CrossLink-NX MIPI-to-Parallel and Parallel-to-MIPI User Guide

Reference Design

FPGA-RD-02318-1.0

July 2025

Disclaimers

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

Inclusive Language

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

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AHB-Lite	Advanced High-Performance Bus Lite
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CSI-2	Camera Serial Interface 2
DE	Data Enable
DPI	Display Pixel Interface
DSI	Display Serial Interface
DUT	Device Under Test
EBR	Embedded Block RAM
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
HS	High Speed
HSYNC	Horizontal Sync
I/O	Input/Output
IP	Intellectual Property
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
LMMI	Lattice Memory Mapped Interface
LP	Low Power
LSB	Least Significant Bit
LUT	Look Up Table
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor
LVDS	Low Voltage Differential Signaling
MC	Main Controller
MIPI	Mobile Industry Processor Interface
MSB	Most Significant Bit
OpenOCD	Open On-Chip Debugger
PLL	Phase Locked Loop
PRBS	Pseudo Random Binary Sequencer
RGB	Red, Green, Blue
RISC-V	Reduced Instruction Set Computer Five
Rx	Receiver
SDK	Software Development Kit
SoC	System On Chip
STA	Static Timing Analysis
Tx	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
VSYNC	Vertical Sync

1. Introduction

The Mobile Industry Processor Interface (MIPI®) D-PHY was developed primarily to support camera and display interconnections in mobile devices. MIPI D-PHY has become the primary high-speed PHY solution in the industry for these applications in smartphones. The MIPI D-PHY interface is typically used alongside the MIPI Camera Serial Interface-2 (CSI-2) and MIPI Display Serial Interface (DSI) protocol specifications. The interface meets the demanding requirements of low power consumption, minimal noise generation, and high noise immunity demanded by mobile phone designs.

MIPI D-PHY is a practical PHY solution for typical camera and display applications, designed to replace traditional parallel buses based on LVCMOS or LVDS. However, many processors, displays, and cameras still use RGB, CMOS, or MIPI Display Pixel Interface (DPI) as their interface.

The Lattice Semiconductor MIPI-to-Parallel and Parallel-to-MIPI reference design allows quick interfacing between a processor with a MIPI DSI and a display with an RGB interface, or between a camera with a MIPI CSI-2 interface and a processor with a parallel interface. This reference design supports conversion using Lattice CrossLink-NX™ devices and is ideal for applications such as wearables, tablets, human machine interfaces, medical equipment, and more.

1.1. Quick Facts

Download the reference design files from the following Lattice reference design web pages:

- MIPI DSI/CSI-2 to Parallel Bridge Reference Design web page
- Parallel to MIPI CSI-2/DSI Display Interface Bridge Reference Design web page

Table 1.1. Reference Design Summary

General	Target Devices	LIFCL-40
	Source Code Format	Verilog, System Verilog
Simulation	Functional Simulation	Performed
	Timing Simulation	Performed
	Test Bench	Available
	Test Bench Format	System Verilog
Software Requirements	Software Tool and Version	Lattice Radiant™ software version 2025.1 Lattice Propel™ software version 2025.1 Lattice Propel Builder version 2025.1
	IP Version	Pixel-to-Byte IP v1.9.1 Byte-to-Pixel IP v1.9.1 CSI-2/DSI D-PHY Receiver IP v2.0.0 CSI-2/DSI D-PHY Transmitter IP v2.3.0 RISC-V MC 2.8.0
Hardware Requirements	Board	Lattice CrossLink-NX Evaluation Board

1.2. Features

Key features of the MIPI-to-Parallel and Parallel-to-MIPI reference design include:

- Compliant with MIPI D-PHY v1.2 and MIPI CSI-2 v1.2 specifications
- Supports MIPI D-PHY interfacing from 160 Mbps up to 1,500 Mbps for soft D-PHY
- Supports 1, 2, or 4 data lanes and one clock lane
- Supports continuous and non-continuous MIPI D-PHY clock modes
- Supports bit rate dynamic reconfiguration
- Supports common MIPI CSI-2 compatible video formats such as RGB888, RAW8, RAW10, RAW12, RAW14, RAW16, YUV_420_8, YUV_420_10, YUV_422_8, and YUV_422_10.

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 (asserted when value is logic 0)
- *_i* are input signals
- *_o* are output signals

1.3.3. Data Ordering and Data Type

The highest bit in a data bus is referred to as the most significant bit (MSB). In MIPI D-PHY, 8-bit parallel data is serialized into a 1-bit data stream on each data lane, where bit 0 is transmitted first.
Table 1.2 lists pixel data order going into Parallel2MIPI module or coming out from MIPI2Parallel module.

Table 1.2. Pixel Data Order

Data Type	Format
RGB	{Red[MSB:0], Green[MSB:0], Blue[MSB:0]}
RAW	RAW[MSB:0]
YUV	YUV[MSB:0]

2. Directory Structure and Files

Figure 2.1 shows the directory structure.

```
.
└── RD02318_soc_w_m2p_p2m/
    ├── docs                                  contains reference design user guide
    ├── fpga_lifcl/
    │   ├── precompiled_file                  contains precompiled bitstream file, memory file
    │   ├── propelsdk/
    │   │   └── propelsdk_soc_w_m2p_p2m_2/
    │   │       ├── src                       contains Propel SDK software source code
    │   │       └── Debug                     Propel SDK generated file
    │   ├── radiant/
    │   │   ├── propelbld_soc_w_m2p_p2m/
    │   │   │   ├── lib                       contains Propel Builder IP and module files -gpio, uart, osc, etc.
    │   │   │   └── propelbld_soc_w_m2p_p2m.sbx  Propel Builder SoC design file
    │   │   ├── src/
    │   │   │   ├── constraints               contains Radiant project constraint files (.sdc, .pdc)
    │   │   │   ├── ip                        contains Radiant project IP files -b2p, p2b, dphy_tx, etc.
    │   │   │   ├── rtl                       contains reference design rtl files -synthesis_directives, parallel2mipi, mipi2parallel, etc.
    │   │   │   └── testbench                 contains reference design testbench files
    │   │   ├── sge                           contains Lattice Propel Software Generation Engine files
    │   │   ├── verification                  contains Lattice Propel verification files
    │   │   └── propelbld_soc_w_m2p_p2m.rdf   Lattice Radiant project file
    │   └── sim/
    │       └── questa                        contains run_sim.do file for simulation purpose
    └── misc                                  contains Excel Calculator Helper tools
```

Figure 2.1. Directory Structure

3. Functional Description

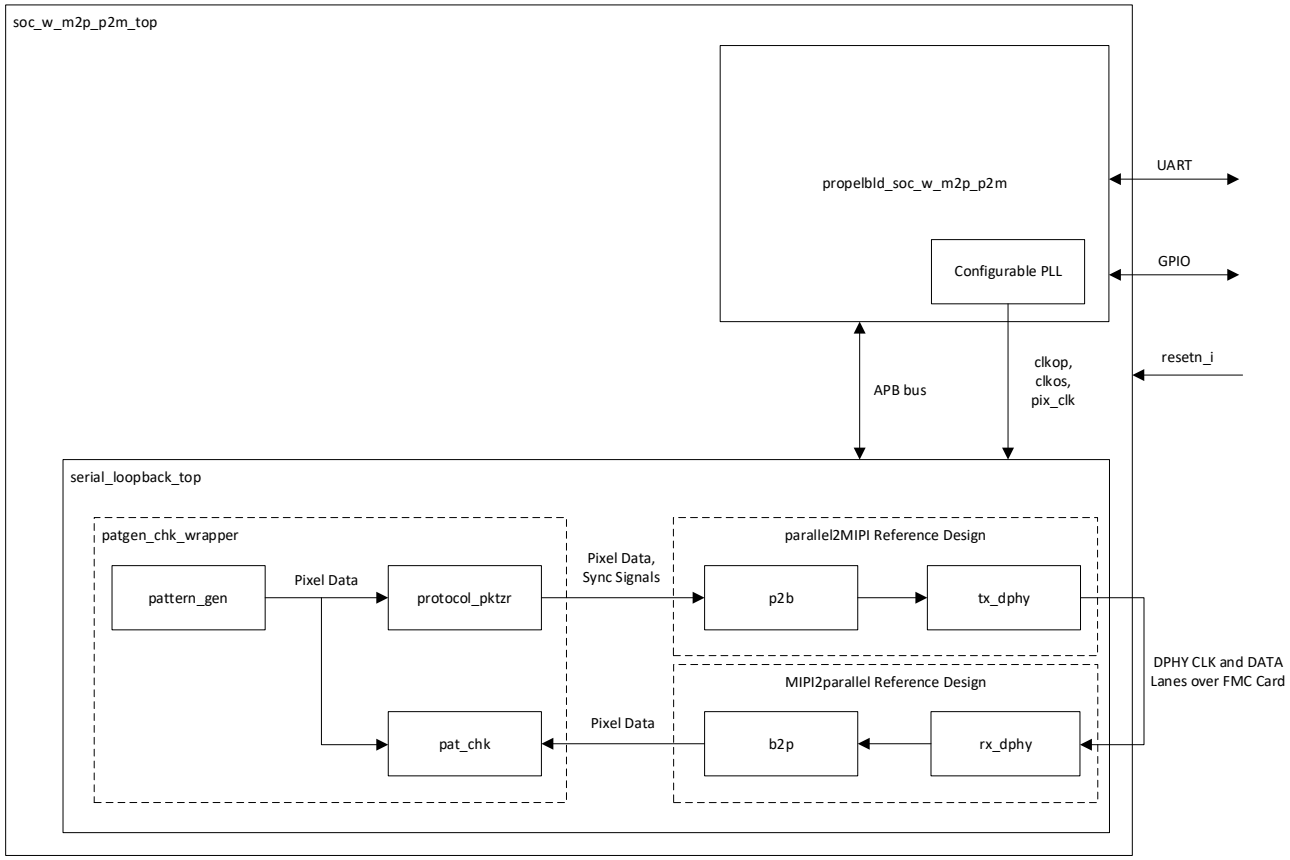


Figure 3.1. Reference Design Block Diagram

3.1. Design Components

The MIPI-to-Parallel and Parallel-to-MIPI reference design initiates and connects propelbld_soc_w_m2p_p2m and serial_loopback_top blocks with the soc_w_m2p_p2m_top module.

The blocks in the MIPI-to-Parallel and Parallel-to-MIPI reference design are described in the subsequent subsections.

3.1.1. propelbld_soc_w_m2p_p2m

The propelbld_soc_w_m2p_p2m module is generated by the Lattice Propel Builder to wrap over the RISC-V and other peripherals together. RISC-V, system memory (system0), and AHB-Lite bus run on a 100 MHz clock, while APB bus runs on a 50 MHz clock. Both clocks are generated by PLL0.

All IPs and connections for the module are generated in the Lattice Propel Builder. Any modification to the module must be made within the Lattice Propel Builder environment. Figure 3.2 shows a high-level block diagram of the module.

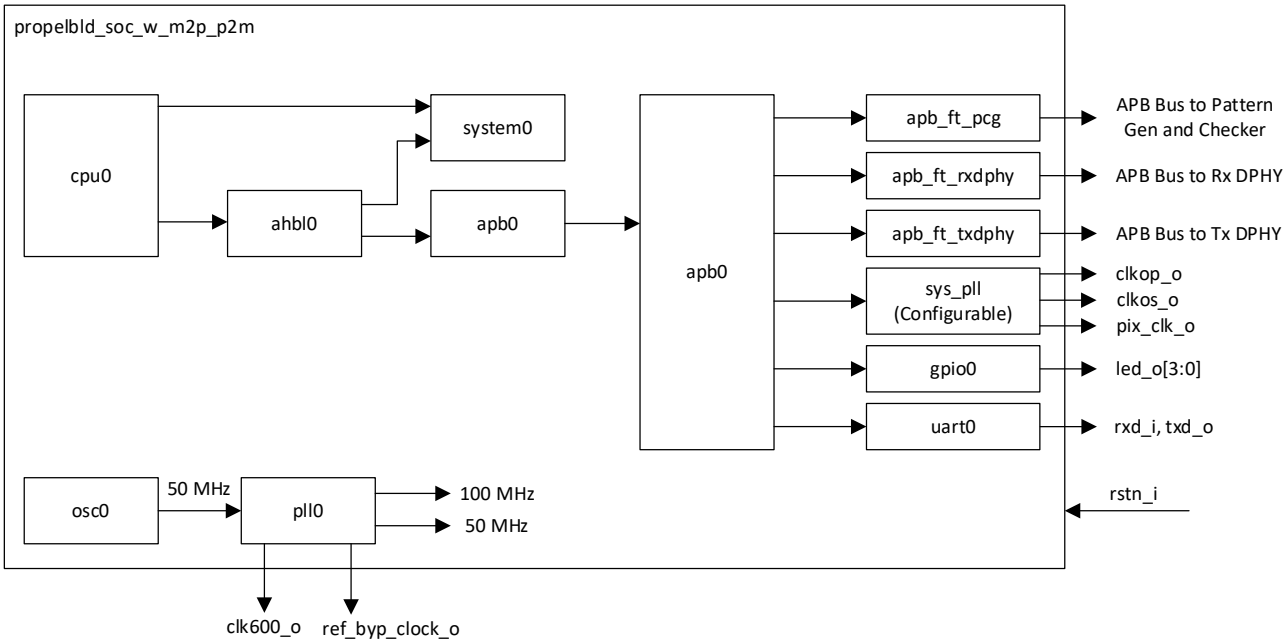


Figure 3.2. propelbld_soc_w_m2p_p2m Block Diagram

The propelbld_soc_w_m2p_p2m module instantiated the blocks described in the subsequent subsections.

3.1.1.1. osc0

The osc0 module is an on-chip oscillator that supplies a 50 MHz clock, serving as the reference clock for pll0 and sys_pll in the propelbld_soc_w_m2p_p2m block.

The following guidelines and parameter settings are required for this reference design:

- HFCLK Enable – ENABLED
- Frequency – 50 MHz
- LFCLK Enabled – DISABLED
- SEDCLK Enable – DISABLED
- CFGLMMICLK Enable – DISABLED

3.1.1.2. pll0

The pll0 module is instantiated to generate the necessary clocks required by the design.

The following guidelines and parameter settings are required for this reference design:

- Configuration Mode – Frequency
- CLK1: Frequency – 50 MHz, sourced from the on-chip oscillator output.
- CLKFB: Feedback Mode – INTCLKOP
- CLKOP: Desired Frequency Value – 100 MHz. Used for cpu0 and AHB bus.
- CLKOS: Desired Frequency Value – 50 MHz. Used for the APB bus.
- CLKOS2: Desired Frequency Value – 600 MHz. Used for the frequency_checker module (optional, for debugging purposes).
- CLKOS3: Desired Frequency Value – 120 MHz. Used as the tx_ref_clk and sync_clk for the parallel2mipi and mipi2parallel modules.

3.1.1.3. sys_pll

The sys_pll module is the PLL module instantiated to generate all the necessary clocks for this reference design.
The following guidelines and parameter settings are required for this reference design:

- Configuration Mode – Divider
- Enable Fractional-N Divider – Checked
- CLKI: Frequency – Dependent on the clki_i frequency. This reference design uses 50 MHz clock from osc0_inst.

- PLL outputs:
  - CLKOP – Generates pll_clkop for D-PHY Tx
  - CLKOS – Generates pll_clkos for D-PHY Tx. The frequency of this clock output is the same as CLKOP, with a 90-degree clock shift.
  - CLKOS2 – Generates the pixel clock for the Pixel-to-Byte IP and Byte-to-Pixel IP
- Provide PLL Reset – Checked
- Provide PLL Lock Signal – Checked
- Select Register Interface – APB. This allows dynamic reconfiguration of the PLL to switch to a different clock frequency when performing dynamic reconfiguration on the D-PHY bitrate.
- Enable Soft Control Register – Checked

3.1.1.4. cpu0

The cpu0 module is a 32-bit RISC-V-based CPU instantiated within the propelbld_soc_w_m2p_p2m module. cpu0 interfaces with other components, such as UART, GPIO, mipi2parallel, parallel2mipi, and others, within the software design. The cpu0 module is also used for UART communication with the host and for controlling the on-board LEDs.

The following guidelines and parameter settings are required for this reference design:

- C Extension for Compressed Instructions – Checked
- M Extension for Integer Mult and Div – Checked
- Debug Enabled – Checked. This option enables the JTAG port on the CPU to allow debugging with OpenOCD.
- PIC Enable – Checked
- Timer Enable – Checked
- PIC and Timer Base Address – 32'hFFFF0000
- Number of Interrupt Requests – 2
- JTAG Channel Selection – 14
- Enable AHBL Data Output Register – Checked

3.1.1.5. ahbl0

The ahbl0 module is an AHB-Lite Interconnect IP that connects the system memory (system0) and other peripherals to the cpu0 module within the propelbld_soc_w_m2p_p2m block. AHBL_M00 bus connects to the AHBL_S01 bus of system0, while the AHBL_M01 bus connects to the ahbl2apb0 module.

The following guidelines and parameter settings are required for this reference design:

- Total AHB-Lite Managers – 1
- Total AHB-Lite Subordinates – 2
- Manager Address Width – 32
- Data Bus Width – 32
- Manager 0 Subordinate 0 Connect Enable – Checked
- Manager 0 Subordinate 1 Connect Enable – Checked
- For other tabs, use parameter default settings.

3.1.1.6. ahbl2apb0

The ahbl2apb0 module is an AHB-Lite to APB Bridge IP that creates a bridge between the APB interconnect (apb0) to the AHB interconnect (ahbl0) in the propelbld_soc_w_m2p_p2m block. The AHBL_S0 bus of the ahbl2apb0 module connects to the AHBL_M01 of the ahbl0 module.

The following guidelines and parameter settings are required for this reference design:

- Address Width – 32 bits
- Data Bus Width – 32 bits
- APB Clock Enable – Checked

3.1.1.7. apb0

The apb0 module is an APB Interconnect IP within the propelbld_soc_w_m2p_p2m block. This APB interconnect module connects a UART module (uart0), a GPIO module (gpio0), and APB feedthrough modules (apb_ft_txdphy, apb_ft_rxdphy, and apb_ft_pcg).

The following guidelines and parameter settings are required for this reference design:

- Total APB Requestor – 1
- Total APB Completers – 6
- Requestor Address Width – 32 bits
- Data Bus Width – 32 bits

3.1.1.8. system0

The system0 module is generated from the System Memory IP, which stores the instruction and data for cpu0 execution instantiated in the propelbld_soc_w_m2p_p2m block.

The following guidelines and parameter settings are required for this reference design:

- Interface – AHBL
- Memory Address Depth – 8192
- Data Bus Width – 32 bits
- Memory Type – EBR
- Port Count – 2
- ECC Enable – Unchecked
- Enable Arbiter – Unchecked
- Enable Data Streamer – Unchecked
- Initialize Memory – Checked
- Initialization File Format – Hex
- Initialization File – The .mem file is generated after each Lattice Propel SDK software build. Locate the file in the *<design directory>/fpga_lifcl/propelsdk/propelsdk_soc_w_m2p_p2m_2/Debug*.

3.1.1.9. uart0

The uart0 module is generated from the UART IP and instantiated within the propelbld_soc_w_m2p_p2m block. The uart0 module enables communication between cpu0 and the host through the UART serial interface. For example, printing logs to the Lattice Propel SDK software console.

The following guidelines and parameter settings are required for this reference design:

- System Clock Frequency – 50 MHz
- Serial Data Width – 8
- Stop Bits – 1
- Parity Enable – Unchecked
- Baud Rate Type – Standard
- UART Standard Baud Rate – 115200
- FIFO Enable – Unchecked
- Rx Ready Enable – Unchecked
- Tx Ready Enable – Unchecked

3.1.1.10. gpio0

The gpio0 module is generated from the GPIO IP and instantiated within the propelbld_soc_w_m2p_p2m block. The gpio0 module allows cpu0 to interact with the FPGA I/O. In this reference design, all four I/O are assigned to the on-board LEDs.

The following guidelines and parameter settings are required for this reference design:

- Number of I/O Lines – 4
- Remove Tri-State Buffer – Checked
- Initial Output Value – 0
- IO Direction – 0xF
- Interface – APB

3.1.1.11. apb_ft

The apb_ft module is generated with the APB Feedthrough IP and instantiated within the propelbld_soc_w_m2p_p2m block. The three instances of this module that are instantiated are apb_ft_rxdphy, apb_ft_txdphy, and apb_ft_pgc. These instances allow APB buses to be exposed from the propelbld_soc_w_m2p_p2m module and manually connected to the serial_loopback_top module at the top level of this reference design (soc_w_m2p_p2m_top).

The following guidelines and parameter settings are required for this reference design:

- Address Width – 10 bits
- Data Bus Width – 32 bits
- Enable PSLVERR signal – Checked
- Export Interface as – Completer
- Memory Map Width – 10 bits

3.1.2. serial_loopback_top

This module is the main hardware design of the MIPI-to-Parallel and Parallel-to-MIPI reference design. This module instantiates the MIPI-to-Parallel (mipi2parallel) and Parallel-to-MIPI (parallel2mipi) modules, Pattern Generator and Checker (patgen_chk_wrapper), and APB to LMMI bridges (apb2lmmi) as described in Figure 3.3. All the IPs in this module are generated using the Lattice Radiant software. To change any IP configuration, you must modify the IP using the Lattice Radiant software.

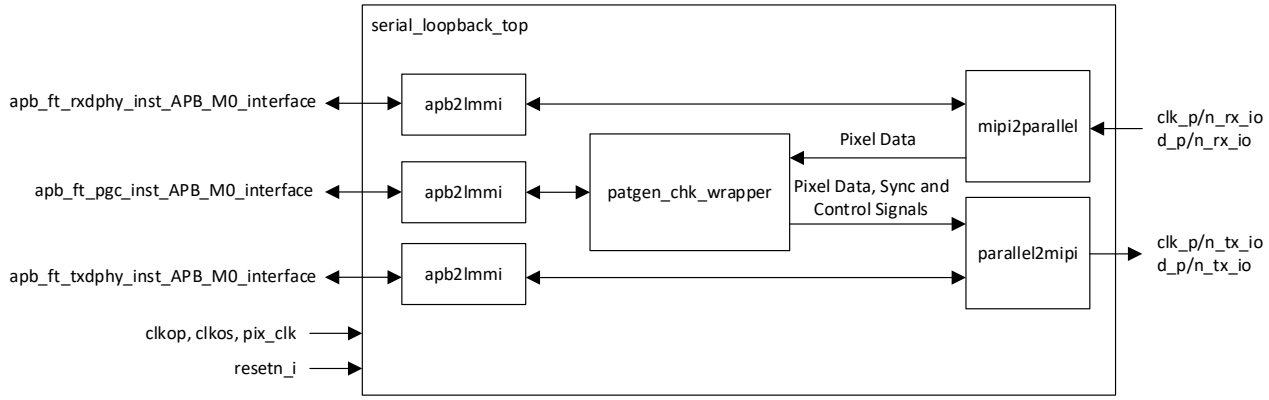


Figure 3.3. serial_loopback_top Block Diagram

3.1.2.1. apb2lmmi

Since most of the IPs, such as rx_dphy and tx_dphy, and the custom Pattern Generator and Checker design implement an LMMI as their control interface, a custom bridge between the APB and LMMI designs is created. This bridge enables these IPs and designs to be controlled by the CPU through software.

Note: To maintain consistency with the design at CPU (the propelbld_soc_w_m2p_p2m module), all APB data widths are standardized to 32 bits. For IPs or modules that use less than 32 bits of data per address, zeros are added as the most significant bits.

All LMMI offsets are mapped to APB addresses by multiplying the offset by a factor of 4. For example, if the LMMI offset is 0x1, the corresponding APB address is 0x4.

3.1.2.2. parallel2mipi

The parallel2mipi module is an RTL wrapper that connects the Pixel-to-Byte Converter IP (p2b) and CSI-2/DSI D-PHY Transmitter IP (tx_dphy).

Figure 3.4 and Figure 3.5 show examples of input bus waveforms to the p2b module that is converted into byte data for the tx_dphy IP. Then, tx_dphy IP converts those byte data into MIPI D-PHY packets to be transmitted out.

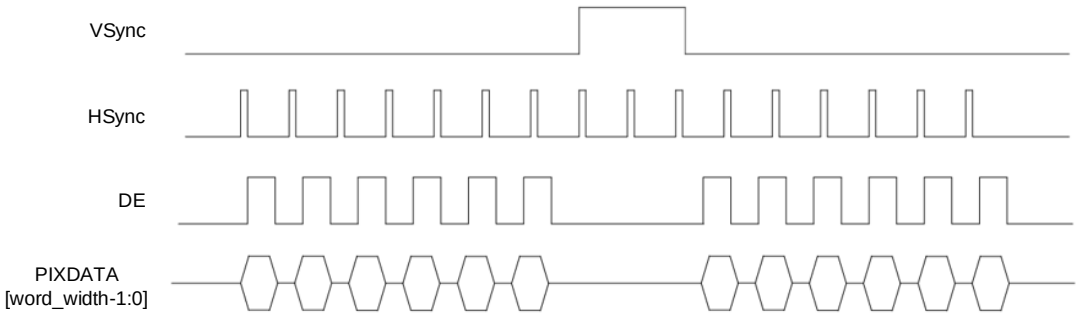


Figure 3.4. Display (DSI) Input Bus Waveform

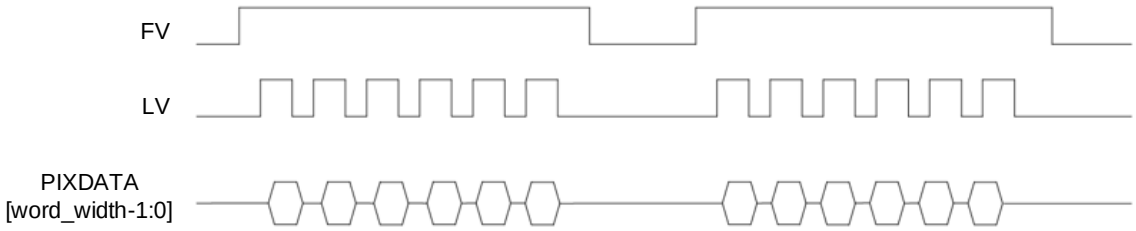


Figure 3.5. Camera (CSI-2) Input Bus Waveform

The submodules in the parallel2mipi module are described in the subsequent subsections.

p2b

This module is instantiated within the Parallel-to-MIPI block to convert pixel data into byte data output according to configurations, such as Tx Interface, Data Type, number of Tx Lanes, and other settings. Refer to the Pixel-to-Byte Converter IP User Guide (FPGA-IPUG-02094) for more details.

The following guidelines and parameter settings are required for this reference design:

- Data Type – Select RGB888, RAW8, RAW10, RAW12, RAW14, RAW16, YUV420_8, YUV422_8, YUV420_10, or YUV422_10. Other types are not supported in this reference design.
- Number of Input Pixel Lanes – Select 1, 2, or 4 input pixel per clock.
- Pixel Clock Frequency – Set to the target pixel clock frequency after dynamic reconfiguration. Default is 100 MHz.
- Enable AXI4-Stream Receiver Interface – OFF
- Receiver Data Rate – Automatically calculated. It is recommended that the value equal to the Transmitter Data Rate to have the same FIFO read and write frequency.
- Tx Interface – Select CSI-2 or DSI. Set the same type as the Tx D-PHY IP.
- DSI Mode – Available only for DSI. Set to Non-Burst Pulses (default).
- Number of Tx Lanes – Select 1, 2, or 4. Set the same value as the Tx D-PHY IP.
- Tx Gear – Select 8. Set the same value as the Tx D-PHY IP.
- Byte Clock Frequency – Set to target byte clock frequency after dynamic reconfiguration. Default is 150 MHz.
- Enable AXI4-Stream Transmitter Interface – OFF
- Transmitter Data Rate – Automatically calculated. It is recommended that the value equal to the Receiver Data Rate to have the same FIFO read and write frequency.
- Enable APB Interface – OFF
- Enable Line Valid Masks Signals – Available for CSI-2. OFF (default).
- Word Count – (NUM_PIXELS × PD_BUS_WIDTH) ÷ 8
  For example: In case of 240 pixel with RGB888, the value is (240 × 24) ÷ 8 = 720.
- Manual Adjust – Unchecked (disabled)

The Pixel-to-Byte Converter IP converts the standard pixel data format to the D-PHY CSI-2/DSI standard based byte data stream. The .ipx file included in the project (p2b/p2b.ipx) can be used to reconfigure the IP per your configuration requirements. If you create this IP from scratch, it is recommended to set the design name to *p2b* so that you do not need to modify the instance name of this IP in the top-level design. Otherwise, you need to modify the names accordingly.

tx_dphy

This module is instantiated within the Parallel-to-MIPI block and created according to the channel conditions, such as number of lanes, bandwidth, and other settings. Refer to the CSI-2/DSI D-PHY Tx IP User Guide (FPGA-IPUG-02080) for more details.

The following guidelines and parameter settings are required for this reference design:

- Tx Interface Type – Select CSI-2 or DSI. Set according to the required configuration.
- D-PHY Tx IP – Select SOFT D-PHY. HARD D-PHY is not supported in the current version of reference design.
- Number of Tx Lanes – Select 1, 2, or 4. Set according to the required configuration.
- Tx Gear – Select Gear 8.
- Bypass Packet Formatter – Unchecked (disabled).
- Enable LMMI Interface – Checked (enabled) to allow register access through LMMI interface.
- Enable AXI4-Stream Interface – Unchecked (disabled).
- EoTp Enable (DSI) – Unchecked (disabled).
- Enable Frame Number Increment in Packet Formatter (CSI-2) – Checked (enabled).
- Frame number MAX Value Increment in Packet Formatter (CSI-2) – set to 1
- Enable Line Number Increment in Packet Formatter (CSI-2) – Unchecked (disabled).
- Extended Virtual Channel ID (CSI-2) – Unchecked (disabled).
- Target Tx Line Rate (Mbps per Lane) – Set according to the required configuration.
- D-PHY Clock Mode – Set Continuous or Non-Continuous according to the required configuration.
- Enable Edge Clock Synchronizer and Divider – Checked (enabled).
- Reference Clock Frequency (MHz) [24–200] – Set the value according to the tx_ref_clk_i frequency in the parallel2mipi block. Default is 120 MHz coming from the pll0_inst module.
- Enable tINIT Counter – Checked (Enabled).
- tINIT Counter Value – 1000 (default value).
- Enable Miscellaneous Status Signals – Checked (enabled).
- Protocol Timing Parameters tab – Default values are recommended (change timing values if required).

This module takes the byte data and outputs DSI/CSI-2 data after serialization in the DSI/CSI-2 High Speed mode. The .ipx file included in the project (tx_dphy/tx_dphy.ipx) can be used to reconfigure the IP per your configuration requirements. If you are creating this IP from scratch, it is recommended to set the design name to *tx_dphy* so that you do not need to modify the instance name of this IP in the top-level design. Otherwise, you need to modify the names accordingly.

3.1.2.3. mipi2parallel

The parallel2mipi is an RTL wrapper that connects the Byte-to-Pixel Converter IP (b2p) and CSI-2/DSI D-PHY Receiver IP (rx_dphy).

The parallel transmit interface consists of clock, pixel data, and control signals. The pixel data width is configurable depending on the data type. The control signals are either data enable (DE), vertical and horizontal sync flags (VSYNC and HSYNC) for MIPI DSI applications, or frame valid and line valid for MIPI CSI-2 applications.

The clock is edge-aligned against data and control signals. All signal transitions occur in sync with the rising edge of the pixel clock, as shown in Figure 3.6 and Figure 3.7.

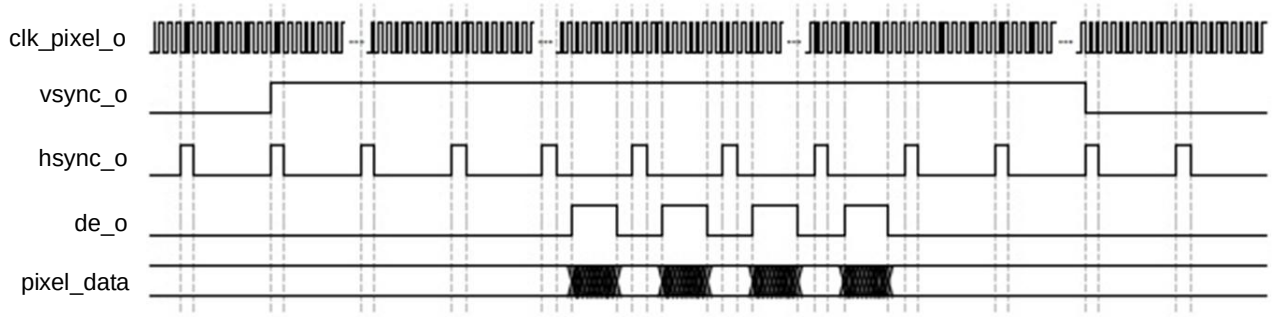


Figure 3.6. Parallel Transmit Interface Timing Diagram (DSI)

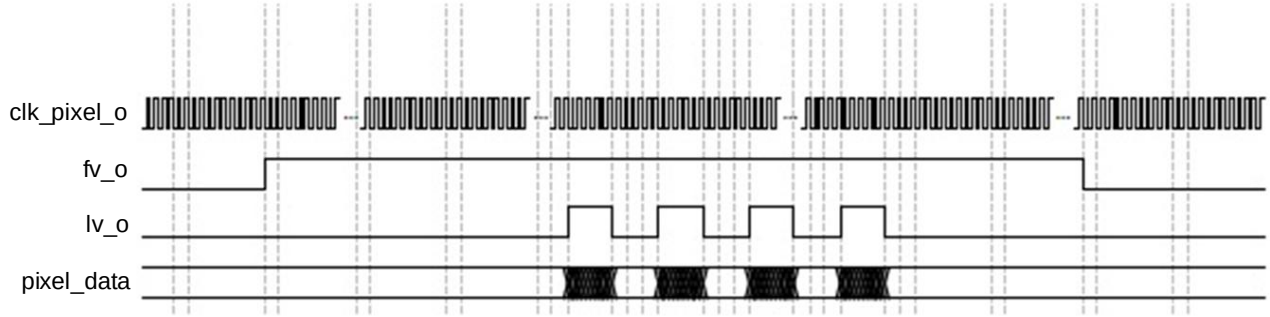


Figure 3.7. Parallel Transmit Interface Timing Diagram (CSI-2)

b2p

This module must be created for the Rx interface according to the required configuration, such as data type, the number of lanes, Rx Gear, and other settings. Refer to the Byte-to-Pixel Converter IP User Guide (FPGA-IPUG-02079) for more details.

The following guidelines and parameter settings are required for this reference design:

- Data Type – Select RGB888, RAW8, RAW10, RAW12, RAW14, RAW16, YUV420_8, YUV422_8, YUV420_10, or YUV422_10. Other types are not supported in this reference design.
- Rx Interface – Select CSI-2 or DSI. Set the same type as the Rx D-PHY IP.
- DSI Mode – Available for DSI. Set to Non-Burst Pulses.
- Number of Rx Lanes – Select 1, 2 or 4. Set the same value as the Rx D-PHY IP.
- Rx Gear – Select 8. Set the same value as the Rx D-PHY IP.
- Byte Clock Frequency – Set to the target byte clock frequency after dynamic reconfiguration. Default is 150 MHz.
- Enable AXI4-Stream Receiver Interface – Unchecked (disabled).
- Number of Output Pixel Lanes – Select 1 or 2
- Camera/Display Control Polarity – Select Positive. If you use Negative polarity, select this option as Positive and use Negative polarity defined from the synthesis_directives.v.
- DSI Sync Packet Delay – Available for DSI. Set to 5 (default value).
- Pixel Clock Frequency – Set to the target pixel clock frequency after dynamic reconfiguration. Default is 100 MHz.
- Pixel-Side Transmitter Interface – Native Interface
- Manual Adjust – Unchecked (disabled).
- FIFO Implementation – EBR
- Word Count – Enter the appropriate value using the following equation:
  Word Count = (NUM_PIXELS × PD_BUS_WIDTH) ÷ 8
  For example: In case of 240 pixel with RGB888, the value is (240 × 24) ÷ 8 = 720.
- Enable Debug Ports – Unchecked (disabled).
- Register Interface – OFF

The Byte-to-Pixel Converter IP converts the D-PHY CSI-2/DSI standard based byte data stream to standard pixel data format. The .ipx file included in the project (b2p/b2p.ipx) can be used to reconfigure the IP per your configuration requirements. If you are creating this IP from scratch, it is recommended to set the design name to *b2p* so that you do not need to modify the instance name of this IP in the top-level design. Otherwise, you need to modify the names accordingly.

rx_dphy

This module must be created for the Rx interface according to the required configuration, such as the number of lanes, bandwidth, and other settings. Refer to CSI-2/DSI D-PHY Rx IP User Guide (FPGA-IPUG-02081) for more details.

The following shows guidelines and parameter settings required for this reference design:

- Rx Interface Type – Select CSI-2 or DSI (set according to the required configuration).
- D-PHY Rx IP – Select SOFT D-PHY. HARD D-PHY is not supported in the current version of reference design.
- Number of D-PHY Data Lanes – Set according to Rx interface configuration (set according to the required configuration).
- Rx Gear – Select Gear 8
- Enable Deskew Calibration Detection – Unchecked (disabled).
- Rx Line Rate – Set according to the Rx interface configuration.
- D-PHY Clock Mode – Select Continuous or Non-continuous (set according to the required configuration).
- Sync Clock Frequency – Set the value according to the sync_clk_i frequency in the mipi2parallel block. Default is 120 MHz coming from the pll0_inst module.
- Enable Lane Aligner Module – Checked (enabled).
- Enable Packet Parser – Checked (enabled).
- Enable AXI4-Stream Interface – Unchecked (disabled).
- Enable LMMI Interface – Checked (enabled).
- Enable Miscellaneous Status Signals – Checked (enabled).
- Enable CRC Check – Unchecked (disabled).
- Customize Data Settle Cycle – Unchecked (disabled).
- Parameters in RX_FIFO Settings tab – Use default settings
  Parameters in Soft PHY tab – Use default settings

This module takes serial CSI-2/DSI data and outputs byte data after de-serialization in the MIPI High Speed mode. The .ipx file included in the project (rx_dphy/rx_dphy.ipx) can be used to reconfigure the IP per your configuration requirements. If you are creating this IP from scratch, it is recommended to set the design name to *rx_dphy* so that you do not need to modify the instance names of these IPs in top level design. Otherwise, you need to modify the names accordingly.

3.1.2.4. patgen_chk_wrapper

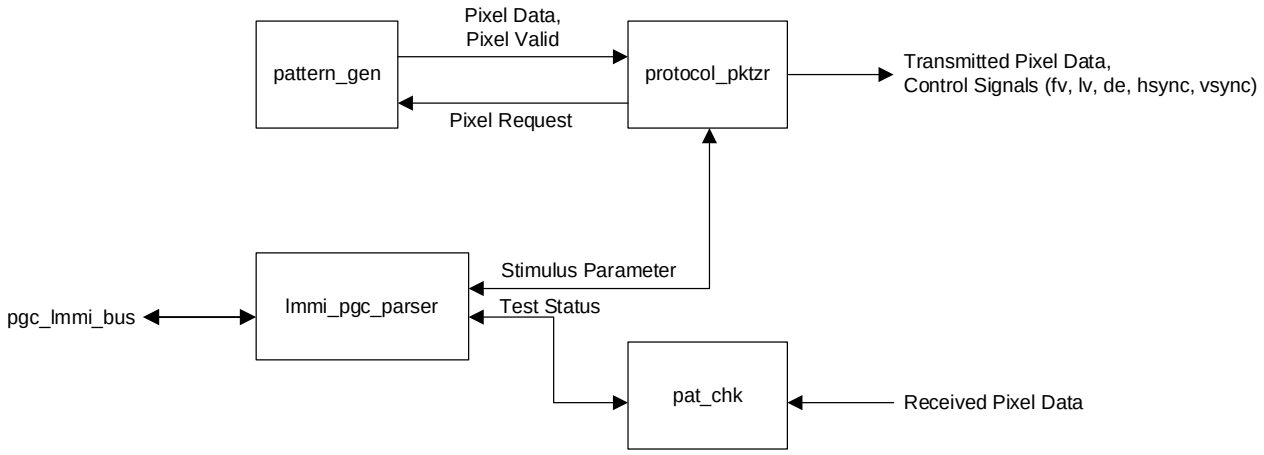


Figure 3.8. patgen_chk_wrapper High Level Block Diagram

This reference design incorporates custom components such as the Pattern Generator (pattern_gen), Pattern Checker (pat_chk), LMMI Parser (lmmi_pgc_parser), and Protocol Packetizer (protocol_pktzr) modules. The patgen_chk_wrapper module acts as a synthetic test stimulus for the Parallel-to-MIPI reference design and compares the stimulus to the received pixel data of MIPI-to-Parallel reference design. Figure 3.8 shows the high-level block diagram and Table 3.1 shows the register map of the patgen_chk_wrapper module.

Table 3.1. patgen_chk_wrapper Register Map

Offset	Name	Access	Description
0x0	PGC_DUT_CONTROL	RW	Resetn signal to DUT
0x1	PGC_TST_CONTROL	RW	Test control register for num_frames, polarity, fifo_empty, sticky_test and drop_frames.
0x2	PGC_TST_STATUS	RO	Test status register for test_done, run_flag, and test_result.
0x3	PGC_VID_TIM_HBP	RW	Video timing horizontal back porch register
0x4	PGC_VID_TIM_HSC	RW	Video timing horizontal sync register
0x5	PGC_VID_TIM_HCT	RW	Video timing horizontal active register
0x6	PGC_VID_TIM_HFP	RW	Video timing horizontal front porch register
0x7	PGC_VID_TIM_VBP	RW	Video timing vertical back porch register
0x8	PGC_VID_TIM_VSC	RW	Video timing vertical sync register
0x9	PGC_VID_TIM_VCT	RW	Video timing vertical active register
0xA	PGC_VID_TIM_VFP	RW	Video timing vertical front porch register

pattern_gen

The Pattern Generator module supports up to two user-selectable pattern types – PRBS or Colorbar. The pattern_gen module generates pixel data according to the specified data width:

- Colorbar is only supported for data type of RGB888 and 1 pixel per clock (PPC) mode.
- The current PRBS implementation only generates 8-bits pseudo-random data. For the data type and pixel per clock combination that requires the output data to be larger than 8 bits, the LSBs are repeated for MSBs.
  For example, if the PRBS generated 0xAB, and the output data width is 16 bits, the output data is 0xABAB. The timing diagram in Figure 3.9 shows a 3-cycle delay between pix_req_i assertion and de-assertion to the valid pixel outputs.

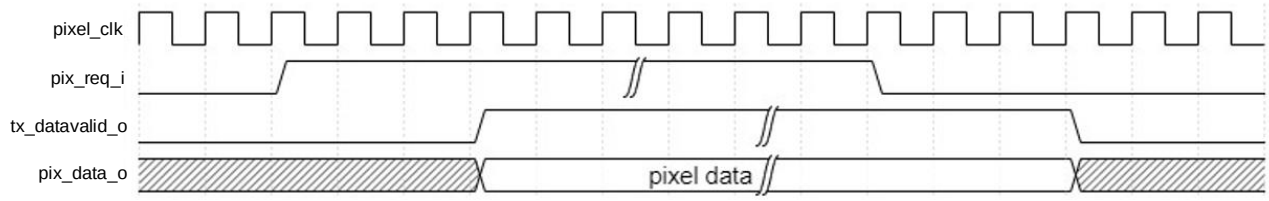


Figure 3.9. pattern_gen Timing Diagram

protocol_pktzr

The Protocol Packetizer module receives pixel data from the Pattern Generator and other signals from LMMI Parser and produces valid CSI-2 (fv and lv) signals into the Pixel-to-Byte module. Figure 3.10 and Figure 3.11 show the timing diagrams of the generated lv_o and fv_o against the defined design parameters described in the Reference Design Parameter Description section.

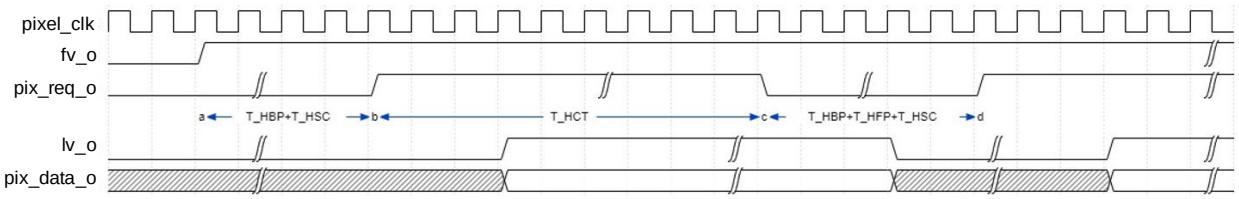


Figure 3.10. protocol_pktzr Timing Diagram Per Line

Figure 3.11. protocol_pktzr Timing Diagram Per Frame

pat_chk

The Pattern Checker module stores the pixel data from the Pattern Generator in a FIFO and compares the pixel data with the incoming data from the Byte-to-Pixel module. If the data matches, the test_result_o signal from the Pattern Checker module is asserted. The signal is de-asserted in the event of a data mismatch.

lmmi_pgc_parser

This module acts as a bridge between control and status signals to the LMMI interface. The default value of each register is defined by the compiler directives as described in the Reference Design Parameter Description section.

3.2. Clocking Scheme

3.2.1. Clocking Overview

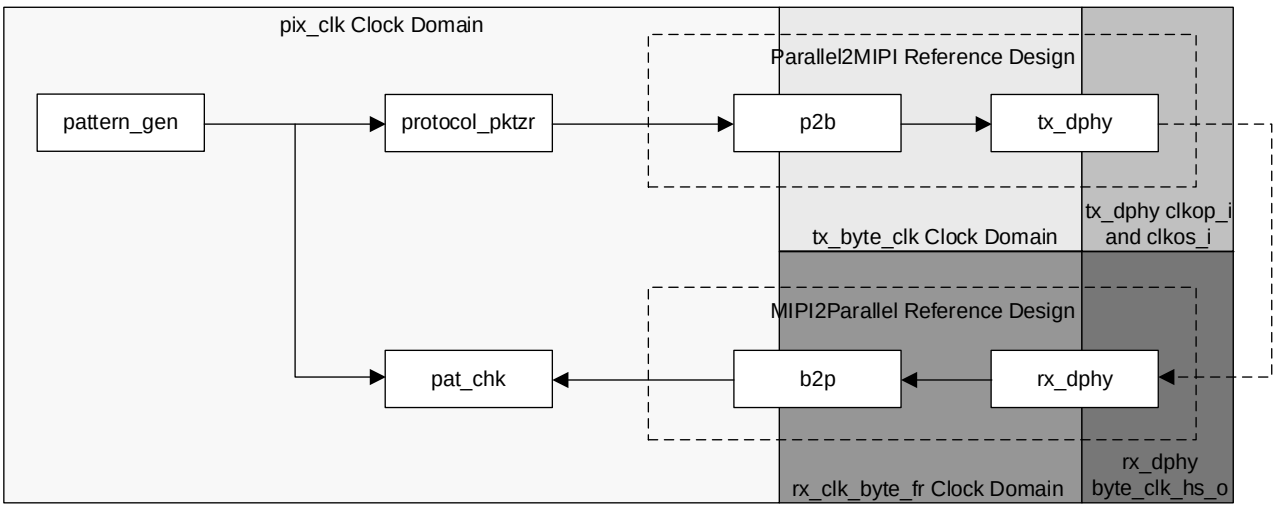


Figure 3.12. Reference Design Clock Domains Block Diagram

Figure 3.12 shows the data path clock domains of the designs:

- The pixel clock domain (pix_clk) is used when the data is already in pixel format. Sync signals also use this domain. This clock is generated by CLKOS2 of the sys_pll module.
- Tx byte clock domain (tx_byte_clk) is used when the data is in byte format and to be consumed by tx_dphy. This clock is generated by the tx_dphy module by dividing the clkop_i pins.

- Tx D-PHY clock domain (tx_dphy clkop_i and clkos_i) is used to transmit D-PHY packets over the D-PHY channel:
  - tx_dphy clkop_i is generated by CLKOP of the sys_pll module.
  - tx_dphy clkos_i is generated by CLKOS of the sys_pll module.
- Rx D-PHY write byte clock (rx_dphy byte_clk_hs_o) is used to capture D-PHY byte data into the built-in FIFO of rx_dphy. This clock is generated by dividing the D-PHY clock lanes by the number of gears.
- Rx D-PHY free-running byte clock (rx_clk_byte_fr) is used to read data from the built-in FIFO of rx_dphy:
  - In the continuous clock mode, this clock is the same as rx_dphy byte_clk_hs_o.
  - In the non-continuous clock mode, this clock comes from byte_clk_o of Tx D-PHY.

3.3. Reset Scheme

3.3.1. Reset Overview

The system level reset is routed to the resetn_i pin of the top-level module as an active-low reset. Asserting this reset asynchronously resets all modules, including sub-modules within the System on Chip (SoC) and serial loopback modules. After the system reset is released, modules in the SoC block (for example, cpu0, osc0, and pll0) and configuration interconnects (AHB-Lite, APB, and LMMI) are out of reset.

The primary reset of serial_loopback_top and the submodules can be controlled using software by modifying the bit 0 of register PGC_DUT_CONTROL (dut_reset_n). To ensure the correctness of the design, this reset must be toggled for at least 100 ns after the bit rate is changed using software.

On top of that, parallel2mipi and mipi2parallel modules use the following signals to reset the block:

```
assign rstn = dut_resetn && dphy_reg_wr_done && resetn_i && pll_locked_i;

//dut_resetn is a software-controlled reset signal that writes to PGC_DUT_CONTROL register
//dphy_reg_wr_done is an optional software-controlled signal after write to D-PHY registers
//resetn_i is an on-board system level reset with push-button
//pll_locked_i is from sys_pll module to check for PLL lock signal after PLL dynamic
reconfiguration
```

All the reset signals are asynchronously asserted, and synchronously de-asserted with the respective clock domains.

4. Reference Design Parameter Description

The MIPI-to-Parallel and Parallel-to-MIPI reference design includes the parameters shown in Table 4.1, Table 4.2, and Table 4.3. You can modify the parameters by editing the synthesis_directives.v file for the design, and the simulation_directives.v file for the RTL simulation testbench.

Note: The parameters must match with the IP configurations respectively.

Table 4.1. Design Parameters in synthesis_directives.v

Parameter	Default Value	Description
MISC_ON	MISC_ON	Enable miscellaneous signals in the design. Comment to disable.
DPHY_DEBUG_ON	DPHY_DEBUG_ON	Enable debug signals in the designs. Comment to disable.
SYNC_POLARITY_POS SYNC_POLARITY_NEG	SYNC_POLARITY_POS	Define polarity for sync signals. Applicable only for DSI.
NUM_DPHY_LANES_1 NUM_DPHY_LANES_2 NUM_DPHY_LANES_4	NUM_DPHY_LANES_4	Number of MIPI D-PHY lanes to be used in the design
DPHY_CLK_MODE_HS_ONLY DPHY_CLK_MODE_HS_LP	DPHY_CLK_MODE_HS_ONLY	Sets the mode for D-PHY clock mode: • HS_ONLY – Continuous Clock Mode • HS_LP – Non-continuous Clock Mode
DPHY_GEAR_8	DPHY_GEAR_8	Number of Gears. For the current version of reference design, only Gear 8 is valid.
PROTOCOL_CSI2 PROTOCOL_DSI	PROTOCOL_CSI2	Defines the protocol to be used
DT_RGB666 DT_RGB888 DT_RAW8 DT_RAW10 DT_RAW12 DT_RAW14 DT_RAW16 DT_YUV420_8 DT_YUV420_10 DT_YUV422_8 DT_YUV422_10	DT_RGB888	Defines the data types to be transmitted or received
NUM_PIX_LANE_1 NUM_PIX_LANE_2 NUM_PIX_LANE_4	NUM_PIX_LANE_2	Defines the number of pixel lanes (or pixel per clock). Note that not all modes are available for a given combination of data types and number of lanes.

Table 4.2. Pattern Generator and Checker Parameters in synthesis_directives.v[1]

Parameter	Default Value	Description
DEFAULT_T_HBP[2]	13'd600	Horizontal back porch period (in pixel clock cycle). The value must be at least 3 or larger.
DEFAULT_T_HSC[2]	13'd44	Horizontal sync porch period (in pixel clock cycle).
DEFAULT_T_HCT[3]	13'd240	Horizontal active period (in pixel clock cycle).
DEFAULT_T_HFP[2]	13'd600	Horizontal front porch period (in pixel clock cycle).
DEFAULT_T_VBP[4,5]	13'd40	Vertical back porch period (in number of lines).
DEFAULT_T_VSC[4,5]	13'd5	Vertical sync porch period (in number of lines).
DEFAULT_T_VCT[4,5]	13'd5	Vertical active period (in number of lines).
DEFAULT_T_VFP[4,5]	13'd10	Vertical front porch period (in number of lines).
DEFAULT_TEST_NUM_FRAMES	8'd2	Number of frames to be tested (inclusive of dropped frames).
DEFAULT_FIFO_EMPTY_FAIL	1'b0	0 – Do not check for checker's FIFO empty as fail 1 – Flag checker's FIFO empty as fail

Parameter	Default Value	Description
DEFAULT_STICKY_TEST_RESULT	1'b1	0 – Test result is not sticky 1 – Test result is sticky
DEFAULT_DROP_FRAMES	4'd1	Number of the first N frames to be ignored by the checker
PATTERN_TYPE	PRBS	PRBS – Uses 8-bit PRBS data for pattern generator COLORBAR – Uses RGB888 color bar for pattern generator[6]

Notes:

1. The directive values are the default values of the pattern generator and checker registers.
2. For CSI-2 mode, DEFAULT_T_HBP + DEFAULT_T_HSC + DEFAULT_T_HFP is used as a horizontal blanking period.
3. The design assumed DEFAULT_HCT is the largest active period for this configuration to calculate the necessary FIFO depth.
4. For CSI-2 mode, DEFAULT_T_VBP + DEFAULT_T_VSC + DEFAULT_T_VFP is used as a vertical blanking period.
5. To speed up simulation, you can reduce vertical timing parameters accordingly.
6. COLORBAR is only supported for RGB888 data type with 1 pixel per clock.

Table 4.3. Simulation Parameters in simulation_directives.v

Parameter	Default Value	Description
REFCLK_PER	8000	Period of the external reference clock in ps
SIM_DBG_MODULE	SIM_DBG_MODULE	Uncomment this directive to enable additional simulation modules for debugging

Table 4.4. Lattice Propel SDK Software Parameter in param_def.h

Parameter	Default Value	Description
BITRATE	1200	Tx and Rx D-PHY target bitrate for dynamic reconfiguration
DATA_TYPE	0x3E	MIPI data type as defined in the Byte-to-Pixel and Pixel-to-Byte IPs. CSI2_RGB888 = 0x24 CSI2_RAW8 = 0x2A CSI2_RAW10 = 0x2B CSI2_RAW12 = 0x2C CSI2_RAW14 = 0x2D CSI2_RAW16 = 0x2E CSI2_YUV420_8 = 0x18 CSI2_YUV420_10 = 0x19 CSI2_YUV422_8 = 0x1E CSI2_YUV422_10 = 0x1F DSI_RGB666 = 0x2E DSI_RGB888 = 0x3E
CLKOP_ODIV	1	Sys_pll's dynamic reconfiguration target output clock divider for PLL CLKOP (clkop_o). This value is minus one (– 1) from the PLL's Divider Desired Value.
CLKOS_ODIV	1	Sys_pll's dynamic reconfiguration target output clock divider for PLL CLKOS (clkos_o). This value is minus one (– 1) from the PLL's Divider Desired Value.
PIXCLK_ODIV	11	Sys_pll's dynamic reconfiguration target output clock divider for PLL CLKOS2 (pix_clk_o). This value is minus one (– 1) from the PLL's Divider Desired Value.
DBG_PRINT	Commented	Enable printf status and debug messages through UART. Comment this parameter to reduce simulation time.
LOOP_COUNT	1	Software run loop count
NUM_FRAMES	2	Number of frames to be transmitted
DROP_FRAMES	1	Number of drop frames

5. Signal Description

The input/output interface signals for the soc_w_m2p_p2m_top.v module are shown in Table 5.1.

Table 5.1. Primary I/O

Port Name	I/O	Width	Description
resetn_i	Input	1	System wide reset signal
uart_rxd_i	Input	1	UART Rx input
uart_txd_o	Output	1	UART Tx output
clk_p_tx_io	Input/Output	1	CSI-2/DSI D-PHY Transmitter Clock pin
clk_n_tx_io	Input/Output	1	CSI-2/DSI D-PHY Transmitter Clock pin
d_p_tx_io[NUM_TX_LANE-1:0][1]	Input/Output	NUM_TX_LANE	CSI-2/DSI D-PHY Transmitter Data pin
d_n_tx_io[NUM_TX_LANE-1:0][1]	Input/Output	NUM_TX_LANE	CSI-2/DSI D-PHY Transmitter Data pin
clk_p_rx_io	Input/Output	1	CSI-2/DSI D-PHY Receiver Clock pin
clk_n_rx_io	Input/Output	1	CSI-2/DSI D-PHY Receiver Clock pin
d_p_rx_io[NUM_RX_LANE-1:0][1]	Input/Output	NUM_RX_LANE	CSI-2/DSI D-PHY Receiver Data pin
d_n_rx_io[NUM_RX_LANE-1:0][1]	Input/Output	NUM_RX_LANE	CSI-2/DSI D-PHY Receiver Data pin
led_o[3:0]	Output	4	General purpose outputs to on-board LEDs
dbg_led_o[1:0]	Output	2	0 – Run test status 1 – Test result status

Note:

1. NUM_RX_LANE and NUM_TX_LANE are set to be equal in this design, which is defined by the NUM_DPHY_LANES_{1-4} synthesis_directives.v.

6. Running the Reference Design

This section describes how to run the MIPI-to-Parallel and Parallel-to-MIPI reference design using the Lattice Radiant software. For more details on the Lattice Radiant software, refer to the Lattice Radiant Software User Guide.

6.1. Opening the Reference Design Project

6.1.1. Radiant Main Project File

To open reference design main project file, follow these steps:

1. Open the Lattice Radiant software.
2. Click **File > Open Project** and from the project database, open the Lattice Radiant software project file (.rdf) from the *<design_directory>/fpga_lifcl/radiant* directory. This opens the reference design project.

Note: The Radiant main project file also contains IPs that are generated by the Propel Builder software. To modify these IPs, you must use the Propel Builder software.

6.1.2. Propel Builder Project File

Follow these steps to open the Propel Builder project of the reference design:

1. Open the Lattice Propel Builder software.
2. Click **File > Open Design** and from the project database, open the .sbx file from the *<design_directory>/fpga_lifcl/radiant/propelbld_soc_w_m2p_p2m* directory. This opens the Lattice Propel Builder project of the reference design.

Note: For each modification on the IP or Propel Builder project, you must click the **Generate** () button for the changes to be reflected in the main Radiant project.

6.1.3. Propel SDK Project File

The reference design also contains software source code to interface with the RISC-V CPU to manage the dynamic reconfiguration flow. Follow these steps to open the software project file:

1. Open the Lattice Propel SDK software.
2. Within the Lattice Propel Laucher interface, set the Workspace to the *<design_directory>/fpga_lifcl/propelsdk* directory and click **Launch**. This opens the software project of the reference design.

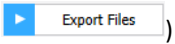
Notes:

- If the project does not appear after setting up the Workspace, you need to import the project by following these steps:
  a. Navigate to **Import Projects > General > Existing Projects into Workspace > Next**.
  b. Assign root directory to *<design_directory>/fpga_lifcl/propelsdk/propelsdk_soc_w_m2p_p2m_2*.
  c. Click **Finish**.
- For every modification on the software project, you must rebuild the software by clicking **Project > Build Project**. This action generates a new .mem file located at *<design_directory>/fpga_lifcl/propelsdk/propelsdk_soc_w_m2p_p2m_2/Debug*. You need to configure system0 in the Propel Builder software to use the .mem file as the Initialization File.

6.2. Compiling and Generating the Bitstream File

This section provides the procedure for creating your FPGA bitstream file using the Lattice Radiant software. However, the reference design package also includes a precompiled FPGA bitstream file, located in the *<design_folder>/fpga_lifcl/precompiled_file* directory, which you can use to run the hardware test directly.

To compile and create a new FPGA bitstream file using the Lattice Radiant software, follow these steps:

1. Open the Lattice Radiant software and reference design project as mentioned in the Radiant Main Project File section.
2. Click the **Export Files** (

) button to generate the bit file. View the log message in the Export Reports folder for the generated bitstream.
3. Locate new bitstream file in the impl1 directory.

7. Simulating the Reference Design

To simulate the design, perform the following steps:

1. Unzip the reference design .zip file.
2. Open the reference design project file (propelbld_soc_w_m2p_p2m.rdf) using the Lattice Radiant software as mentioned in the Radiant Main Project File section.
3. To modify the stimulus sent (resolution, blanking periods, and other data), you can manually modify the compiler directives, as documented in Table 4.1 and Table 4.2.
4. You can also define the SIM_DBG_MODULE directive, as described in Table 4.3, to enable additional debug modules to assist in RTL simulation debugging.
5. Open the QuestaSim simulator from the Radiant software.
6. Using QuestaSim Transcript, navigate to the *<design_directory>/fpga_lifcl/sim* directory.
7. Run the script using the following command:

```
VSIM 12> do run_sim.do
```

7.1. Simulation Results

When the simulation succeeds, logs are printed in the Transcript window, as shown in Figure 7.1.

```
#            0 Debug mode - Compare RX and TX DPHYs payload data
#            0 TX will be sending frame 0. This frame will be ignored.
#   5000000000 De-assert Global Reset
#   5000000000 Wait for RISC-V to release reset for DUT.
#   5000000000 DUT is now out of reset. Now wait for the run flag to trigger.
# 232197459000 TX will be sending frame 1. This frame will be tested.
# 265944678000 Test is now running. Wait for the test to complete
# 276672939000 TX will be sending frame 2. This frame will be tested.
# 276674937000 Test is now complete
# 276674937000: TEST PASSED!
```

Figure 7.1. Simulation Log

Figure 7.2 shows some of the signals after running the script. You can add more signals in QuestaSim and rerun the simulation by issuing the following command:

```
restart -f;
run -all;
```

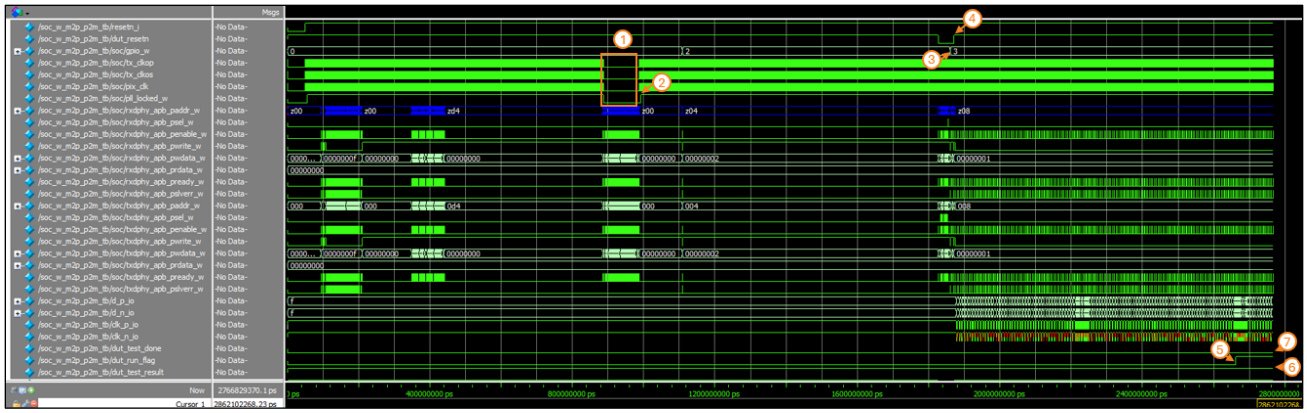


Figure 7.2. Simulation Waveform

The following provides a high-level explanation of the simulation waveform shown above:

1. Sys_pll configuration is in progress.
2. PLL configuration is completed. The sys_pll output is now running at 600 MHz for PLL CLKOP and CLKOS (with a 90-degree phase shift), and 150 MHz for CLKOS2 after the pll_locked signal is asserted.
3. Software writes to the D-PHY Tx and Rx IP registers (tx_tlpx, tx_tclk_hszero, tx_tclk_post, and others) to update the bitrate to 1,200 Mbps. GPIO[0] is asserted once completed.
4. Reset signals to the parallel2mipi and mipi2parallel modules are deasserted. Both D-PHY Tx and Rx are now running at 1,200 Mbps.
5. The test begins, comparing the transmitted and received pixel data.
6. The test_result signal remains asserted, indicating that the pixel data matches.
7. The test has completed.

8. Implementing the Reference Design on Board

The MIPI-to-Parallel and Parallel-to-MIPI reference design can be evaluated using the Lattice CrossLink-NX Evaluation Board. Additional FMC Loopback Card is required to form a loopback path between Tx D-PHY and Rx D-PHY.

8.1. Hardware Setup

Figure 8.1 shows the full setup required to implement MIPI-to-Parallel and Parallel-to-MIPI reference design on the Lattice CrossLink-NX Evaluation Board.

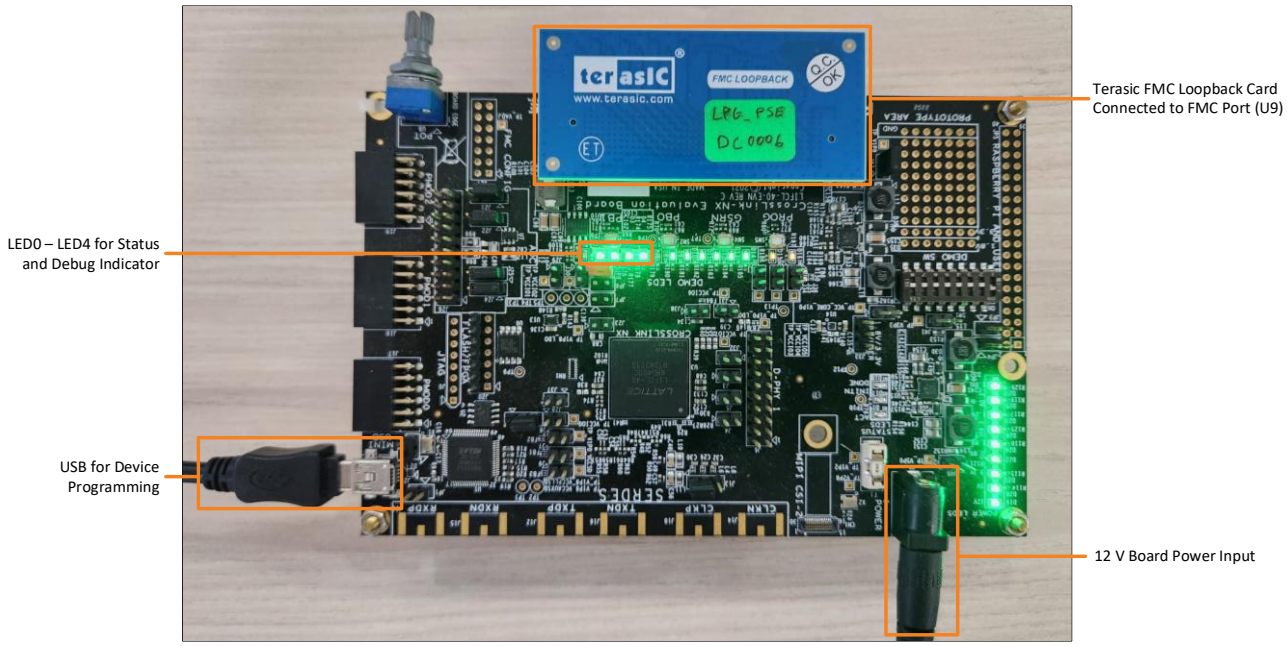


Figure 8.1. Lattice CrossLink-NX Evaluation Board with Additional Hardware

8.1.1. Enabling 1.2 V Support

The CrossLink-NX Evaluation Board must be reworked to enable 1.2 V support for VCCIO3 (bank 3) and VCCIO4 (bank 4) for MIPI usage. Refer to CrossLink-NX Evaluation Board User Guide for the full schematic diagram.

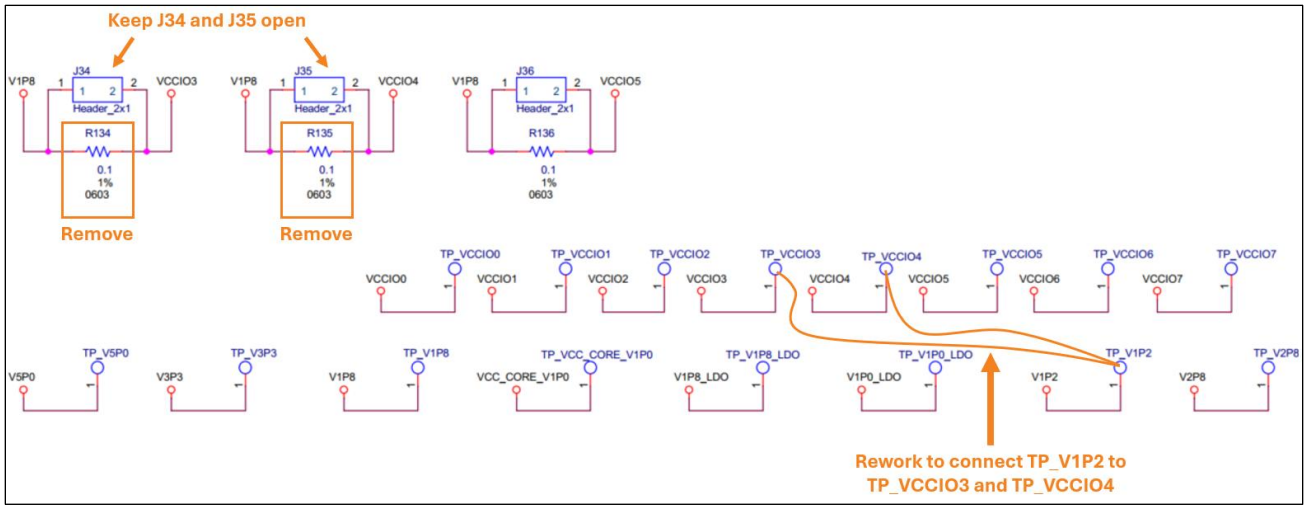


Figure 8.2. Enabling 1.2 V on VCCIO3 and VCCIO4

8.1.2. Enabling UART (Optional)

UART can be used for serial communication to print software status messages through the built-in Serial Terminal of the Propel SDK. The CrossLink-NX Evaluation Board disables the UART communication by default and requires the following hardware modifications to enable the feature:

- Install two 0 Ω resistors at R16 and R17
- Remove the resistors at R18 and R19

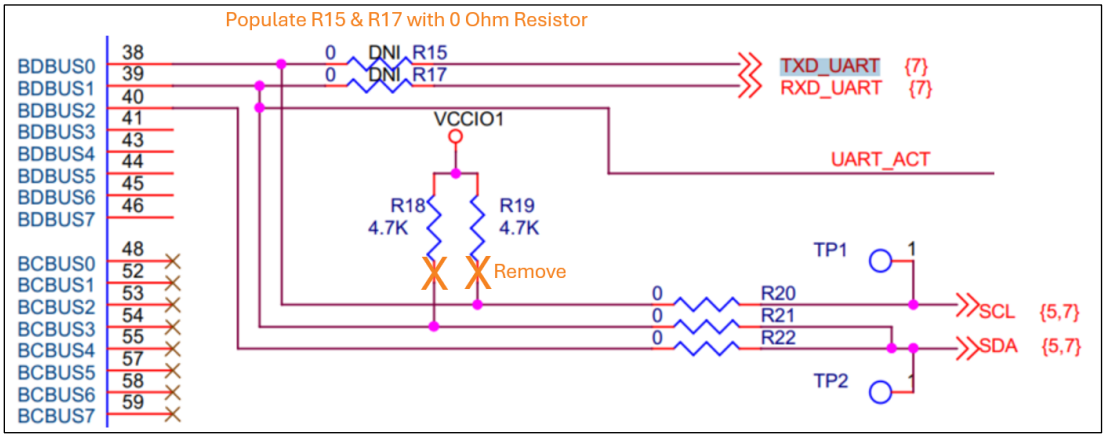


Figure 8.3. Enabling UART Communication

8.2. Programming the Board

To program the board, follow these steps:

1. Modify the reference design configuration and complete the compilation to generate bitstream, as described in the Compiling and Generating the Bitstream File section. Alternatively, you may use the precompiled bitstream file that comes with this reference design.
2. Connect a USB cable from the Lattice CrossLink-NX Evaluation Board to the host machine.
3. Launch and set up the Lattice Radiant Programmer as shown in Figure 8.4.
4. Click the **Program Device** ( ) icon to start programming the FPGA device.

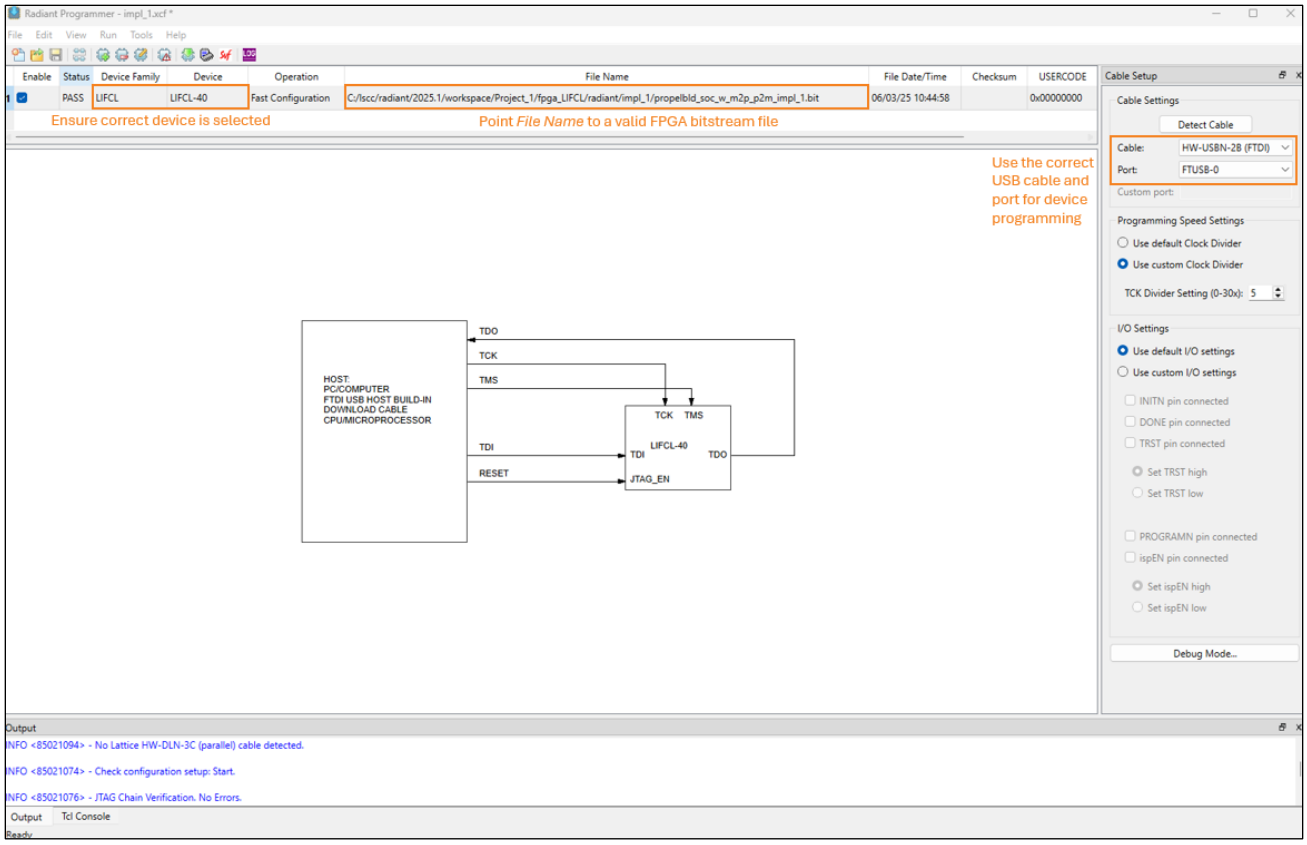


Figure 8.4. The Lattice Radiant Programmer Setup

8.3. Connecting to the Lattice Propel Software

1. Launch the Lattice Propel software version 2025.1 and set the workspace to the *<design_directory>/fpga_lifcl/propelsdk* directory, as mentioned in the Propel SDK Project File section.

   Note: Ensure the board has been programmed. Refer to the Programming the Board section for steps.

2. Launch the Serial Terminal as shown in the example in Figure 8.5. Note that the serial port may vary from host to host.

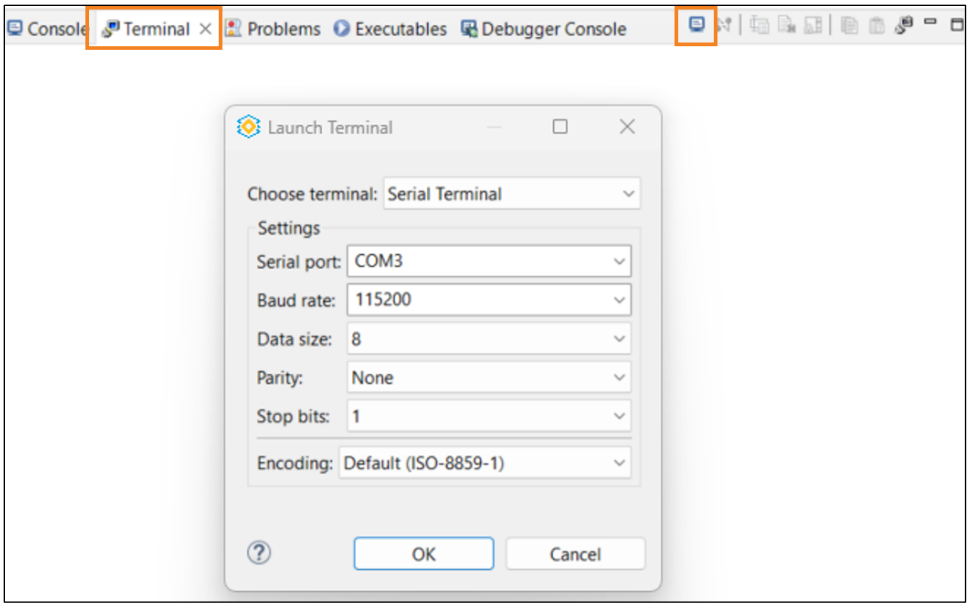


Figure 8.5. Launching Serial Terminal

3. Set up cable connection configuration for OpenOCD by navigating to **Run > Debug Configurations > GDB OpenOCD Debugging > propelsdk_soc_w_m2p_p2m_2 Debug**.
4. Select the port that is connected to the USB cable, as shown in Figure 8.6.

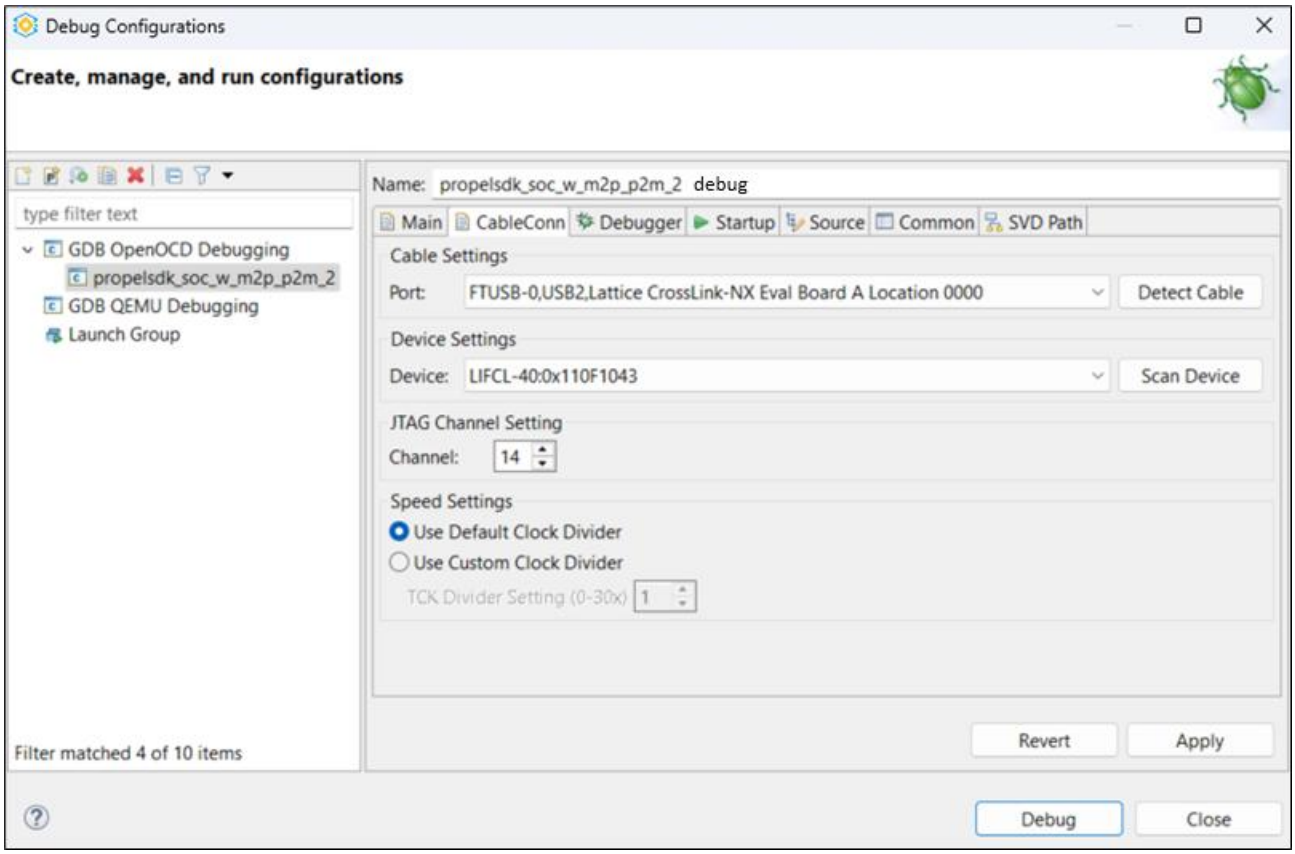


Figure 8.6. Cable Connection Setup – Cable Connection Tab

5. Apply and close the **Debug Configuration** window.
6. You can now write C code or use the existing example code in main.c.
7. Compile and build the design by navigating to **Project > Build Project**.
8. Click the **Run** ( ) button to run the code.

8.4. Hardware Test Result

For the demo hardware test, the design dynamically configures the PLL and D-PHY Tx and Rx to a 1,200 Mbps bitrate, while simultaneously comparing the input and output pixel data of the Pixel-to-Byte and Byte-to-Pixel modules.

To view the result, monitor the on-board LEDs (LED0 to LED4). The LED statuses are described in Table 11.1.

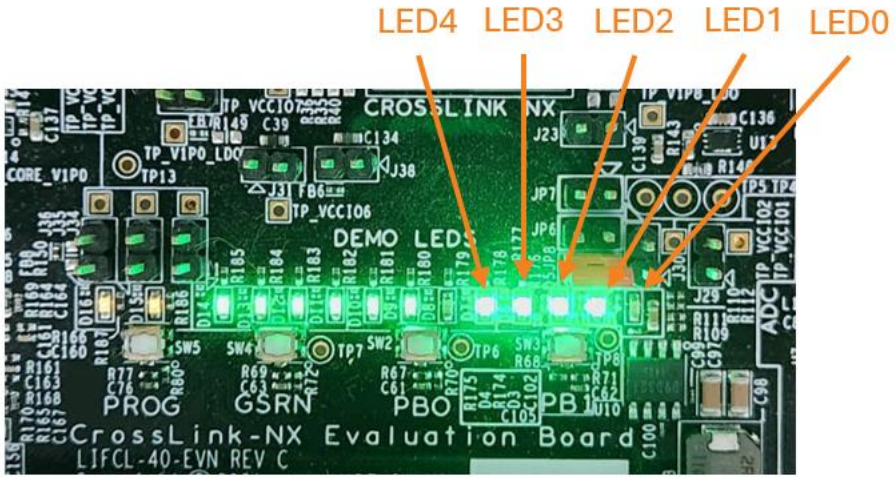


Figure 8.7. LED Status on Successful Hardware Test

Note: Optionally, you could also view the test results through the Serial Terminal by enabling the DBG_PRINT in the software's param_def.h. To view it, go to the **Terminal** tab of the Lattice Propel SDK and analyze the results based on the printout log.

9. Customizing the Reference Design

9.1. Using the Excel Calculator Helper

The reference design includes a simple calculator that helps you determine the D-PHY clock, byte clock, and pixel clock frequencies required by the MIPI D-PHY Tx and Rx, as well as the Byte-to-Pixel and Pixel-to-Byte IP blocks. The calculator can be found in the *<design_directory>/misc* directory.

	A	B	C
1	**Items**	**Values**	**Notes**
2	Line Rate (Mbps)	1200	Must be between 160 and 1500Mbps
3	Number of Lanes	4	1, 2 or 4 lanes
4	Number of Gear	8	Fixed to 8
5	Interface	CSI-2	Select CSI-2 or DSI
6	Data Type	RGB888	Be aware that not all DT supports 2/4 PPC. Check the configuration in b2p/p2b IP GUI
7	Pixel Per Clock	2	1, 2, or 4 PPC
8	Number of bits per pixel clock	48	
9	D_PHY clock (MHz)	600.0000	Propel Builder sys_pll's CLKOP/CLKOS frequency. Note that CLKOS need to be 90 degree shifted
10	Pixel Clock Freq (MHz)	100.0000	Propel Builder sys_pll's CLKOS2 frequency.
11	Byte_clk_fr (MHz)	150.0000	Propel Builder pll0's CLKOS3 frequency, used by RX DPHY as a read clock. Only used by non-continuous clock mode. For continuous clock mode, RX DPHY uses write clock as the read clock as well
12			
13	**LEGEND**		
14	must be filled by user		

Figure 9.1. Excel Calculator Tool

9.2. Dynamic Configuration to Other Bitrates

The reference design provides an example of dynamic configuration at a 1,200 Mbps bitrate. To change to other bitrates, refer to the following steps.

9.2.1. Lattice Radiant Software

1. Modify synthesis_directives.v as per your design requirements, such as the number of D-PHY lanes, the selected protocol, and the video data type.
2. Ensure the parameters defined in the Byte-to-Pixel and Pixel-to-Byte IPs match those in synthesis_directives.v. After updating the IP parameters, make sure to click **Generate**.
3. Confirm that the Tx and Rx D-PHY IPs use the same number of lanes, interface type, and clock mode as defined in synthesis_directives.v. After updating the IP parameters, make sure to click **Generate**.

9.2.2. Lattice Propel SDK

1. Update param_def.h to define the dynamic reconfiguration parameters, including the target bitrate, video data type, and sys_pll settings required for reconfiguration. You can open the sys_pll IP in the Propel Builder software to determine the correct Divider Desired Values for CLKOP/CLKOS (D-PHY clock) and CLKOS2 (pixel clock) based on the target frequency.
2. Rebuild the software after completing the updates. This will generate a .mem file in the Debug folder of the software project.

9.2.3. Lattice Propel Builder Software

1. Open system0 and add the newly generated .mem file as the Initialization File.
2. Click the **Generate** () icon to regenerate the project. The Radiant project is refreshed.

9.3. Standalone MIPI-to-Parallel and Parallel-to-MIPI Reference Design

You can implement standalone MIPI-to-Parallel and Parallel-to-MIPI reference designs. Note that there are no standalone RTL simulation or standalone hardware test available for each of the design. You can verify your target configurations using the MIPI-to-Parallel and Parallel-to-MIPI reference design first before proceeding to extract the standalone MIPI-to-Parallel or Parallel-to-MIPI designs.

9.3.1. MIPI-to-Parallel Standalone Design

1. For MIPI-to-Parallel standalone design (static configuration), you may copy the following files into your new project file:
   - mipi2parallel.v
   - synthesis_directives.v
   - b2p.ipx
   - rx_dphy.ipx
2. You may assign the reset_n_i signal input of the mipi2parallel module to an external push button reset.
3. Refer to the serial_loopback_top module *(<design_directory>/fpga_lifcl/radiant/src/rtl/serial_loopback_top.v)* and the Clocking Scheme section for guidance on clock sourcing for the design.

9.3.2. Parallel-to-MIPI Standalone Design

1. For Parallel-to-MIPI standalone design (static configuration), you may copy the following files into your new project file:
   - Parallel2mipi.v
   - synthesis_directives.v
   - p2b.ipx
   - tx_dphy.ipx
2. You may assign the reset_n_i signal input of the parallel2mipi module to an external push button reset.
3. Refer to the serial_loopback_top module *(<design_directory>/fpga_lifcl/radiant/src/rtl/serial_loopback_top.v)* and the Clocking Scheme section for guidance on clock sourcing for the design.

10. Resource Utilization

Resource utilization depends on the configuration used. Table 10.1 shows the resource utilization examples under certain configurations targeting LIFCL-40 devices. This table is for reference only and actual usage may vary.

Table 10.1. Resource Utilization

Configuration	LUT4	FPU Register	EBR	I/O Buffers
4-lanes, continuous clock mode, CSI-2, RGB888, 2 pixel/clock	9821	6052	29	33
4-lanes, non-continuous clock mode, DSI, RGB888, 2 pixel/clock	9861	6111	31	33
2-lanes, non-continuous clock mode, CSI-2, RAW12, 4 pixel/clock	8921	5460	27	25
4-lanes, continuous clock mode, DSI, RGB888, 4 pixel/clock	9641	5879	30	25

11. Debugging

This section lists possible issues and suggested troubleshooting steps that you can follow.

11.1. Debug Tools

You can use various tools to debug the MIPI-to-Parallel and Parallel-to-MIPI reference design issues.

11.1.1. LED Debug

The reference design uses on-board LEDs (LED0 to LED4) to simplify monitoring of its runtime status during hardware test.

Table 11.1. On-Board LED Status

LED	Status	Description
LED0	On	Test is running
	Off	Test is not running
LED1	On	Test passed
	Off	Test failed
LED2	On	Software is done writing to D-PHY registers
	Off	Software is not done writing to D-PHY registers
LED3	On	Software is done configuring PLL registers
	Off	Software is not done configuring PLL registers
LED4	On	Dynamic reconfiguration test completed
	Off	Dynamic reconfiguration test not completed

11.1.2. Reveal Analyzer

The Reveal™ Analyzer continuously monitors signals within the FPGA for specific conditions, ranging from simple to complex. When a trigger condition occurs, the Reveal Analyzer saves signal values preceding, during, and following the event for analysis, including a waveform presentation. The data can be saved in the following format:

- Value change dump file (.vcd) that can be used with tools such as QuestaSim™.
- ASCII tabular format that can be used with tools such as Microsoft® Excel.

Before running the Reveal Analyzer, use the Reveal Inserter to add Reveal modules to your design. In these modules, specify the signals to monitor, define the trigger conditions, and other preferred options. The Reveal Analyzer supports multiple logic analyzer cores using hard/soft JTAG interface. You can have up to 15 modules, typically one for each clock region of interest. When the modules are set up, regenerate the bitstream data file to program the FPGA.

During debug cycles, this tool uses a divide and conquer method to narrow down the problem areas into many small functional blocks to control and monitor the status of each block.

Refer to Reveal User Guide for Radiant Software for details on how to use the Reveal Analyzer. Refer to the simulation wave.do file from the Simulation Results section to identify some of the critical signals that can be viewed with the Reveal Analyzer.

11.1.3. Propel Terminal

The Lattice Propel SDK includes a built-in terminal tool with serial support for microcontroller debugging. You can utilize this tool to monitor software flow status, such as printf messages after completing PLL register writes, D-PHY register writes, and dynamic reconfiguration. For detailed instructions on using this feature, refer to the Connecting to the Lattice Propel Software section.

11.1.4. Internal Debug Module

The reference design includes the following internal modules to assist in debugging user designs in case any issues or errors arise. You can use the Reveal Analyzer to monitor the output signals of these modules:

- frequency_checker module – This module measures the clock frequency connected to its in_clock input port. In the parallel2mipi block, it measures byte_clk_o from the Tx D-PHY, and in the mipi2parallel block, it measures rx_clk_byte_hs.
- pixelcounter module – This module counts the output pixels of the mipi2parallel block based on a valid signal.
- linecounter module – This module counts the number of lines from the mipi2parallel block, also based on a valid signal.

12. Known Limitations

- Only RGB888, RAW8, RAW10, RAW12, RAW14, RAW16, YUV420_8, YUV422_8, YUV420_10, and YUV422_10 data types are supported.
- The data bandwidth for pixel clock and byte clocks domains (pix_clk, tx_byte_clk, rx_clk_byte_fr, and byte_clk_hs_o) must be equal.
- Dynamic reconfiguration on lanes is not supported in this reference design.
- You need to wait at least 100 ns to release the reset after performing dynamic reconfiguration.

References

- Byte-to-Pixel Converter IP User Guide (FPGA-IPUG-02079)
- Pixel-to-Byte Converter IP User Guide (FPGA-IPUG-02094)
- CSI-2/DSI D-PHY Tx IP User Guide (FPGA-IPUG-02080)
- CSI-2/DSI D-PHY Rx IP User Guide (FPGA-IPUG-02081)
- Reveal User Guide for Radiant Software
- MIPI DSI/CSI-2 to Parallel Bridge Reference Design web page
- Parallel to MIPI CSI-2 / DSI Display Interface Bridge Reference Design web page
- CrossLink-NX web page
- Lattice Radiant Software web page
- Lattice Propel Design Environment web page
- Lattice Solutions Reference Designs 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, please refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, July 2025

Section	Change Summary
All	Initial release.

www.latticesemi.com