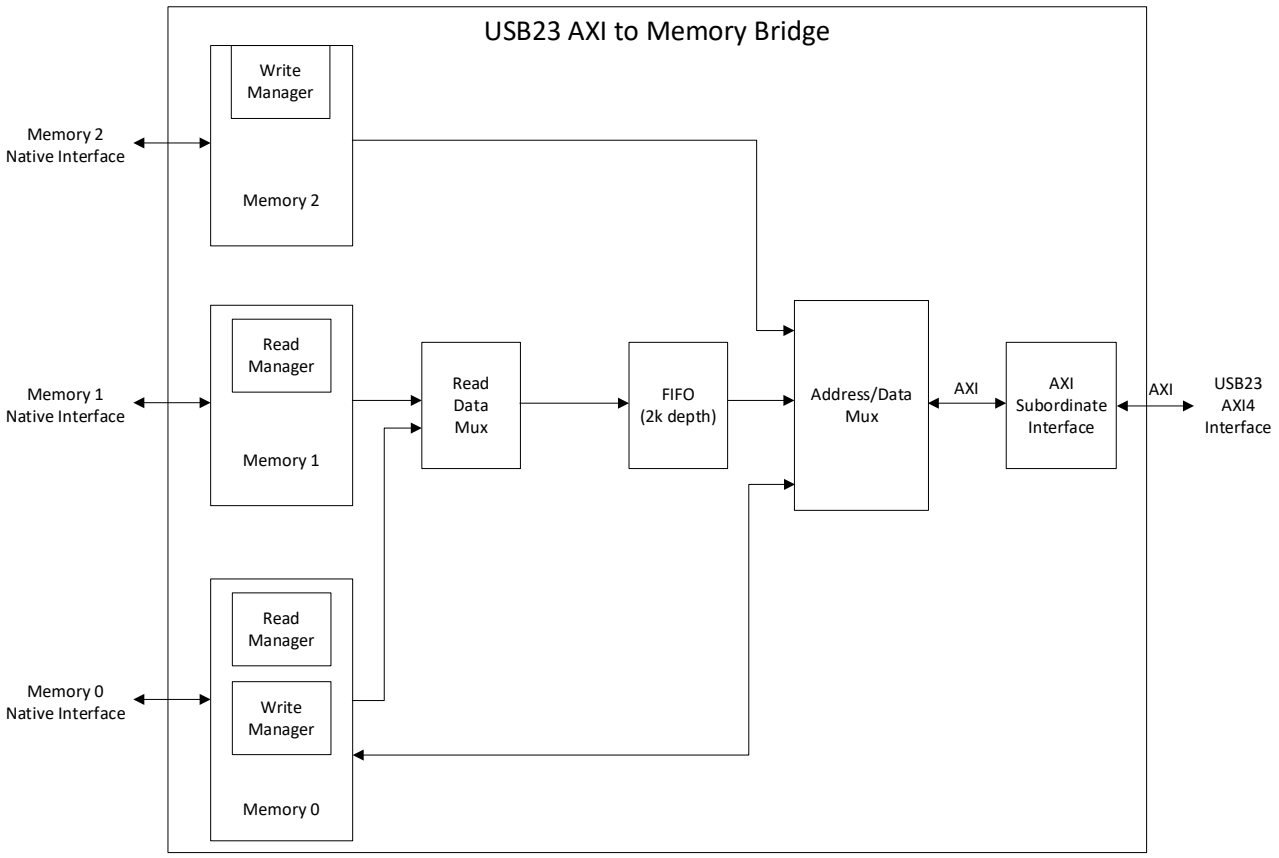

Figure 3.37. USB23 AXI Bridge to Memory Bridge Architecture

3.1.30.2. Block Description

The functional blocks are described in the following sections.

AXI Subordinate Interface

The AXI subordinate interface is connected to the AXI4 bus of the USB23 Controller.

Address/Data Mux

This block decodes the write address or read address of the specific memory block.

FIFO

This FIFO acts as the intermediate storage when the AXI manager is not ready to accept data. The FIFO size is 2048 KB, 256 in depth, and 64-bits data width.

Read Data Mux

This block decodes the memory block that requires a read operation based on the read address.

Memory 0

This block contains both a write manager and a read manager, enabling it to perform both read-and-write operations.

Memory 1

This block contains a read manager that is designed to handle read operations.

Memory 2

This block contains a read manager that is designed to handle read operations.

3.1.30.3. IP Core Port and Parameters

This section provides details of the USB23 AXI manager to Memory Bridge IP ports and interface.

Clock and Global Reset

The table below describes ports for clock and global reset.

Table 3.1. Clock and Reset Ports

Signal Name	Width	Direction	Description
clk	1	Input	Global clock.
reset_n	1	Input	Global reset, active low.

Native Memory Interface

The table below describes ports for the native memory interface.

Table 3.2. Native Memory Interface Ports

Signal Name	Width	Direction	Description
wr_req_o	1	Input	Write request.
wr_addr_8_o	18	Output	Write address.
wr_data_o	64	Output	Write data.
wr_byte_en_o	8	Output	Write byte enable.
rd_data_i	64	Input	Read data.
rd_req_o	1	Output	Read request.
rd_addr_8_o	18	Output	Read address.

AXI4 Interface

The table below describes ports of the native AXI4 interface.

Table 3.3. AXI4 Interface Ports

Signal Name	Width	Direction	Description
axis_mi_awid_i	8	Input	Write Transaction ID: A unique identifier for the write transaction.
axis_mi_awaddr_i	32	Input	Write Address: The address of the write operation. It is used by the subordinate to determine where to write data.
axis_mi_awlen_i	8	Input	Write Burst Length: The number of transfers in a burst that is measured in bytes.
axis_mi_awsize_i	3	Input	Write Burst Size: The size of each transfer in the burst, such as 8 bit, 16 bit, and 32 bit.
axis_mi_awburst_i	2	Input	Write Burst Type: The type of burst transfer, such as fixed, incrementing, or wrapping.
axis_mi_awvalid_i	1	Input	Write Address Valid: Indicates that a valid write address is available from the manager.
axis_mi_awready_o	1	Output	Write Address Ready: Indicates that the subordinate is ready to accept the write address.
axis_mi_wdata_i	64	Input	Write Data: The data being written to the subordinate. The subordinate stores this data at the address specified in AWADDR.
axis_mi_wstrb_i	8	Input	Write Strobes: A set of byte-enable signals that indicate which bytes of the data are valid for writing.
axis_mi_wlast_i	1	Input	Write Last: This signal indicates the last transfer in a write burst.
axis_mi_wvalid_i	1	Input	Write Data Valid: Indicates that valid data is available on the WDATA bus for the subordinate.
axis_mi_wready_o	1	Output	Write Data Ready: Indicates that the subordinate is ready to accept the write data.

Signal Name	Width	Direction	Description
axis_mi_bid_o	8	Output	Write Transaction ID: A unique identifier for the write transaction that is used to match the response with the request.
axis_mi_bresp_o	2	Output	Write Response: The status of the write operation. It can indicate a successful operation represented by OKAY, or an error, such as SLVERR.
axis_mi_bvalid_o	1	Output	Write Response Valid: Indicates that the subordinate has a valid response for the write transaction.
axis_mi_bready_i	1	Input	Write Response Ready: Indicates that the manager is ready to receive the write response from the subordinate.
axis_mi_arid_i	8	Input	Read address ID: This signal is the identification tag for the read address group of signals.
axis_mi_araddr_i	32	Input	Read address: The read address gives the address of the first transfer in a read burst transaction.
axis_mi_arlen_i	8	Input	Burst length: This signal indicates the exact number of transfers in a burst.
axis_mi_arsize_i	3	Input	Burst size: This signal indicates the size of each transfer in the burst.
axis_mi_arburst_i	2	Input	Burst type: The burst type and the size information determine how the address for each transfer within the burst is calculated.
axis_mi_arvalid_i	1	Input	Read address valid: This signal indicates that the channel is signaling valid read address and control information.
axis_mi_arready_o	1	Output	Read address ready: This signal indicates that the subordinate is ready to accept an address and associated control signals.
axis_mi_rready_i	1	Input	Read Data Ready: Indicates that the manager is ready to accept read data from the subordinate.
axis_mi_rid_o	8	Output	Read Transaction ID: A unique identifier for the read transaction that is used to match the response with the request.
axis_mi_rdata_o	64	Output	Read Data: The data being read from the subordinate. The manager reads this data at the address specified in ARADDR.
axis_mi_rresp_o	2	Output	Read Response: The status of the read operation. It can indicate a successful operation, represented by OKAY, or an error, such as SLVERR.
axis_mi_rlast_o	1	Output	Read last: This signal indicates the last transfer in a read burst.
axis_mi_rvalid_o	1	Output	Read Data Valid: Indicates that valid read data is available for the manager to read.

3.1.31. Dual Boot Wrapper

This dual boot wrapper module consists of the following IP or modules that are responsible for the dual boot functionality, which has been added in the design:

- Internal oscillator—This IP is an oscillator module. It generates clock sources, sys_clk and lmmi_clk, to the LMMI host controller and CONFIG_LMMIA primitive.
- CONFIG_LMMIA—Lattice Memory Mapped Interface (LMMI) interfaces to the configuration block.
- LMMI Host Controller—This controller implements a state machine controller to send the necessary commands to the CONFIG_LMMIA block, and sends the boot address to the MULTIBOOT primitive to boot the desired alternate pattern stored in the internal or external SPI flash.
- MULTIBOOT Primitive—This primitive is a wrapper for the interface to perform the multi-boot functionality. It enables the booting to load the desired alternate pattern through sending the Refresh command to the CONFIG_LMMIA block.

For more information about the dual boot feature, refer to the Multi-Boot User Guide for Nexus Platform (FPGA-TN-02145).

3.2. Clocking Scheme

The CrossLinkU-NX USB3 Vision Reference Design use a PLL to generate the 90 MHz system clock, 80 MHz pixel clock for image signal processing and 60 MHz USBPHY clock. All these clocks are generated from the 60 MHz crystal oscillator on LIFCL-33U Evaluation Board. An overview of the design clocking scheme is shown in the figure below.

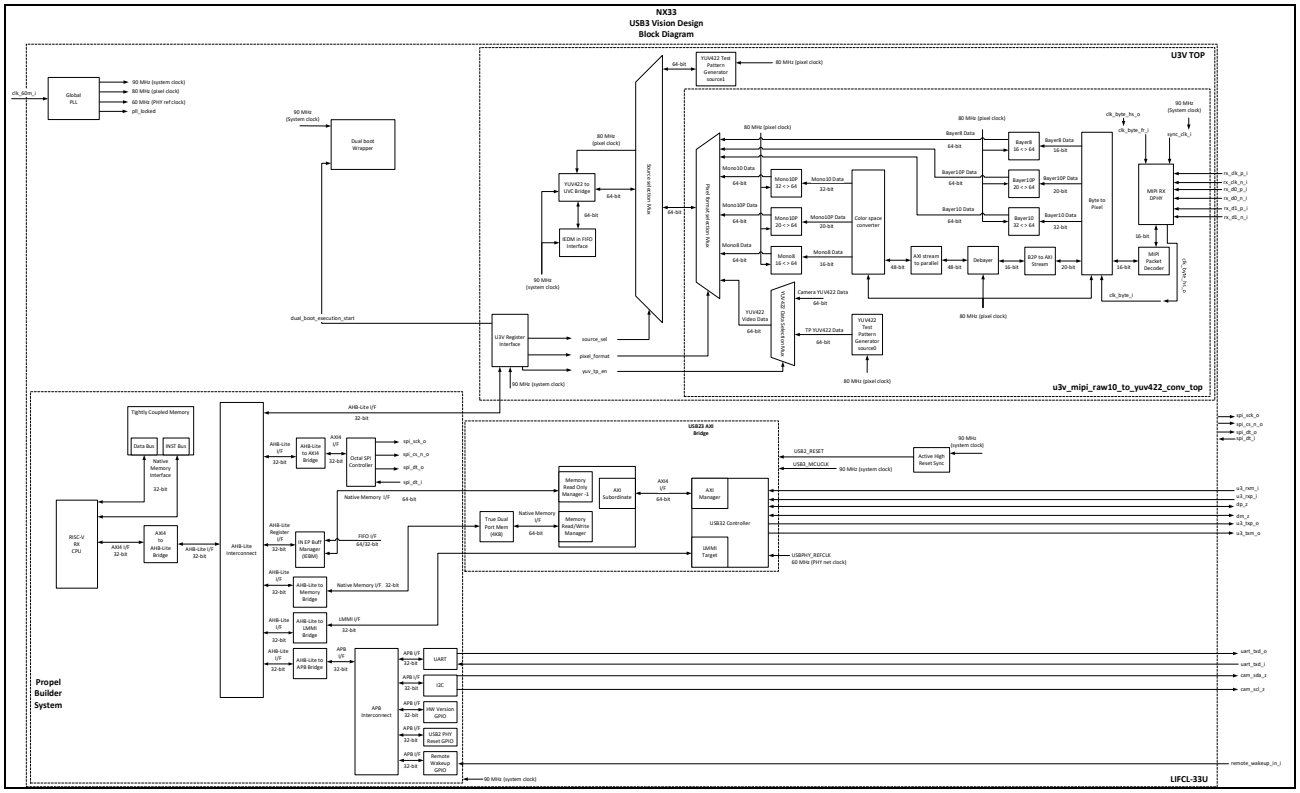

Figure 3.38. USB3 Vision Reference Design Clocking Scheme

3.3. Reset Scheme

The overview of reset scheme for U3V Reference Design is shown in Figure 3.39. The following table summarizes the reset domains in the design.

Table 3.4. Reset Domains

Reset	Source	Destination
cpu_reset_n_90m	PLL lock signal synchronized to PLL clkos_o (CPU clock).	RISC-V CPU rstn_i and MIPI RX DPHY.
riscv_system_reset_n	RISC-V RX.	All components in the Propel software subsystem.
riscv_system_reset_n_90m	riscv_system_reset_n synchronized to PLL clkos_o (CPU clock).	USB23 Controller and the AXI Bridge subsystem, YUV422 to UVC Bridge and IEBM In FIFO interface.
pixel_clk_reset_n	riscv_system_reset_n synchronized to PLL clkos2_o (Pixel clock).	All video IP in the U3V subsystem.
mipi_byte_clk_reset_n	riscv_system_reset_n synchronized to clk_byte_hs_o from the MIPI RX DPHY.	MIPI packet decoder and Byte-to-Pixel Converter IP.

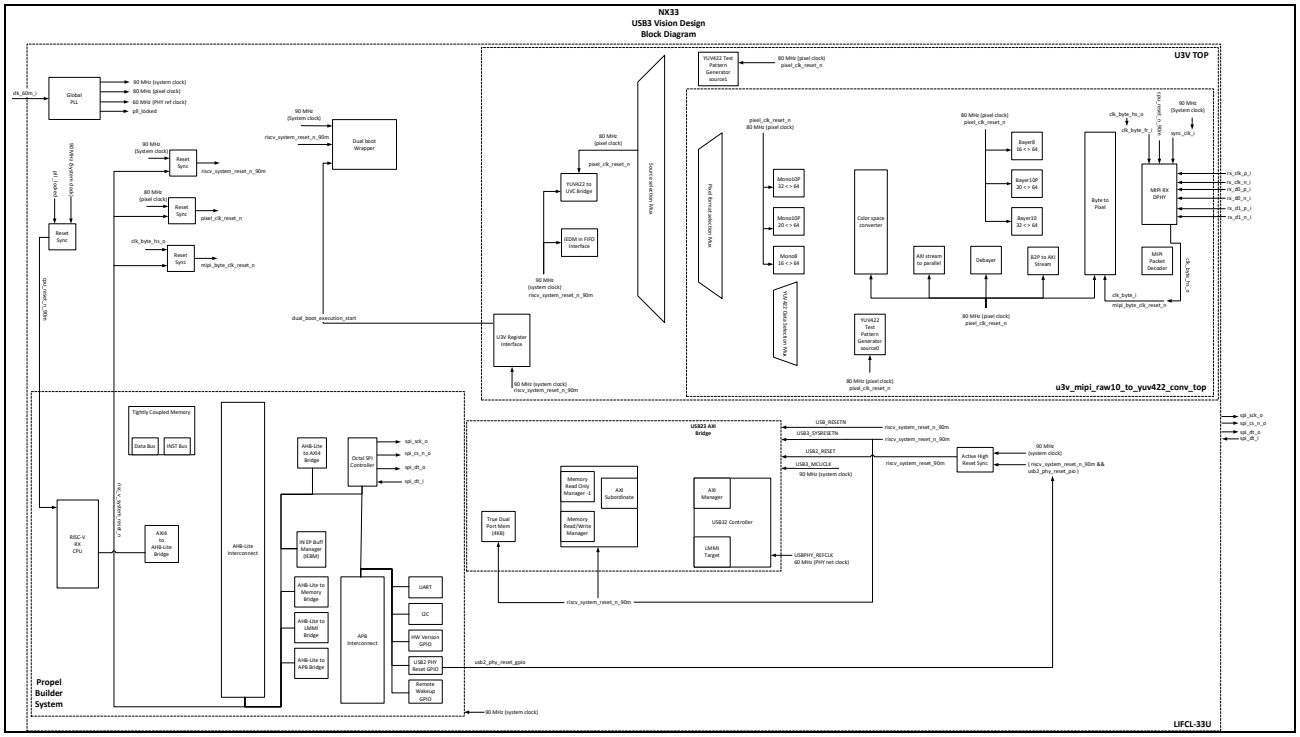

Figure 3.39. Reset Scheme

3.4. Design Operations

This section describes the design operations of the reference design.

3.4.1. USB3 Vision Enumeration

USB3 Vision is an interface standard for industrial cameras using USB 3.0. The enumeration process of a USB3 Vision device follows the standard USB enumeration process with additional steps specific to the USB3 Vision protocol.

After the physical connection is established, the following USB standard enumeration process occurs:

- Device Detection:
  - The host detects the presence of the USB3 Vision camera.
  - The host issues a Reset on the USB bus to ensure the device is in a known state.
- Device Descriptor Request—The host requests the Device Descriptor (PID, VID, class, and subclass) using GET_DESCRIPTOR requests. This helps identify the device as a USB3 Vision camera from the class and subclass field sent by the USB device in response to this request.
- Address Assignment—The host assigns a unique address to the device.
- Configuration Descriptor Request—The host requests the Configuration Descriptor to determine available interfaces and endpoints. It sets the device to the configured state.
- Interface Descriptor Processing—The host retrieves the Interface Descriptor supported by the USB3 Vision device, which includes USB class and subclass values specific to USB3 Vision. In response to the interface descriptor request from the USB host, the USB device provides details about the supported interfaces, such as control interface, streaming interface, event interface, and so on.

After the standard USB enumeration is completed, the host software (such as a USB3 Vision-compliant) interacts with the camera using GenICam and USB3 Vision Protocol.

The following list describes the USB3 Vision specific enumeration:

- Discovery of USB3 Vision Camera:
  - The host software checks if the device supports USB3 Vision Protocol by reading the GenICam XML Descriptor.
  - The XML file describes the camera register map, features, and control parameters.
- Vendor-Specific Control Transfers:
  The host sends Vendor-Specific Requests over Control Transfers (Endpoint 0) to:
  - Read the device capabilities.
  - Retrieve the GenICam XML file.
  - Configure the camera operational settings.
- Bulk Endpoints Setup
  The USB3 vision device exposes Bulk IN and Bulk OUT endpoints:
  - Bulk IN Endpoint: Used for streaming image data to the host.
  - Bulk OUT Endpoint: Used for sending control commands to the camera.

After the camera is enumerated, the USB host queries the supported pixel formats, resolutions, and frame rates. The camera can be configured dynamically via the GenICam interface. The host can start image acquisition by triggering data transfer over bulk endpoints. Then USB3 vision device starts sending the camera data over the BULK IN endpoints and the USB host software processes the data in real time.

3.4.2. USB3 Vision Interfaces

This section intends to give user basic information about USB3 Vision design data flow from USB perspective. The USB3 Vision Interfaces implemented in this reference design follows the definition in USB3 Vision Specifications. This reference design assumes that you have basic familiarity with USB3 Vision standard and USB interface.
This reference design has the following interfaces:

- Device Control Interface (DCI)
- Device Event Interface (DEI)
- Device Streaming Interface (DSI)

These interfaces are discussed in the following subsections.

3.4.2.1. Device Control Interface

This interface is used to control the USB3 Vision aspects of the device. It follows the Generic Control Protocol (GenCP), which is part of the GenICam Standard.

The following register maps are required and defined in the USB3 Vision design. All these registers are managed via the DCI:

- Technology Agnostic Bootstrap Register Map (ABRM)
- Technology Specific Bootstrap Register Map (SBRM)
- Event Interface Register Map (EIRM)
- Streaming Interface Register Map (SIRM)

Refer to USB3 Vision specification on details about each of these registers.

In this reference design, the USB3 Vision registers are implemented as structures in the RISC-V Zephyr firmware code.

The structures are defined in the *usbd_u3v.c* file that includes U3VABRM, U3VSBRM, U3VEIRM, and U3VSIRM.

The offset of these registers are defined in the *usbd_u3v.h* file and are listed below:

```
#define U3V_ABRM_ADDR       0x00000000
#define U3V_SBRM_ADDR       0x00050000
#define U3V_EIRM_ADDR       0x00060000
#define U3V_SIRM_ADDR       0x00070000
```

The first 64 KB of the bootstrap register map is used for ABRM. Therefore, ABRM has base offset 0x0. While rest register maps can be started on a free manufacturer-specific address space after address 0xFFFF.

To read or write any register, the host sends command over Bulk OUT endpoint and device responses to that command over Bulk IN endpoint. In this reference design, the RISC-V firmware decodes command packet received over OUT endpoint, takes necessary action, and prepares response to be sent over IN endpoint. For more information, refer to the Control and Event Channel Protocol section.

3.4.2.2. Device Event Interface

This interface is used to pass transmission of asynchronous events from the device to the host. Events are generated only when the *Event Enable* flag in EI Control Register (part of EIRM) is set.

This reference design supports two types of events:

- U3V Event Test—This is used to test device event interface. This is sent when *TriggerEventTest* bit in Event Test Control register (part of EIRM) is set.
- Start of Frame event test—This is used to send event at the start of frame. This is controlled via its corresponding event notification register. This register is a part of XML-based registers.

One Bulk IN endpoint is used to send event packets to the host. The RISC-V firmware prepares data for this Bulk IN endpoint. Based on request received from the host via the DCI, the firmware decides when this event is sent. For more information on how protocols are used, refer to the Control and Event Channel Protocol section.

3.4.2.3. Device Streaming Interface

This interface is used to pass video streaming data. One Bulk IN endpoint is used to pass this data. Registers controlling this interface are defined in SIRM.

The *Stream Enable* flag that is available in the SI control register (part of SIRM) is used to control streaming. The host sets the *Stream Enable* bit to start streaming. Once that is set, the device streams data over the stream endpoint. When the host wants to stop streaming, it clears the *Stream Enable* bit.

When you start the video streaming using any U3V application (such as eBUS Player), the host sends a command to enable the *Stream Enable* flag. When you stop the streaming from any U3V application, the host sends a command to clear the *Stream Enable* flag. Follow the same method described earlier to perform register access via the DCI.

To allow the separation of user requested payload data and USB3 Vision protocol specific data, USB3 Vision standard uses the concept of a ***leader*** - ***payload*** - ***trailer*** data sequence. Streaming control registers (SIRM) are used to pass information about leader/payload/trailer layout. These registers are also controlled via the DCI. Refer to the U3V specification to get idea about various registers related to this.

Image resolution and pixel format can also be changed from a U3V application if supported by the device. This reference design supports this feature. Registers controlling image resolution and pixel format are part of XML-based registers. Based on XML file content, U3V application knows which registers to be updated whenever you update image resolution and/or pixel format. These registers are accessed via the DCI.

3.4.3. Streaming Data Transfer via IEBM

This section provides a high-level overview of how streaming data from a camera sensor or test pattern generator module is transmitted to a USB host. Kindly have a look at the following image.

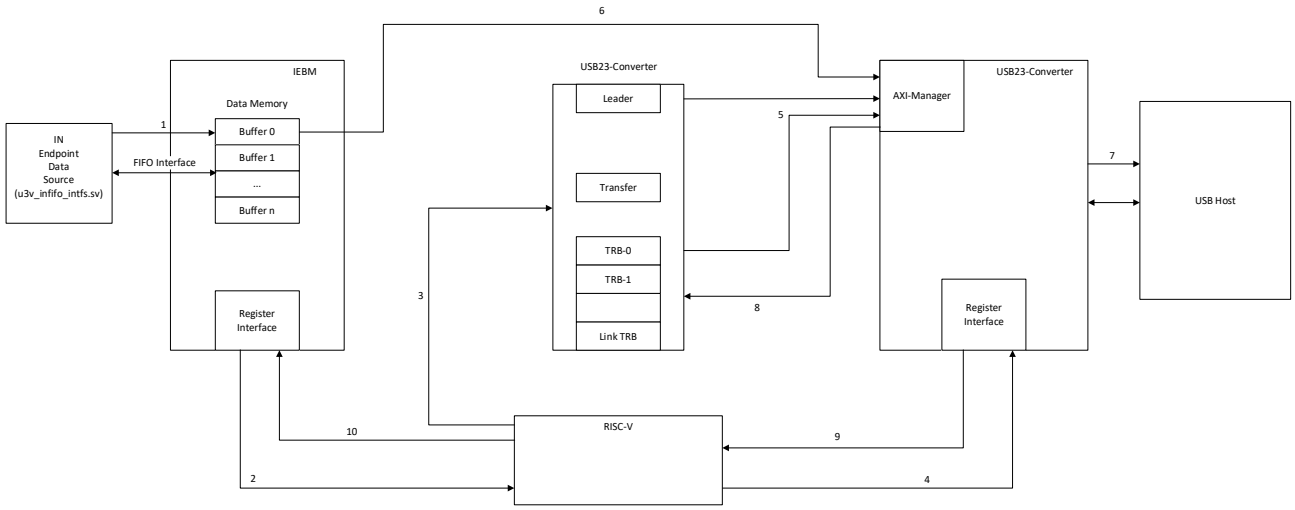

Figure 3.40. Data Flow During Streaming

The IEBM block is configured to accept data from the IN endpoint data source.

Based on the figure above, the streaming data transfers happen in the following sequences:

1. Data from the IN endpoint data source is written into the internal buffer of the IEBM via the FIFO interface. Information about the start of frame and end of frame is also passed to the IEBM to facilitate preparation of Leader and Trailer data.
2. After a buffer is filled, the IEBM generates an interrupt to the RISC-V processor. Meanwhile, if the IEBM has a free buffer available, it switches to that buffer, allowing the IN endpoint data source to continue filling data via the FIFO interface.
3. Upon receiving the interrupt from the IEBM, the RISC-V processor collects information about the buffer that has been handed over.
    a. If this is the first buffer in a frame, firmware prepares leader data and writes it into corresponding on-chip memory. It also prepares the corresponding Transfer Request Block (TRB) in the TRB buffer. If this is not the first buffer, the firmware skips preparing the TRB for the leader.
    b. Firmware prepares TRB for the data buffer and writes it into the TRB buffer. This TRB contains information about the data buffer pointer and size. It also contains some other information useful for the USB23 Controller.
    c. If this is the last buffer in a frame, the firmware prepares trailer data and writes it into the corresponding on-chip memory. It also prepares the corresponding TRB in the TRB buffer. If this is not the last buffer, the firmware skips preparing TRB for the trailer.
4. After the TRB buffer is updated, the RISC-V processor informs the USB23 controller about the availability of the TRB(s) by writing to the appropriate registers via the LMMI interface.
5. The USB23 controller fetches the TRB(s) from the TRB buffer.
6. Based on the information in the TRB, the USB23 controller fetches data either from the IEBM data buffer or from on-chip memory. Leader and trailer data reside in on-chip memory while video data reside in IEBM internal buffer.
7. Upon receiving a request from the USB host for the corresponding IN endpoint, the USB23 controller transmits the data.
8. Once all the data corresponding to the TRB is transmitted to the USB host, the USB23 controller updates the TRB in the TRB buffer.
9. After updating the TRB, the USB23 controller generates an interrupt to inform the RISC-V processor that the TRB has been processed.

10. The RISC-V processor handles the interrupt generated by the USB23 controller. If interrupt is for the TRB associated with IEBM buffer, the RISC-V processor returns the buffer to the IEBM, allowing the IN endpoint data source to reuse the buffer for new data.

3.4.4. Control and Event Channel Protocol

The DCI and DEI operate according to the Generic Control Protocol (GenCP). GenCP is part of the GenICam standard.

It follows the following data transfer layout.

Prefix – Common Command Data (CCD) – Specific Command Data (SCD)

Prefix: USB3 Vision defines the Prefix as 0x43563355 for the control channel and 0x45563355 for the event channel.

Common Command Data: GenCP defines a technology agnostic Common Command Data (CCD) section. It can include a command request or a command acknowledge. The following table lists the command format.

Table 3.5. Common Command Data Format

Width (Bytes)	Offset (Bytes)	Description
2	0	Flags
2	2	Command ID (command_id)
2	4	Length
2	6	Request ID (request_id)

Specific Command Data: The length and content of Specific Command Data (SCD) depends on the command ID (command_id) used.

USB3 Vision devices and the host software use the SCDs as defined by GenCP.

Example 1: Reading Manifest Table Address Register

Technology Agnostic Bootstrap Register Map (ABRM) has one register at offset 0x001D0 that specifies Manifest Table Address. Suppose the host wishes to read its content. For this, the host sends command via Bulk OUT endpoint. It will have the following format:

Table 3.6. Read Manifest Table Data Structure

Section	Width (Bytes)	Offset (Bytes)	Description
Prefix	4	0	Prefix. 0x43563355 for control channel. This Prefix contains the ASCII string *U3VC*, which stands for USB3 Vision Control (with 'U' in the LSB).
CCD	2	4	Flags. 0x4000 to indicate that acknowledgement is required.
	2	6	Command. 0x0800 to indicate READMEM_CMD. (This follows GenCP specification.)
	2	8	Payload length. 0x000C to indicate that the payload has 12 bytes of data. In this example, SCD is 12 bytes.
	2	10	Request ID. This is based on a request. For example, 0x0259.
SCD	8	12	Register address. 0x00000000000001d0 to indicate Manifest Table Address. Refer to the USB 3 Vision specification.
	2	20	Reserved. This is set to 0.
	2	22	Read length. 8 to indicate 8 bytes are read.

Upon receiving the above data from Bulk OUT endpoint, the RISC-V firmware decodes this data. Upon decoding this, it identifies that this command is to read the Manifest Table Address from ABRM. It collects data from the corresponding structure and prepares a response to be sent over to the Bulk IN endpoint. The following response will be sent over Bulk IN endpoint as shown in table below.

Table 3.7. Read Manifest Table Response Data Structure

Section	Width (Bytes)	Offset (Bytes)	Description
Prefix	4	0	Prefix. 0x43563355 for control channel. This Prefix contains the ASCII string *U3VC* standing for USB3 Vision Control (with 'U' in the LSB).
CCD	2	4	Flags. 0x0000 which represents U3V_STATUS_GENCP_SUCCESS. For example, status is successful.
	2	6	Command. 0x0801 to indicate READMEM_ACK. (This follows GenCP specification.)
	2	8	Payload length. 0x0008 to indicate payload has 8 bytes of data. In this example, SCD is of 8 bytes.
	2	10	Request ID. This is based on a request. Let's take 0x0259 as an example.
SCD	8	12	Data. 0x0000000000080000. This represents the Manifest Table Address. This relates to U3V_MANIFEST_TABLE_ADDR macro value available in *lsc_usb3vision.h*.

Example 2: FrameStart Event Format

This example shows the event data format that is being used to send FrameStart event. You control this event generation from the eBUS Player or similar application. This section describes what happens at the FPGA side.

To enable this event, the host sets the following fields in corresponding registers:

- *Event Enable* flag in EI Control Register (part of EIRM) is set.
- *Event Notification Enable* flag in Event notification register (part of XML-based register) is set.

Both registers are written in the same way as explained in Example 1.

After these registers are written, the device sends event at the start of frame. This event data is sent over corresponding Bulk endpoint and has the structure shown in the following table.

Table 3.8. Frame Start Event Data Structure

Section	Width (Bytes)	Offset (Bytes)	Description
Prefix	4	0	Prefix. 0x45563355 for event channel. This prefix contains the ASCII string "U3VE" standing for USB3 Vision Event (with 'U' in the LSB).
CCD	2	4	Flags. 0x0000 to indicate that acknowledgement is not required.
	2	6	Command. 0x0C00 to indicate EVENT_CMD. This follows GenCP specification.
	2	8	Payload length. 0x0014 to indicate that the payload has 20 bytes of data. In this example, SCD is of 20 bytes.
	2	10	Request ID. This is based on request.
SCD	2	12	Event size. 0 as Multiple Events in a single command is not supported.
	2	14	Event ID.

Section	Width (Bytes)	Offset (Bytes)	Description
			0x9C40 that indicates Frame Start event. This mapping is managed in an XML file.
	8	16	Timestamp. This indicates the timestamp value.
	8	24	Event-specific data. This example passes 8 bits of frame ID.

Events are generated only when the *Event Enable* flag in EI Control Register (part of EIRM) is set.

3.4.5. XML File

The USB3 Vision standard relies on the GenICam standard to describe the features supported by the camera. This description takes the form of an XML device description file respecting the syntax defined by the GenApi module of the GenICam standard.

For more information on the GenICam standard, refer to the GenICam website.

This XML file is retrieved and interpreted by the host software to enumerate the features supported by the device. The XML device description file provides the mapping between a device feature and the device registers supporting it.

3.4.5.1. Device Discovery and Manifest Table Fetch

After the USB3 Vision camera is connected and enumerated by the host, the host queries the camera for its manifest table. The host sends a Control Request to read the manifest table, which provides details about the location and size of the XML file.

3.4.5.2. Fetching the XML File

The manifest table contains the necessary information for the host to locate the XML file such as its address. Based on the location provided, the host initiates a bulk read transfer (or control transfer) to retrieve the XML file in chunks. USB3 Vision allows high-speed data transfer so the host can fetch large XML files efficiently using bulk transfer over the control endpoint. The Address of the Manifest Table is defined at register offset 0x1D0. This is a part of ARBM. The manifest table contains a list of Manifest Entries as shown in the table below.

Table 3.9. Manifest Table Layout

Width (Bytes)	Offset (Bytes)	Support	Access	Description
8	0	M	R	**MT Entry Count** Number of entries in the Manifest Table.
64	8	M	R	**Manifest Entry 0** First entry in the Manifest Table.
64	8 + 64	O	R	**Manifest Entry 1** Second entry in the Manifest Table.
…	…			…
64	8 + n*64	O	R	**Manifest Entry n** (N+1)th entry in the Manifest Table.

By reading the manifest table, the Host gets information about the number of XML files supported by device. Each Manifest Entry describes the properties of a single file. Manifest Entry follows the format according to GenCP standard as shown in table below.

Table 3.10. Manifest Entry Layout

<table>
<tr><th>Width (Bytes)</th><th>Offset (Bytes)</th><th>Description</th></tr>
<tr><td>4</td><td>0</td><td>GenICam File Version

<table>
<tr><th>Bit offset (lsb << x)</th><th>Width (bits)</th><th>Description</th></tr>
<tr><td>0</td><td>16</td><td>File-Subminor Version
Subminor version of the GenICam file referenced in this entry.</td></tr>
<tr><td>16</td><td>8</td><td>File-Minor Version
Minor version of the GenICam file referenced in this entry.</td></tr>
<tr><td>24</td><td>8</td><td>File-Major Version
Major version of the GenICam file referenced in this entry.</td></tr>
</table></td></tr>
<tr><td>4</td><td>4</td><td>Schema / Filetype / Fileformat

<table>
<tr><th>Bit offset (lsb << x)</th><th>Width (bits)</th><th>Description</th></tr>
<tr><td>0</td><td>3</td><td>File Type
File type of the file this manifest entry points to.
0 = Device XML
This is the “normal” GenICam device xml containing all device features. This is the one file provided in GenCP until version 1.1.
1 = Buffer XML
This optional XML-file contains only the chunkdata related nodes. This allows the consumer to instantiate one nodemap per buffer in case the buffers containing chunk data and so work on multiple buffers in parallel.
2-7 = reserved</td></tr>
<tr><td>3</td><td>7</td><td>Reserved
Set to 0.</td></tr>
<tr><td>10</td><td>6</td><td>File Format
The file format of the file this entry points to.
0 = Uncompressed GenICam XML file
1 = ZIP containing a single GenICam XML file
2-63 = reserved</td></tr>
<tr><td>16</td><td>8</td><td>Schema-Minor Version
Minor Version of the GenICam Schema the GenICam file complies with.</td></tr>
<tr><td>24</td><td>8</td><td>Schema-Major Version
Major Version of the GenICam Schema the GenICam file complies with.</td></tr>
</table></td></tr>
<tr><td>8</td><td>8</td><td>Register Address
Register Address at which the file can be read from.</td></tr>
<tr><td>8</td><td>16</td><td>File Size
Size of the file this manifest entry points to in bytes.</td></tr>
<tr><td>20</td><td>24</td><td>SHA1-Hash</td></tr>
</table>

Width (Bytes)	Offset (Bytes)	Description
		SHA1 Hash of the file or 0 in case the hash is not available.
20	44	Reserved Set to 0.

Based on the manifest entry, the host gets information about register address from which this file can be read and file size. The host reads the entire XML file via the DCI. The USB3 Vision device uses a zip format of XML file to send to the host when host sends a read request to read an XML file. This zipped XML file raw data is stored using an array U3VXML defined in the *lsc_usb3vision_xml.c* file.

3.4.5.3. Parsing and Using the XML File

After the XML file is completely downloaded to the host, it is parsed using the GenICam standard. This XML file defines the camera device features, controls, and capabilities, such as resolutions, pixel format, frame rates, and many more. The host can then use this information to control the camera device and configure it.

Example Usage of XML File

The following table describes the U3V Mandatory Features in the XML file based on the definitions from the USB3 Vision Specification.

Table 3.11. USB3 Vision Mandatory Features for the Camera XML Description File

Feature Name	Name	Access	Units	Description
Width	IInteger	R/(W)	Pixels	Width of the image output by the camera on the stream channel.
Height	IInteger	R/(W)	Pixels	Height of the image output by the camera on the stream channel. For linescan sources, this represents the maximum height of the frame created by combining consecutive lines.
PixelFormat	IEnumeration	R/(W)	—	Format of the pixel output by the camera on the stream channel.

From XML file, the host can retrieve the device supported setting Width of output image and which register offset shall be used to inform device to set width of an output image.

If USB Vision device supports configuration of width of output image. When it receives request from the host via corresponding register and if that value is supported by the device, it configures camera sensor to the requested width. If camera configuration is successful, it returns GENCP_SUCCESS as a status code. If received width is invalid, GENCP_WRONG_CONFIG status is returned.

The corresponding register is a part of XML-based registers in this reference design. WidthRegAddr member of U3VXMLIMPREG structure represents this register in our case. This structure has been defined in the *usbd_u3v.c* file.

There is a mapping for the device feature defined in an XML file in the RISC V firmware. These XML features are implemented in firmware based on XML Inquiry register(U3VXMLINQREG) and XML Implementation register (U3VXMLIMPREG) sets.

U3VXMLINQREG: This register set helps to get an idea about the features that are supported by the U3V device. Offset and name of these registers are retrieved from the XML file. After reading the XML file, the host reads these registers to get information about the supported features.

U3VXMLIMPREG: This register set represents the implementation of the corresponding supported features defined in U3VXMLINQREG. The host performs read or write operations on this register to configure the U3V device.

Example:

XML has TestPatternInq_RegAddr with register offset 0x24 as shown in the figure below.

```
<Integer Name="TestPatternInq_RegAddr">
  <Value>0x00010024</Value>
</Integer>
```

Figure 3.41. Test Pattern Inquiry Register Offset in XML File

For this address offset 0x24, the value 0x00000900 has been assigned in the RISC-V firmware as shown in the figure below.

```
U3VXMLINQREG_S U3VXMLINQREG =
{
    .DeviceControlInqRegAddr = 0x00000043,          /**< Offset: 0x0 <br> DeviceControlInq_RegAddr :
    .ReverseInqRegAddr = 0x00018001,                /**< Offset: 0x4 <br> ReverseInq_RegAddr - 0x0180
    .WidthMinRegAddr = 0x00000000,                  /**< Offset: 0x8 <br> WidthMin_RegAddr - 0x0 */
    .HeightMinRegAddr = 0x00000000,                 /**< Offset: 0xC <br> HeightMin_RegAddr - 0x0 */
    .HunitRegAddr = 0x00000010,                     /**< Offset: 0x10 <br> Hunit_RegAddr - 0x10 */
    .VunitRegAddr = 0x00000002,                     /**< Offset: 0x14 <br> Vunit_RegAddr - 0x02 */
    .RegionSelectorInqRegAddr = 0x40000001,         /**< Offset: 0x18 <br> RegionSelectorInq_RegAddr
    .PixelFormatInq0RegAddr = 0x00000581,           /**< Offset: 0x1C <br> PixelFormatInq0_RegAddr -
    .PixelFormatInqRegAddr = 0x00330009,            /**< Offset: 0x20 <br> PixelFormatInq_RegAddr -0x
    .TestPatternInqRegAddr = 0x00000900,            /**< Offset: 0x24 <br> TestPatternInq_RegAddr - 0
    .FrameSpecInfoInqRegAddr = 0x000003DF,          /**< Offset: 0x28 <br> FrameSpecInfoInq_RegAddr -
    .FrameSpecInfoROIPositionInqBit = 0x03000300    /**< Offset: 0x2C <br> FrameSpecInfoROIPositionIn
```

Figure 3.42. Data Structure—Value of Test Pattern Inquiry Register

When the host wants to get the device supported test patterns, the host sends read command with address 0x00010024, where 0x00010000 is a base offset of U3VXMLINQREG and 0x24 is an offset of TestPatternInq_RegAddr. The device sends data 0x00000900 in response to the read command. This data indicates the supported test patterns that the device has.

When you want to change any of the supported test patterns, the host application sends corresponding write command with address 0x00030030 where 0x00030000 indicates the base address of U3VXMLIMPREG and 0x30 is an offset of TestPattern_RegAddr. This TestPattern_RegAddr is defined in XML as shown the figure below.

```
<Integer Name="TestPattern_RegAddr">
  <Value>0x00030030</Value>
</Integer>
```

Figure 3.43. Test Pattern Register Offset in XML File

Firmware receives data from an application for test pattern and writes it at the address offset of 0x30 (TestPatternRegAddr) in U3VXMLIMPREG as shown below.

```
U3VXMLIMPREG_S U3VXMLIMPREG =
{
    .AcquisitionStartRegAddr = 0x00000000,                  /**< Offset: 0x0 <br> AcquisitionStart_RegAddr - RW */
    .AcquisitionStopRegAddr = 0x00000000,                   /**< Offset: 0x4 <br> AcquisitionStop_RegAddr - RW */
    .DeviceTemperatureRegAddr = 0x00000000,                 /**< Offset: 0x8 <br> DeviceTemperature_RegAddr - R */
    .DeviceResetRegAddr = 0x00000000,                       /**< Offset: 0xC <br> DeviceReset_RegAddr - W */
    .FindMeRegAddr = 0x00000000,                            /**< Offset: 0x10 <br> FindMe_RegAddr - W */
    .DeviceLinkThroughputLimitModeRegAddr = 0x00000000,     /**< Offset: 0x14 <br> DeviceLinkThroughputLimitMode_RegAddr    - RW */
    .RegionModeRegAddr = 0x00000000,                        /**< Offset: 0x18 <br> RegionMode_RegAddr - RW */
    .WidthRegAddr = U3V_IMAGE_SIZE_X_DEFAULT,               /**< Offset: 0x1C <br> Width_RegAddr    - RW */
    .HeightRegAddr = U3V_IMAGE_SIZE_Y_DEFAULT,              /**< Offset: 0x20 <br> Height_RegAddr - RW */
    .OffsetXRegAddr = 0x00000000,                           /**< Offset: 0x24 <br> OffsetX_RegAddr - RW */
    .OffsetYRegAddr = 0x00000000,                           /**< Offset: 0x28 <br> OffsetY_RegAddr - RW */
    .LinePitchRegAddr = 0x00000000,                         /**< Offset: 0x2C <br> LinePitch_RegAddr - RW */
    .TestPatternRegAddr = 0x00000000,                       /**< Offset: 0x30 <br> TestPattern_RegAddr - RW */
    .ReverseXRegAddr = 0x00000000,                          /**< Offset: 0x34 <br> ReverseX_RegAddr - RW */
```

Figure 3.44. Data Structure—Value of Test Pattern Register

3.4.5.4. Video Pipeline Operation

The block diagram in the following figure shows the video IP path in this reference design.

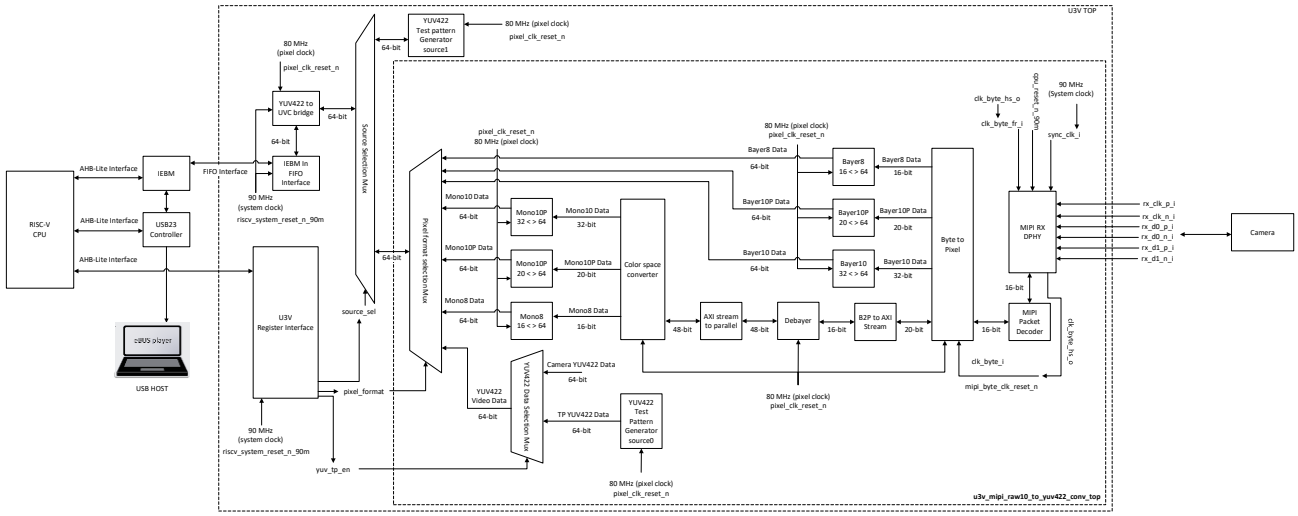

Figure 3.45. U3V Design Video IP Path Block Diagram

3.4.5.5. USB Video Path Overview

The USB video path processes the MIPI serial data that is received from the camera sensor and converts it into a suitable USB video format that can be streamed to the host system. This process involves multiple stages of data transformation that is handled by different IP cores. The following lists the step-by-step breakdown of the pipeline:

1. MIPI Serial to Parallel Conversion
   - The RX DPHY IP core receives MIPI serial data coming from the camera sensor.
   - It converts the high-speed serial stream into parallel pixel data, making it suitable for further processing.
2. Byte-to-Pixel Conversion
   - The Byte-to-Pixel Converter IP core takes the MIPI byte data and reorganizes it into a pixel-formatted structure.
   - The output format from B2P is typically Bayer format, which needs further processing before display or streaming.
3. Bayer to RGB Conversion
   - The Debayer IP core processes the Bayer-formatted pixel data and converts it into a full RGB image.
   - This is an essential step to reconstruct a meaningful image from raw sensor data.
4. RGB to YUV422 Conversion
   - The Color Space Conversion IP core takes the RGB pixel data and converts it into YUV422 format, which is widely used for video compression and streaming.
   - The YUV test pattern generator module produces predefined solid color patterns in YUV422 format for testing and debugging.
5. Supported Pixel Formats
   - As per the block diagram, the system supports multiple pixel formats after the YUV422 conversion step. The available formats include:
     - YUV format: YUV422. Used for compressed video transmission.
     - Monochrome (Grayscale) formats: MONO8, MONO10, and MONI10P
     - Bayer (Raw Sensor) Formats: BAYER8, BAYER10, and BAYER10P
6. Data Selection and Transmission to USB
   - Depending on the pixel_format selection value set by you, the appropriate pixel data is selected using a multiplexer (MUX).
   - The selected pixel data is then passed to the IEBM buffer, which stores camera data or YUV test pattern data. After one buffer is filled, it is handed over to the RISC-V CPU for further processing.

3.4.5.6. Changing Video Properties

When user change the pixel format from the eBUS player, the following activities will be performed.

1. The eBUS Player sends a request to the GenICam-compliant USB3 Vision device to change the pixel format. Let's say the user has selected the MONO10 format. This request is transmitted via the USB3 Vision control channel, following the GenICam standard. The request includes the PixelFormat register value corresponding to MONO10, ensuring the camera correctly interprets and updates its internal settings.
2. The RISC-V CPU, which acts as the embedded processing unit inside the USB3 Vision device, receives the pixel format update request. Upon receiving this request, the CPU:
   a. Processes the command, extracts the requested pixel format, and verifies whether MONO10 is supported.
   b. Stores the GenICam-compliant pixel format value internally for reference.
   c. Generates an acknowledgment response and sends it back to the USB host via the USB3 Vision protocol, confirming that the requested pixel format update has been successfully applied.
3. After successfully processing the request, the RISC-V CPU writes the updated GenICam-compliant pixel format value into the U3V register interface module. The format update is communicated using the AHB-Lite interface, which connects the CPU with the register interface module.
4. In this case, because the selected format is MONO10, the RISC-V CPU will not enable the YUV test pattern generator. This Pixel Format Data Selection Multiplexer is responsible for dynamically selecting the correct data format based on the updated PixelFormat register value. The selected MONO10 data is then passed to the subsequent processing modules for further handling, such as transmission over the USB3 pipeline.

3.4.5.7. Selecting YUV Test Pattern

The YUV test pattern data can be generated by the design. This is useful as an alternative data source besides the camera sensor. To select the YUV test pattern, follow these steps:

1. Select the YUV422 pixel format in the eBUS Player, indicating that the image data must be transmitted in a YUV422 color space.
2. Enable the test pattern mode by setting it to *TestImage_YUV422* within the eBUS Player. This instructs the U3V device to generate a YUV422 test pattern instead of using real sensor data.
3. After the RISC-V CPU receives and processes both requests:
   a. It updates the PixelFormat register with the GenICam-compliant YUV422 format value, ensuring that the correct encoding scheme is selected.
   b. It enables the YUV422 test pattern module, which generates predefined YUV422 test image data.
4. As a result, the Pixel Format Data Selection Multiplexer ('Pixel Format Data Selection Mux') identifies that the test pattern mode is active and selects the YUV422 test pattern data instead of actual sensor data. The selected test pattern data is then passed to the subsequent processing modules for further handling.

3.4.6. SPI Flash Operation

The SPI flash is used in this design to store the FPGA bitstream and the USB3 Vision device GUID. You can write to the SPI flash from the host computer using a Windows application (*USBTestApp.exe*). The commands are sent from the host to the FPGA via a USB connection. The figure below shows the flowchart for SPI File write operation.

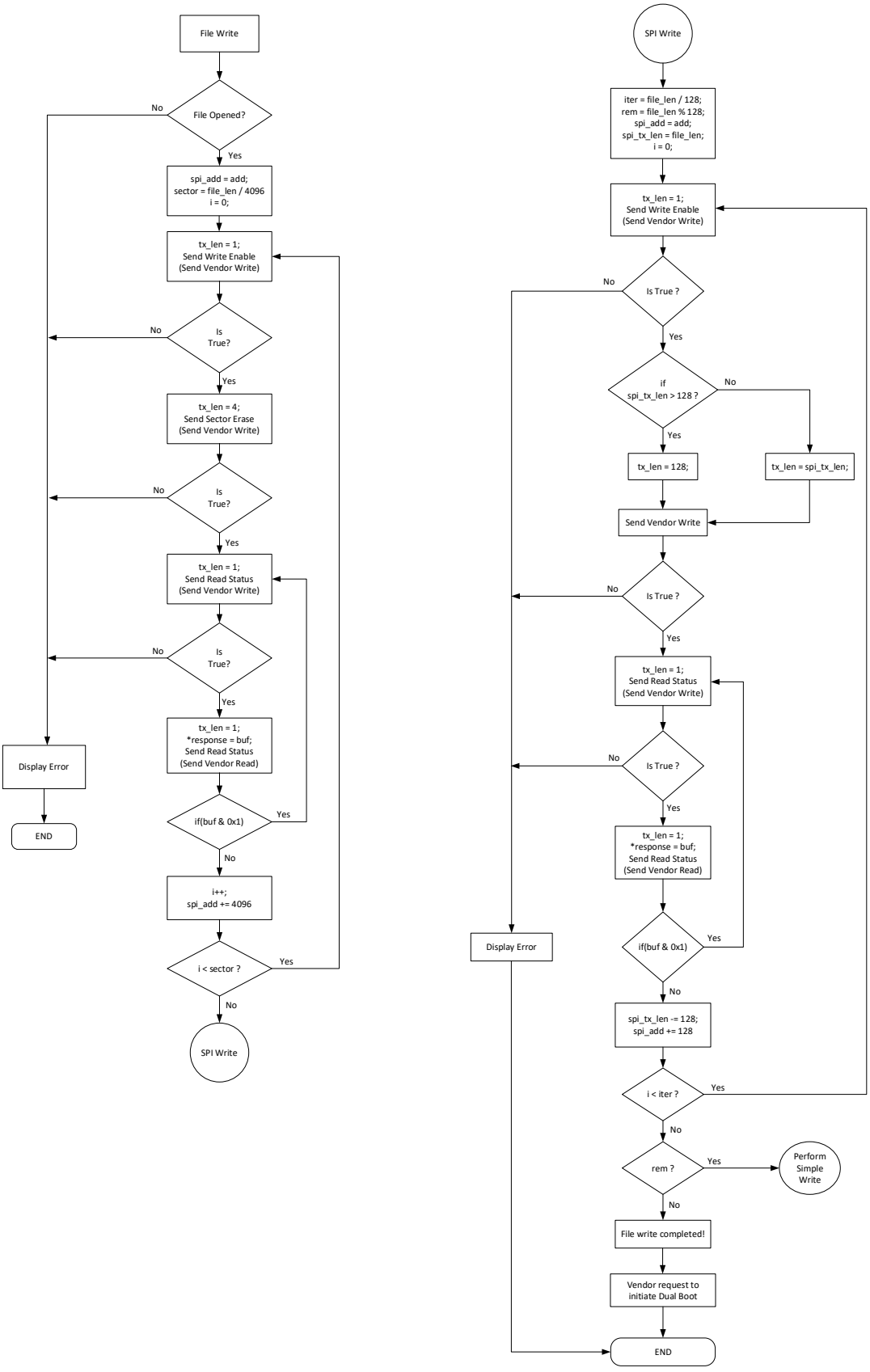

Figure 3.46. SPI File Write Operation Flow Chart

Notes:

1. It is assumed that there is a bit file running on the device that supports SPI Operations for USB Test Application.
2. You must have a USB Test application and *U3V_GUID.bit* file for usage.

At the end of the SPI flash write operation, the USB Test application sends a vendor-specific USB request to initiate the dual boot sequence. The dual boot sequence is a process to re-program the FPGA into the newly written bit file in the SPI flash. On the firmware side, when the RISC-V processor receives this request, it toggles a dedicated GPIO signal. This action triggers the dual boot process in the hardware.

4. U3V Reference Design Signal Description

The input and output interface signals for the CrossLinkU-NX USB3 Vision Reference Design are shown in the table below.

Table 4.1. Primary I/O

Port Name	Direction	LIFCL-33U Ball	Description
clk_60m_i	Input	H8	60 MHz input clock form the on-board crystal oscillator.
dp_z	Input/Output	D7	USB2.0 positive differential data line.
dm_z	Input/Output	E7	USB2.0 negative differential data line.
u3_rxp_i	Input	A8	USB3.0 positive differential data line for receiver.
u3_rxm_i	Input	B8	USB3.0 negative differential data line for receiver.
vbus_z	Input/Output	E5	Power signal.
u2_reset_ext_z	Input/Output	—	Synchronous reset for USB 2.0.
REFINCLKEXTP_i	Input	F8	USB3.0 PHY external positive differential clock.
REFINCLKEXTM_i	Input	E8	USB3.0 PHY external negative differential clock.
u3_txp_o	Output	A7	USB3.0 positive differential data line for the transmitter.
u3_txm_o	Output	A6	USB3.0 negative differential data line for the transmitter.
cam_scl_z	Input/Output	J6	I2C controller serial clock line (SCL) for the camera sensor.
cam_sda_z	Input/Output	H6	I2C controller serial data line (SDA) for the camera sensor.
cam_en_o	Output	L7	Camera sensors enable.
rx_clk_p_i	Input/Output	F7	Camera positive differential clock line.
rx_clk_n_i	Input/Output	E6	Camera negative differential clock line.
rx_d0_p_i	Input/Output	M7	Camera positive differential data0 line.
rx_d0_n_i	Input/Output	M6	Camera negative differential data0 line.
rx_d1_p_i	Input/Output	N7	Camera positive differential data1 line.
rx_d1_n_i	Input/Output	N6	Camera negative differential data1 line.
uart_rxd_i	Input	F1	UART input for RISC-V microcontroller debugging.
uart_txd_o	Output	G1	UART output for RISC-V microcontroller debugging.
remote_wakeup_in_i	Input	B1	Remote wakeup GPIO input pin.
spi_cs_n_o	Output	B3	SPI Chip select line.
spi_sck_o	Output	B4	SPI serial clock line.
spi_dt_o	Output	D4	SPI data output line.
spi_dt_i	Input	D3	SPI data input line.

5. Running the U3V Demo on Evaluation Board

This section describes how to run the U3V demo using prebuilt binary files on the LIFCL-33U Evaluation Board.

5.1. Extracting Reference Design Files

Before you start running the demo, download the design archive file from the Lattice Semiconductor website.

Extract the *CrossLinkU-NX-U3V-RD-Source-RevA.zip* file to your local directory, example to *C:\workspace*. This path is referred as *<my_workspace> hereafter, for example <my_workspace> = C:\workspace\ CrossLinkU-NX-U3V-RD-Source-RevA*.

5.2. Pleora eBUS Player Installation

This reference design requires installation of Pleora eBUS Player to stream the USB3 Vision video. Download the eBUS software version 6.2.8.5877 from the Pleora eBus Player download page and install it into your PC or laptop.

5.3. CrossLinkU-NX LIFCL-33U Evaluation Board Connection

Connect the following to the CrossLinkU LIFCL-33U Evaluation Board:

- Connect Raspberry Pi Camera Module 2 to the CN1 connector.
- Connect the Micro USB port (J2) of the LIFCL-33U Evaluation Board to your PC by using a Micro USB.
- Make sure that Jumper J5 is not connected.

5.4. Programming FPGA Bitstream to Flash

In case the LIFCL-33U Evaluation Board must be programmed or re-programmed, refer to the following steps:

1. Launch the Radiant Programmer. Click the **Operation** area to bring up the Device Properties window.

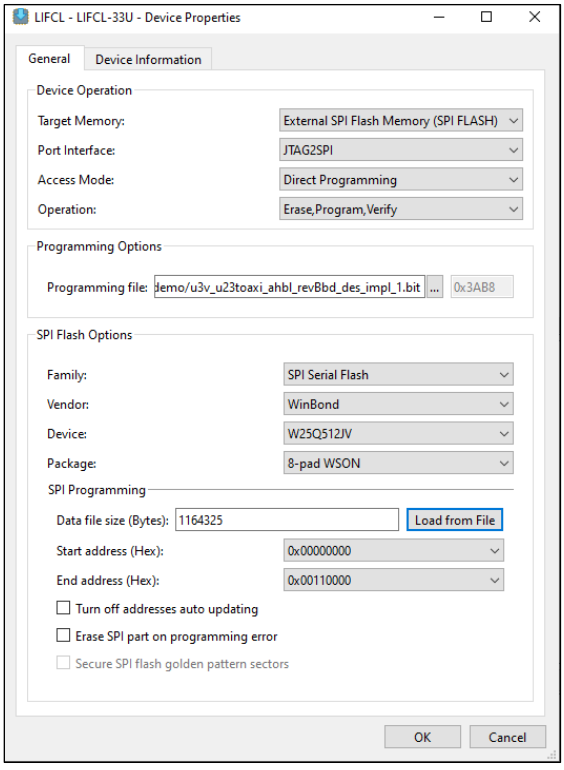

Figure 5.1. Device Properties Options

2. Select options for **Target Memory**, **Port Interface**, **Access Mode,** and **Operation**.
3. Select the bitstream file from the *<my_workspace>\hardware\ref_design\demo\u3v_u23toaxi_ahbl_revBbd_des_impl_1.bit* under **Programming files**.
4. Choose the **SPI Flash Options**.
5. Click **OK** after configuration.
6. Click the **Program Device** icon from the toolbar to perform the device programming. The following figure shows a successful operation.

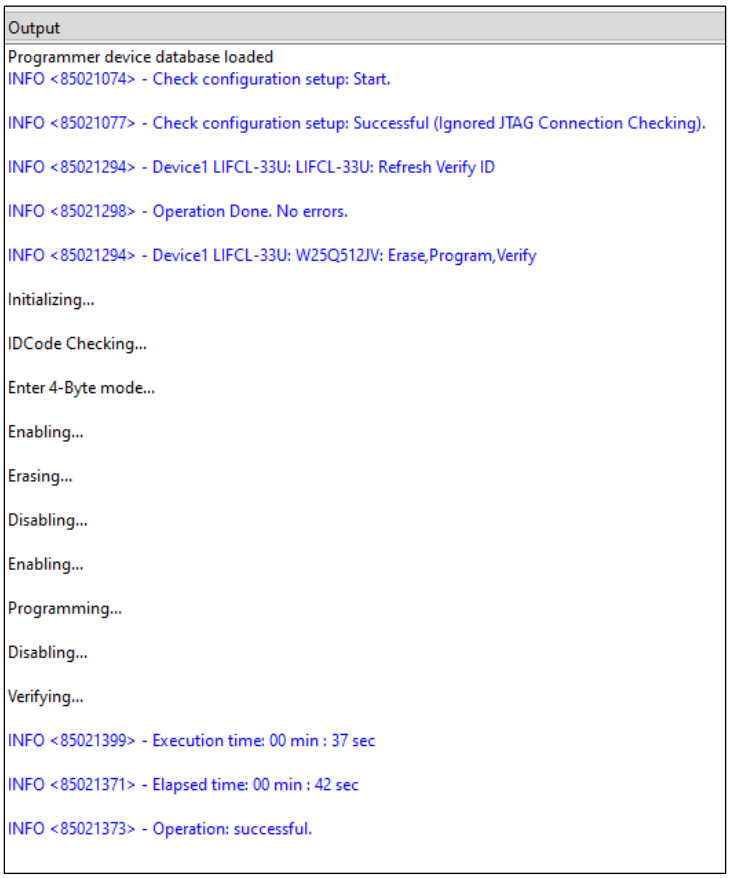

Figure 5.2. Successful Device Programming Operation

5.5. Connect Evaluation Board to PC

Once the bitstream programming is successful, connect the USB-C port on the evaluation board to PC by following these steps:

1. Disconnect the Micro USB cable from PC and the board J2 connector.
2. Connect the USB type-C connector CN2 to your PC using the USB-C to USB type-C cable or the USB-C to USB type-A 9-pin cable.
3. The evaluation board must be enumerated as **USB3 Vision Device**, which can be found in the Windows Device Manager. The figure below shows that the USB3 Vision Device is enumerated.

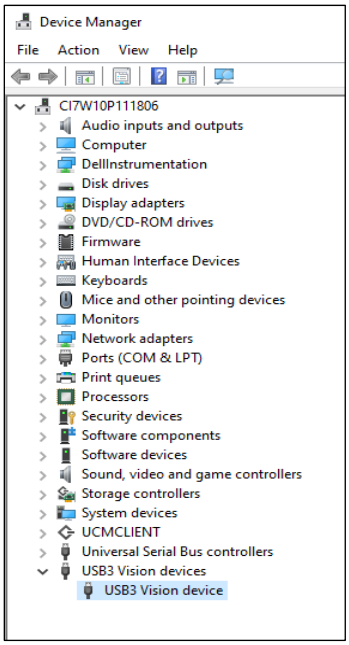

Figure 5.3. U3V USB Enumeration

5.6. USB3 Vision Demonstration

This section lists the steps on how to see camera/YUV test pattern video data on the eBUS Player.

5.6.1. Device Connection to eBUS Player

To connect the device to the eBUS Player, follow these steps:

1. Launch the eBUS Player software.
2. Click on the **Select / Connect** tab to connect the USB3 Vision device to the eBUS Player application.
3. Select **U3V Camera-NX33U [2AC100000000]** device and click on **OK**.

5.6.2. Video Streaming from Camera Sensor

To start video streaming from the Raspberry Pi camera sensor, follow these steps:

1. Click on the **Device control** tab.
2. Configure the following settings available under the **ImageFormatControl** tab:
   a. Change the width and height, by typing in the following allowable values:
      - Width = 1,280, Height = 720
      - Width = 1,920, Height = 1,080
      - Width = 3,280, Height = 2,160

      Note: The demo bitstream file that is provided only supports 1080P resolution. For more information on the support for other resolutions, refer to the Changing to the Other Supported Resolutions section.
   b. Double-click on the **PixelFormat** to change the format to any option available in the drop-down list.
   c. Set **TestPattern** to **Off** to view the camera sensor video.
3. Click on the **Play** button to see the video from the camera sensor. The video FPS and USB throughput (Mbps) are shown at the bottom of the eBUS Player.
4. Click the **Stop** button.
5. Click the **Disconnect** button before you remove the USB-C cable from the computer.

5.6.3. Video Streaming from YUV422 Test Pattern

The reference design contains a YUV422 test pattern generator that generates solid color pattern video. This is useful to stream high-bandwidth frame rates over the USB, which is not limited by the Raspberry Pi sensor bandwidth. Follow these steps to enable the test pattern:

1. Click on the **Device control tab**.
2. Set **TestPattern** mode to **TestImage_YUV422**.
3. Click the **Play** button.
4. Click the **Stop** button. Note that you must click on the **Stop** button if the video streaming is on-going.
5. Click the **Disconnect** button before you remove the USB-C cable from the computer.
6. Select YUV422 test image mode to see the YUV422 test pattern on the eBUS Player. The eBUS Player flashes various solid color frames.

5.6.4. SPI File Write Operation Using USB Test Application

This section lists the steps on how to use **USB Test application** for loading bit file into SPI flash and updating U3V GUID for the device.

5.6.4.1. Download Demo Bitstream File in SPI Flash Using USB Test Application

The following steps demonstrate how to write bitstream file into the SPI flash using the USB Test application.

1. Launch the *USBTestApp.exe* in the *software\utilities\USBTestApp* directory.

   Note: Make sure the USB-C cable is connected to the board.

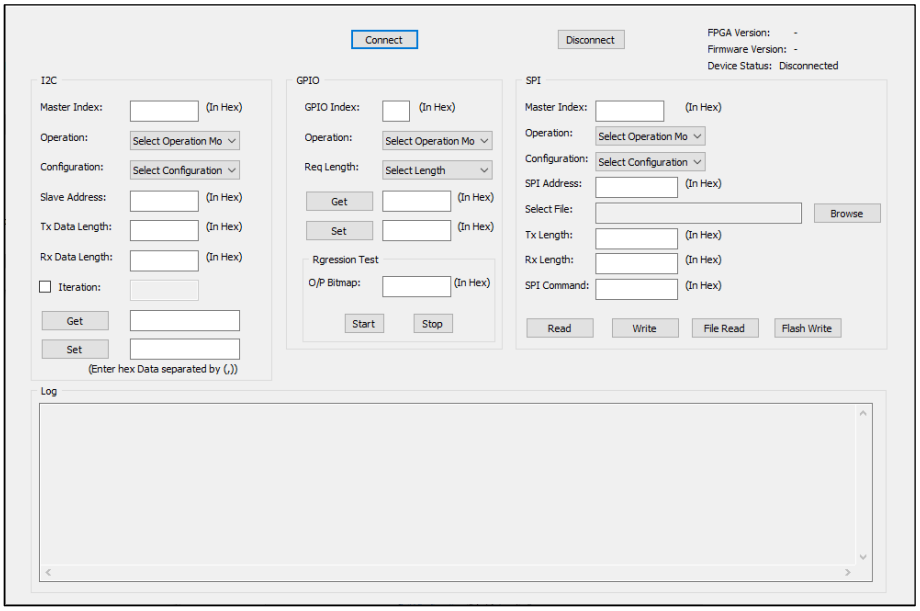

Figure 5.4. USB Test Application

2. Click on the **Connect** button to connect to the device. On successful connection, you will see the information highlighted in the figure below.

Figure 5.5. Device Connected to USB Test Application

3. To write a new bitstream file to SPI flash, set the following GUI parameters:
   - SPI Address = 0x0
   - Select File = Browse and select the bitstream in *<my_workspace>\hardware\ref_design\demo\u3v_u23toaxi_ahbl_revBbd_des_impl_1.bit*

Note: Make sure you select a bitstream file that supports both USB and image upgrade functionality.

If a bitstream without this support is selected, the SPI flash upgrade will not be possible. In this case, the only way is to use the JTAG programmer.

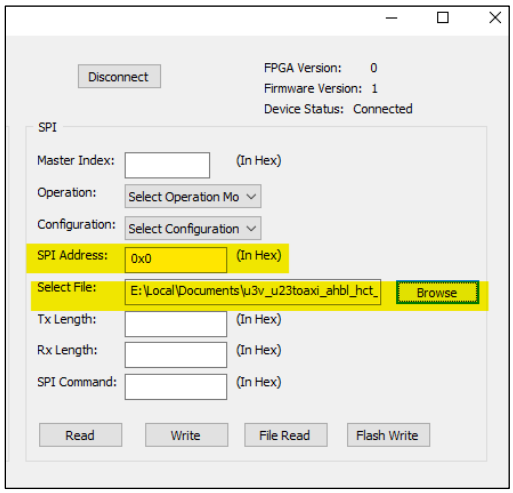

Figure 5.6. SPI File Write Settings

4. Click on the **Flash Write** button to load the file in the flash at the given SPI address and wait for the write operation to be completed. The write operation may take some time depending on the size of the bit file. The message *file data verified!* is displayed in the Log section when the write operation is successful.

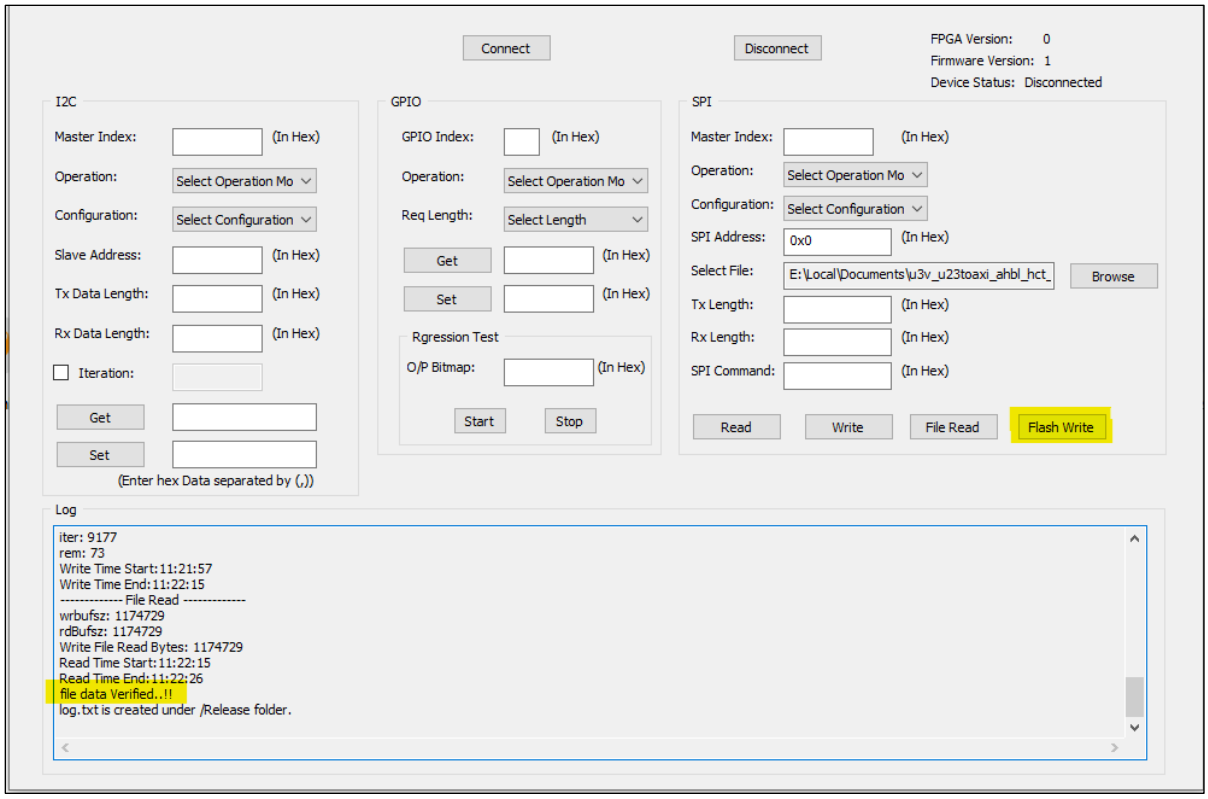

Figure 5.7. File Write Successful

5. The FPGA is automatically re-configured to the newly written *.bit* file. In this demo, the U3V bit is used. Therefore, the same operations described in the Device Connection to eBUS Player section, Video Streaming from Camera Sensor section, and Video Streaming from YUV422 Test Pattern section are used to verify that the reconfiguration is successful.

5.6.4.2. Update U3V GUID in SPI Flash Using USB Test Application

The U3V GUID value represents the ID for the device. It is stored in the offset 0x500000 of the SPI flash. The GUID value is saved as a binary file (*U3V_GUID.bit*) and can be updated using the USB Test Application.

The following lists the steps for updating the GUID:

1. Open the *U3V_GUID.bit* file in *software\utilities\USBTestApp* with any text editor.
2. Change the last 8 digit of the serial number to your new GUID and save it. The following figure shows that the default 8-digits of the serial number is 00000000.

   Note: The serial number must be a hexadecimal value within the range 00000000 to 7FFFFFFF, as this is the only range supported by eBUS. According to the specifications, it is mandatory to keep the leading 4 digits as *2CA1*.

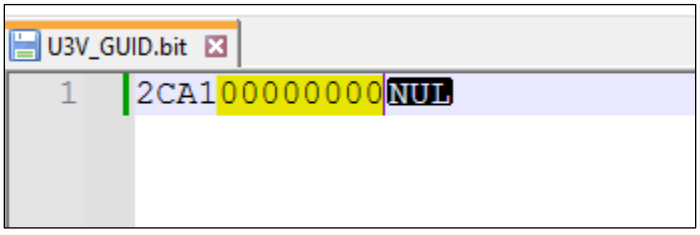

Figure 5.8. U3V GUID Bitstream File Contents

3. Open the USB Test Application. Click on the **Connect** button to connect the device to the SLS Application. On successful connection, you will see the information highlighted in the figure below.

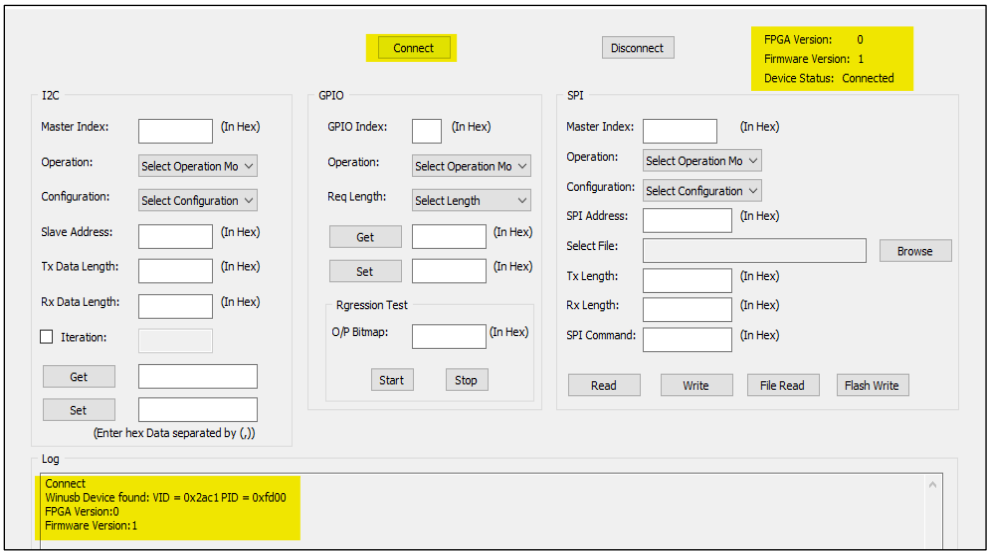

Figure 5.9. Successful Connection

4. Make the following changes highlighted in the figure below:
   - SPI Address = 0x500000
   - Select File = *U3V_GUID.bit*

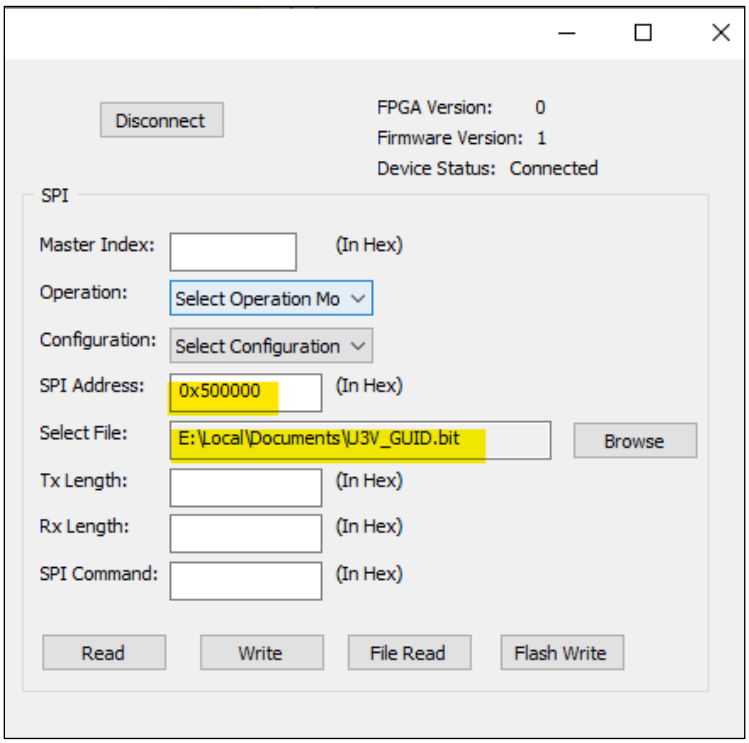

Figure 5.10. U3V GUID Bit Selection

5. Click on the **Flash Write** button to load the file in the flash at the given SPI address. On successful write operation, the log displays *file data verified..!!* message.

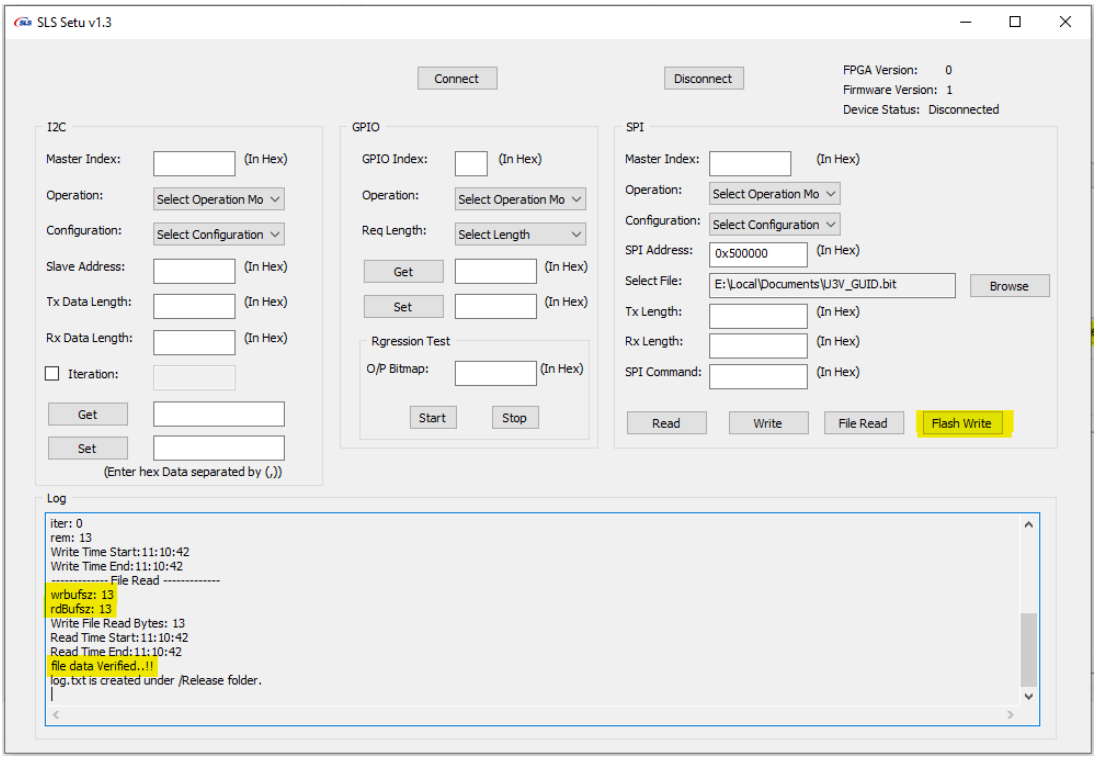

Figure 5.11. File Write Verified

6. Re-connect the device to the eBUS Player. The new GUID must be reflected in the **Device Selection** list.

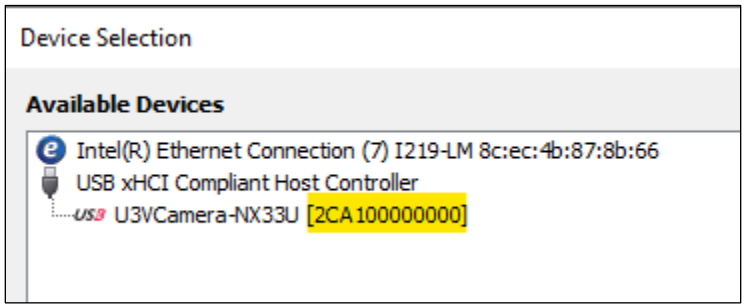

Figure 5.12. Device Selection List is Updated with New GUID

6. Building the Reference Design

This section describes how to build the U3V reference design using the Lattice Propel® and Lattice Radiant® software. The process of building the reference design is needed when you want to generate the binary files from source files especially after you made changes to the reference design.

For more details on these software, refer to *the Lattice Propel SDK Builder User Guide* on the Lattice Propel Design Environment web page, and the *Lattice Radiant Software User Guide* on the Lattice Radiant Software web page.

Note: Building this reference design requires the project source code. Contact Lattice Sales or Technical Support to get access to the complete USB3 Vision design source code.

6.1. Running Propel Builder Design

The U3V reference design uses the hardened USB block inside the LIFCL-33U, which requires a RISC-V microcontroller to handle the USB hard IP enumeration and USB traffic control. This section describes how to generate the RISC-V design, including the peripheral soft IP cores in the Propel Builder software.

6.1.1. Install IPs on Local from File

Before opening the U3V Propel Builder project, there are four soft IP cores that must be installed into the IP on Local section in the Propel Builder software. These are custom IP cores that are created in this reference design that are not available in the IP Server. Therefore, these IP cores must be installed to the IP on Local section. To perform the soft IP installation process, launch the Propel Builder software v2024.2 and follow these steps:

1. Right-click **IP on Local** and select **Install IP from file…**, as shown in the figure below.

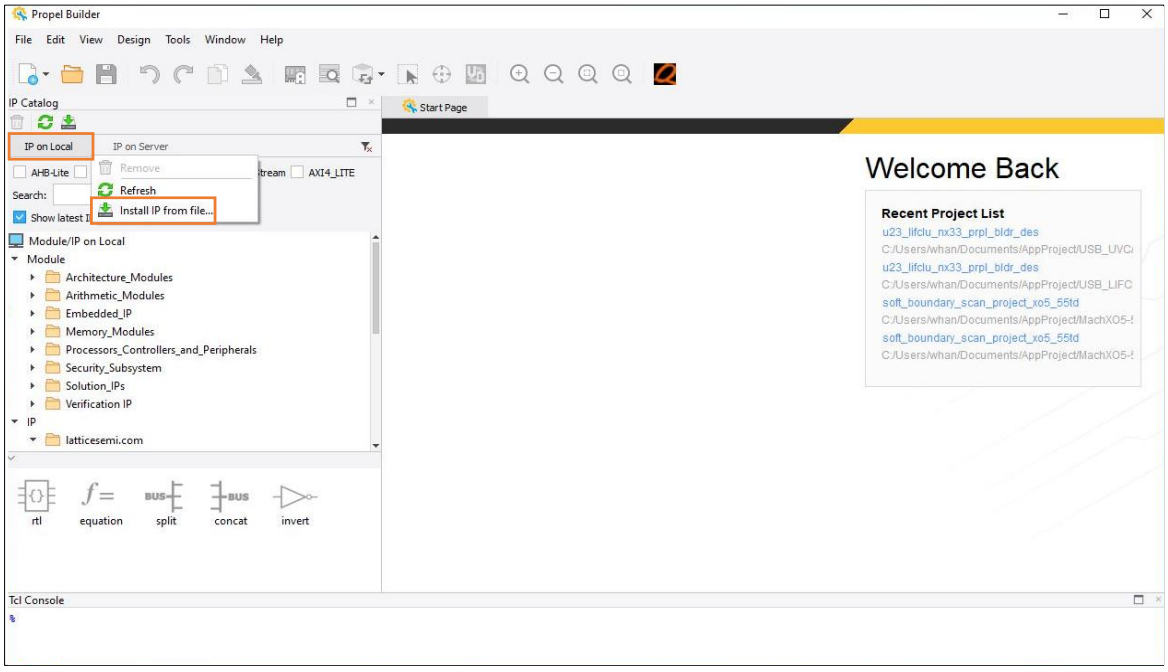

Figure 6.1. Install IP from File

2. In the popped-up file browser, select the following: *<my_workspace>\hardware\ip_packager\latticesemi.com_ahbl_to_axi4lite_bridge_1.0.0.0.ipk*.
3. Click **OK**.
4. In the **IP License Agreement** window, click **Accept** to finish the IP installation as shown in the figure below.

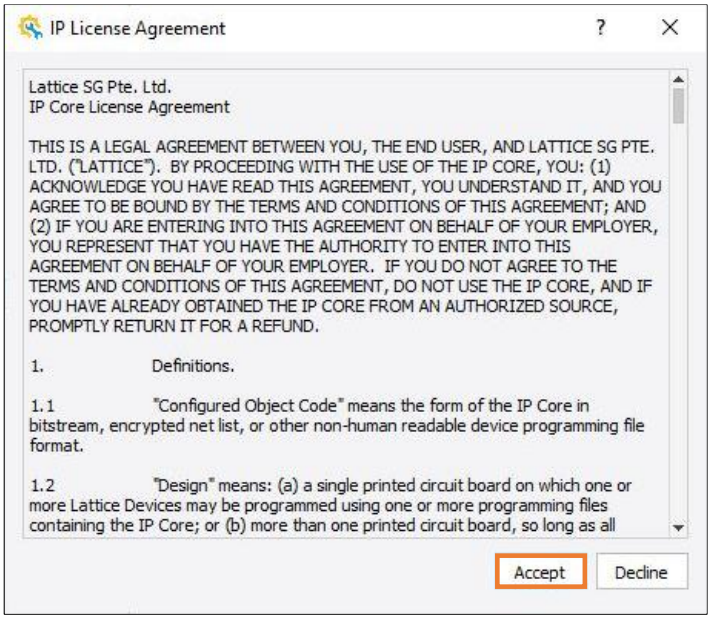

Figure 6.2. Accept IP License Agreement

5. Repeat step 1 to step 4 to install the rest of the IP cores listed below:
   - *<my_workspace>\hardware\ip_packager\latticesemi.com_ahbl_to_lmmi_bridge_1.1.0.0.ipk.*
   - *<my_workspace>\hardware\ip_packager\latticesemi.com_ahbl_to_mem_bridge_1.0.0.0.ipk.*
   - *<my_workspace>\hardware\ip_packager\latticesemi.com_in_ep_buffer_manager_2.2.1.0.ipk.*

   After completing the installation process, these IP cores mentioned above are available in the IP on Local section, as shown in the figure below.

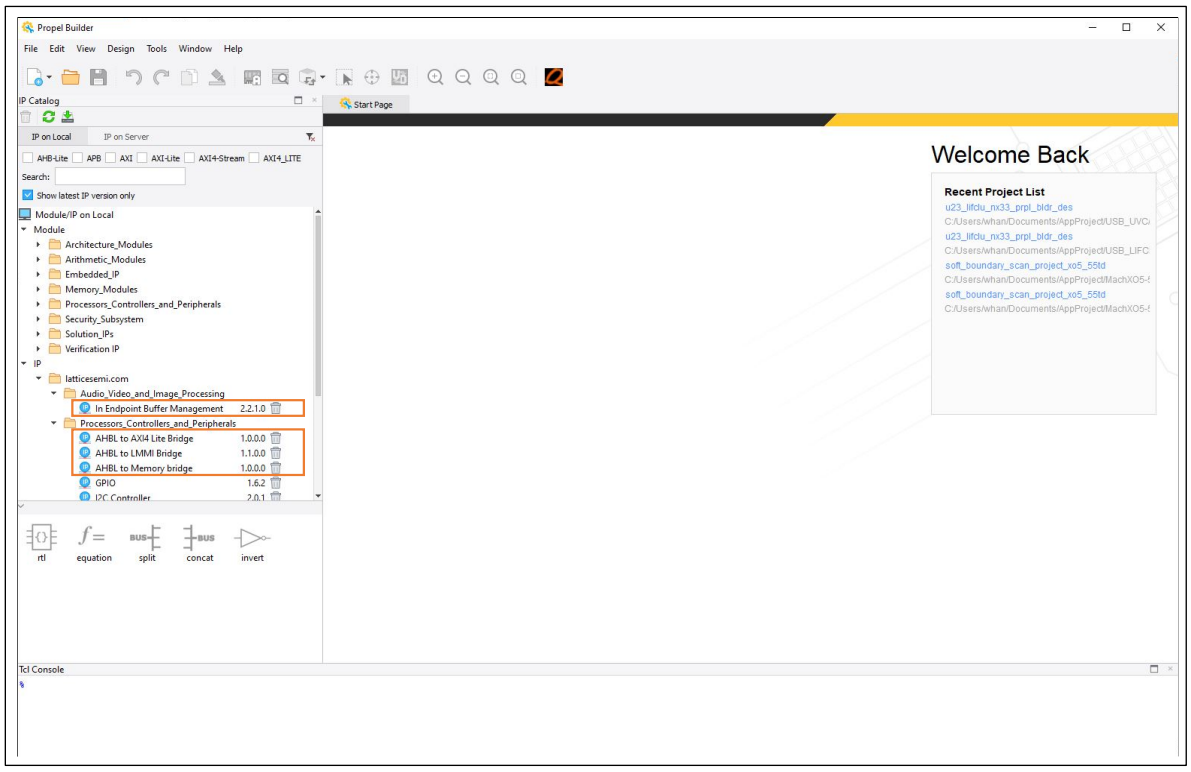

Figure 6.3. IP Cores Installed from Files

6.1.2. Opening the Propel Builder Design

To open the Propel Builder software project in the Lattice Propel Builder, follow these steps:

1. Launch the Propel Builder software tool.
2. Click **File > Open Design** and navigate through the Open sbx window to open the following: *<my_workspace>\hardware\ref_designs\u3v_pbs\u3v_pbs\u3v_pbs.sbx.*

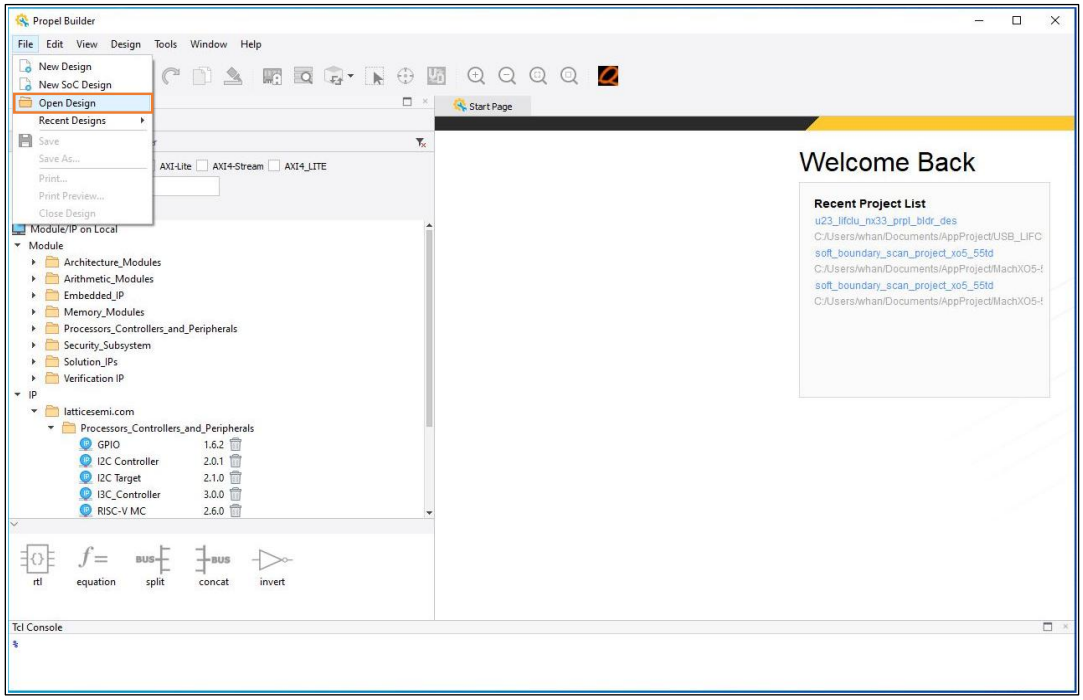

Figure 6.4. Open the Propel Builder Design

6.1.3. Generate the Propel Builder System

To generate the Propel Builder design, click on the **Generate** icon as shown in the figure below. No errors should be observed in the **TCL Console** of the Propel Builder software.

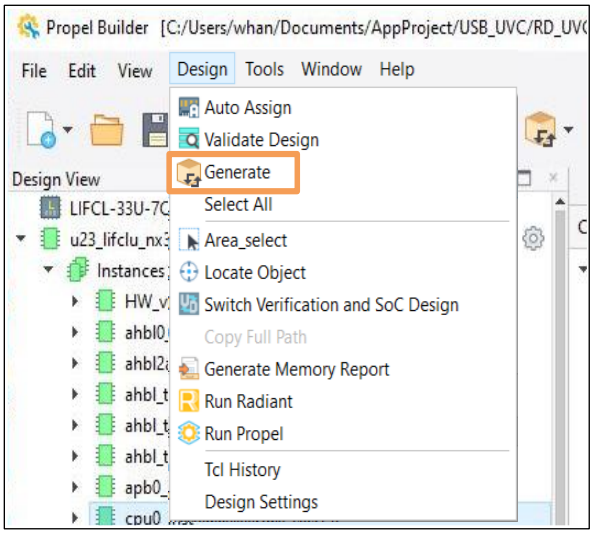

Figure 6.5. Generate the Propel Builder System

6.2. Running the Radiant Software Project

This section describes the procedure to create your FPGA bitstream file using the Lattice Radiant software v2024.2.

1. Launch the **Radiant 2024.2** from the Windows Start menu.
   Note: Do not launch the Radiant software from within the Propel Builder software as this will overwrite the top-level setup of the Radiant software.
2. In the Radiant software, click **File** -> **Open** -> **Project.**
3. Navigate to *<my_workspace>\hardware\ref_designs* and select *u3v_u23toaxi_ahbl_revBbd_des.rdf*. Click **Open.**
4. In the Radiant software, click **Export Files** to start the bitstream compilation.

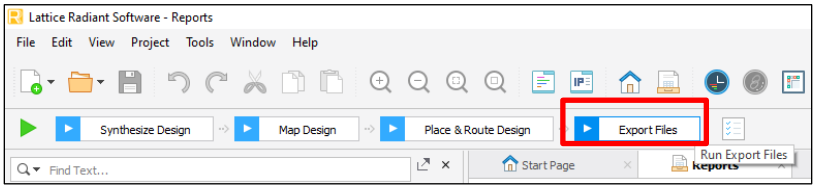

Figure 6.6. Export Files to Start Bitstream Compilation

The compilation may take some time to complete. Observe that the *.bit* file is generated or updated in the project path (*<my_workspace>\hardware\ref_designs \impl_1*) after the compilation is completed successfully.

Note: The generated *.bit* file does not contain the RISC-V firmware. The following sections provide instructions on how to build the Zephyr firmware for RISC-V and to combine the firmware into the *.bit* file.

6.3. Building RISC-V Zephyr

This reference design uses Zephyr RTOS as the RISC-V firmware for USB3 Vision operations. This section describes how to set up the Zephyr build environment, apply the git patches from this design, and compile the Zephyr binary.

6.3.1. Set Up Zephyr Build Environment

For instructions on how to set up the Zephyr build environment, refer to the Getting Started Guide on the Zephyr website.

You may choose the corresponding guide based on the machine you have (Ubuntu or Windows or MacOS).
Note : This design uses Zephyr SDK version 0.17.0.

The following lists important notes or workarounds that you can follow in case you face some issues during the setup:

- Use Windows Command Prompt instead of Power Shell. Run as Administrator when it is instructed, otherwise run as Regular User.
- When installing the dependencies using Python PIP3, you may see the following error:
  ERROR: Could not install packages due to an OSError: [WinError 2]
  Append `--user` to the `pip3` command to resolve it.

  For example:

```
pip3 install -r %HOMEPATH%\zephyrproject\zephyr\scripts\requirements.txt --
user
```

To test whether your environment can successfully build Zephyr, try the following sample:

```
west build -p auto -b reel_board samples\basic\blinky
```

If there are errors: "*error: conflicting types for […]*" when setting up the Zephyr build environment, follow these steps to resolve the error:

1. Open the *.config* file located in C:\%HOMEPATH%\zephyrproject\zephyr\build\zephyr, edit these two variables:

```
CONFIG_PICOLIBC=n
CONFIG_MINIMAL_LIBC=y
```

2. Rerun the build command using the following command:

```
west build -b reel_board samples\basic\blinky
```

Note: "`-p always/auto`" is removed to avoid pristine build overriding the edited *.config* file.

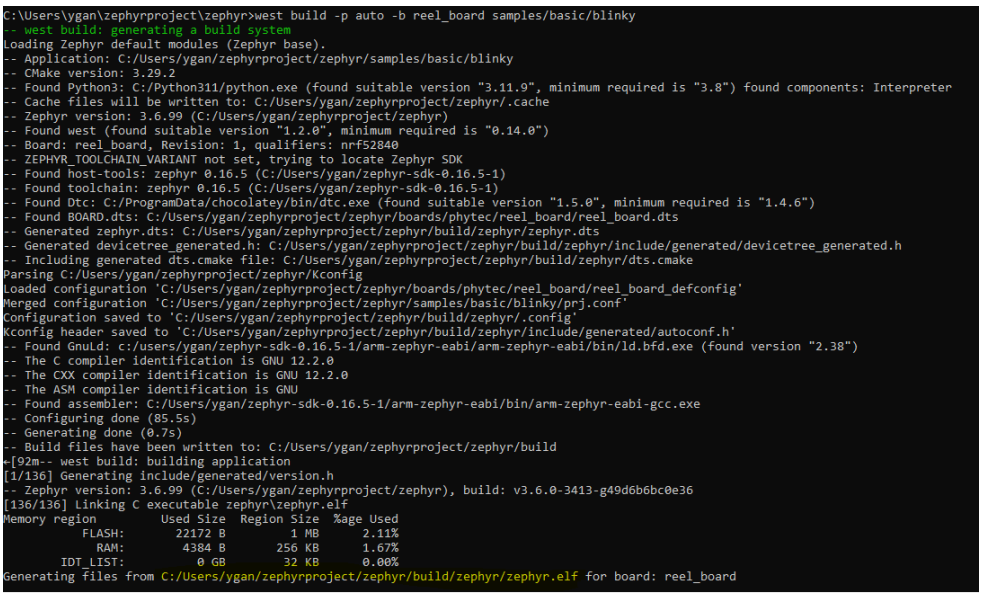

Figure 6.7. Zephyr Build Environment

6.3.2. Apply Patch and Build Zephyr Binary

Before applying the patch, make sure all the steps provided in the section above are followed. The following instructions are based on Windows environment:

1. Open a *cmd.exe* terminal window as a regular user. Go to the *zephyr* folder located in the *zephyrproject* folder.

   Note: This assumes that you set up the Zephyr environment in *HOMEPATH (C:\Users\<username>).*

   ```
   cd %HOMEPATH%\zephyrproject\zephyr
   ```

2. Reset the Zephyr repository to LTS release version 3.7.0.

   ```
   git reset --hard v3.7.0
   ```

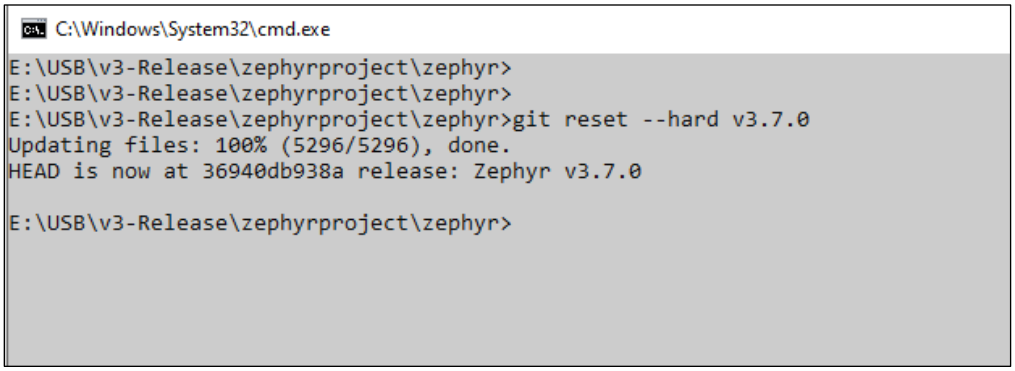

Figure 6.8. Reset the Zephyr Repository

3. Verify that the repository has been successfully reset to the required version. The Zephyr repository must be at tag v3.7.0.

   ```
   git log --oneline -10
   ```

```
E:\USB\v3-Release\zephyrproject\zephyr>git log --oneline -10
36940db938a (HEAD -> main, tag: v3.7.0) release: Zephyr v3.7.0
07ed4b04e31 doc: release: Finalize v3.7.0 release notes and migration guide
04836856423 doc: release: Add v3.7.0 to the list of supported releases
51c39d7d3f4 doc: release-notes-3.7: add ctcc/nrf52840
58f3714d011 boards: ct: ctcc: add M.2 picture and fix pictures quality
257f9dcb091 doc: 3.7 migration guide: Remove empty headings
4b26e2698df doc: 3.7 migration guide: Minor fixes
edb8409e01a doc: 3.7 release_notes: Remove empty headings
48f66a69977 doc: 3.7 release_notes: Remove mention of never introduced options
66feacc781b doc: 3.7 release_notes: Fix minor issues

E:\USB\v3-Release\zephyrproject\zephyr>
```

Figure 6.9. Zephyr Repository is at Tag v3.7.0

4. Apply the patch using the following command:
   ```
   git am --3way --ignore-space-change --keep-cr LSC_Patches_Combined.patch
   ```
   Note: Type in the actual path where the *.patch* file is located. For example: *<my_workspace>\zephyr\LSC_Patches_Combined.patch*.

```
E:\USB\v1-CrossLinkNX33U\zephyrproject\zephyr>git am --3way --ignore-space-change --keep-cr "LSC_Patches_Combined.patch"
Applying: LSC Patches Combined
E:\USB\v1-CrossLinkNX33U\zephyrproject\zephyr>
```

Figure 6.10. Apply Zephyr Patch

5. Run the command to build the Zephyr project. After building the project successfully, the command generates *zephyr.elf* and *zephyr.bin* files as shown below:
   ```
   west build -p always -b clunx_eval_rev_b samples\subsys\usb\testusb_lattice
   ```
   The generated files are in the following path: *%HOMEPATH%\zephyrproject\zephyr\build\zephyr*.

6.3.3. Regenerate Bitstream Using the ECO Editor

After you have successfully built the Zephyr source code to obtain the *zephyr.elf* file, follow these steps to re-generate the FPGA bitstream that contains the Zephyr binary as RISC-V firmware:

1. In the Windows command prompt, change the directory to: *%HOMEPATH%\zephyrproject\zephyr\build\zephyr*. You will find the *zephyr.bin* file in this directory.
2. At the command prompt, type the following command to convert the *zephyr.bin* file to *zephyr.mem* file that is needed by the Radiant software.
   Note: This command assumes that you installed the Propel tool in the path listed below. Otherwise, change the path accordingly.

   C:\lscc\propel\2024.2\sdk\tools\bin\srec_cat.exe zephyr.bin -Binary -byte-swap 4 -DISable Header -Output zephyr.mem -MEM 32
3. To merge the software application file (*.mem*) with the programming file (*.bit*) using the ECO Editor within the Radiant software, follow these steps:.
   a. Select **Tools > ECO Editor** to open the ECO Editor in the Radiant software.
      Note that the ECO Editor tool will only be available after design compilation, which requires you to complete the procedure to create your FPGA bitstream file using the Radiant software.
   b. In the **Memory Initialization** tab, double-click on the Attribute column of row 1.5.0 located in the top red box shown in Figure 6.12.
   c. In the Memory Initialization Settings window, make the selection shown in Figure 6.12.

      Note: The *zephyr.mem* file is in *%HOMEPATH%\zephyrproject\zephyr\build\zephyr* path. Click **OK** when you have completed the memory initialization settings.
   d. Click **File > Save** to save the changes made in the ECO Editor. When prompted, click **Save to proceed**.

e. The **Export Files** icon turns blue. Click on it to rerun the **Export Files** step to generate the new bitstream with Zephyr binary integrated.

Figure 6.11. Rerun Export Files to Generate New Bitstream

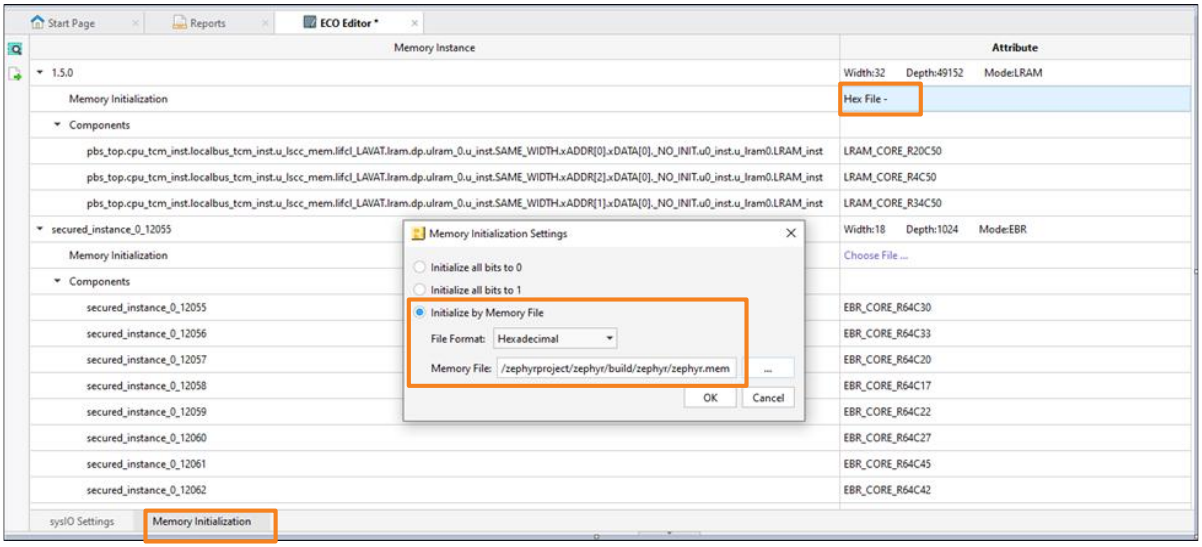

Figure 6.12. ECO Editor Settings

4. Program the updated bitstream file to the evaluation board by following the instructions in the Programming FPGA Bitstream to Flash section.

7. Customizing the Reference Design

This section provides a description for camera support and resolution adjustment. It also provides a way to enable or disable the YUV422 test pattern. For more information on customization or modification of the UVC reference design, contact Lattice Sales.

7.1. Changing the Camera Resolution

The current Lattice B2P and Debayer IP cores are configured to support up to a maximum resolution of 2160P in the Radiant software design. Higher camera resolutions are not supported due to limitations in the Debayer and B2P IP cores. If both IP cores support higher resolutions in the future, you must upgrade or configure the IP cores with the following modifications and recompile the Radiant software design.

7.1.1. Updating Radiant IP Parameters Configuration

7.1.1.1. Updating Lattice Byte-to-Pixel IP Core Parameters

Set the **Word Count** value according to the requirement. To set the **Word Count** field, refer to Figure 3.30.

The word count for 1920 (W) x 1080 (H) resolutions can be calculated based on the equation below:

$$WordCount = \frac{(1920 * 10)}{8} = 2400$$

The value *10* refers to bits per pixel. For example, RAW10 has 10 bits per pixel.

7.1.1.2. Updating Lattice Debayer IP Core Parameters

To update the parameters of the Debayer IP, follow these steps:

1. Under the **DEBAYER** setting, set the **Horizontal size value**, which is the IMAGE_WIDTH value according to the requirement. For details of the configuration, refer to Figure 3.31.
2. Under **Test parameters** setting, set the **Horizontal pixel size** and **Vertical pixel size** according to the desired resolution. For details of the configuration, refer to Figure 3.31.

7.2. Changing to the Other Supported Resolutions

The demo bitstream that is provided only supports 1080P resolution. To change the resolution to other supported resolutions (for example, 720P and 2160P), follow these steps:

1. Modify the Zephyr code.
2. Update the bitstream to the new RISC-V Zephyr firmware.
   For guidance on how to build the RISC-V Zephyr firmware and update the FPGA, refer to the Building RISC-V Zephyr section.

7.2.1. Modifying the Zephyr Code

Modify the macros in the *usbd_u3v.h* file located in *%HOMEPATH%\zephyrproject\zephyr\include\zephyr\usb\classdirectory*, as shown in the figure below.

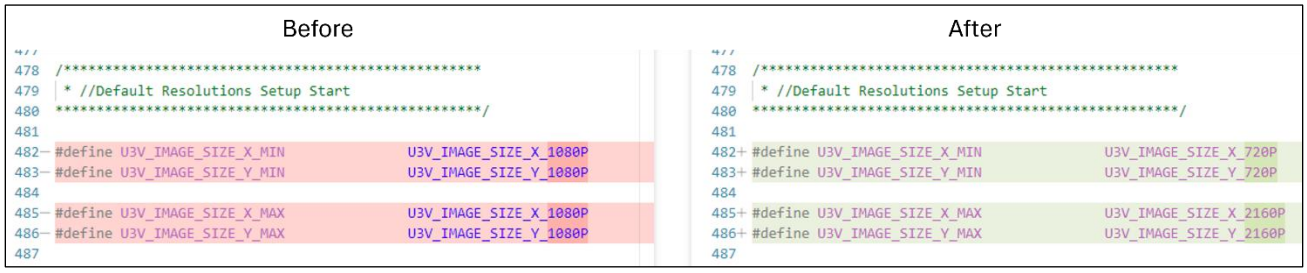

Figure 7.1. Zephyr Code Change to Support Other Resolutions

7.3. Utilizing eBus Player RGB Filtering Feature

The RGB Filtering feature in eBus Player allows precise color adjustments on the displayed image by modifying the Red, Green, and Blue channels independently. This is useful for correcting color balance, enhancing image visibility, or compensating for lighting conditions during testing.

To apply RGB filtering, follow these steps:

1. Ensure the device is connected correctly, and the Play button is clicked.
2. From the main menu, go to **Tools > Image Filtering** to open Image Filtering.
3. In the **RGB Filtering** section, select the **Enabled** checkbox to activate color adjustment controls.
4. Adjust the following color parameters:
   - **Gains:** Multipliers applied to the R, G, and/or B channel values. Increasing gain amplifies the intensity of the respective color channel.
   - **Offsets:** Values added to the R, G, and/or B channel values. Use offsets to shift color levels without changing contrast.
   - **White Balance:** Click **White Balance** to automatically adjust gains so that the last acquired image is balanced to an average gray tone.
     When adjustments are made, the image preview updates immediately in the Display section, providing real-time visual feedback.

Figure 7.2. Example of Corrected Image Colors with the RGB Filtering Feature

5. When you are satisfied with the color correction, close the dialog box to apply the changes.

Appendix A. Resource Utilization

The table below shows the resource utilization of the U3V reference design for LIFCL-33U-9CTG104C using Synplify Pro® of the Lattice Radiant software v2024.2.

Table A.1. Resource Utilization for LIFCL-33U-9CTG104C

Module/Resource Utilization		LUTs	Registers	EBRs	Large RAMs	DSP MULT
RISC-V System	RISC-V RX Micro Controller	6,862	3,449	18	0	6
	TCM System Memory	243	115	0	3	0
	Bus Controller & Converters	920	488	0	0	0
	IEBM	475	531	0	2	0
	Lattice I2C Controller	606	514	0	0	0
	Octal SPI Controller	1,367	967	2	0	0
	GPIOs	207	176	0	0	0
	UART	253	145	0	0	0
Image Signal Processing	MIPI DPHY/CSI	643	383	5	0	0
	Byte-to-Pixel Converter	501	292	3	0	0
	B2P to AXI Stream	178	72	0	0	0
	Image Debayer	5,923	5,240	17	0	6
	AXI Stream to Parallel	7	54	0	0	0
	Color Space Converter	288	396	0	0	12
	YUV422 Test Pattern	733	218	0	0	0
	YUV422 to UVC Bridge	324	193	3	0	0
	IEBM IN FIFO Interface	9	18	0	0	0
	Data Width Convertor	602	964	0	0	0
	USB23 AXI to Memory Bridge	235	175	6	0	0
USB3 Vision	U3V Register Interface	438	442	0	0	0
Misc	Dual Boot	53	177	0	0	0
	Reset Generation and Clock Domain Crossing	172	417	0	0	0
Total		21,039	15,426	54	5	24
Percentage Over the Device (%)		76.1	62.59	84.38	100	37.5

Appendix B. IN Endpoint Buffer Manager

This section provides technical information about the IEBM IP.

B.1. IP Ports and Parameters

This section provides details of the IEBM IP ports and interface.

AHB-Lite Interface

This interface is used to access the registers of the IP. This AHB-Lite subordinate must be connected to the processor. This interface is synchronous to the AHB-Lite clock, ahbl_clk. The following table describes the ports of the AHB-Lite subordinate interface.

Table B.1. AHB-Lite Subordinate Interface

Signal Name	Width	Direction	Description
ahbl_clk	1	Input	AHB-Lite clock.
ahbl_reset_n	1	Input	AHB-Lite reset, active low.
ahbl_haddr_i	8	Input	AHB-Lite address.
ahbl_hburst_i	3	Input	AHB-Lite burst type.
ahbl_hmastlock_i	1	Input	AHB-Lite manager lock.
ahbl_hprot_i	4	Input	AHB-Lite protection control.
ahbl_hready_i	1	Input	AHB-Lite ready.
ahbl_hsel_i	1	Input	AHB-Lite subordinate select.
ahbl_hsize_i	3	Input	AHB-Lite size.
ahbl_htrans_i	2	Input	AHB-Lite transfer type.
ahbl_hwdata_i	32	Input	AHB-Lite write data.
ahbl_hwrite_i	1	Input	AHB-Lite transfer direction. When HIGH, this signal indicates a write transfer. When LOW, it indicates a read transfer.
ahbl_hrdata_o	32	Output	AHB-Lite read data.
ahbl_hreadyout_o	1	Output	AHB-Lite ready out indicates the transfer has finished on the bus.
ahbl_hresp_o	1	Output	Valid lines in the One Frame register.

AXI Interface

This subordinate interface is used to read the data from IP data memory. This interface is synchronous to the AXI clock, axi_clk. The table below describes the ports of the AXI subordinate interface.

Table B.2. AXI Subordinate Interface

Signal Name	Width	Direction	Description
axi_clk	1	Input	AXI clock.
axi_reset_n	1	Input	AXI reset, active low.
axi_to_mem_araddr_i	8[1]	Input	AXI Read address.
axi_to_mem_arburst_i	2	Input	AXI Read burst type.
axi_to_mem_arid_i	8	Input	AXI Read address ID.
axi_to_mem_arlen_i	8	Input	AXI Read burst length.
axi_to_mem_arsize_i	3	Input	AXI Read burst size.
axi_to_mem_arvalid_i	1	Input	AXI Read address valid.
axi_to_mem_arready_o	3	Output	AXI Read address ready.
axi_to_mem_awaddr_i	8[1]	Input	AXI Write address.
axi_to_mem_awburst_i	2	Input	AXI Write burst type.
axi_to_mem_awid_i	8	Input	AXI Write address ID.

Signal Name	Width	Direction	Description
axi_to_mem_awlen_i	8	Input	AXI Write burst length.
axi_to_mem_awsize_i	3	Input	AXI Write burst size.
axi_to_mem_awvalid_i	1	Input	AXI Write address valid.
axi_to_mem_awready_o	1	Output	AXI Write address ready.
axi_to_mem_wdata_i	64[1]	Input	AXI Write data.
axi_to_mem_wlast_i	1	Input	AXI Write last.
axi_to_mem_wstrb_i	8[2]	Input	AXI Write strobe.
axi_to_mem_wvalid_i	1	Input	AXI Write valid.
axi_to_mem_wready_o	1	Output	AXI Write ready.
axi_to_mem_bready_i	1	Input	AXI Write response ready.
axi_to_mem_bid_o	8	Output	AXI Write response ID.
axi_to_mem_bresp_o	2	Output	AXI Write response.
axi_to_mem_bvalid_o	1	Output	AXI Write response valid.
axi_to_mem_rready_i	1	Input	AXI Read ready.
axi_to_mem_rdata_o	64[1]	Output	AXI Read data.
axi_to_mem_rid_o	8	Output	AXI Read ID.
axi_to_mem_rlast_o	1	Output	AXI Read last.
axi_to_mem_rresp_o	2	Output	AXI Read response.
axi_to_mem_rvalid_o	1	Output	AXI Read valid.

Notes:

1. The AXI address width and data width are configurable. Based on the IEBM_AXI4_ADDR_WIDTH_I and IEBM_AXI4_DATA_WIDTH_I parameter selection, this value changes.
2. Depending on the IEBM_AXI4_DATA_WIDTH_I selection, this value also changes. For example, if the selected AXI4 data width is 16, this value is 2. If the selected AXI4 data width is 32, this value is 4. If the selected AXI4 data width is 64, this value is 8.

FIFO Interface

This interface is used to communicate with the external FIFO controller, efm. The external FIFO controller can read or write into the endpoint buffer using this FIFO interface. This interface is synchronous to the external FIFO controller, clock, efm_clk. The table below describes the ports of the FIFO interface. For more information, refer to the FIFO Write Operation section.

Table B.3. FIFO Interface

Signal Name	Width	Direction	Description
ieds_clk	1	Input	External FIFO controller clock.
ieds_fifo_wr_req_i	1	Input	Write request.
ieds_fifo_wr_data_i	32[1]	Input	Write data.
ieds_fifo_wr_byte_en_i	4[2]	Input	Byte enable.
ieds_hand_over_partial_filled_fifo_pl_i	1	Input	Partial transfer request pulse.
ieds_zero_len_xfer_req_pl_i	1	Input	Zero length transfer request pulse.
ieds_fifo_invalid_access_o	1	Output	Invalid FIFO access.
ieds_fifo_wr_full_o	1	Output	FIFO is full.
ieds_fifo_wr_almost_full_o	1	Output	FIFO is almost full.
ieds_fifo_wr_allowed_o	1	Output	Write is allowed.
ieds_none_bufr_pending_o	1	Output	No buffer is pending.
ieds_fifo_being_flushed_o	1	Output	FIFO is being flushed.
ieds_fifo_bytes_can_be_written_o	10	Output	Number of bytes that can be written.

Notes:

1. FIFO data width is configurable. Based on the corresponding selection, this value changes.
2. Depending upon FIFO data width selection, this value changes. For example, if the selected FIFO data width is 16, this value is 2. If the selected FIFO data width is 32, this value is 4. If the selected FIFO data width is 64, this value is 8.

Miscellaneous Ports

This table describes the miscellaneous ports of the IP.

Table B.4. FIFO Interface

Signal Name	Width	Direction	Description
timestamp_i	64	Input	Timestamp value.
valid_lines_per_frame_i	13	Input	This indicates the valid lines in one frame.
first_bufr_in_xfer_i	1	Input	First buffer in transfer indication.
last_bufr_in_xfer_i	3	Input	Last buffer in transfer indication.

User Configurable Parameters

This table describes the list the user configurable parameters for the IP.

Table B.5. User Configurable Parameters

Parameter	Description
IEBM_AXI4_ADDR_WIDTH_I	This parameter defines the address width of the AXI interface.
IEBM_AXI4_DATA_WIDTH_I	This parameter defines the read and write data width of AXI4 interface. Valid values: 32, 64, and 128.
IEBM_AXI4_BYTE_EN_WIDTH_I	This parameter defines the byte enable width of the AXI interface. Calculated based on the IEBM_AXI4_DATA_WIDTH_I/8.
IEBM_AHBL_ADDR_WIDTH_I	This parameter defines the address width of the AHB-Lite interface.
IEBM_AHBL_DATA_WIDTH_I	This parameter defines the read and write data width of AHB-Lite interface. Valid values: 32.
IEBM_HW_MAX_BUFRS_I	This parameter defines the number of total buffers allocated for hardware. Valid values: 1 to 15.
IEBM_HW_SINGLE_BUFR_SIZE_I	This parameter defines the In Endpoint Buffer Manager's single buffer depth in terms of bytes. Valid values: 1 to 4,096.
IEBM_BUFR_WR_DATA_WIDTH_I	This parameter defines the In Endpoint Buffer Manager write data width in terms of bits. Valid values: 32, 64, 128.
IEBM_BUFR_WR_BYTE_EN_WIDTH_I	This parameter defines the In Endpoint Buffer Manager write data width. Valid values: 4, 8, 16.
IEBM_IP_MAJOR_VER_I	This parameter defines the major version for the IP.
IEBM_IP_MINOR_VER_I	This parameter defines the minor version for the IP.

B.2. Register Offset Map

This section provides register offset map for the IN Endpoint Buffer Manager IP.

Table B.6. IEBM Register Offset Map

Register	Offset	Description
IP_VERSION	0x00	IP version register.
SCRATCH	0x04	Scratch register.
INT_EN	0x08	Interrupt enable register.
INT_SRC	0x0C	Interrupt source register.
BUFR_CNFG	0x10	Buffer configuration register
HW_PARAMS_INFO	0x14	Hardware parameters information register.
CTRL	0x18	Control register.
BUFR_TRACKER_INFO	0x1C	Buffer tracker information register.

Register	Offset	Description
BUFR_AVAILABILITY_INFO	0x20	Buffer availability information register.
BUFR_XCHNG_CTRL	0x24	Buffer exchange control register.
SEL_BUFR_INFO_FOR_FW	0x28	Selected buffer information for firmware register.
Reserved	0x2C	Reserved.
TIMESTAMP_W0	0x30	Timestamp lower bytes register.
TIMESTAMP_W1	0x34	Timestamp upper bytes register.
VALID_LINES_IN_FRAME	0x38	Valid Lines in one frame register.

B.3. Register Details

IP_VERSION

This register contains information about the IP version.

Table B.7. IP_VERSION, Offset = 0x00

Bit	Access	Default Value	Description
31:24	RO	Varies upon IP release.	IP major version.
23:16	RO	Varies upon IP release.	IP minor version.
15:0	RO	0x0.	Reserved.

SCRATCH

This register is used for testing and debugging purpose. When being read, the register returns the same value that is written to it.

Table B.8. SCRATCH, Offset = 0x04

Bit	Access	Default Value	Description
31:0	RW	0x0	This field can be used for testing and debugging purposes. When being read, the register returns the same value that is written to it. Typically, this is used to check whether firmware can communicate with the IP properly during initial design development phase.

INT_EN

This register indicates the interrupt enable that controls the reporting of interrupt to the software. It contains direct mapping to the INT_SRC register. When a bit is set, the corresponding interrupt is active.

Table B.9. INT_EN, Offset = 0x08

Bit	Access	Default Value	Description
31:1	RO	0x0	Reserved.
0	RW	0x0	Enable buffer is available to firmware for processing interrupt. 1: Enable buffer available to firmware for processing interrupt. 0: Disable buffer available to firmware for processing interrupt.

INT_SRC

This register indicates the interrupt source, which defines the events that determine the interrupt generation.

Table B.10. INT_SRC, Offset = 0x0C

Bit	Access	Default Value	Description
31:1	RO	0x0	Reserved.
0	RO	0x0	Buffer is available to firmware for processing interrupts. This field indicates that there is at least one buffer available to firmware for further processing. This field remains high as long as no_of_bufrs_pending_to_be_processed_by_fw is not zero.

BUFR_CNFG

This register allows firmware to configure buffer parameters according to different requirements. The firmware must update this register during the initialization, configuration, or reconfiguration stages.

Warning: The firmware must only modify this register when Configured Flag is zero. Failing to meet this requirement may cause unexpected behavior.

Table B.11. BUFR_CNFG, Offset = 0x10

Bit	Access	Default Value	Description
31:22	RO	0x0	Reserved.
21:17	RW	HW_MAX_BUFRS	FW_ALLOCATED_TOTAL_BUFRS This field defines the total number of buffers allocated by the firmware for this endpoint. This field must be less than or equal to HW_MAX_BUFRS, the hardware parameter that describes maximum buffers that can be used for this endpoint. This field must not be zero. The firmware must use maximum buffers allowed by hardware design, that is, upon HW_MAX_BUFRS. The firmware may use smaller number for debugging purpose.
16:0	RW	HW_SINGLE_BUFR_MAX_DEPTH	FW_ALLOCATED_SINGLE_BUFR_DEPTH This field defines single buffer depth in terms of bytes. This field must be less than or equal to HW_SINGLE_BUFR_MAX_DEPTH, the hardware parameter that defines the maximum depth of a single buffer in terms of bytes. This field must not be zero. This field must be in multiples of the endpoint's maximum packet size. This ensures data packet to be sent on the USB bus does not split across multiple buffers. Examples: • HW_SINGLE_BUFR_MAX_DEPTH is 2048 and endpoint's maximum packet size is 512 bytes. In this case, valid values for this field are 512, 1024, 1,536, and 2048. • HW_SINGLE_BUFR_MAX_DEPTH is 512 and endpoint's maximum packet size is 64 bytes. In this case, valid values for this field are 64, 128, 192, 256, 320, 384, 448, and 512. • HW_SINGLE_BUFR_MAX_DEPTH is 4096 and endpoint's maximum packet size is 700 bytes. In this case, valid values for this field are 700, 1400, 2100, 2800, and 3500.

HW_PARAMS_INFO

This register contains information about parameters set in the hardware design.

Table B.12. HW_PARAMS_INFO, Offset = 0x14

Bit	Access	Default Value	Description
31:28	RO	Based on the hardware parameters	HW_MAX_BUFRS This field indicates the hardware parameter value that describes the maximum buffers that can be used for this endpoint.
27	RO	0x0	Reserved.
26:10	RO	Based on the hardware parameters	HW_SINGLE_BUFR_MAX_DEPTH This field indicates the hardware parameter value that defines the maximum depth of a single buffer in terms of bytes. This field must be in multiples of 8.
9:8	RO	0x0	Reserved.
7:0	RO	Based on the hardware parameters	HW_FIFO_DATA_WIDTH This field indicates the hardware parameter value for the data width of FIFO interface in terms of bits.

CTRL

This register is used by the firmware to control bridge IP functionality. The firmware access this register for IP initialization, configuration, or reconfiguration steps.

Table B.13. CTRL, Offset = 0x18

Bit	Access	Default Value	Description
31:2	RO	0x0	Reserved.
1	RW	0x0	Flush Buffers flush_bufrs The firmware sets this field to request the bridge IP to flush corresponding buffers. Upon detecting this field going high, the IP takes the following actions: 1. Resets the IN Endpoint Buffer Manager Buffer Tracker and Firmware Buffer Tracker to 0. 2. Loads the number of buffers available to the IN Endpoint Buffer Manager with a value equals to the FW_ALLOCATED_TOTAL_BUFRS field. 3. Resets the number of buffers pending to be processed by the firmware to 0. 4. Resets other appropriate local variables. 5. Asserts one signal for 10 clock cycles of the FIFO clock to indicate to the IN Endpoint Buffer Manager about the re-initialization process. That block must reinitialize its local variables for a fresh start. 6. The IP resets this field. 7. The firmware can only write 1 to this field. Writing 0 has no effect. This field is automatically cleared by the IP.
0	RW	0x0	Configured. This field indicates that the firmware has configured or reconfigured the bridge IP and the IN endpoint data supplier block can now access the FIFO interface. The firmware must configure or reconfigure the bridge IP every time the corresponding endpoint is configured or re-initialized. The endpoint is configured or re-initialized upon the following requests: SET configuration, SET interface, and Clear feature of Endpoint Halt.

BUFR_TRACKER_INFO

This register is used to track buffer pointers.

Table B.14. BUFR_TRACKER_INFO, Offset = 0x1C

Bit	Access	Default Value	Description
31:4	RO	0x0	Reserved.
3:0	RO	0x0	Firmware Buffer Tracker fw_bufr_tracker. This field indicates the buffer that is either being accessed or is to be accessed by firmware. This field is incremented by one after one buffer is processed by the firmware. This rolls over to 0 once it reaches the value that equals to FW_ALLOCATED_TOTAL_BUFRS. In other words, the value of this field is in a range 0 to (FW_ALLOCATED_TOTAL_BUFRS-1). This is reset to 0 when the flush buffers bit of the control register is set.

BUFR_AVAILABILITY_INFO

This register is used to get information about buffer availability.

Table B.15. BUFR_AVAILABILITY_INFO, Offset = 0x20

Bit	Access	Default Value	Description
31:5	RO	0x0	Reserved.
4:0	RO	0x0	Number of buffers pending to be processed by firmware no_of_bufrs_pending_to_be_processed_by_fw. This field indicates the number of buffers that are yet to be processed by the firmware. Upon initialization or re-initialization, this field is reset to 0. This is incremented by one once one buffer is handed over by the IN Endpoint Buffer Manager to the firmware. This is decremented by one after one buffer is processed by the firmware. For example, the corresponding transfer request block (TRB) is submitted to the USB23 IP.

BUFR_XCHNG_CTRL

This register is used to exchange buffers between the IP and the firmware.

Table B.16. BUFR_XCHNG_CTRL, Offset = 0x24

Bit	Access	Default Value	Description
31:2	RO	0x0	Reserved.
1	RW	0x0	Handover Buffer to IN Endpoint Buffer Manager handover_bufr_to_iebm. The firmware sets this bit when it receives information from the USB23 IP that this buffer has been consumed. Upon detecting this, the number of buffers available to IN Endpoint Buffer Manager increments by 1. This bit is auto-cleared by the IP.
1	RW	0x0	The firmware processed one buffer fw_processed_one_bufr. The firmware sets this bit when it has one buffer processed. In other words, when it has submitted corresponding TRB to the USB IP. Upon detecting this, the number of buffers pending to be processed by firmware field decrements by 1 and the Firmware Buffer Tracker field increments by 1. This bit is auto-cleared by the IP.

SEL_BUFR_INFO_FOR_FW

This register indicates the selected buffer information to the firmware.

Table B.17. SEL_BUFR_INFO_FOR_FW, Offset = 0x28

Bit	Access	Default Value	Description
31:23	RO	0x0	Reserved.
22	RO	0x0	First buffer in transfer indicator. The firmware knows whether the handed over buffer from the external controller is the first buffer in the transfer or not. 0: The selected buffer is not the first buffer in transfer. 1: This is the first buffer in the transfer.
21	RO	0x0	Last buffer in transfer indicator. The firmware knows that the handed over buffer from the external controller is the last buffer in the transfer or not. 0: The selected buffer is not the last buffer in transfer. 1: This is the last buffer in the transfer.
20:4	RO	0x0	Buffer size.
3:0	RO	0x0	Selected buffer number, same value as Firmware Buffer Tracker. The firmware uses this to find offset. Buffr_Offset = MEMORY_BASE_ADDRESS + (Selected buffer number * HW_ALLOCATED_SINGLE_BUFR_DEPTH) For example: The selected buffer number or Firmware Buffer Tracker is 2. HW_ALLOCATED_SINGLE_BUFR_DEPTH is 40,96. MEMORY_BASE_ADDRESS is 0. Buffr_Offset = 8,192.

TIMESTAMP

The use case of this register is to know the current time of the ongoing transfer operation. The timestamp register is a 64-bit register. Therefore, two separate 32-bit registers are used for the timestamp register. This is in terms of clock cycles. The firmware reads this register and calculates the current time based on the frequency of the operating clock.

TIMESTAMP_W0 – Timestamp lower word register

TIMESTAMP_W1 – Timestamp upper word register

For example:

If this register shows the value of 1055275, which is 0x00101A2B in hexadecimal, with TIMESTAMP_W0 = 0x1A2B and TIMESTAMP_W1 = 0x0010, and the operating clock frequency is 60 MHz, given a clock period of 11.11 ns, then the current time id calculated is as follows:

TIMESTAMP * (clock period) = 1055275 * 11.11 ns = 11.724 ms.

TIMESTAMP_W0

Timestamp lower word register, 32 bits.

Table B.18. TIMESTAMP_W0, Offset = 0x30

Bit	Access	Default Value	Description
31:0	RO	0x0	Timestamp_w0. This indicates the lower 32 bits information for the current timestamp value.

TIMESTAMP_W1

Timestamp upper word register, 32 bits.

Table B.19. TIMESTAMP_W0, Offset = 0x34

Bit	Access	Default Value	Description
31:0	RO	0x0	Timestamp_w1. This indicates the upper 32-bits information for the current timestamp value.

VALID_LINES_IN_FRAME

This register provides the information regarding the total number of lines that has been received and written in the buffers.

Table B.20. BUFR_XCHNG_CTRL (Offset = 0x24)

Bit	Access	Default Value	Description
31:13	RO	0x0	Reserved.
12:0	RO	0x0	active_lines_in_frame This indicates the total number of lines that has been received and written into the buffers.

B.4. IN Endpoint Buffer Management Architecture

The buffer management architecture is shown in the figure below.

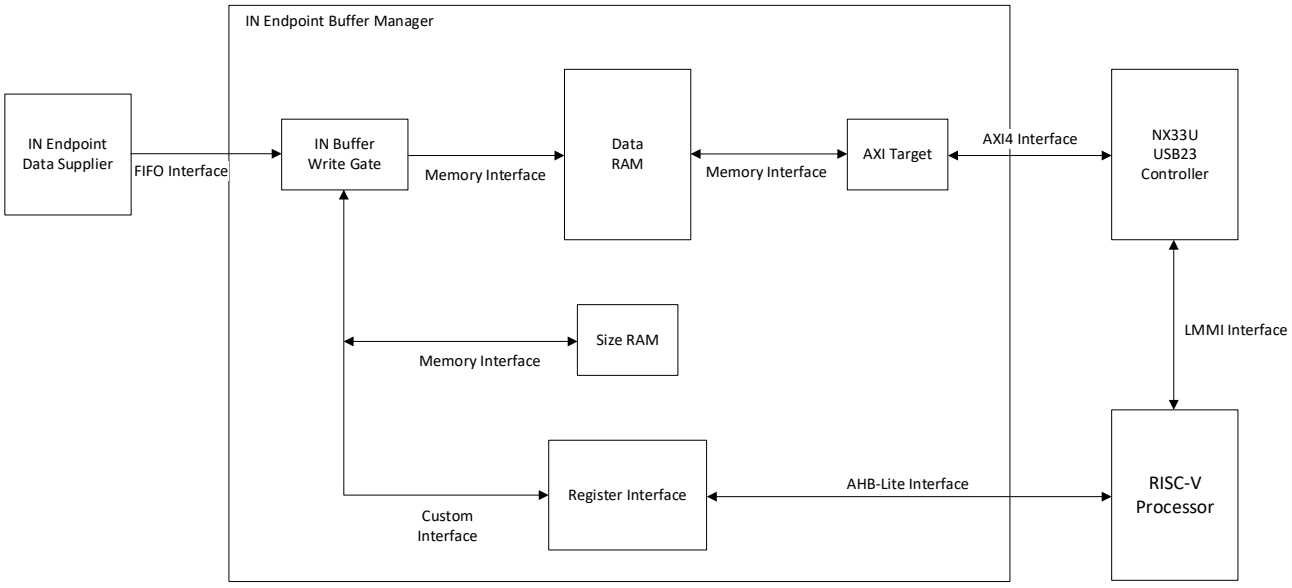

Figure B.1. IN Endpoint Buffer Management Block Diagram

B.5. Buffer Read and Write Management Flow

This section describes how one buffer is written into the data RAM and read from data RAM. Buffer data is written by the IN Endpoint Buffer Manager and it is read back from data RAM by the CrossLinkU-NX USB23 Controller.

Buffer Write Flow

The buffer write operation contains the following steps:

1. The IN FIFO write gate module checks whether the buffer is available or not. When you examine the write allowed signal, ieds_fifo_wr_allowed_o, if it is high, it indicates that the access is granted to the IN Endpoint Buffer Manager.
2. After the access is granted by the IN FIFO write access gate module, the IN Endpoint Buffer Manager stores the data into the IN Data RAM until one buffer becomes full.
3. When one buffer becomes full, corresponding buffer length information is stored in the Size RAM. Also, the corresponding buffer-related information is updated in the registers.
4. Upon completion, the Buffer Available to firmware for processing interrupt is generated.
5. When an interrupt is generated, proceed to step 1.
6. The flow chart below shows how one buffer is written into data RAM.

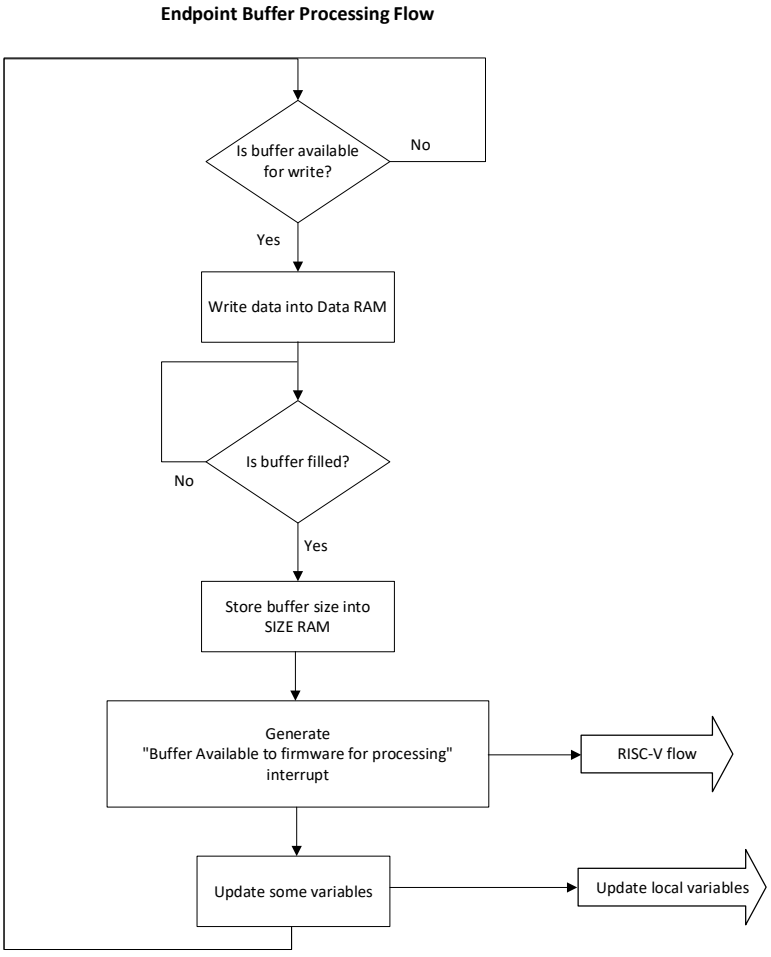

Figure B.2. Buffer Write Operation Flow

Buffer Read Flow

After one buffer is written into the Data RAM, the corresponding interrupt and buffer information is updated in the register through the register interface module. The following lists the steps for the RISC-V firmware to process one buffer.

1. The Buffer Available to firmware for processing interrupt is generated. This interrupt remains high until no_of_bufrs_pending_to_be_processed_by_fw is not zero.
2. The firmware reads the no_of_bufrs_pending_to_be_processed_by_fw field from the Buffer Availability Information register and stores it into the local variable called bufr_avail_for_fw_processing.
3. The firmware checks whether the bufr_avail_for_fw_processing is zero. If it is zero, it means the whole TRB of the buffer has been initiated by the firmware. Go to step 1 and wait for an interrupt.
4. If the bufr_avail_for_fw_processing variable is not zero, that means the firmware must prepare the TRB.
5. For TRB, the firmware needs two fields. The first field is the TRB starting address that is decided upon the buffer number. The second field is the TRB size, which is the buffer size field of the SEL_BUFR_INFO_FOR_FW register. Therefore, read the SEL_BUFR_INFO_FOR_FW register to know the buffer size and buffer number. With this information, the firmware must prepare the TRB for selected buffer.
6. Set the Firmware Processed one buffer bit in the Buffer Exchange Control register.
7. Decrement the local variable bufr_avail_for_fw_processing by 1 and go back to step 3.
8. In parallel to this, after the data packet is sent from the USB23 Controller to the USB HOST, the host sends the ACK in response to the data packet being sent. After that, the USB23 Controller generates an xfer complete event. Based on that event, the firmware must set the handover_bufr_to_ iebm bit.

The flow chart below shows how one buffer is read from the data RAM.

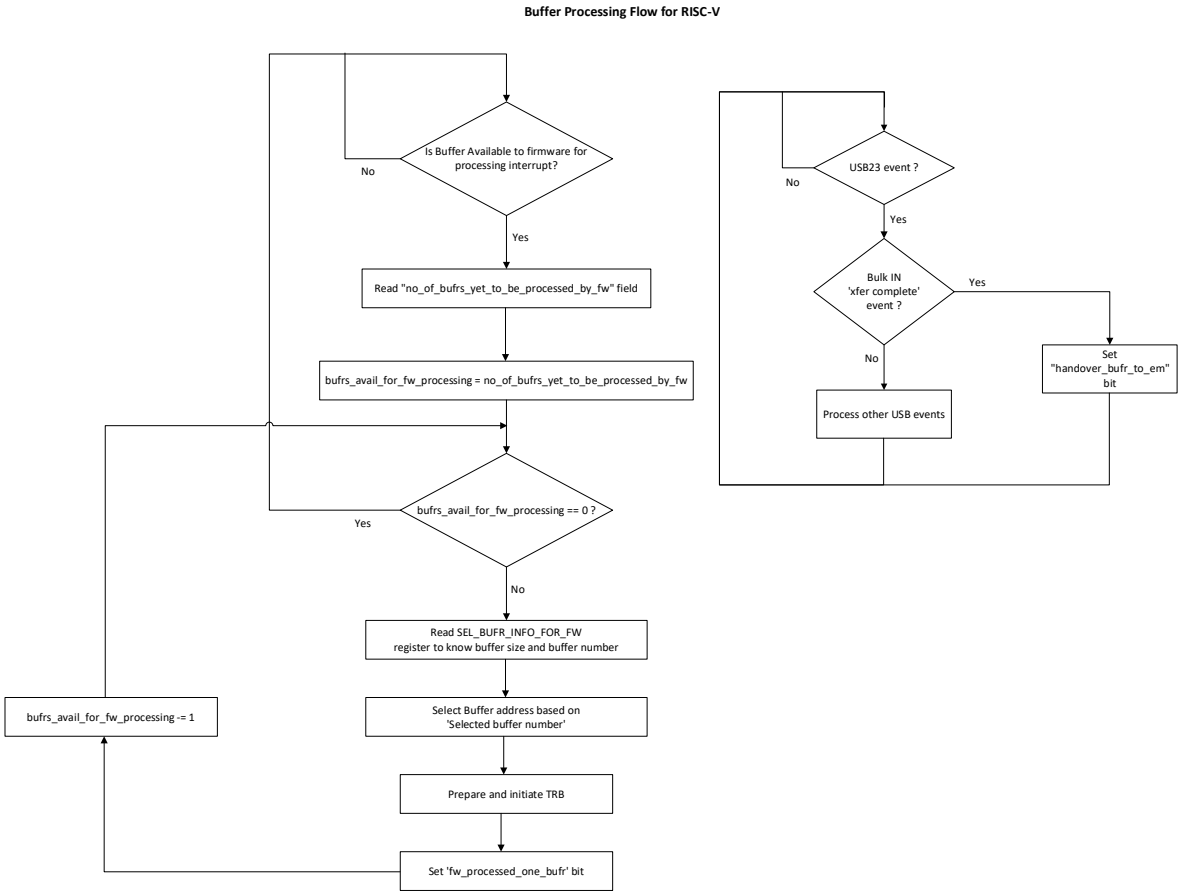

Figure B.3. Buffer Read Operation Flow

B.6. Core Operation

This section describes how firmware can configure and reconfigure buffers.

IP Core Configuration or Reconfiguration

To configure or reconfigure the IP, follow these steps:

1. Disable interrupt generation by writing 0 to the INT_EN register.
2. Reset the Configured bit in the CTRL register.
3. Load the Firmware Buffer Configuration register with appropriate values.
4. Set the Flush buffers bit in the CTRL register.
5. Poll until the Flush buffers bit in the CTRL register is reset by the IP.
6. If the firmware plans to use the interrupt mode, enable interrupt generation by writing appropriate value to the INT_EN register.
7. Set the Configured bit in the CTRL register.

FIFO Write Operation

This section describes how IN Endpoint Data Source (IEDS) performs FIFO write operation. The following table describes ports that are used during the FIFO write operation.

Table B.21. Input Ports Related to FIFO Write Operation

Port	Direction	Description
Ieds_fifo_wr_req_i	Input	Write Request Assert this signal to request a write operation. Do not assert this signal if any of the following conditions is true: • FIFO is being flushed – ieds_fifo_being_flushed_o is high. • FIFO is full – ieds_fifo_wr_full_o is high. • FIFO write operation is not allowed – ieds_fifo_wr_allowed_o is low.
ieds_fifo_wr_data_i	Input	Write Data It holds the data to be written in the FIFO when the write request is asserted.
ieds_fifo_wr_byte_en_i	Input	Write Byte Enable It holds the byte enable bits when the write request is asserted. Make sure the following conditions are met: • This is intended to be used only during partial transfer. • In all other cases, all bits in this signal must be kept asserted when write request is high. **Note:** During partial transfer, if the number of bytes to be written is less than the data width of the bus, all bytes must be adjacent and start from the least significant position only. For example, you want to write 0xABCD as the last transfer on the 32-bit bus. ieds_fifo_wr_data_i is 0x0000ABCD and ieds_fifo_wr_byte_en_i is 0x3. ieds_fifo_wr_byte_en_i must not be other than 0x3.
ieds_hand_over_partial_filled_fifo_pl_i	Input	Hand Over Partially Filled FIFO Pulse Assert the signal for one clock cycle to hand over the partially filled buffer. It means, whenever the last data has been written into FIFO and the ieds_fifo_wr_full_o signal is low on the next clock cycle of it, this signal must be asserted for one clock cycle. Make sure the following conditions are met: • This must be an active high pulse of a single clock cycle. • After you assert this signal, the buffer is handed over to the firmware for processing and ieds_fifo_wr_allowed_o goes low after one clock cycle. Wait for the ieds_fifo_wr_allowed_o signal to go high again

Port	Direction	Description
		before initiating the next write operation.
ieds_zero_len_xfer_req_pl_i	Input	Zero Length Packet Transfer Pulse Assert the signal for one clock cycle to transfer zero length packet. In a special case, when the last data has been written into the FIFO and ieds_fifo_wr_full_o signal becomes high at the next clock cycle, it is mandatory to assert this signal for one clock cycle when ieds_fifo_wr_allowed_o becomes high again.
ieds_fifo_wr_full_o	Output	Full When asserted, the IN FIFO is considered full. **Note:** Do not perform write operation when the FIFO is full.
ieds_fifo_wr_almost_full_o	Output	Almost Full Asserted when there is space for only single write operation. It is used as an early indication of the full signal.
ieds_fifo_wr_allowed_o	Output	Write Allowed Asserted when FIFO write operation is allowed. This signal would go low for one clock cycle after either ieds_fifo_wr_full_o, or ieds_hand_over_partial_filled_fifo_pl_i, or ieds_zero_len_xfer_req_pl_i goes high. This signal also goes low when ieds_zero_len_xfer_req_pl_i goes high. Make sure the following conditions are simultaneously met: • The Configured bit of the CTRL register of this IP is set by the firmware. • At least one buffer is available with the FIFO controller, In Endpoint Data Source, to write data. Example: Assume four buffers are available to a FIFO controller, the IN Endpoint Data Source. After the first buffer is written by IEDS and the FIFO full signal goes high, this signal goes low. It takes around three to four efm_clk clock cycles to go high again. During this period, the first buffer is handed over to the firmware and the buffer pointer switches to the next buffer. Same as the first buffer, after you write the second buffer and FIFO full signal goes high, this signal goes low again for three to four efm_clk clock cycles. Therefore, if all buffers are filled in this manner, this signal goes low and would go high again only when at-least one buffer is sent by the USB device against the IN request from the USB host. **Notes:** • When FIFO is full and this signal is low, it takes around three to four clock cycles of the FIFO clock to go HIGH again if at-least one buffer is available to the IEDS. • Do not perform write operation when this signal is low.
ieds_fifo_invalid_access_o	Output	Invalid Access Pulse Asserted when IN FIFO is being accessed improperly. The following lists the conditions considered as invalid access to the FIFO: • Data is being written into the FIFO while write operation is not allowed. That is, ieds_fifo_wr_req_i is high while ieds_fifo_wr_allowed_o is low. • Data is being written into FIFO when FIFO is full. That is, ieds_fifo_wr_req_i is high while ieds_fifo_wr_full_o is high.
ieds_none_bufr_pending_o	Output	None Buffer Pending Asserted when there is no buffer pending for processing. This can be used to know whether there is any buffer pending to be transmitted.
ieds_fifo_being_flushed_o	Output	FIFO Being Flushed This signal is asserted during the IP initialization process. This indicates the FIFO is being flushed. All buffers are available to the external FIFO controller once FIFO is flushed.

Port	Direction	Description
		Note: Do not perform a write operation when this signal is high.
ieds_fifo_bytes_can_be_written_o	Output	Number of Bytes that Can Be Written This indicates the number of bytes that can be written in the FIFO. An IEDS uses this information to know how many bytes can be written until the FIFO becomes full. **Note:** This is considered valid only when ieds_fifo_wr_allowed_o is high.

FIFO Write Examples

This section contains examples showing different cases for FIFO write operations, such as full transfer and partial transfer. For these examples, the following configuration is used:

- USB 2.0 speed
- Endpoint's maximum packet size: 512 bytes
- Number of IEBM buffers: 4
- Write data width: 32 bits

Some typical FIFO write operation examples are shown in Figure B.4 to Figure B.9.

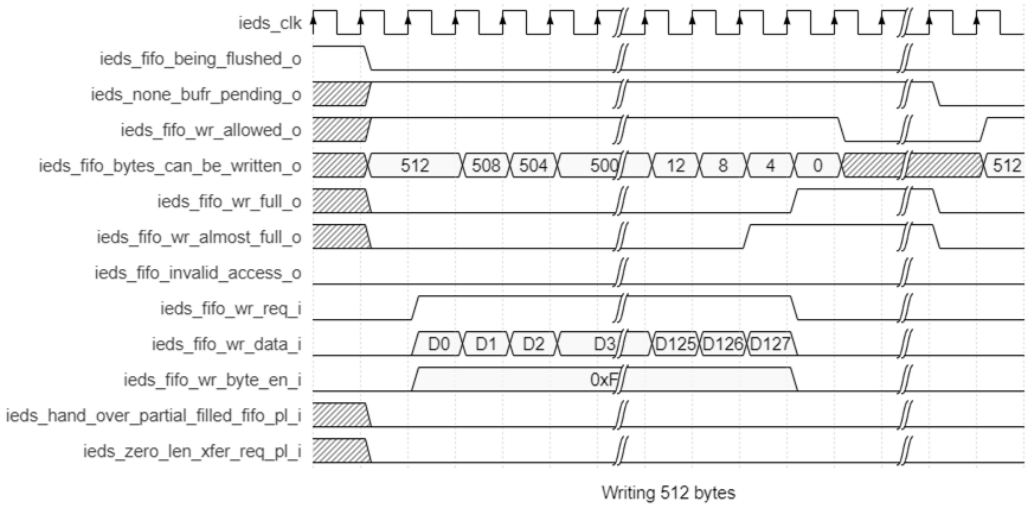

Figure B.4. FIFO Operation Example – Writing 512 Bytes

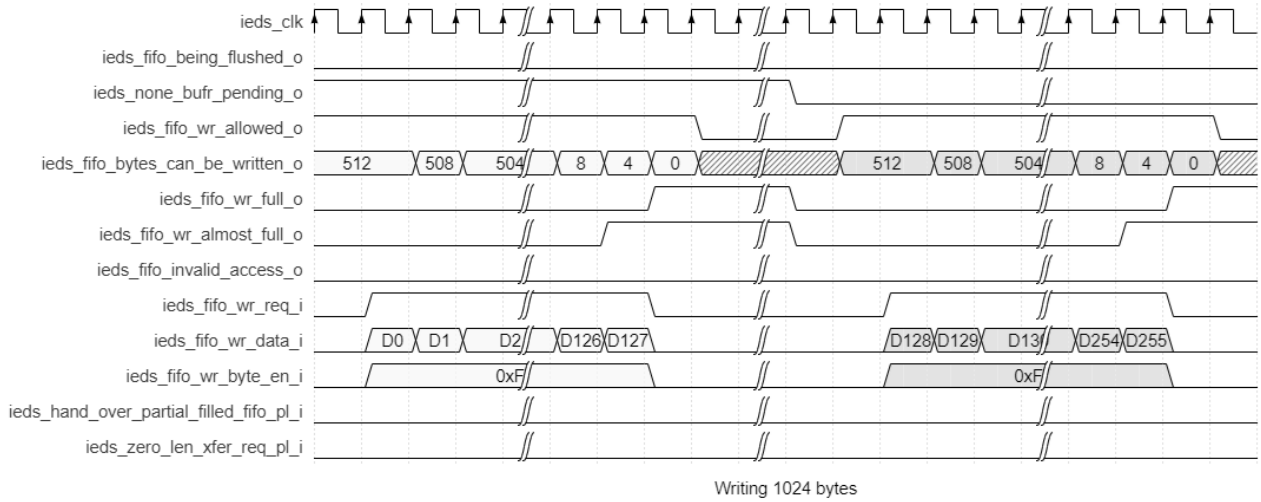

Figure B.5. FIFO Operation Example – Writing 1,024 Bytes

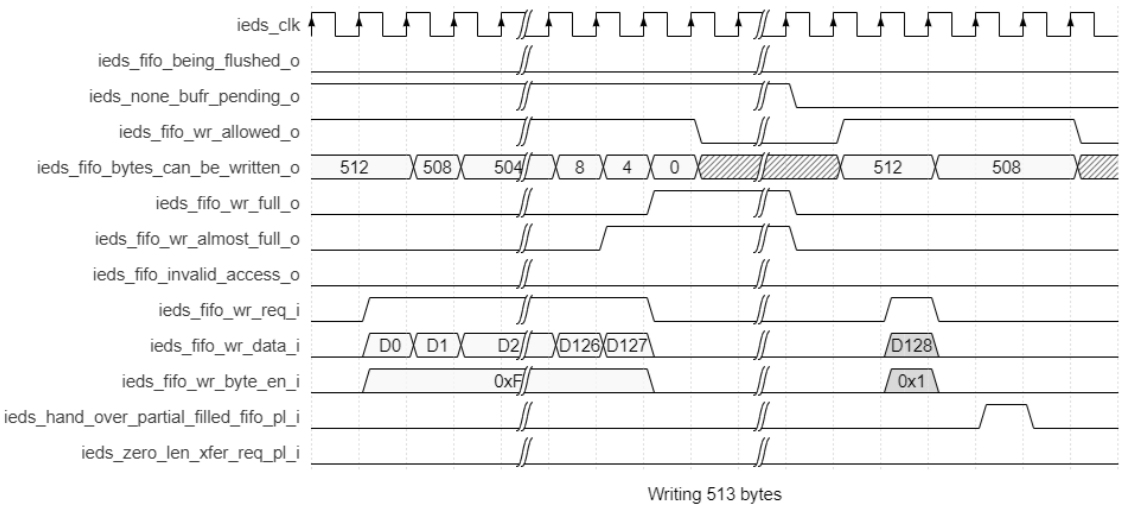

Figure B.6. FIFO Operation Example – Writing 513 Bytes

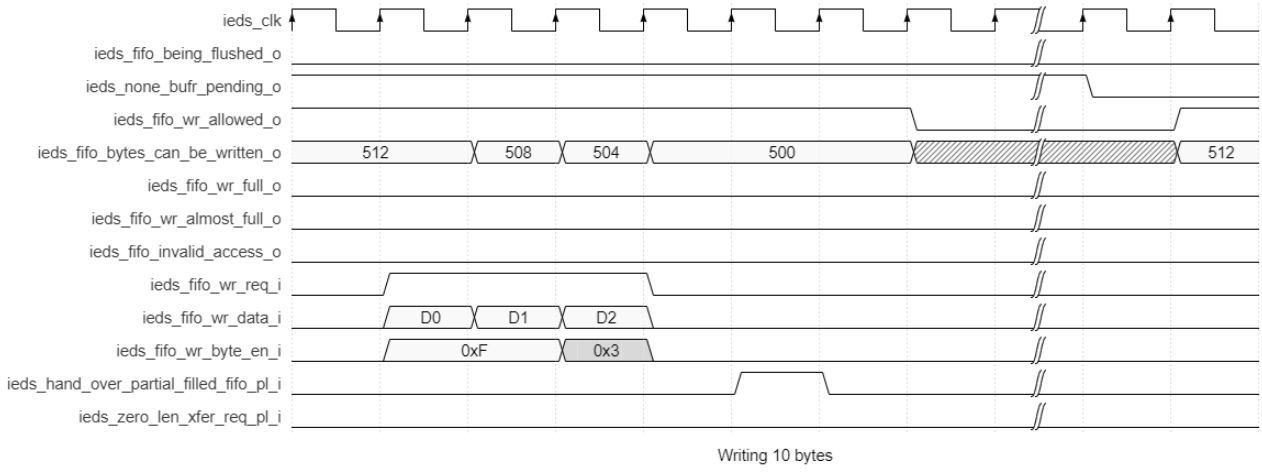

Figure B.7. FIFO Operation Example – Writing 10 Bytes

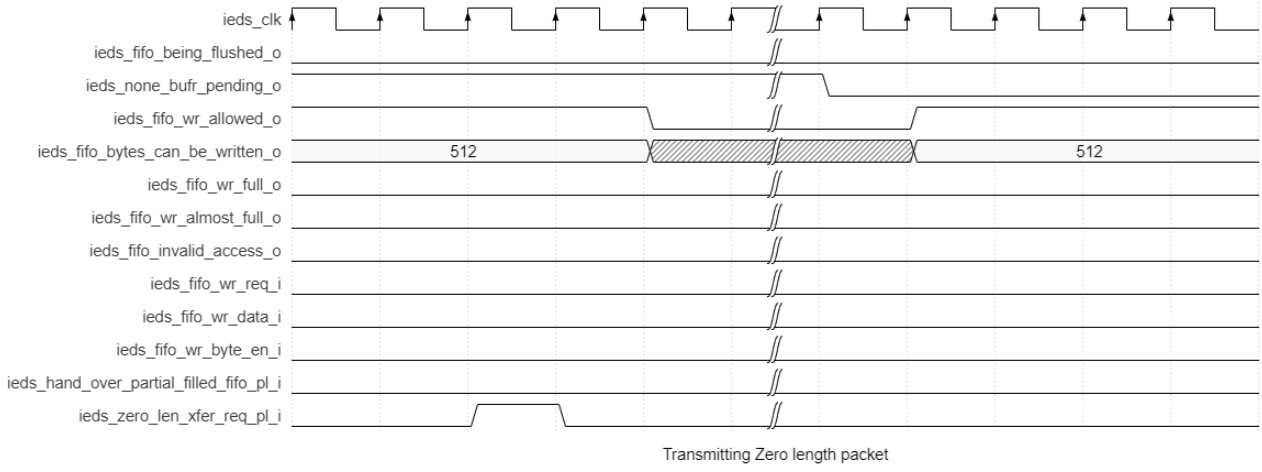

Figure B.8. FIFO Operation Example – Zero Length Packet Request

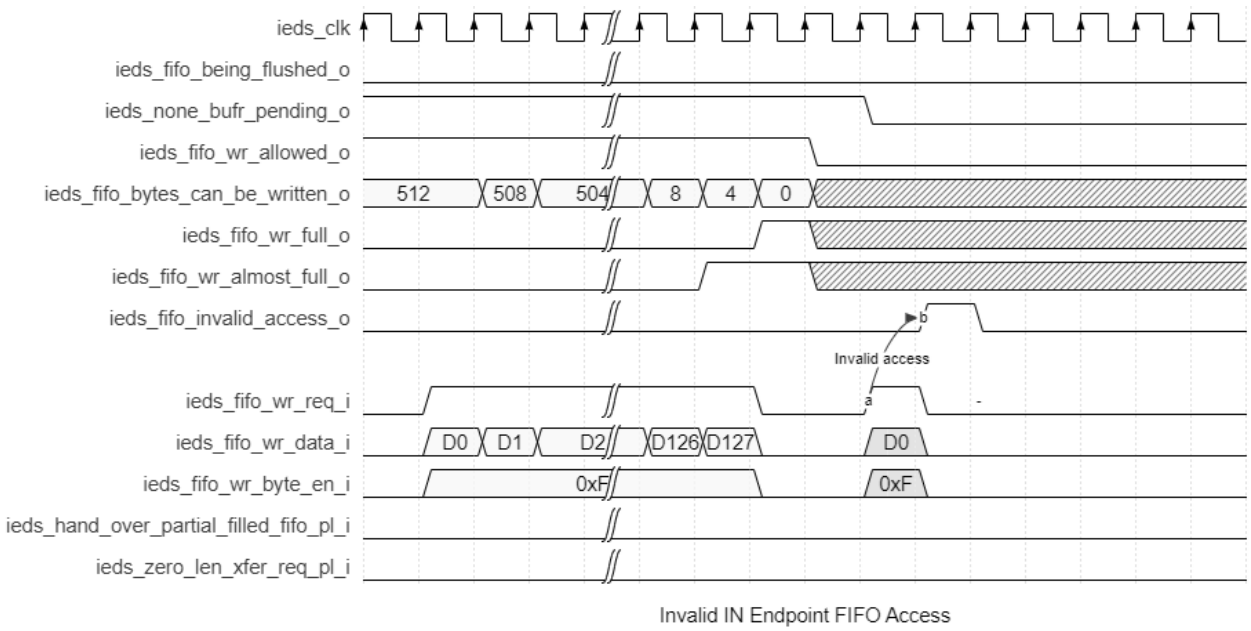

Figure B.9. FIFO Operation Example – Invalid IN FIFO Access

References

- Lattice Radiant Software User Guide
- Lattice Propel SDK User Guide (FPGA-UG-02218)
- Lattice Propel Builder User Guide (FPGA-UG-02219)
- RISC-V MC CPU IP User Guide (FPGA-IPUG-02267)
- AHB-Lite Interconnect Module User Guide (FPGA-IPUG-02051)
- AHB-Lite to APB Bridge Module User Guide (FPGA-IPUG-02053)
- System Memory Module User Guide (FPGA-IPUG-02073)
- GPIO IP User Guide (FPGA-IPUG-02076)
- SPI Controller IP User Guide (FPGA-IPUG-02069)
- SPI Target IP User Guide (FPGA-IPUG-02070)
- I2C Controller IP User Guide (FPGA-IPUG-02071)
- I2C Target IP User Guide (FPGA-IPUG-02072)
- UART 16550 IP User Guide (FPGA-IPUG-02100)
- USB 2.0/3.2 IP User Guide (FPGA-IPUG-02237)
- CrossLinkU-NX web page
- Lattice Radiant Software web page
- Lattice Propel Design Environment web page
- Lattice Insights web page 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, February 2026

Section	Change Summary
Introduction	Updated the term *IEMB* to *IEBM* in Table 1.1. Summary of the Reference Design.
Building the Reference Design	Updated the Set Up Zephyr Build Environment section.
Customizing the Reference Design	Added the Utilizing eBus Player RGB Filtering Feature section.

Revision 1.0, July 2025

Section	Change Summary
All	Initial release.

www.latticesemi.com