Pixel-to-Byte Converter IP

User Guide

FPGA-IPUG-02026-1.3

March 2020

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 and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. 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. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. 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.

Contents

Figures

Tables

1. Introduction

The Lattice Semiconductor Pixel-to-Byte Converter IP converts a standard parallel video interface to DSI or CSI-2 data for Lattice Semiconductor CrossLink™ and CrossLinkPlus™ devices.

The increasing demand for better displays makes bridging applications very popular. Mobile Industry Processor Interface (MIPI®) D-PHY has become the industry primary high-speed PHY solution for camera and display interconnection in mobile devices. It is typically used in conjunction with MIPI Camera Serial Interface-2 (CSI-2) and MIPI Display Serial Interface (DSI) protocol Specifications. It meets the requirements of low-power, low noise generation, and high noise immunity that mobile phone designs demand.

MIPI D-PHY is designed to replace traditional parallel bus based on LVCMOS or LVDS. However, many processors and displays/cameras still use an RGB or CMOS as interface. So, to connect to a MIPI D-PHY IP, a converter logic is required to convert the parallel interface into MIPI D-DPHY byte packet compatible format.

This document describes the use of Lattice FPGA technology for applications requiring conversion of parallel interface to MIPI D-PHY byte packet compatible format. This design can be used in multiple configurations.

This document is for Pixel-to-Byte Converter IP design version 1.3.

Figure 1.1. Sample Parallel Interface to MIPI DSI System Diagram

1.1. Quick Facts

Table 1.1 provides quick facts about the Pixel-to-Byte Converter IP for CrossLink and CrossLinkPlus devices.

Table 1.1. Pixel-to-Byte Converter IP Quick Facts

		Pixel-to-Byte Converter IP Configuration			
		1-Pixel, Gear 8, 4-Lane, RGB888 Configuration	**1-Pixel, Gear 8, 4-Lane, RGB666 Configuration**	**1-Pixel, Gear 8, 4-Lane, RAW12 Configuration**	**1-Pixel, Gear 8, 4-Lane, RAW10 Configuration**
IP Requirements	FPGA Families Supported	CrossLink/CrossLinkPlus			
Resource Utilization	Targeted Device	LIF-MD6000-6MG81I			
	Data Path Width	24-bit parallel input, 64-bit parallel output	18-bit parallel input, 64-bit parallel output	12-bit parallel input, 64-bit parallel output	10-bit parallel input, 64-bit parallel output
	LUTs	184	488	159	563
	sysMEM™ EBRs	4	2	4	4
	Registers	266	434	336	635
	HW MIPI Block	0	0	0	0
Design Tool Support	Lattice Implementation	Lattice Diamond® 3.11. SP1			
	Synthesis	Lattice Synthesis Engine			
		Synplify Pro® N-2018.03L-SP1-1			
	Simulation	Aldec® Active HDL™ 10.5 Lattice Edition			

1.2. Features

The key features of the Pixel-to-Byte Converter IP include:

- Supports RGB888, RGB666, RAW8, RAW10, RAW12, YUV420/YUV422 8/10-bit video formats
- Converts 1, 2, 4, 6, 8, or 10 pixels per pixel clock into MIPI D-DPHY byte packet compatible format
- Supports byte arrangement for 1, 2, or 4 MIPI D-PHY data lanes

1.3. Conventions

1.3.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL. This includes radix indications and logical operators.

1.3.2. Data Ordering and Data Types

The most significant bit within the pixel data is the highest index.

1.3.3. Signal Names

Signal names that end with:

- *_n* are active low
- *_i* are input signals
- *_o* are output signals
- *_io* are bidirectional signals

2. Functional Description

The Pixel-to-Byte Converter IP converts a standard parallel video interface to either DSI or CSI-2 byte packets. The input interface for the design consists of a pixel clock and pixel bus. For DSI, it also consists of vertical and horizontal synchronization flags, and a data enable. For CSI-2, it also consists of a frame and line valid flags.

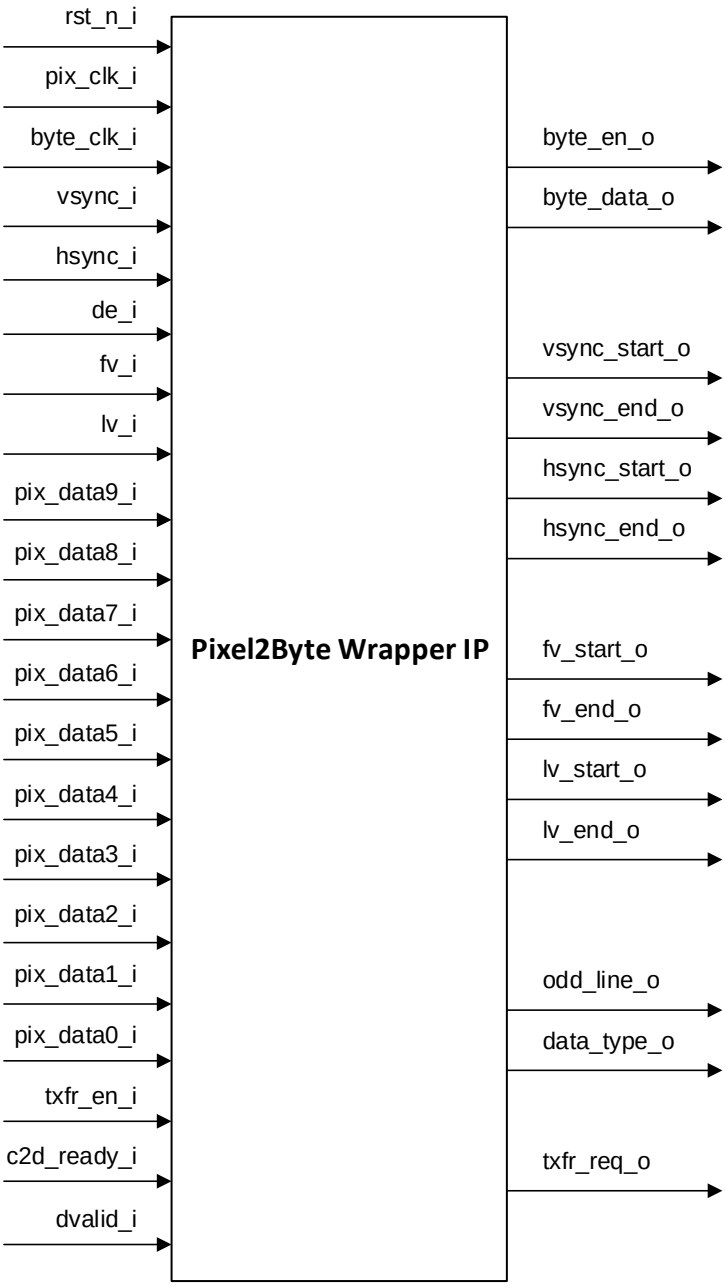

Figure 2.1. Top-level Block Diagram

Table 2.1. Pixel-to-Byte Converter IP Pin Function Description

Pin Name	Direction	Function Description
Clocks and Reset		
rst_n_i	I	Asynchronous active low system reset. 0 – System on reset
pix_clk_i	I	Input pixel clock.
byte_clk_i	I	Input byte clock.
Pixel Domain		
pix_data0_i	I	Input pixel data 0. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data1_i[1]	I	Input pixel data 1. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data2_i[1]	I	Input pixel data 2. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data3_i[1]	I	Input pixel data 3. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data4_i[1]	I	Input pixel data 4. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data5_i[1]	I	Input pixel data 5. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data6_i[1]	I	Input pixel data 6. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data7_i[1]	I	Input pixel data 7. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666

Pin Name	Direction	Function Description
		12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data8_i[1]	I	Input pixel data 8. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
pix_data9_i[1]	I	Input pixel data 9. Bus width depends on the data type selected. 24-bit bus width – RGB888 18-bit bus width – RGB666 12-bit bus width – Raw12 10-bit bus width – Raw10, YUV420/422 10-bit 8-bit bus width – Raw8, YUV420/422 8-bit
de_i[2]	I	Input data enable for parallel interface.
hsync_i[2]	I	Input horizontal sync for parallel interface.
vsync_i[2]	I	Input vertical sync for parallel interface.
fv_i[3]	I	Input frame valid for parallel interface.
lv_i[3]	I	Input line valid sync for parallel interface.
Byte Domain		
vsync_start_o[2]	O	Pulse signal used to indicate Vsync start.
vsync_end_o[2]	O	Pulse signal used to indicate Vsync end.
hsync_start_o[2]	O	Pulse signal used to indicate Hsync start.
hsync_end_o[2]	O	Pulse signal used to indicate Hsync end.
fv_start_o[3]	O	Pulse signal used to indicate frame start.
fv_end_o[3]	O	Pulse signal used to indicate frame end.
lv_start_o[3]	O	Pulse signal used to indicate line start.
lv_end_o[3]	O	Pulse signal used to indicate line end.
byte_en_o	O	Indicates valid output byte data.
byte_data_o	O	64-bit output data in byte format.
Miscellaneous		
c2d_ready_i	I	Ready signal for transmit request
odd_line_o[4,5]	O	Indicates if current line is odd or even; used for YUV420 data type. 0 – Output value for even line 1 – Output value for odd line
data_type_o[5]	O	6-bit output that indicates the data type based from MIPI DSI and CSI2 Specifications
txfr_en_i[6,7]	I	Enable flag from outside the IP to indicate that byte data can already be sent out from pixel2byte. 0 – do not send output byte data yet
txfr_req_o[6,7]	O	When asserted, it indicates that valid data (HSYNC, VSYNC, DE) is received and IP is ready to process the data.

Notes:

1. Available only when the number of input pixels data is more than 1. See Table 2.3 for more details.
2. Available only if data interface is DSI.
3. Available only if data interface is CSI-2.
4. Available only for YUV420 data type.
5. Can be turned on if *Enable miscellaneous status signals* is selected in the Pixel to Byte Converter UI, or MISC_ON is defined in the <*instance_name*>_params.v file.
6. Turned on if ENABLE HANDSHAKE SIGNAL is selected in the interface upon IP generation, or TXFR_SIG is defined in the defines file. These signals are mandatory and are always available.
7. See the Interface and Timing Diagram section for details on their functionality.

2.1. Interface and Timing Diagram

Figure 2.2 and Figure 2.3 shows the timing diagram of display and camera parallel input interface, respectively. It follows the standard DSI/CSI-2 interface protocol with VSYNC, HSYNC, Data Enable for DSI and Frame Valid, Line Valid for CSI-2, pixel data, all clocked by pixel clock. The number of pixels per pixel clock depends on the number of input pixel clocks selected during design configuration. See Table 2.3 for the list of supported configurations.

Input pixel data is converted to byte data compatible with MIPI D-PHY packet with a bus width fixed to 64-bit. LSB of input pixel data is transmitted first. Byte arrangement depends on the gearing and targeted number of MIPI D-PHY lanes. Each 16-bit of the total bus width corresponds to lane number of the target Tx lanes.

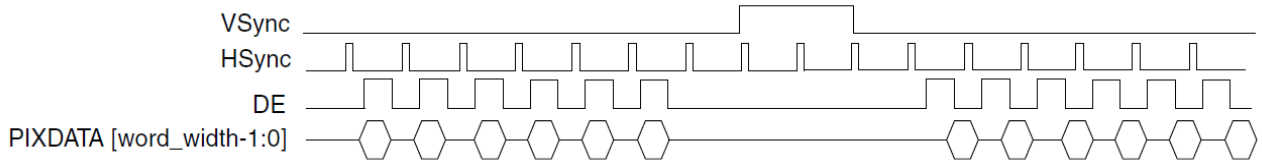

Figure 2.2. Display Parallel Input Interface Timing Diagram (DSI)

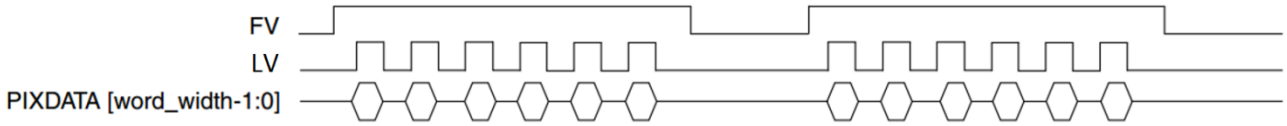

Figure 2.3. Camera Sensor Parallel Input Interface Timing Diagram (CSI-2)

Table 2.2. Output Byte data bus Width Allocation

Tx Lane Number	Valid Bits
1	[15:0]
2	[31:16]
3	[47:32]
4	[63:48]

LSB is the first valid output byte. Depending on the gearing, each byte is placed accordingly. Unused bytes are padded with 0s.

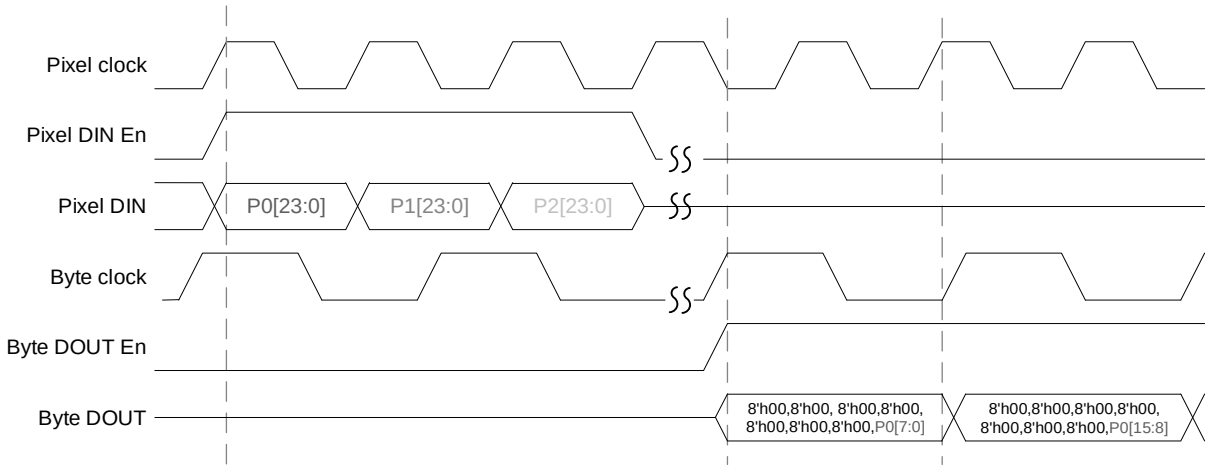

Figure 2.4. Sample Input to Output Timing Diagram (RGB888, Gear 8, 1 Tx lane, 1 Pixel per Pixel Clock)

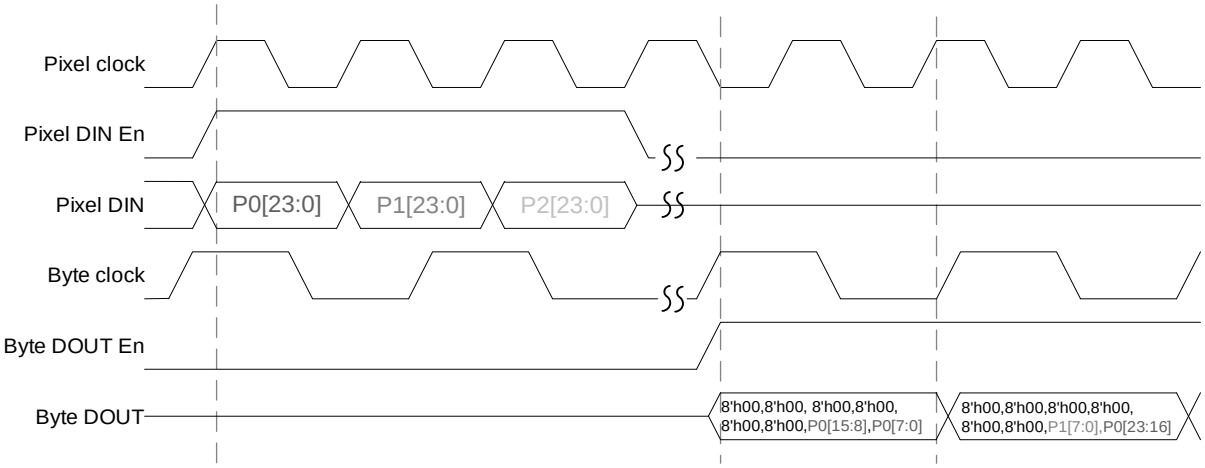

Figure 2.5. Sample Input to Output Timing Diagram (RGB888, Gear 16, 1 Tx lane, 1 Pixel per Pixel Clock)

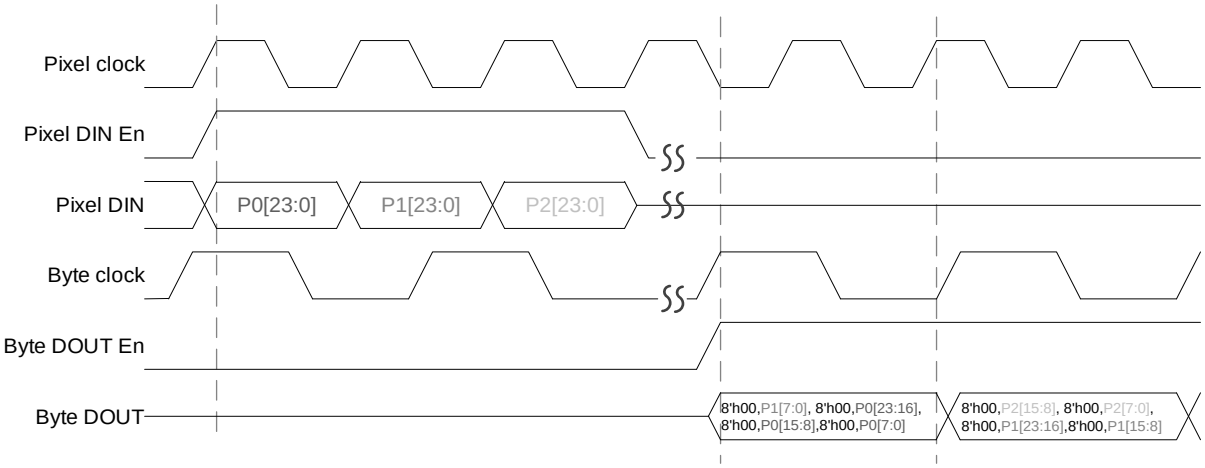

Figure 2.6. Sample Input to Output Timing Diagram (RGB888, Gear 8, 4 Tx lanes, 1 Pixel per Pixel Clock)

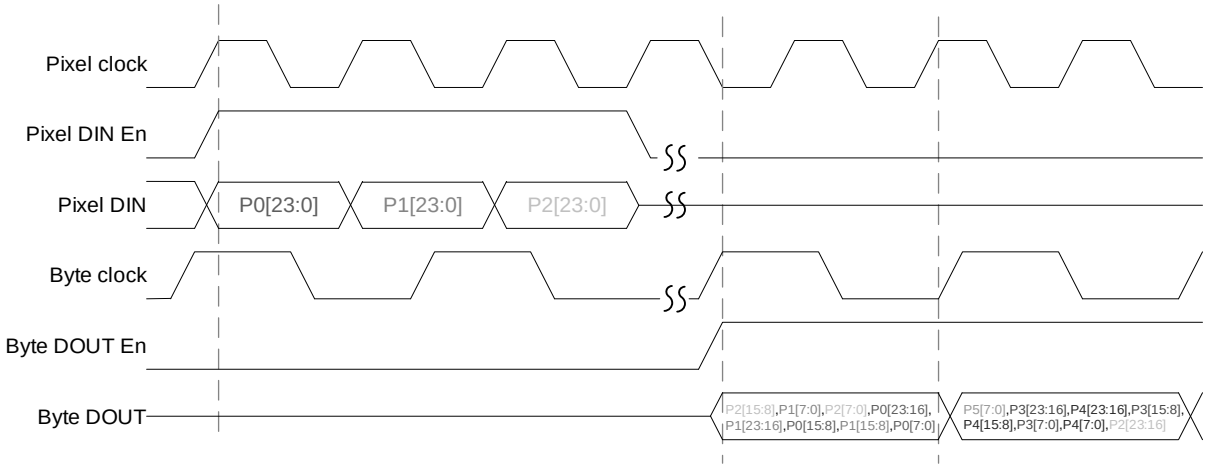

Figure 2.7. Sample Input to Output Timing Diagram (RGB888, Gear 16, 4 Tx lanes, 1 Pixel per Pixel Clock)

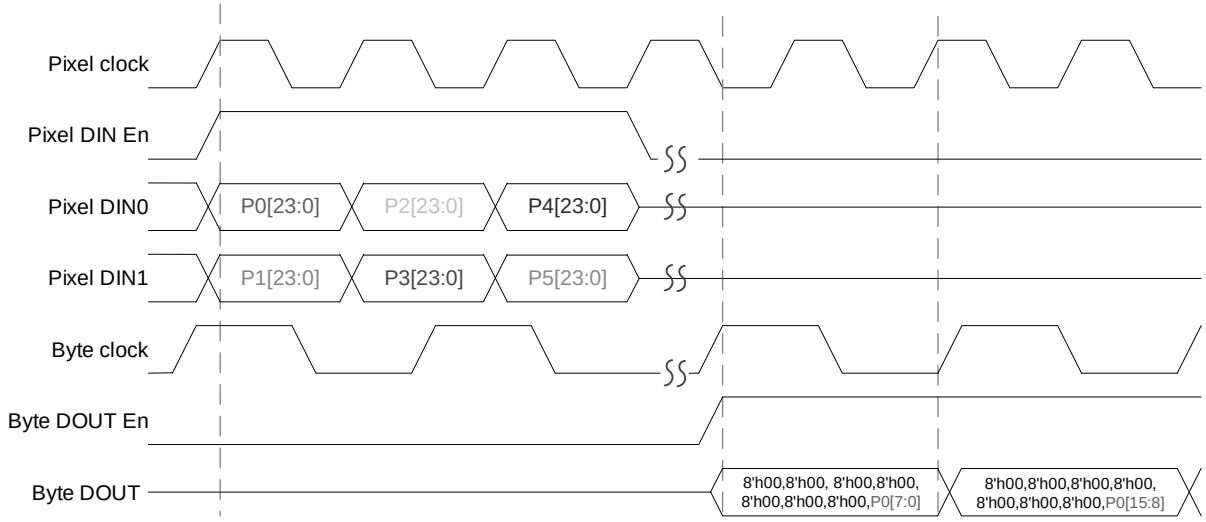

Figure 2.8. Sample Input to Output Timing Diagram (RGB888, Gear 8, 1 Tx lane, 2 Pixels per Pixel Clock)

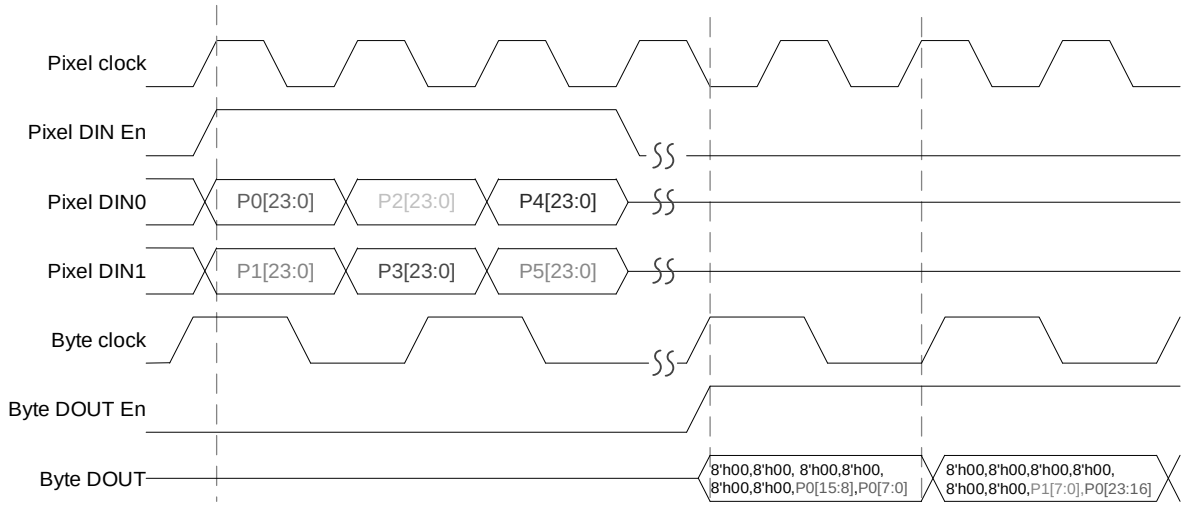

Figure 2.9. Sample Input to Output Timing Diagram (RGB888, Gear 16, 1 Tx lane, 2 Pixels per Pixel Clock)

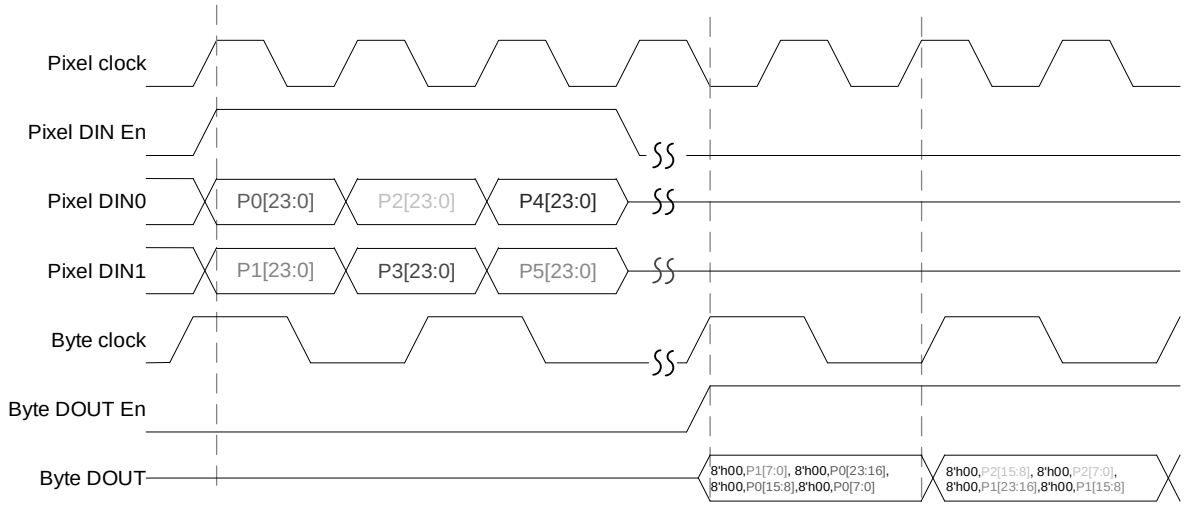

Figure 2.10. Sample Input to Output Timing Diagram (RGB888, Gear 8, 4 Tx lanes, 2 Pixels per Pixel Clock)

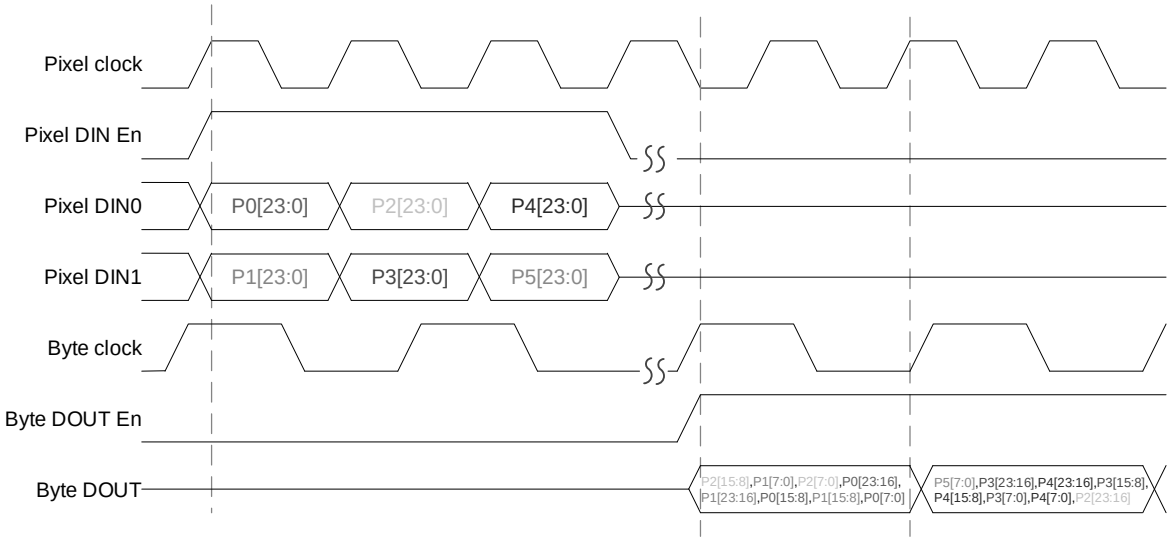

Figure 2.11. Sample Input to Output Timing Diagram (RGB888, Gear 16, 4 Tx lanes, 2 Pixels per Pixel Clock)

As Pixel-to-Byte Converter IP is interfaced to other MIPI D-PHY related IPs, input signal *txfr_en_i* is expected to be asserted when MIPI D-PHY lanes are already in High Speed (HS) mode. HS mode refers to the state after $T_{HS-ZERO}$ until just before $T_{HS-TRAIL}$. See Figure 2.12.

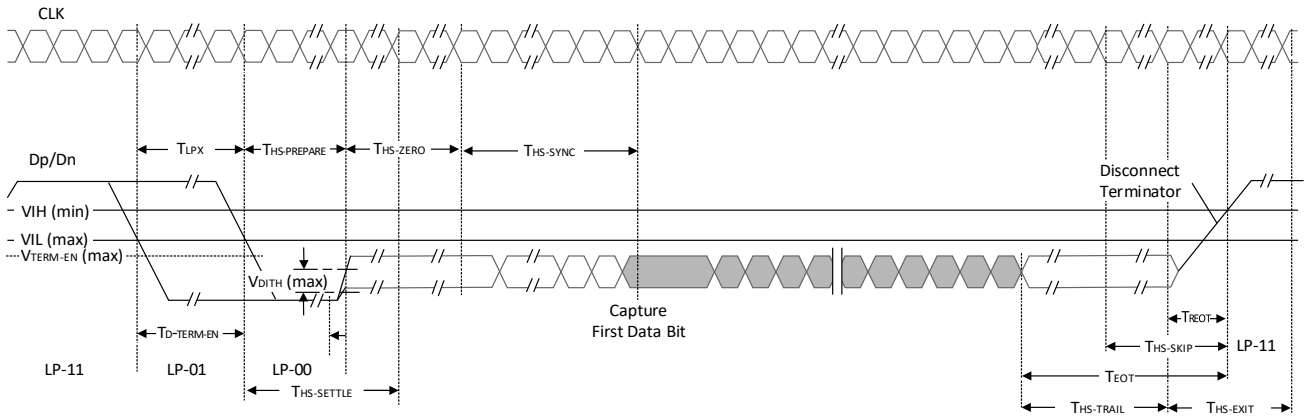

Figure 2.12. MIPI D-PHY High Speed Transmission Timing Diagram

Output signal *txfr_req_o* of the IP is asserted just before T_{LPX} when c2d_ready_i signal is Hi. The input signal *txfr_en_i* should be asserted just right after $T_{HS-ZERO}$ until D-PHY lanes go to $T_{HS-TRAIL}$. The distance between assertion of *txfr_req_o* and *txfr_en_i* should be approximately the minimum requirement for $T_{LPX}+T_{HS-PREPARE}+T_{HS-ZERO}$. Refer to Table 14 Global Operation Timing Parameters of MIPI Alliance Specification for D-PHY, version 1.1, for their corresponding values. Approximately, 200 ns+1 byte clock should suffice. See Figure 2.13.

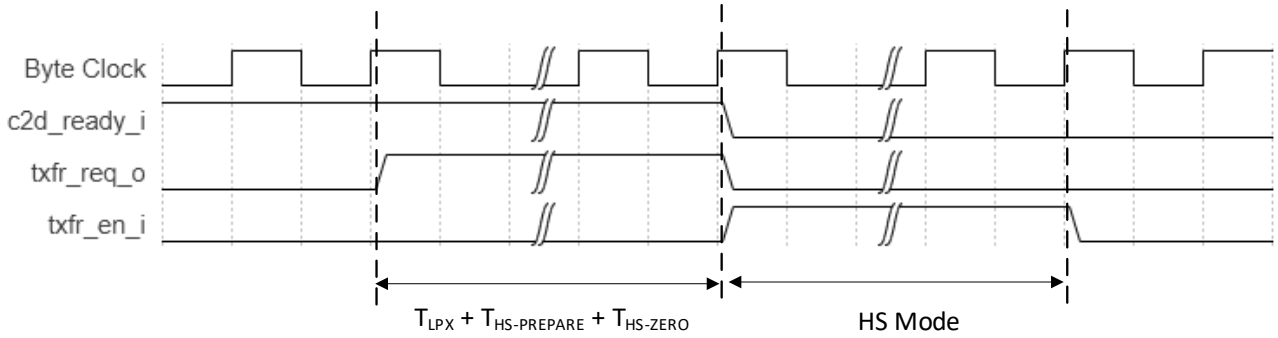

Figure 2.13. Handshake Signals Timing Diagram

2.2. Supported Configurations

Table 2.3 lists the supported configurations of the Pixel-to-Byte Converter.

Table 2.3. List of Supported Configurations

Number of Pixels per Pixel Clock	Interface	Data Type	Tx Lane	Tx Gear
1 Pixel per Pixel clock	DSI	RGB666	1	8, 16
			2	8
			4	8
		RGB888	1	8, 16
			2	8, 16
			4	8
	CSI-2	RAW10 YUV420 10-bit YUV422 10-bit	1	8, 16
			2	8
			4	8
		RAW12	1	8, 16
			2	8
			4	8
		RAW 8 YUV420 8-bit YUV422 8-bit	1	8
			2	8
			4	8
		RGB888	1	8, 16
			2	8, 16
			4	8
2 Pixels per Pixel clock	DSI	RGB666	2	16
			4	8, 16
		RGB888	4	8, 16
	CSI-2	RAW10 YUV420 10-bit YUV422 10-bit	2	16
			4	8
		RAW12	2	16
			4	8
		RAW 8 YUV420 8-bit YUV422 8-bit	1	16
			2	16
			4	8
		RGB888	4	16
4 Pixels per Pixel clock	DSI	RGB666	4	16
		RGB888	4	16

Number of Pixels per Pixel Clock	Interface	Data Type	Tx Lane	Tx Gear
	CSI-2	RAW10	4	8
		RAW12	4	8
6 Pixels per Pixel clock	CSI-2	RAW10	4	8
		RAW12	4	8
8 Pixels per Pixel clock	CSI-2	RAW10	4	16
		RAW12	4	16
10 Pixels per Pixel clock	CSI-2	RAW10	4	16
		RAW12	4	16

2.3. Clock, Reset and Initialization

2.3.1. Reset and Initialization

Active low reset is used in the design with synchronous release. This is the system reset input connected to Pixel-to-Byte module.

Follow this initialization and reset sequence:

1. Assert active low system reset for at least three clock cycles of the slower clock (pixel clock or byte clock).
2. IP is ready to process data after reset.

2.3.2. Clock Domains and Clock Domain Crossing

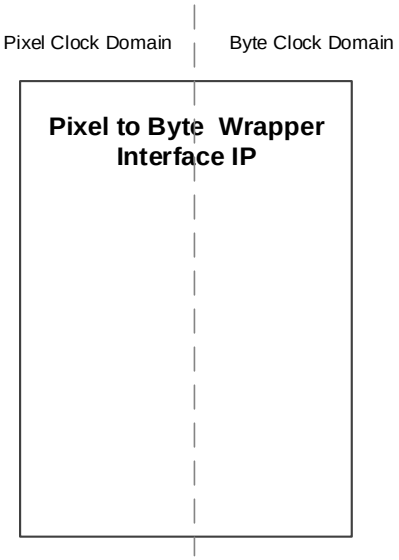

Figure 2.14. Clock Domain Crossing Block Diagram

Table 2.4. Clock Domain Crossing

Clock Domain Crossing	Handling Approach
Pixel Clock to Byte Clock	Parameterized Module Interfacing FIFO IP

The general formula for computing the required clocks of the IP (clocks used in the computation are in frequency domain):

$$\frac{Byte\ Clock}{Pixel\ Clock} = \frac{bits\ per\ pixel * pixel\ per\ pixclk}{no.\ of\ TX\ lane * TX\ gear}$$

Example:
IP configuration: RGB888 data type; 1 Pixel per Pixel Clock, Tx Gear 8, 4 Tx Lanes

$$\frac{Byte\ Clock}{Pixel\ Clock} = \frac{24 * 1}{4 * 8}$$

$$\frac{Byte\ Clock}{Pixel\ Clock} = \frac{3}{4}$$

IP configuration: RGB666 data type; 1 Pixel per Pixel Clock, Tx Gear 8, 1 Tx Lane

$$\frac{Byte\ Clock}{Pixel\ Clock} = \frac{18 * 1}{1 * 8}$$

$$\frac{Byte\ Clock}{Pixel\ Clock} = \frac{9}{4}$$

The internal buffer depth and width is set based on data type. Due to this, the relationship between the pixel and the byte clock must meet the requirement described by the equation above.

2.4. Design and Module Description

2.4.1. Pixel-to-Byte Wrapper Module

The *pixel2byte_wrapper.v* module instantiates pixel2byte and synchronizer modules. pixel2byte module is the core module which does pixel-to-byte packet conversion.

Synchronizers are two-level synchronizers used to sync the system reset into different clock domains before it is used in the system.

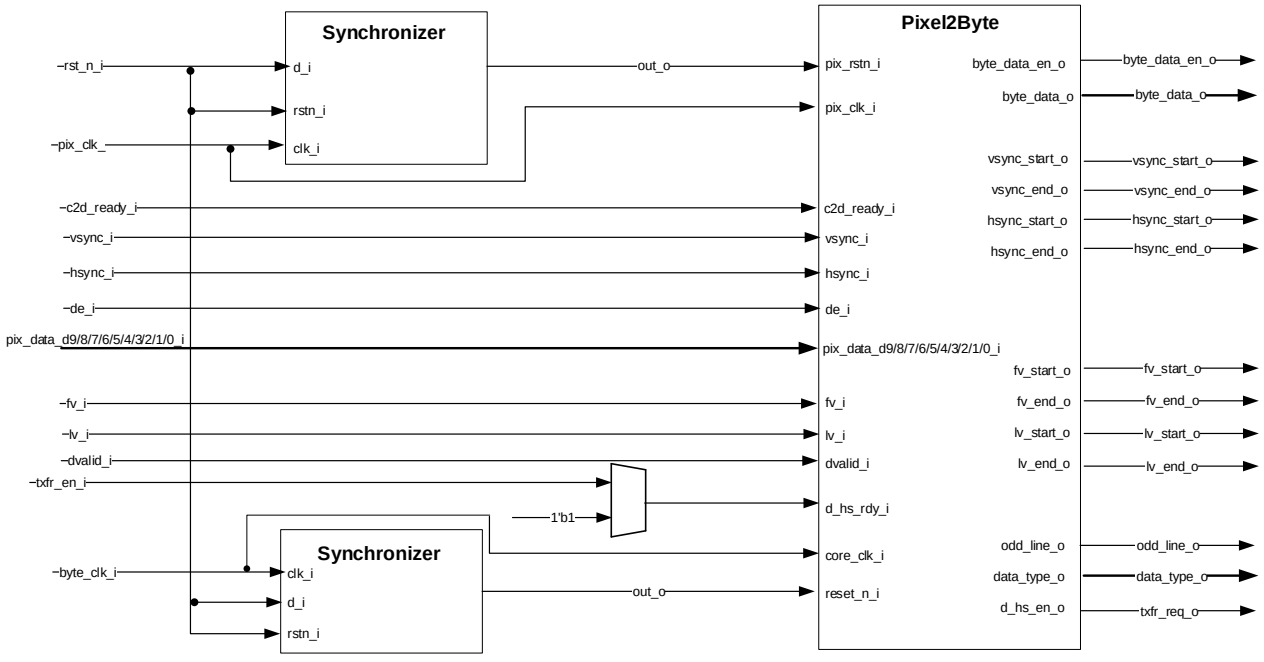

Figure 2.15. Pixel-to-Byte Wrapper Block Diagram

2.4.2. Pixel-to-Byte Module

Pixel2byte module instantiates the different wrapper modules for the pixel-to-byte converter logic for each data type.

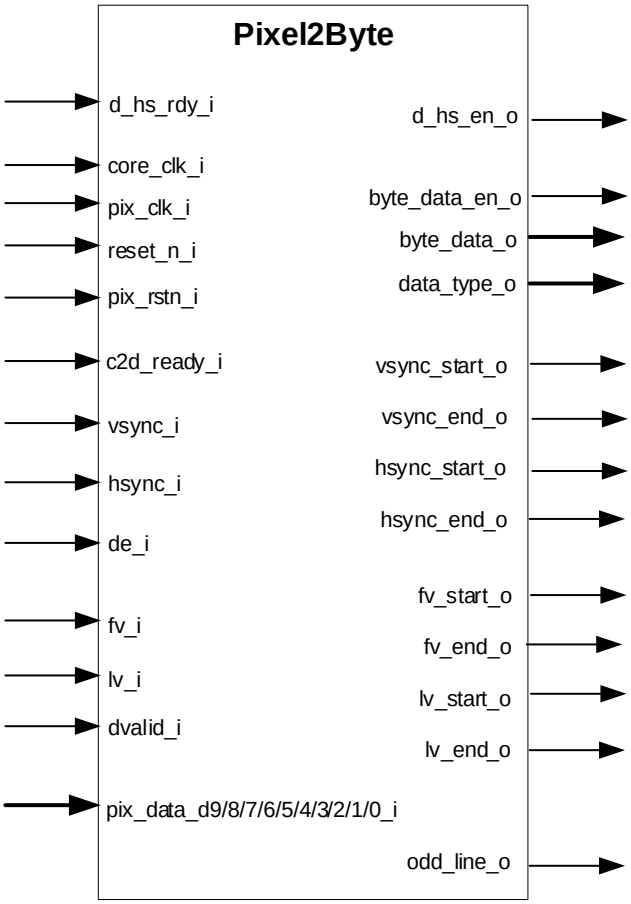

Figure 2.16. Pixel-to-Byte Block Diagram

Table 2.5. Pixel-to-Byte Pin List Summary

Port Name	Width	Dir	Type	Description
d_hs_rdy_i	1	I	LVCMOS	Enable flag from outside the IP that indicates that byte data can already be sent out from pixel2byte. 0 – do not send output byte data yet
pix_clk_i	1	I	LVCMOS	Input pixel clock.
core_clk_i	1	I	LVCMOS	Input byte clock.
reset_n_i	1	I	LVCMOS	Asynchronous active low system reset with synchronized release to byte clock. 0 – System on reset
pix_rstn	1	I	LVCMOS	Asynchronous active low system reset with synchronized release to pixel clock. 0 – System on reset
vsync_i	1	I	LVCMOS	Input vertical sync for parallel interface.
c2d_ready_i	1	I	LVCMOS	Ready signal for transmit request
hsync_i	1	I	LVCMOS	Input horizontal sync for parallel interface.
de_i	1	I	LVCMOS	Input data enable for parallel interface.
fv_i	1	I	LVCMOS	Input frame valid for parallel interface.
lv_i	1	I	LVCMOS	Input line valid for parallel interface.

Port Name	Width	Dir	Type	Description
dvalid_i	1	I	LVCMOS	Input data enable for parallel interface. Can be tied to lv_i if not used.
pixdata_d9_i	WORD_WIDTH	I	LVCMOS	Input pixel data 9. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d8_i	WORD_WIDTH	I	LVCMOS	Input pixel data 8. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d7_i	WORD_WIDTH	I	LVCMOS	Input pixel data 7. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d6_i	WORD_WIDTH	I	LVCMOS	Input pixel data 6. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d5_i	WORD_WIDTH	I	LVCMOS	Input pixel data 5. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d4_i	WORD_WIDTH	I	LVCMOS	Input pixel data 4. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d3_i	WORD_WIDTH	I	LVCMOS	Input pixel data 3. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit

Port Name	Width	Dir	Type	Description
pixdata_d2_i	WORD_WIDTH	I	LVCMOS	Input pixel data 2. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d1_i	WORD_WIDTH	I	LVCMOS	Input pixel data 1. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
pixdata_d0_i	WORD_WIDTH	I	LVCMOS	Input pixel data 0. Bus width depends on the data type selected. 24 – RGB888 18 – RGB666 12 – Raw12 10 – Raw10, YUV420/422 10-bit 8 – Raw8, YUV420/422 8-bit
vsync_start_o	1	O	LVCMOS	Pulse signal used to indicate Vsync start occurred.
vsync_end_o	1	O	LVCMOS	Pulse signal used to indicate Vsync end occurred.
hsync_start_o	1	O	LVCMOS	Pulse signal used to indicate Hsync start occurred.
hsync_end_o	1	O	LVCMOS	Pulse signal used to indicate Hsync end occurred.
fv_start_o	1	O	LVCMOS	Pulse signal used to indicate Frame valid start occurred.
fv_end_o	1	O	LVCMOS	Pulse signal used to indicate Frame valid end occurred.
lv_start_o	1	O	LVCMOS	Pulse signal used to indicate Line valid start occurred.
lv_end_o	1	O	LVCMOS	Pulse signal used to indicate Line valid end occurred.
odd_line_o	1	O	LVCMOS	Used to indicate if current line if odd or even; used for YUV420 data type. 0 – Even line 1 – Odd line
byte_data_en_o	1	O	LVCMOS	Indicates valid output byte data.
byte_data_o	4*DATA_WIDTH	O	LVCMOS	Output byte data.
data_type_o	6	O	LVCMOS	Indicates data type.
d_hs_en_o	1	O	LVCMOS	Transmission request. When asserted, it indicates that valid data (HSYNC, VSYNC, DE) is received and IP is ready to process the data.

2.4.3. Synchronizer Module

Synchronizer is a two-level synchronizer used to sync the input data into a different clock domain. In the design, this is used to synchronize the system reset into different clock domains before it is used in the system.

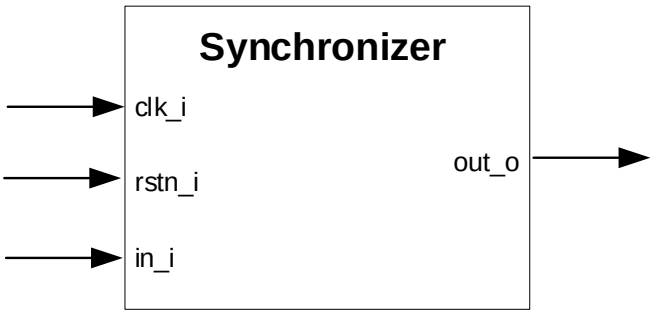

Figure 2.17. Synchronizer Block Diagram

Table 2.6. Synchronizer Pin List Summary

Port Name	Width	Dir	Type	Description
clk_i	1	I	LVCMOS	Input clock. Input data is synchronized with this clock.
rstn_i	1	I	LVCMOS	Asynchronous active low reset.
in_i	1	I	LVCMOS	Input data.
out_o	1	O	LVCMOS	Output data.

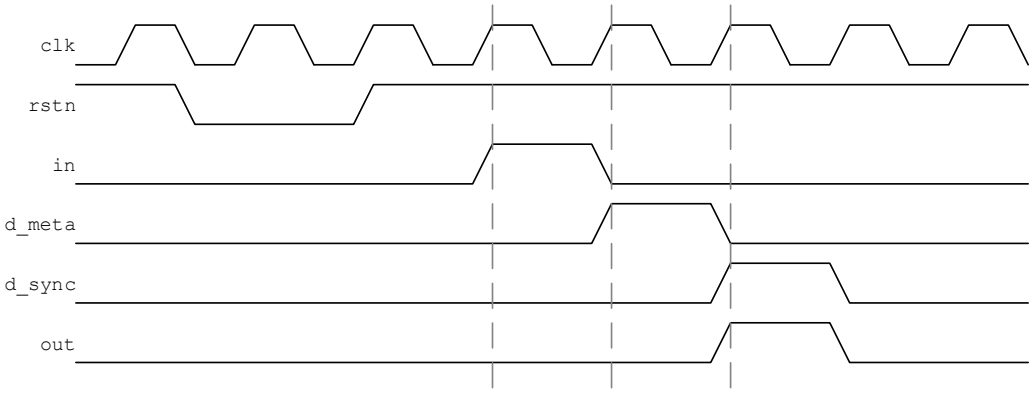

Figure 2.18. Synchronizer Timing Diagram

3. Configuration Settings

Table 3.1 lists the parameters used to generate the Pixel-to-Byte Converter IP. All parameters are either set automatically or input in the interface during the Pixel-to-Byte Converter IP generation.

Table 3.1. Pixel-to-Byte Converter IP Interface Parameter Settings

Parameter	Attribute	Options	Description
Tx Interface	User-Input	DSI, CSI-2	Set the Tx interface.
Number of Tx Lanes	User-Input	1, 2, 4	Set the target number of MIPI D-PHY Tx lanes.
Tx Gear	User-Input	8, 16	Specify the target Tx gearing.
Data Type	User-Input	RGB888, RGB666, RAW8, RAW10, RAW12, YUV420 8-bit, YUV422 8-bit, YUV420 10-bit, YUV422 10-bit	Specify the data type depending on the Data Format selected
Number of Input Pixels	User-Input	1,2,4,6,8,10	Specify the number of input pixels per pixel clock.

4. IP Generation and Evaluation

This section provides information on how to generate the Lattice Pixel-to-Byte Converter IP code using the Lattice Diamond Clarity Designer and how to run simulation, synthesis, and hardware evaluation.

4.1. Licensing the IP

The Pixel-to-Byte Converter IP is available free of charge, but an IP-specific license is required to enable full, unrestricted use of the Pixel-to-Byte Converter IP in a complete, top level design.

Request your license by going to the link http://www.latticesemi.com/en/Support/Licensing and request the free Lattice Diamond license. In this form, select the desired CrossLink/CrossLinkPlus IP for your design.

You may download and generate the Pixel-to-Byte Converter IP and fully evaluate the IP through functional simulation and implementation (synthesis, map, place and route) without an IP license. The Pixel-to-Byte Converter IP also supports Lattice's IP hardware evaluation capability, see the Hardware Evaluation section for further details.

HOWEVER, THE IP LICENSE IS REQUIRED TO ENABLE TIMING SIMULATION, TO OPEN THE DESIGN IN DIAMOND EPIC TOOL, OR TO GENERATE BITSTREAMS THAT DO NOT INCLUDE THE HARDWARE EVALUATION TIMEOUT LIMITATION.

4.2. Getting Started

The Pixel-to-Byte Converter IP is available for download from the Lattice IP Server using the Clarity Designer tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP has been installed, the IP is available in the Clarity Designer user interface as shown in Figure 4.1.

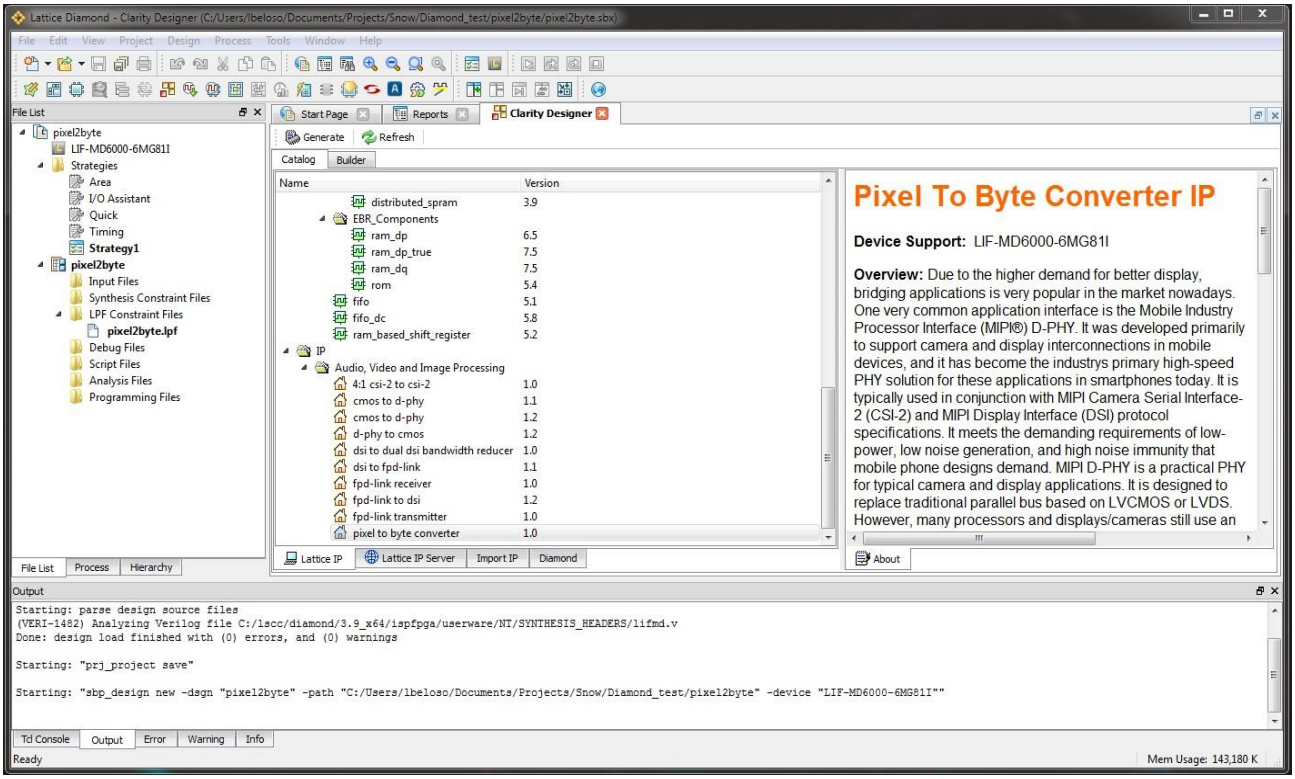

Figure 4.1. Clarity Designer Window

4.3. Generating IP in Clarity Designer

The Clarity Designer tool is used to customize modules and IPs and place them into the device architecture. Besides configuration and generation of modules and IPs, Clarity Designer can also create a top module template in which all generated modules and IPs are instantiated.

The procedure for generating Pixel-to-Byte Converter IP in Clarity Designer is described below.

Clarity Designer is started from the Diamond design environment.

To start Clarity Designer:

1. Create a new Diamond project for CrossLink or CrossLinkPlus family devices.
2. From the Diamond main window, choose **Tools > Clarity Designer**, or click  in Diamond toolbox. The **Clarity Designer** project dialog box is displayed.
3. Select and/or fill out the following items as shown in Figure 4.2.
   - **Create new Clarity design** – Select this to create a new Clarity Design project directory in which the Pixel-to-Byte Converter IP is generated.
   - **Design Location – Clarity Design project directory path.**
   - **Design Name** – Clarity Design project name.
   - **HDL Output** – Hardware Description Language Output Format (Verilog HDL).

   The Clarity Designer project dialog box also allows you to open an existing Clarity Designer project by selecting the following:
   - **Open Clarity design** – Open an existing Clarity Design project.
   - **Design File** – Name of existing Clarity Design project file with .sbx extension.
4. Click **Create**. A new Clarity Designer project is created.

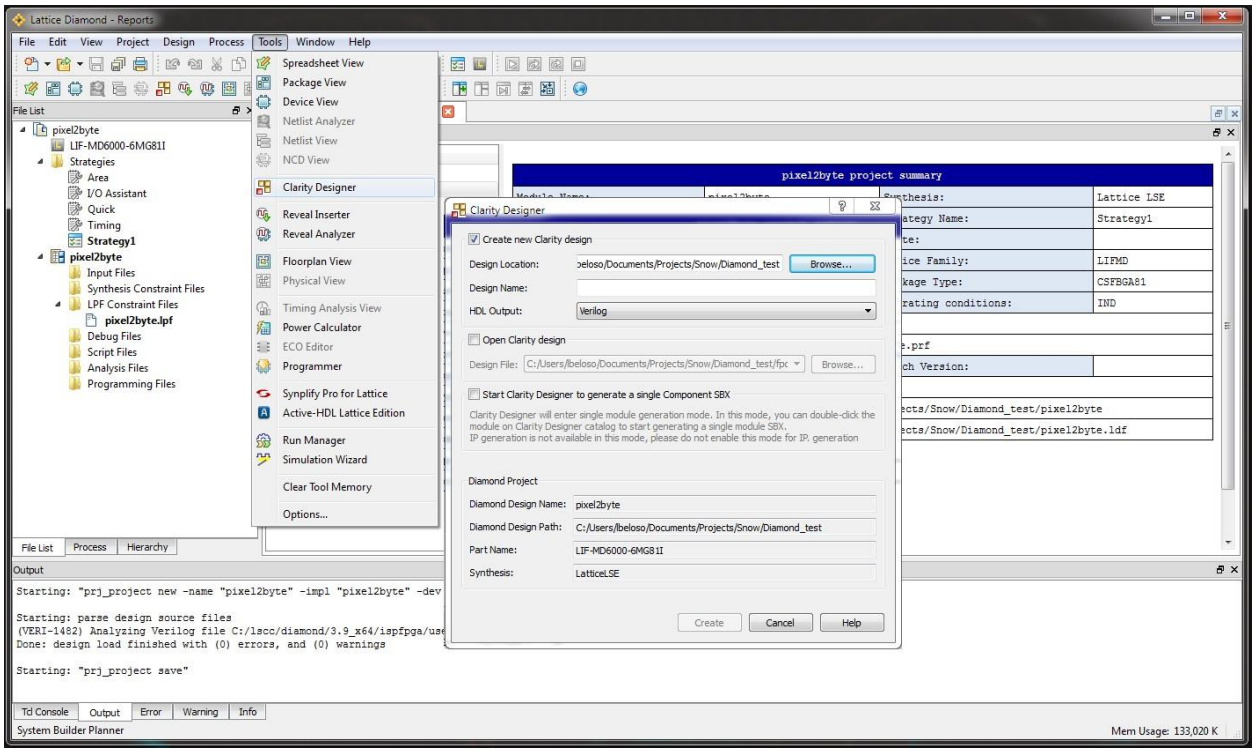

Figure 4.2. Starting Clarity Designer from Diamond Design Environment

To configure Pixel-to-Byte Converter IP in Clarity Designer:

1. Double-click **Pixel to Byte Converter** in the IP list of the System Catalog view. The **pixel to byte converter** dialog box is displayed as shown in Figure 4.3.

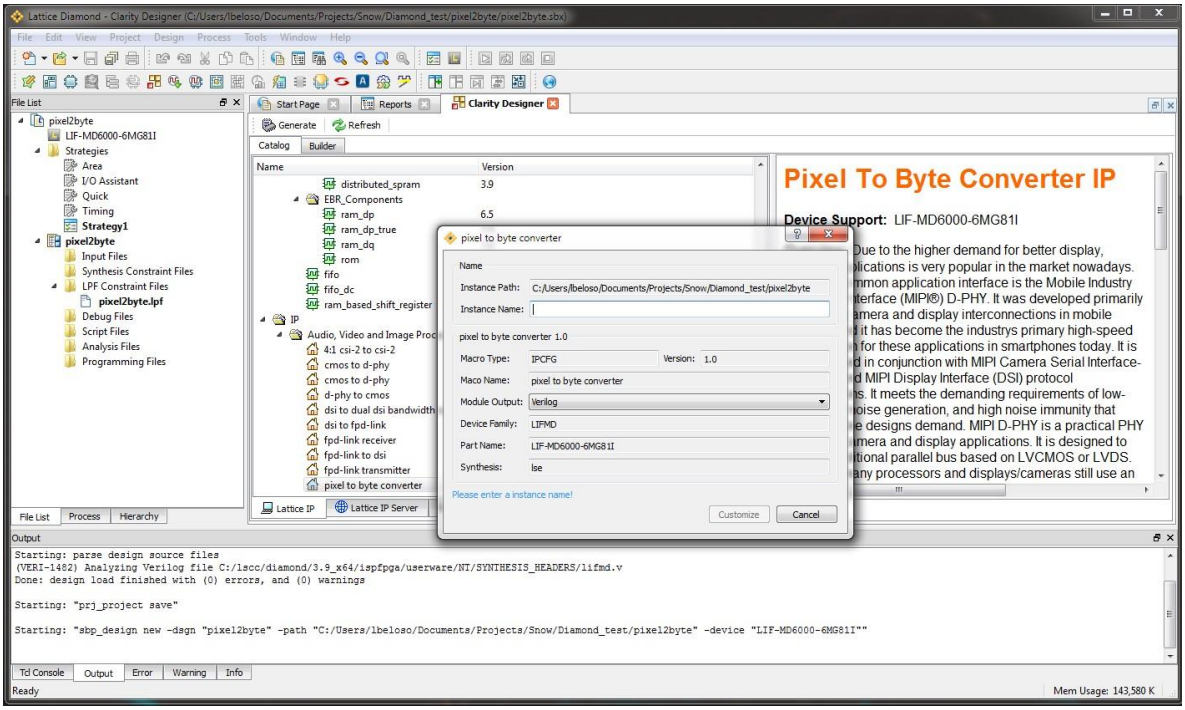

Figure 4.3. Configuring Pixel-to-Byte Converter IP in Clarity Designer

2. Enter the **Instance Name**.
3. Click the **Customize** button. An IP configuration user interface is displayed as shown in Figure 4.4. From this dialog box, you can select the IP configuration specific to your application.
4. Input valid values in the required fields in the **Configuration** tab.
5. After selecting the required parameters, click the **Configure** button.
6. Click **Close**.
7. Click Generate in the toolbox. Clarity Designer generates all the IPs and modules, and creates a top module to wrap them.

For detailed instructions on how to use the Clarity Designer, refer to the Lattice Diamond software user guide.

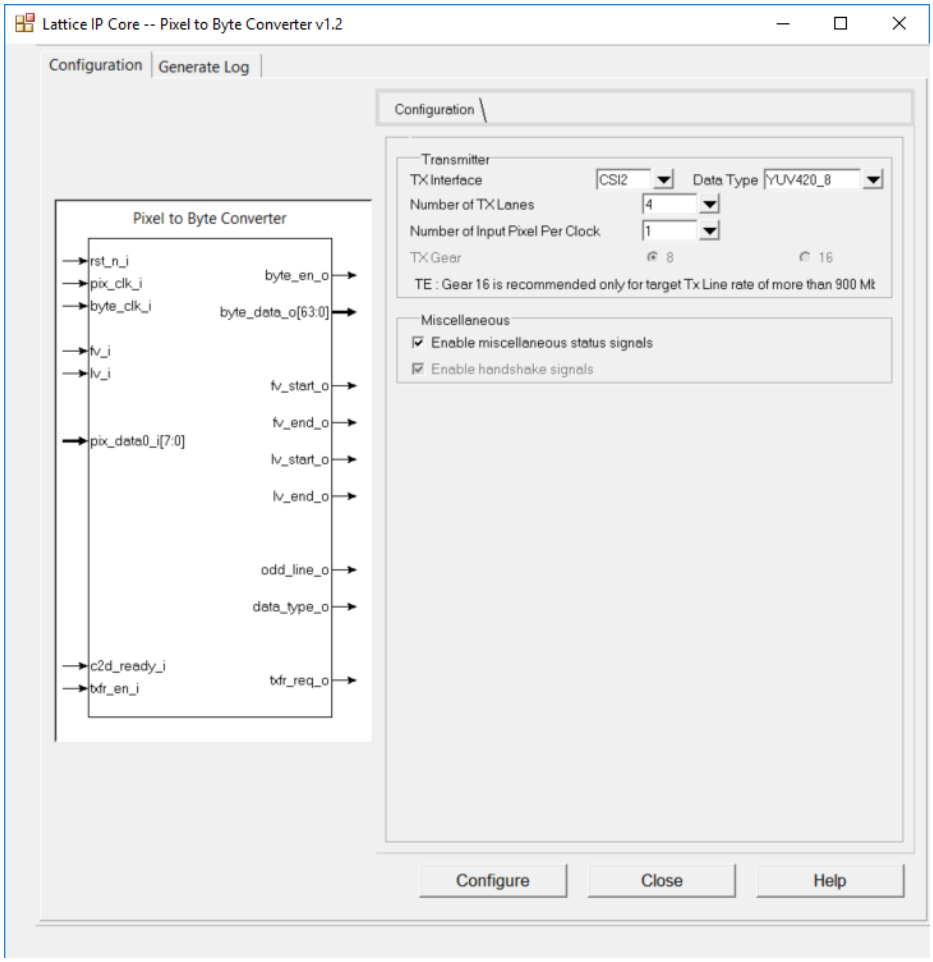

Figure 4.4. Configuration Tab in IP User Interface

4.4. Generated IP Directory Structure and Files

Figure 4.5 shows the directory structure of generated IP and supporting files.

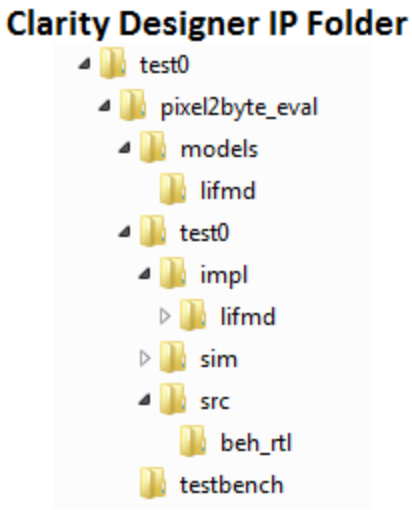

Figure 4.5. Pixel-to-Byte Converter IP Directory Structure

The design flow for the IP created with Clarity Designer uses a post-synthesized module (NGO) for synthesis and a protected model for simulation. The post-synthesized module and protected model are customized when you configure the IP and are created automatically when the IP is generated.

Table 4.1 provides a list of key files and directories created by Clarity Designer and how they are used. The post-synthesized module (NGO), the protected simulation model, and all other files are also generated based on your configuration and provided as examples to use or evaluate the IP.

Table 4.1. Files Generated in Clarity Designer

File	Description
<instance_name>.v	Verilog top-level module of Pixel-to-Byte Converter IP used for both synthesis and simulation.
*<instance_name>*_*.v	Verilog submodules for simulation. Files that do not have equivalent black box modules are also used for synthesis.
*<instance_name>*_*_beh.v	Protected Verilog models for simulation.
*<instance_name>*_* _bb.v	Verilog black box modules for synthesis.
*<instance_name>*_*.ngo	User interface configured and synthesized modules for synthesis.
*<instance_name>*_params.*v*	Verilog parameters file which contains required compiler directives to successfully configure IP during synthesis and simulation.
<instance_name>.lpc	Lattice Parameters Configuration file. This file records all the IP configuration options set through Clarity Designer. It is used by IP generation script to generate configuration-specific IP. It is also used to reload parameter settings in the IP user interface in Clarity Designer when it is being reconfigured.
*<instance_name>*_inst.v/vhd	Template for instantiating the generated soft IP top-level in another user-created top module.

Aside from the files listed in the tables, most of the files required to evaluate the Pixel-to-Byte Converter IP are available under the directory *<pixel2byte_eval>*, including the simulation model. Lattice Diamond project files are also included under the folder at *<pixel2byte_eval>**<instance_name>**impl**<device_family>**<synthesis_tool>*\, where *<device_family>* can either be lifmd for CrossLink or lifmdf for CrossLinkPlus devices.

The *<instance_name>* folder (test0 folder in Figure 4.5) contains files/folders with content specific to the *<instance_name>* configuration. This directory is created by Clarity Designer each time the IP is generated and regenerated with the same file name. A separate *<instance_name>* directory is generated for IPs with different names, such as *<my_IP_0>,**<my_IP_1>*, and others.

The folder *<instance_name>,* the *pixel2byte_eval* and sub directories provide files supporting Pixel-to-Byte Converter IP evaluation that includes files/folders with content that is constant for all configurations of the Pixel-to-Byte Converter IP. The *pixel2byte_eval* directory is created by Clarity Designer the first time the IP is generated, when multiple Pixel-to-Byte Converter IPs are generated in the same root directory and updated each time the IP is regenerated.

You can use the prebuilt Diamond projects provided at *<project_root>**pixel2byte_eval**<instance_name>**impl**<device_family>**<synthesis_tool>*\ to evaluate the implementation (synthesis, map, place and route) of the IP in Lattice Diamond tool. The *src* directory contains the behavioral models of the black-boxed modules and the *models* directory provides library elements.

4.5. Running Functional Simulation

To run simulations using Active-HDL:

1. Under the **Tools** menu, select **Active-HDL**.
2. Modify the tb_params.v file located in *\<project_dir>\pixel2byte_eval\testbench*.

   Update testbench parameters to customize data size and/or other settings.

   // SIP_PCLK - Used to set the period of the input pixel clock (in ps)

   `define SIP_PCLK 7500.0

   // SIP_BCLK - Used to set the period of the input byte clock (in ps)

   `define SIP_BCLK 6667.0

   See additional information about testbench parameters in Table 4.2. It is required to update the SIP_PCLK, SIP_BCLK, and NUM_BYTES to be consistent with the target for simulations.
3. In Active-HDL window, under the **Tools** tab, select **Execute Macro**.
4. Select the .do file *\<project_dir>\pixel2byte_eval\<instance_name>\sim\aldec*run.do*.
5. Click **OK**.
6. Wait for the simulation to finish.

Table 4.2 lists the testbench directives which can be modified by setting the define in the *tb_params.v* file.

Table 4.2. Testbench Compiler Directives

Compiler Directive	Description
SIP_PCLK	Sets the period of the input pixel clock (in ps). Must include at least 1 decimal.
SIP_BCLK	Sets the period of the input byte clock (in ps). Must include at least 1 decimal.
NUM_FRAMES	Sets the number of video frames.
NUM_LINES	Sets the number of lines per frame.
HFRONT	Number of cycles before HSYNC signal asserts (Horizontal Front Blanking).
HPULSE	Number of cycles HSYNC signal asserts.
HBACK	Number of cycles after HSYNC signal asserts (Horizontal Rear Blanking).
VFRONT	Number of cycles before VSYNC signal asserts (Vertical Front Blanking).
VPULSE	Number of cycles VSYNC signal asserts.
VBACK	Number of cycles after VSYNC signal asserts (Vertical Rear Blanking).
NUM_BYTES	Number of bytes sent per line.

4.6. Simulation Strategies

This section describes the simulation environment which demonstrates basic Pixel-to-Byte Converter functionality. Figure 4.6 shows the block diagram of simulation environment.

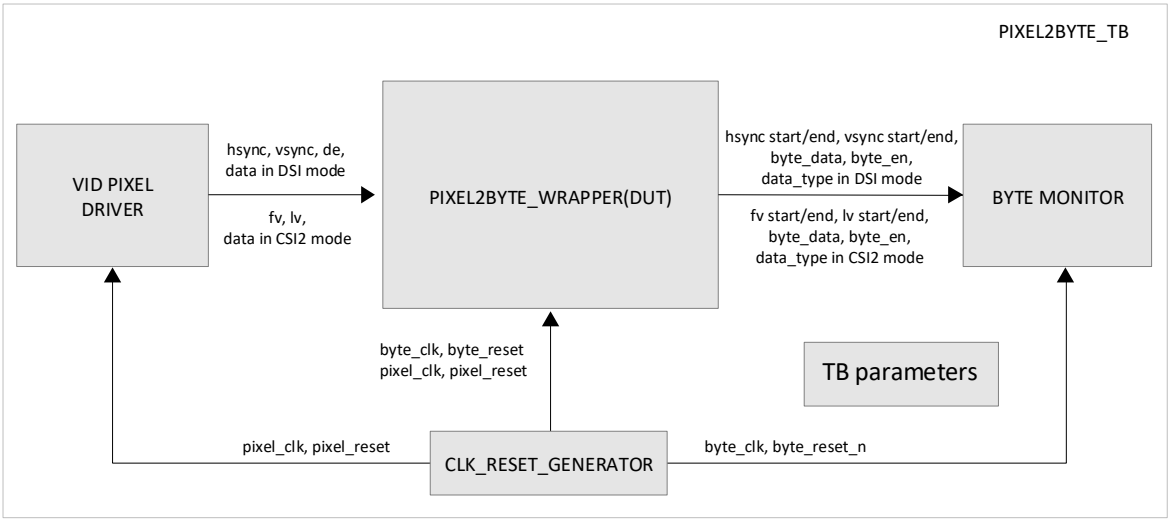

Figure 4.6. Simulation Environment Block Diagram

4.7. Simulation Environment

The simulation environment is made up of a pixel clock domain input driver instance connected to the input of Pixel-to-Byte Converter IP instance in the testbench. The pixel clock domain input driver is configured based on Pixel-to-Byte Converter IP configurations and testbench parameters. After reset, the testbench drives video pixel data in either DSI/CSI2 mode based on configuration. Output data of the DUT is monitored by Byte monitor.

Input pixel data and output byte data are logged into *input_data.log* and *output_data.log* files, respectively. You can compare these files to check if data is being transmitted properly. See the Running Functional Simulation section for details on how to set testbench parameters.

Figure 4.7 shows an example simulation of CSI-2 configuration.

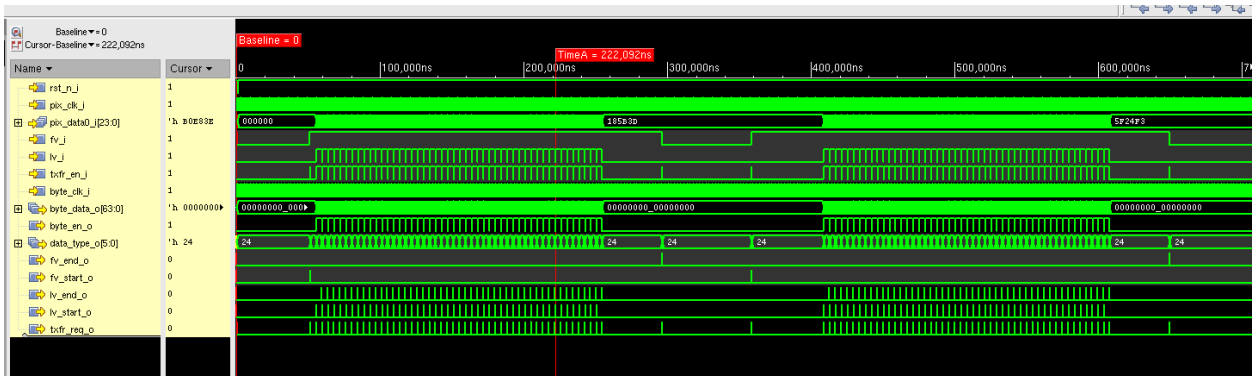

Figure 4.7. CSI-2 Configuration

Figure 4.8 shows an example simulation of DSI configuration.

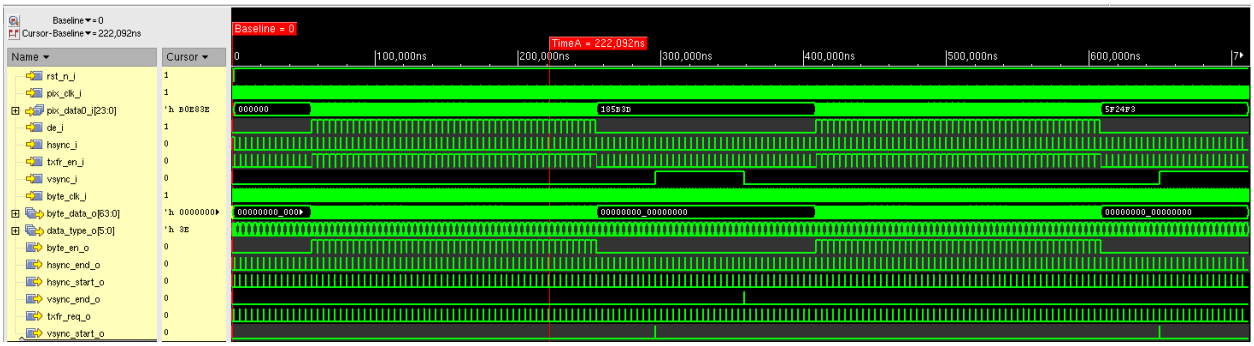

Figure 4.8. DSI Configuration

4.8. Instantiating the IP

The core modules of the Pixel-to-Byte Converter IP are synthesized and provided in NGO format with black box Verilog source files for synthesis. A Verilog HDL source file named *<instance_name>_pixel2byte.v* is the black box core module. The top-level file *<instance_name>.v* instantiates *<instance_name>_pixel2byte.v* and *<instance_name>_synchronizer.v*.

A Verilog instance template *<instance_name>_inst.v* or VHDL instance template *<instance_name>_inst.vhd* is also provided as a guide if the design is to be included in another top level module.

You do not need to instantiate the IP instances one by one manually. The top-level file and other Verilog source files are provided in *<project_dir>*. These files are refreshed each time the IP is regenerated.

4.9. Synthesizing and Implementing the IP

In Clarity Designer, the Clarity Designer project file (.sbx) is added to Lattice Diamond as a source file after all IPs are generated. Note that default Diamond strategy (.sty) and default Diamond preference file (.lpf) are used. When using the .sbx approach, import the recommended strategy from *<project_dir>**pixel2byte_eval**<instance_name>**impl**<device_family>**lse* or *<project_dir>**pixel2byte_eval**<instance_name>**impl**<device_family>**synplify* directories and set it as active strategy. A sample preference file (.lpf) is also included in the directory. All required files are invoked automatically. You can directly synthesize, map and place/par the design in the Diamond design environment after the cores are generated.

Push-button implementation of this top-level design with either Synplify or Lattice Synthesis Engine is supported via the Diamond project files *<instance_name>_top.ldf* which is located in *<project_dir>**pixel2byte_eval**<instance_name>**impl**<device_family>**<synthesis_tool>*\ directory.

To use the pre-built Diamond project files:

1. Choose **File > Open > Project**.
2. In the **Open Project** dialog box browse to *<project_dir>**pixel2byte_eval**<instance_name>**impl**<device_family>**<synthesis_tool>*\.
3. Select and open *<instance_name>_top.ldf*. At this point, all of the files needed to support top-level synthesis and implementation are imported to the project.
4. Select the **Process** tab in the left-hand user interface window.
5. Implement the complete design via the standard Diamond user interface flow.

4.10. Hardware Evaluation

The Pixel-to-Byte Converter IP supports Lattice IP hardware evaluation capability. You can create versions of the IP that operate in hardware for a limited period of time without requiring the request of an IP license. It may also be used to evaluate the IP in hardware in user-defined designs.

4.10.1. Enabling Hardware Evaluation in Diamond

Choose Project > Active Strategy > Translate Design Settings. The hardware evaluation capability may be enabled or disabled in the Strategy dialog box. It is enabled by default.

4.11. Updating/Regenerating the IP

The Clarity Designer user interface allows you to update the local IPs from the Lattice IP server. The updated IP can be used to regenerate the IP in the design. To change the parameters of the IP used in the design, the IP must also be regenerated.

4.11.1. Regenerating an IP in Clarity Designer

To regenerate IP in Clarity Designer:

1. In the **Builder** tab, right-click the IP instance to be regenerated and select **Config** from the menu as shown in Figure 4.9.

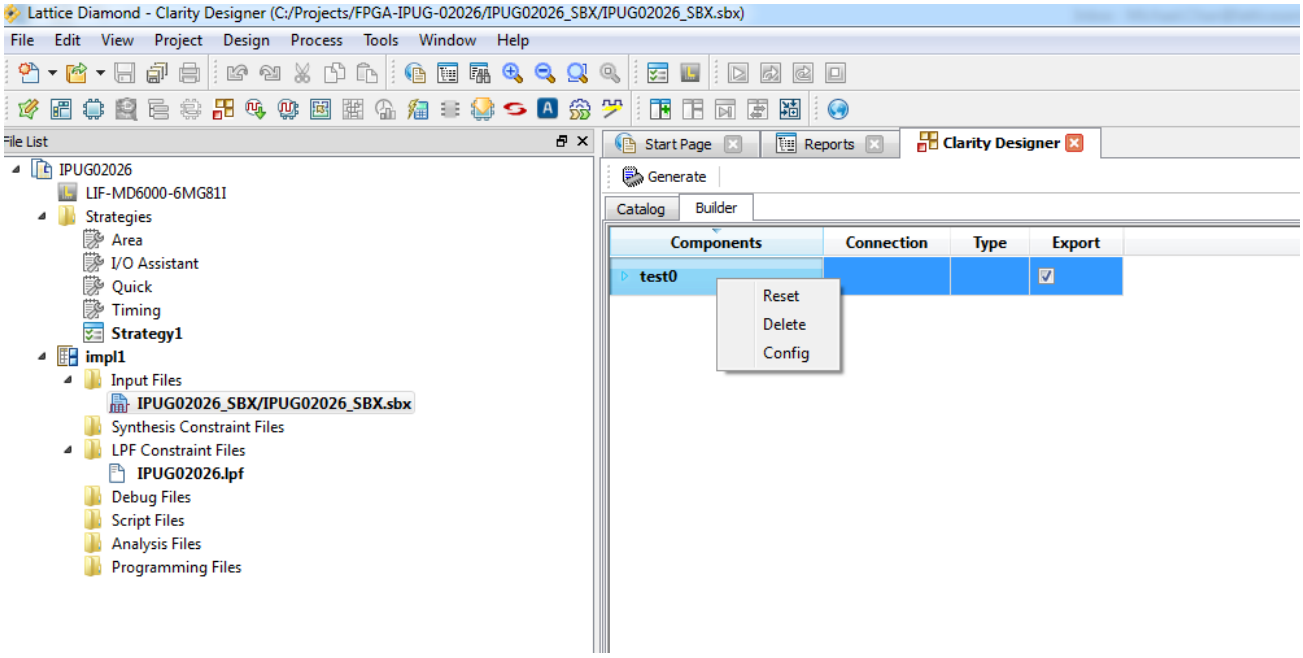

Figure 4.9. IP Regeneration in Clarity Designer

2. The IP Configuration user interface is displayed. Change the parameters as required and click the **Configure** button.
3. Click 

in the toolbox. Clarity Designer regenerates all the instances which are reconfigured.

References

For more information about CrossLink and CrossLinkPlus devices, refer to CrossLink Family Data Sheet (FPGA-DS-02007) and CrossLinkPlus Family Data Sheet (FPGA-DS-02054).

Software documentation:

- Clarity Designer User Manual
- Lattice Diamond User Guide

For further information on interface standards, refer to:

- MIPI Alliance Specification for D-PHY, version 1.1, November 7, 2011, www.mipi.org
- MIPI Alliance Specification for Display Serial Interface, version 1.1, November 22, 2011, www.mipi.org
- MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2), version 1.1, July 18, 2012, www.mipi.org

Technical Support Assistance

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

Appendix A. Resource Utilization

Table A.1. lists resource utilization information for Lattice CrossLink FPGA using the Pixel-to-Byte Converter IP.

Clarity Designer is the Lattice IP configuration utility, and is included as a standard feature of the Diamond tool. For details about the usage of Clarity Designer, refer to the Clarity Designer and Diamond help system.

For more information on the Diamond design tools, visit the Lattice web site at www.latticesemi.com/Products/DesignSoftwareAndIP.

Table A.1. Resource Utilization[1]

IP User-Configurable Parameters	Slices	LUTs	Registers	sysMEM EBRs	Target f_{MAX} (MHz)[2]
1 pixel, Gear 8, 4 lanes, RGB888 configuration	189	184	266	4	150
1 pixel, Gear 8, 4 lanes, RGB666 configuration	414	488	434	2	150
1 pixel, Gear 8, 4 lanes, RAW12 configuration	215	159	336	4	150
1 pixel, Gear 8, 4 lanes, RAW10 configuration	504	563	635	4	150
1 pixel, Gear 8, 4 lanes, YUV420 8-bit configuration	202	115	328	4	150

Notes:

1. Performance and utilization data target an LIF-MD6000-6MG81I device using Lattice Diamond 3.9 and Lattice Synthesis Engine software. Performance may vary when using a different software version or targeting a different device density or speed grade within the CrossLink family. This does not show all possible configurations of the Pixel-to-Byte Converter IP.
2. The f_{MAX} values are based on byte or pixel clock, whichever is faster based on configuration.

Appendix B. What is Not Supported

- The IP does not support configuration through registers.
- The IP has the following limitation:
- The number of total bytes transmitted per line (pixel_width*total number of pixels per line/8) must follow byte divisibility as listed in Table B.1.
- The hsync_i is synchronized in the byte clock domain using two sequential flip-flops. Because of this, it is possible to miss the hsync_i if its duration is less than one byteclock period. You need to adjust the increase hsync pulse duration in order to detect the hsync start and end.

Table B.1. Number of Bytes Limitation

Tx Lanes	Tx Gear	Byte Divisibility
4	8	4
	16	8
2	8	2
	16	4
1	8	1
	16	2

Revision History

Revision 1.3, IP Version 1.3, March 2020

Section	Change Summary
Functional Description	• Added Supported Configurations subsection to move Table 2.3. • Updated Figure 2.1, Figure 2.13, Figure 2.15, Figure 2.16, and Figure 2.17. • Updated Table 2.1, Table 2.5, and Table 2.6. • Updated Interface and Timing Diagram content. • Removed Table 2.6 Pixel-to-Byte Parameter List.
Configuration Settings	• Changed section name from Compiler Directives and Parameter Settings to *Configuration Settings*. • Removed RTL Compiler Directives and Parameter Settings section.
IP Generation and Evaluation	Updated Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4.

Revision 1.2, IP Version 1.3, October 2019

Section	Change Summary
Disclaimer	Newly added section.
All	Added CrossLinkPlus device support. Minor adjustments in style and formatting.
References	Updated.

Revision 1.1, IP Version 1.0, April 2019

Section	Change Summary
Introduction	Specified that this user guide can be used for IP design versions 1.x.
IP Generation and Evaluation	In Licensing the IP, modified the instructions for requesting free license.
Appendix B. What is Not Supported	Corrected formatting and table number.
Revision History	Updated revision history table to new template.
All	Minor adjustments in style and formatting.

Revision 1.0, IP Version 1.0, July 2017

Section	Change Summary
All	Initial release.

www.latticesemi.com