



SubLVDS to MIPI CSI-2 Image Sensor Bridge

Reference Design

FPGA-RD-02061-1.1

September 2019

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

Acronyms in This Document	5
Supported Device and IP	6
1. Introduction	7
1.1. Features	7
1.2. Block Diagram and Clock Distribution	7
1.3. RX and TX Permutations	8
2. Parameters and Port List	10
2.1. Synthesis Directives	10
2.2. Simulation Directives	12
2.3. Top-Level I/O	13
3. Design and Module Description	15
3.1. sensor_sync	15
3.2. rx_sublvds	15
3.3. trim_ctrl	18
3.4. pixel2byte	21
3.5. lane_ctrl	23
3.5.1. Communication Control	23
3.5.2. Bus Width and Byte Data Assignment Conversion	24
3.6. tx_dphy	25
3.7. int_gpll	26
3.8. i2c_slave	27
4. Design and File Modification by User	30
4.1. Top-level RTL	30
4.2. Pixel-to-Byte IP	30
4.3. TX D-PHY IP	30
5. Design Simulation	31
6. Known Limitations	34
7. Design Package and Project Setup	35
8. Resource Utilization	37
References	38
Technical Support Assistance	38
Revision History	39

Figures

Figure 1.1. SubLVDS to MIPI CSI-2 Image Sensor Bridge Block Diagram	7
Figure 1.2. Bandwidth and Clock Frequency Calculator	9
Figure 3.1. rx_sublvds IP Creation in Clarity Designer #1	15
Figure 3.2. rx_sublvds IP Creation in Clarity Designer #1	16
Figure 3.3. SubLVDS Input Global Timing (RAW10, 10 Lanes)	17
Figure 3.4. trim_ctrl Global Timing (RAW10, 10 Lanes)	19
Figure 3.5. Trimming in the Beginning of the Line (RAW10, 10 Lanes)	19
Figure 3.6. pixel2byte IP Creation in Clarity Designer	21
Figure 3.7. Global Timing of pixel2byte	22
Figure 3.8. Line Transactions of pixel2byte	22
Figure 3.9. Global Timing of lane_ctrl	23
Figure 3.10. Handshake to Transfer Short Packet	23
Figure 3.11. Handshake to Transfer Long Packet	24
Figure 3.12. Byte Data Assignment Conversion by lane_ctrl	24
Figure 3.13. tx_dphy IP Creation in Clarity Designer	25
Figure 3.14. GPLL IP Creation	26
Figure 3.15. I ² C IP Creation #1	27
Figure 3.16. I ² C IP Creation #2	28
Figure 3.17. I ² C IP Creation #3	28
Figure 5.1. Script File Modification	31
Figure 5.2. Global Timing of 4-Lane RX and 2-Lane TX	32
Figure 5.3. Global Timing with Sensor Slave Mode	32
Figure 5.4. Global Timing with Sensor Slave Mode	33
Figure 7.1. Directory Structure	35
Figure 7.2. Project Files	36

Tables

Table 1.1. RX and TX Permutations	8
Table 2.1. Synthesis Directives	10
Table 2.2. Simulation Directives	12
Table 2.3. Simulation Directives	13
Table 3.1. Sync Code Details	18
Table 3.2. Granularity of h_active_unit and WC	20
Table 3.3. Byte Data Reallocation	24
Table 3.4. I ² C Slave Register Map	29
Table 8.1. Resource Utilization Examples	37

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AP	Application Processor
CMOS	Complementary Metal Oxide Semiconductor
CSI-2	Camera Serial Interface 2
DDR	Double Data Rate
EAV	End of Active Video
FV	Frame Valid
GPLL	General Purpose PLL
HS	High Speed
I ² C	Inter-Integrated Circuit
ISP	Image Signal Processor
LP	Low Power
LV	Line Valid
LVDS	Low Voltage Differential Signal
MIPI	Mobile Industry Processor Interface
OSCI	Internal Oscillator
PLL	Phase Locked Loop
RD	Reference Design
RX	Receiver
SAV	Start of Active Video
TX	Transmitter
WC	Word Count
XHS	Horizontal Sync Pulse
XVS	Vertical Sync Pulse

Supported Device and IP

This reference design supports following devices with IP versions shown below.

Device Family	Part Number	Compatible IP
CrossLink	LIF-MD6000 LIA-MD6000	SubLVDS Receiver IP version 1.1 and 1.2 Pixel-to-Byte Converter IP version 1.1 and 1.2 D-PHY Transmitter IP version 1.1 and 1.2
CrossLinkPlus	LIF-MDF6000	SubLVDS Receiver IP version 1.2 Pixel-to-Byte Converter IP version 1.2 D-PHY Transmitter IP version 1.2

CrossLink refers to both CrossLink and CrossLinkPlus in this document unless noted.

1. Introduction

Many Image Signal Processors (ISP) or Application Processors (AP) use the Mobile Industry Processor Interface (MIPI®) Camera Serial Interface 2 (CSI-2) standard for image sensor inputs. However, some high-resolution CMOS image sensors use a proprietary SubLVDS output format.

The Lattice Semiconductor SubLVDS to MIPI CSI-2 Image Sensor Bridge reference design for CrossLink™ devices solves the mismatch between SubLVDS output image sensor and an ISP/AP using CSI-2 interface.

1.1. Features

- Supports 4-, 6-, 8-, or 10-lane SubLVDS input to 1-, 2-, or 4-lane MIPI CSI-2 output
- Supports input lane bandwidth of up to 1.2 Gbps (in the case of 4-lane configuration) and output lane bandwidth of up to 1.5 Gbps
- Image cropping option
- Dynamic parameter setting through I²C

1.2. Block Diagram and Clock Distribution

Figure 1.1 shows the block level diagram of the SubLVDS to MIPI CSI-2 Image Sensor Bridge reference design. It contains three major IPs and interfacing modules between them. Image data from the sensor come in along with the SubLVDS clock in double data rate (DDR) fashion. This clock is divided by 4 or 8 to generate pixel clock according to RX Gear. Pixel clock is fed to trim_ctrl and Pixel-to-Byte IP modules. On the other hand, pixel clock is fed to TX D-PHY IP as a reference clock and TX D-PHY IP creates MIPI clock using its internal PLL. MIPI clock is divided by 8 or 16 to generate byte clock and byte clock is fed to Pixel-to-Byte IP and lane_ctrl module. GPLL is required in case that pixel clock cannot drive TX D-PHY PLL directly due to the input frequency requirement of D-PHY PLL. In some configurations, the data bus going to TX D-PHY is half of the data bus coming out from Pixel-to-Byte IP. In that case, the byte clock generated by TX D-PHY must be 2x of the byte clock used in Pixel-to-Byte IP and the original byte clock (hs_byte_clk) is divided by two and fed to Pixel-to-Byte IP and lane_ctrl. The lane_ctrl module takes care of bus width difference. When the image sensor is in slave mode, FPGA has to feed the sync signals to the sensor. In that case, the sensor_sync module takes the clock from the sensor to generate sync signals. I²C slave module is optional and the internal oscillator is used to feed the clock to I²C slave module in this example.

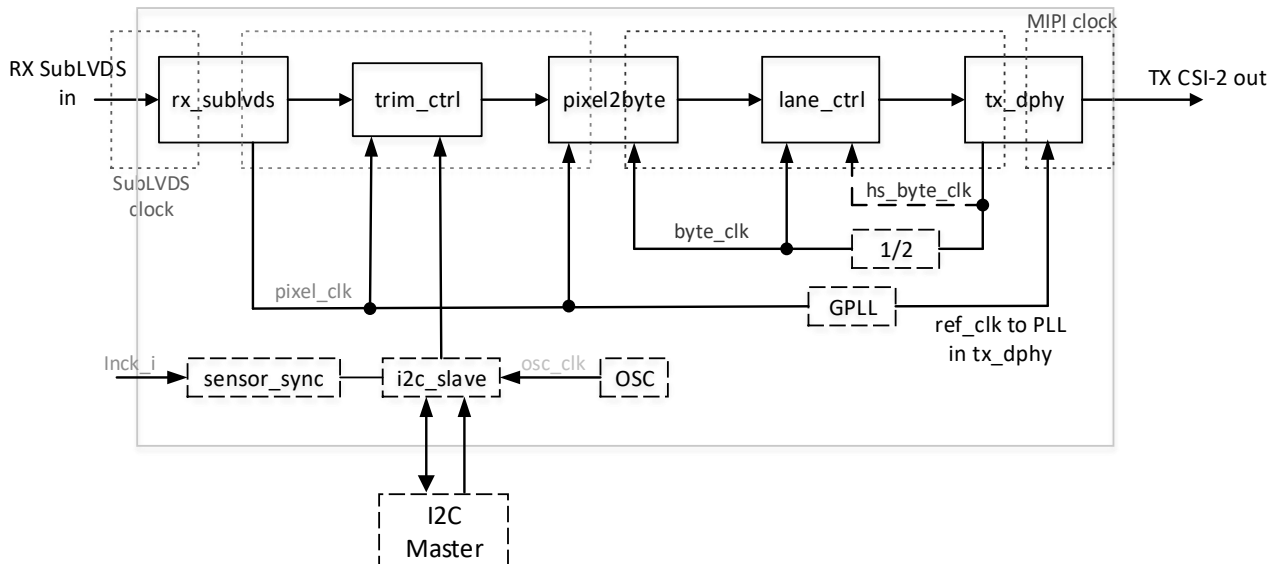


Figure 1.1. SubLVDS to MIPI CSI-2 Image Sensor Bridge Block Diagram

1.3. RX and TX Permutations

Table 1.1 shows the available permutations of RX and TX configurations. Same permutations apply to both RAW10 and RAW12. RX Gear 16 is supported only in case of 4 lanes on RX due to the limitation of Byte-to-Pixel IP. In addition, Byte-to-Pixel IP supports only 4 lanes on TX. To overcome these limitations, the different TX Gear setting is applied in D-PHY TX IP to handle 1 lane or 2 lane outputs, which requires faster byte clock ($hs_byte_clk = 2x$ or $4x$ of $byte_clk$). Currently, permutations require $4x$ of $byte_clk$ (grayed out in the table) are not supported due to the unknown necessity.

Table 1.1. RX and TX Permutations

D-PHY Type	RX Lane Count	RX Gear	TX Gear Setting in Pixel-to-Byte IP	TX Lane Count	TX Gear	hs_byte_clk Ratio Against byte_clk
RAW10 RAW12	4	8	8	4	8	—
				2	16	—
				1	16	2x
		16	16	4	16	—
				2	16	2x
				1	16	4x
	6	8	8	4	8	—
				2	16	—
				1	16	2x
	8	8	16	4	16	—
				2	16	2x
				1	16	4x
	10	8	16	4	16	—
				2	16	2x
				1	16	4x

The Excel sheet (*sublvds2csi2_clock.xlsx*) is provided to calculate the pixel clock, byte clock, and others, from RX bandwidth and other information. This sheet can be useful to configure IPs. A sample entry is shown in Figure 1.2. By setting five rows shown in the table, pixel clock, byte clock, and TX bandwidth are automatically calculated. Those can be used to configure D-PHY TX IP. In the following situations, the byte clock that comes out from D-PHY TX IP is hs_byte_clk and half of this clock is $byte_clk$ fed to Pixel-to-Byte IP (and $lane_ctrl$):

$$2 \times (\text{TX Gear setting in Pixel-to-Byte IP}) = (\text{Number of TX Lanes in D-PHY TX}) \times (\text{TX Gear in D-PHY TX}).$$

When the pixel clock frequency is not a multiple of 24 – 30 MHz, GPLL is required to generate an appropriate reference clock (a multiple of 24 – 30 MHz) to be fed to TX D-PHY IP. For example, when pixel clock is 36 MHz, GPLL can generate $2x$ of pixel clock, which is 72 MHz. This is $3x$ of 24 MHz, which TX D-PHY can accept. This new reference clock has to be a multiple of 24 – 30 MHz and below 150 MHz. You can enable GPLL instantiation by defining `USE_GPLL` in `synthesis_directive.v`, but GPLL configuration must be modified according to the pixel clock frequency and required reference clock frequency.

SubLVDS to MIPI CSI-2 Image Sensor Bridge RD Frequency Calculator

SubLVDS RX	Data Type	RAW10	
	Number of RX Lanes	10	
	RX Gear	8	16 is for 4-lane only
	RX Line Rate	600	
	Sub LVDS Clock Frequency	300	MHz
	Pixel Clock Frequency	75	MHz
Pixel2Byte	Number of TX Lanes	4	always 4
	Number of Input Pixel Per Clock	10	
	TX Gear	16	
D-PHY TX	TX Line Rate (total)	6000	Mbps
	Number of TX Lanes	4	
	TX Line Rate (per lane)	1500	Mbps (Max 1500)
	TX Gear	16	(auto setting by Clarity)
	Byte Clock Frequency	93.75	MHz
	Reference Clock Frequency	75	MHz

Set by user

no GPLL required

If Reference Clock Frequency of D-PHY TX is not a multiple of 24-30 MHz, then GPLL is required.
In that case, GPLL output must be a multiple of 24-30 MHz and below 150 MHz.

Figure 1.2. Bandwidth and Clock Frequency Calculator

2. Parameters and Port List

There are two directive files for this reference design:

- synthesis_directives.v – used for design compilation by Lattice Diamond® and for simulation.
- simulation_directives.v – used for simulation.

You can modify these directives according to your own configuration. The settings in these files must match SubLVDS RX IP, Byte-to-Pixel IP, and TX D-PHY IP settings created by Clarity Designer.

2.1. Synthesis Directives

Table 2.1 shows the synthesis directives that affect this reference design. These are used for both synthesis and simulation. Some parameter selections are restricted by other parameter settings.

Table 2.1. Synthesis Directives

Category	Directive	Remarks
Image Sensor control ¹	SENSOR_MODE_MASTER	Use <i>SLAVE</i> when sync signals (xvs_o/xhs_o) must be sent from FPGA to the image sensor. Only one of these two directives must be defined.
	SENSOR_MODE_SLAVE	
Image Sensor Sync Polarity ¹	SENSOR_SYNC_NEG	Polarity setting for sync signals to the image sensor. Only effective when SENSOR_MODE_SLAVE is defined. Only one of these two directives must be defined.
	SENSOR_SYNC_POS	
XVS (Vertical Sync) assertion period ¹	XVS_LENGTH_XHS	Select the active pulse length of xvs_o between xhs_o and one horizontal line. Only applicable in case of SENSOR_MODE_MASTER. Only one of these two directives must be defined.
	XVS_LENGTH_LINE	
Total line count ¹	V_TOTAL {value}	Total line count for one frame including blanking. Only effective when SENSOR_MODE_SLAVE is defined. Value must be 12'd10 – 12'd4095.
Total horizontal cycle ¹	H_TOTAL {value}	Total cycle count for one line including blanking in the unit of inck_i. Only effective when SENSOR_MODE_SLAVE is defined. Value must be 12'd10 – 12'd4095.
XHS (Horizontal Sync) pulse cycle ¹	XHS_LENGTH {value}	Active pulse width of xhs_o. Only effective when SENSOR_MODE_SLAVE is defined. Value must be 8'd1 – 8'd255.
RX Data Type	RAW10	Define the data type on RX channel. Only one of these two directives must be defined.
	RAW12	
RX channel lane count	NUM_RX_LANE_4	Number of lanes in RX channel. Only one of these four directives must be defined.
	NUM_RX_LANE_6	
	NUM_RX_LANE_8	
	NUM_RX_LANE_10	
RX SubLVDS Clock Gear	RX_GEAR_8	RX SubLVDS Clock Gear. Only one of these directives must be defined. Gear 16 is allowed only in 4-lane configuration.
	RX_GEAR_16	
Use GPLL ²	USE_GPLL	Use GPLL to create a reference clock to be fed to PLL of TX D-PHY IP.
TX D-PHY Clock mode ³	TX_CLK_MODE_HS_LP	TX D-PHY Clock mode. Only one of these two directives must be defined.
	TX_CLK_MODE_HS_ONLY	
TX channel lane count	NUM_TX_LANE_1	Number of lanes in TX channel. Only one of these three directives must be defined.
	NUM_TX_LANE_2	
	NUM_TX_LANE_4	
TX D-PHY Clock Gear	TX_GEAR_8	Number of TX Clock Gear on RX channel. Only one of these directives must be defined.
	TX_GEAR_16	
Parameter set by I ² C	USE_I2C	Define this to use I ² C I/F to set the parameters on the fly.

Category	Directive	Remarks
I ² C Slave Address (MSB) ⁴	I2C_SLAVE_ADR_MSB {value}	Define MSB 5bits of I ² C Slave Address. Value must be 5'h00 – 5'h1F. Applicable only when USE_I2C is defined. This value overwrites the value set in Clarity when IP is created.
Software Reset Register ⁵	SW_RST_N {value}	Default value of the software reset register of I ² C Slave module. Value must be 1'b0 or 1'b1. Applicable only when USE_I2C is defined. Active low.
Top Line Trimming ^{6, 7}	TOP_TRIM {value}	Define the number of lines to be trimmed before TX. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 6'd1 – 6'd63.
Vertical Active Lines on TX ⁷	V_ACTIVE {value}	Define the number of active lines to be sent on TX. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 12'd1 – 12'd4095.
Left Pixel Unit Trimming ^{8, 10}	LEFT_TRIM_UNIT {value}	Define the number of pixel units to be trimmed before TX. 1 pixel unit = number of RX lanes. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 10'd0 – 10'd1023.
Left Pixel Trimming ^{8, 10}	LEFT_TRIM_LANE {value}	Define the number of pixels to be trimmed before TX. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 4'd0 – 4'd9 and less than the RX lane count.
Horizontal Active Pixel units on TX ^{9, 10}	H_ACTIVE_UNIT {value}	Define the number of active pixels to be sent on TX. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 10'd1 – 10'd1023. The value must be even in case of TX_GEAR_16 or NUM_TX_CH_2.
Active Word Count ¹¹	WC {value}	Define the number of byte count of active pixels to be sent to TX. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 16'd1 – 16'd65535.
Virtual Channel ID	VC {value}	Define the virtual Channel ID. This value is used as a fixed value when USE_I2C is not defined and used as the default value of the I ² C register when USE_I2C is defined. Value must be 2'd0 – 2'd3.

Notes:

1. Refer to the image sensor data sheet for proper settings.
2. Refer to the [int_gpll](#) section for details.
3. HS_LP mode means non-continuous clock mode and HS_ONLY means continuous clock mode.
4. LSB two bits of I2C slave address is automatically set when I²C IP is created by Clarity.
5. Logical OR between this register and system reset (reset_n_i) is used to reset modules other than I²C slave module.
6. The first line is always trimmed by SubLVDS RX IP and value = 0 is not allowed.
7. (TOP_TRIM + V_ACTIVE) cannot exceed the vertical active line count of the incoming RX data. It is your responsibility to manage this.
8. Number of pixels trimmed from the left edge is ((LEFT_TRIM_UNIT x number of RX lanes x (RX_GEAR / 8)) + LEFT_TRIM_LANE).
9. Active pixel count sent to TX is (H_ACTIVE_UNIT x number of RX lanes x (RX_GEAR / 8)).
10. ((LEFT_TRIM_UNIT + H_ACTIVE_UNIT) x number of RX lanes x (RX_GEAR / 8) + LEFT_TRIM_LANE) cannot exceed the horizontal active pixel count of the incoming RX data. It is your responsibility to manage this.
11. $WC \leq (H_ACTIVE_UNIT \times \text{number of RX lanes} \times (RX_GEAR / 8)) \times (RAW \text{ number } (10 \text{ or } 12)) / 8$. Refer to the [trim_ctrl](#) section for details.

2.2. Simulation Directives

Table 2.2 shows the simulation directives for this reference design. Some parameter selections are restricted by other parameter settings including Table 2.1.

Table 2.2. Simulation Directives

Category	Directive	Remarks
RX SubLVDS clock period	PIX_CLK {value}	RX SubLVDS clock period in ps.
INCK clock period	INCK_PERIOD {value}	INCK clock period in ps. Applicable only when SENSOR_MODE_SLAVE is defined.
Number of frames to run	NUM_FRAMES {value}	Number of video frames fed by testbench.
Number of active lines ¹	NUM_LINES {value}	Number of RX active video lines per frame.
Number of active pixels ²	NUM_PIXELS {value}	Number of RX active video pixels per line.
Vertical Blanking from XVS to active line ¹	VFRONT_BLNK {value}	Number of blanking lines before the active video line.
Vertical Blanking after active line ¹	VREAR_BLNK {value}	Number of blanking lines after the active video line.
Horizontal Blanking period ²	HB_PERIOD {value}	Horizontal Blanking period in SubLVDS clock cycles.
Software Reset Register ³	I2C_SW_RST_N {value}	Write value to the software reset register of I ² C Slave module. Value must be 1'b0 or 1'b1. Applicable only when USE_I2C is defined. Active low.
Top Line Trimming ^{4, 5}	I2C_TOP_TRIM {value}	Write value to the top trim register of I ² C Slave module. Value must be 6'd1 – 6'd63. Applicable only when USE_I2C is defined.
Vertical Active Lines on TX ⁵	I2C_V_ACTIVE {value}	Write value to the vertical active line register of I ² C Slave module. Value must be 12'd1 – 12'd4095. Applicable only when USE_I2C is defined.
Left Pixel Unit Trimming ^{6, 8}	I2C_LEFT_TRIM_UNIT {value}	Write value to the pixel unit trim register of I ² C Slave module. Value must be 10'd0 – 10'd1023. Applicable only when USE_I2C is defined.
Left Pixel Trimming ^{6, 8}	I2C_LEFT_TRIM_LANE {value}	Write value to the pixel trim register of I ² C Slave module. Value must be 4'd0 – 4'd9 and less than the RX lane count. Applicable only when USE_I2C is defined.
Horizontal Active Pixel units on TX ^{7, 8}	I2C_H_ACTIVE_UNIT {value}	Write value to the horizontal active pixel unit register of I ² C Slave module. Value must be 10'd1 – 10'd1023. Applicable only when USE_I2C is defined.
Active Word Count ⁹	I2C_WC {value}	Write value to the word count register of I ² C Slave module. Value must be 16'd1 – 16'd65535. Applicable only when USE_I2C is defined.
Virtual Channel ID	I2C_VC {value}	Write value to the virtual channel ID register of I ² C Slave module. Value must be 2'd0 – 2'd3. Applicable only when USE_I2C is defined.
Total line count ¹⁰	I2C_V_TOTAL {value}	Write value to the total line count register of I ² C Slave module. Value must be 12'd10 – 12'd4095. Applicable only when USE_I2C and SENSOR_MODE_SLAVE are defined.
Total horizontal cycle ¹⁰	I2C_H_TOTAL {value}	Write value to the total horizontal cycle count register of I ² C Slave module. Value must be 12'd10 – 12'd4095. Applicable only when USE_I2C and SENSOR_MODE_SLAVE are defined.
XHS (Horizontal Sync) pulse cycle ¹⁰	I2C_XHS_LENGTH {value}	Write value to the XHS pulse length register of I ² C Slave module. Value must be 8'd1 – 8'd255. Applicable only when USE_I2C and SENSOR_MODE_SLAVE are defined.

Notes:

1. Total number of lines per frame is (NUM_LINES + VFRONT_BLNK + VREAR_BLNK).
2. In the case of SENSOR_MODE_MASTER, total number of SubLVDS clock cycles per line is (((NUM_PIXELS/NUM_RX_LANE) + 8) x (RAW number (10 or 12)) / 2) + HB_PERIOD).
3. Logical OR between this register and system reset (reset_n_i) is used to reset modules other than I²C slave module.

4. The first line is always trimmed by SubLVDS RX IP and value = 0 is not allowed.
5. $(I2C_TOP_TRIM + I2C_V_ACTIVE)$ cannot exceed the vertical active line count of the incoming RX data. It is your responsibility to manage this.
6. Number of pixels trimmed from the left edge is $((I2C_LEFT_TRIM_UNIT \times \text{number of RX lanes} \times (RX_GEAR / 8)) + I2C_LEFT_TRIM_LANE)$.
7. Active pixel count sent to TX is $(I2C_H_ACTIVE_UNIT \times \text{number of RX lanes} \times (RX_GEAR / 8))$.
8. $((I2C_LEFT_TRIM_UNIT + I2C_H_ACTIVE_UNIT) \times \text{number of RX lanes} \times (RX_GEAR / 8) + I2C_LEFT_TRIM_LANE)$ cannot exceed the horizontal active pixel count of the incoming RX data. It is your responsibility to manage this.
9. $I2C_WC \leq (I2C_H_ACTIVE_UNIT \times \text{number of RX lanes} \times (RX_GEAR / 8)) \times (\text{RAW number (10 or 12)}) / 8$. Refer to the [trim_ctrl](#) section for details.
10. Refer to the image sensor data sheet for proper settings.

2.3. Top-Level I/O

Table 2.3 shows the top level I/O of this reference design. Actual I/O depend on the customer's channel and lane configurations. All necessary I/O ports are automatically declared by compiler directives.

Table 2.3. Simulation Directives

Port Name	Direction	Description
Reset		
reset_n_i	I	Asynchronous active low system reset
Control Interface (conditional)		
inck_i	I	Clock to control XVS and XHS. Only used in case of SENSOR_MODE_SLAVE.
xvs_o	O	Vertical Sync signal to the image sensor. Only used in case of SENSOR_MODE_SLAVE.
xhs_o	O	Horizontal Sync signal to the image sensor. Only used in case of SENSOR_MODE_SLAVE.
Control Interface (optional)		
scl	I/O	I ² C clock. Only used in case of USE_I2C.
sda	I/O	I ² C data Only used in case of USE_I2C.
SubLVDS RX Interface		
clk_p_i	I	Positive differential RX SubLVDS input clock
clk_n_i	I	Negative differential RX SubLVDS input clock
d0_p_i	I	Positive differential RX SubLVDS input data 0
d0_n_i	I	Negative differential RX SubLVDS input data 0
d1_p_i	I	Positive differential RX SubLVDS input data 1
d1_n_i	I	Negative differential RX SubLVDS input data 1
d2_p_i	I	Positive differential RX SubLVDS input data 2
d2_n_i	I	Negative differential RX SubLVDS input data 2
d3_p_i	I	Positive differential RX SubLVDS input data 3
d3_n_i	I	Negative differential RX SubLVDS input data 3
d4_p_i	I	Positive differential RX SubLVDS input data 4 (in case of 6/8/10-lane configuration)
d4_n_i	I	Negative differential RX SubLVDS input data 4 (in case of 6/8/10-lane configuration)
d5_p_i	I	Positive differential RX SubLVDS input data 5 (in case of 6/8/10-lane configuration)
d5_n_i	I	Negative differential RX SubLVDS input data 5 (in case of 6/8/10-lane configuration)
d6_p_i	I	Positive differential RX SubLVDS input data 6 (in case of 8/10-lane configuration)
d6_n_i	I	Negative differential RX SubLVDS input data 6 (in case of 8/10-lane configuration)
d7_p_i	I	Positive differential RX SubLVDS input data 7 (in case of 8/10-lane configuration)
d7_n_i	I	Negative differential RX SubLVDS input data 7 (in case of 8/10-lane configuration)
d8_p_i	I	Positive differential RX SubLVDS input data 8 (in case of 10-lane configuration)
d8_n_i	I	Negative differential RX SubLVDS input data 8 (in case of 10-lane configuration)
d9_p_i	I	Positive differential RX SubLVDS input data 9 (in case of 10-lane configuration)
d9_n_i	I	Negative differential RX SubLVDS input data 9 (in case of 10-lane configuration)

Port Name	Direction	Description
CSI-2 TX Interface		
clk_p_o	O	Positive differential TX CSI-2 output clock
clk_n_o	O	Negative differential TX CSI-2 output clock
d0_p_o	O	Positive differential TX CSI-2 output data 0
d0_n_o	O	Negative differential TX CSI-2 output data 0
d1_p_o	O	Positive differential TX CSI-2 output data 1 (in case of 2/4-lane configuration)
d1_n_o	O	Negative differential TX CSI-2 output data 1 (in case of 2/4-lane configuration)
d2_p_o	O	Positive differential TX CSI-2 output data 2 (in case of 4-lane configuration)
d2_n_o	O	Negative differential TX CSI-2 output data 2 (in case of 4-lane configuration)
d3_p_o	O	Positive differential TX CSI-2 output data 3 (in case of 4-lane configuration)
d3_n_o	O	Negative differential TX CSI-2 output data 3 (in case of 4-lane configuration)

3. Design and Module Description

The top-level design (sublvds2csi2.v) consists of the following modules:

- sensor_sync (conditional)
- rx_sublvds
- trim_ctrl
- pix2byte
- lane_ctrl
- tx_dphy
- int_gp11 (conditional)
- i2c_slave (optional)

The top-level design has a reset synchronization logic.

3.1. sensor_sync

This module is instantiated when SENSOR_MODE_SLAVE is defined to feed the horizontal and vertical sync signals (xhs_o, xvs_o) to the image sensor using the external clock (inck_i). Sync pulse polarity and interval can be changed by directives described in the [Synthesis Directives](#) and [Simulation Directives](#) sections. [Figure 5.4](#) shows xvs_o having one line length of active pulse by XVS_LENGTH_LINE directive. This module has no interaction with other modules except for i2c_slave. The following parameters are taken as input data:

- v_total_i[11:0] – Total number of lines per frame including blanking lines.
- h_total_i[11:0] – Total number of clock cycles per line including blanking period.
- xhs_length[7:0] – Pulse length of xhs_o. In case that XVS_LENGTH_XHS is defined, this value also applies to xvs_o active pulse length.

3.2. rx_sublvds

This module must be created for RX channel according to channel conditions, such as the number of lanes, bandwidth, and others. [Figure 3.1](#) and [Figure 3.2](#) show an example of IP interface settings in Clarity for the SubLVDS Image Sensor Receiver Submodule IP. You can use the sbx file (rx/rx.sbx) included in the sample project and re-configure according to your needs. Refer to [SubLVDS Image Sensor Receiver Submodule IP User Guide \(FPGA-IPUG-02023\)](#) for details.

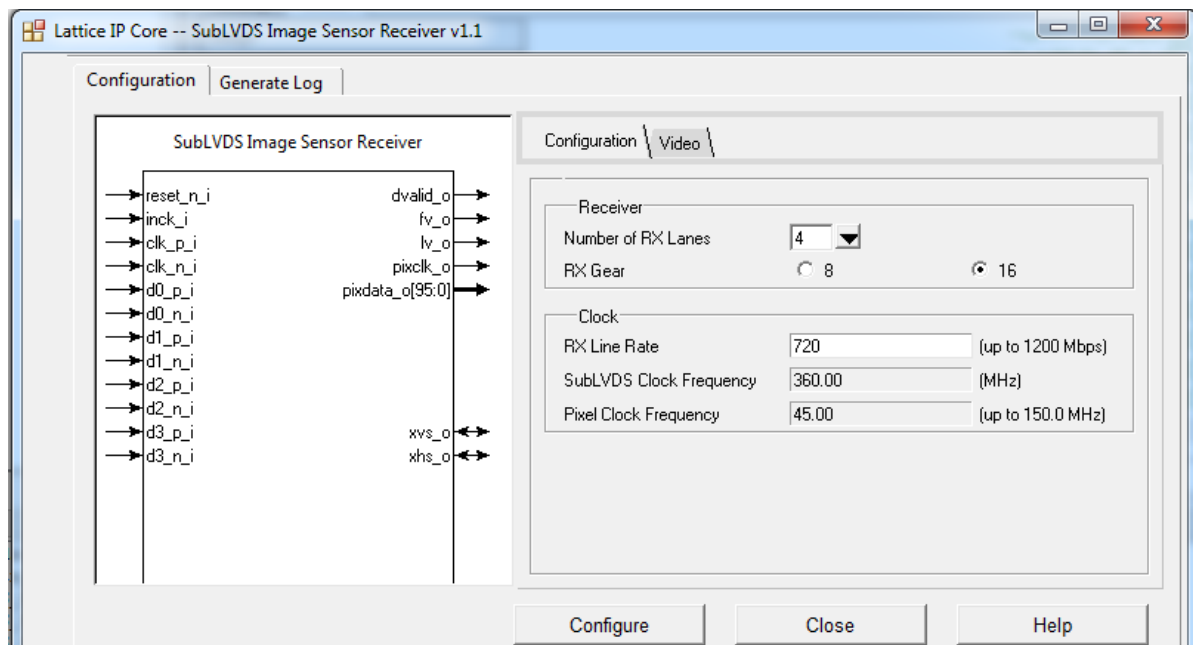


Figure 3.1. rx_sublvds IP Creation in Clarity Designer #1

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

- Number of RX Lanes – Set according to channel configuration. The value must match NUM_RX_LANE_* setting (4, 6, 8, or 10).
- RX Gear – Select 8 or 16; 16 is only applicable to 4-lane configuration and automatically selected when RX Line Rate is set above 900 Mbps.
- RX Line Rate – Set according to channel configuration. The following are the maximum values for different lane configurations:
 - 4-Lane, Gear 8 – 900 Mbps
 - 4-Lane, Gear 16 – 1200 Mbps
 - 6-Lane, Gear 8 – 600 Mbps
 - 8-Lane, Gear 8 – 720 Mbps
 - 10-Lane, Gear 8 – 600 Mbps
- Data Type – Select RAW10 or RAW12.
- Image Sensor Mode – Always select Master. Slave mode is handled by sensor_sync module.

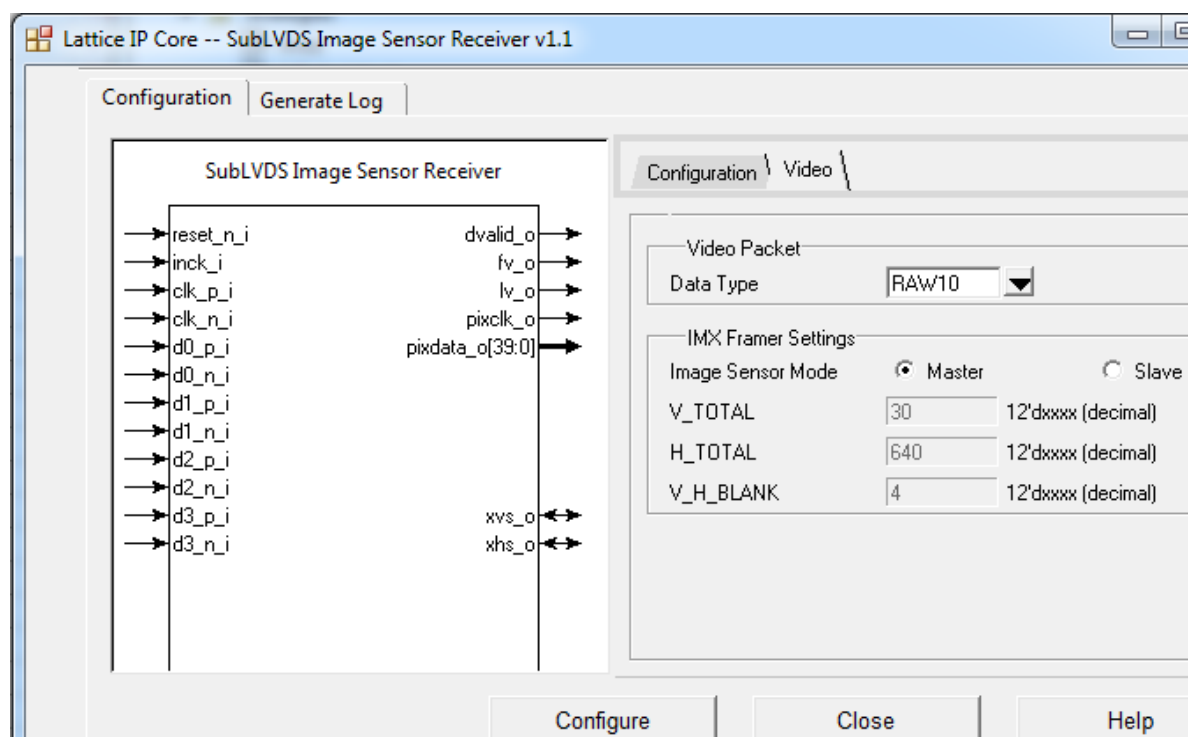


Figure 3.2. rx_sublvds IP Creation in Clarity Designer #1

This module takes serial SubLVDS data from image sensor and outputs pixel data after de-serialization. In case that you generate this IP from scratch, it is recommended to set the design name to rx and the module name to rx_sublvds so that you do not need to modify the instance name of this IP in the top-level design as well as the simulation setup file. Otherwise, you have to modify the names accordingly.

Figure 3.3 shows an example global timing of RAW10 in 10-lane configuration. This IP takes data between SAV (Start of Active Video) and EAV (End of Active Video) as active video data and outputs along with the assertion of dvalid. In case of Gear 8, the pixel clock frequency is $\frac{1}{4}$ of the incoming SubLVDS clock. Incoming data is DDR, which means 8 bits of data are de-serialized every pixel clock cycle. In both RAW10 and RAW12 cases, a single pixel clock cycle is not enough to retrieve one pixel data on each lane. Therefore, dvalid has on and off cycles. In case of RAW10, dvalid is asserted 4 out of every 5 pixel clock cycles (8 bits x 5 = 40 bits: 4 pixel data). In case of RAW12, dvalid is asserted 2 out of every 3 pixel clock cycles (8 bits x 3 = 24 bits: 2 pixel data).

Table 3.1 shows the sync code details. Both SAV and EAV comes from four words of data. The first word is always all 1 and second and third words are always all 0. The fourth word determines the type of the sync codes.

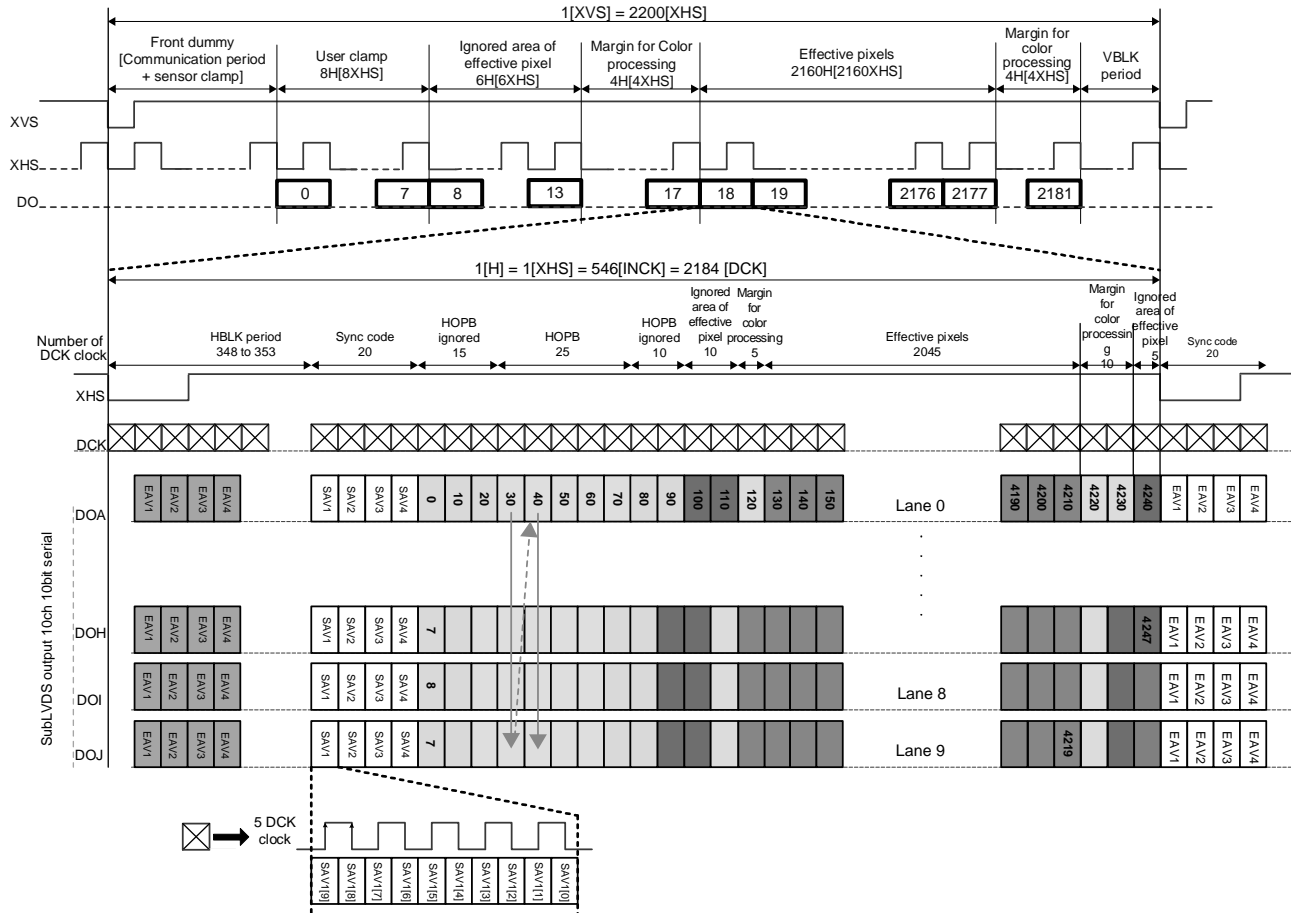


Figure 3.3. SubLVDS Input Global Timing (RAW10, 10 Lanes)

Table 3.1. Sync Code Details

LVDS Output Bit No.		Sync code				
12-Bit Output	10-Bit Output	First Word	Second Word	Third Word	Fourth Word	
11	9	1	0	0	1	
10	8	1	0	0	0	
9	7	1	0	0	V	1: Blanking line 0: Except blanking line
8	6	1	0	0	H	1: End sync code 2: Start sync code
7	5	1	0	0	P3	Protection bits
6	4	1	0	0	P2	
5	3	1	0	0	P1	
4	2	1	0	0	P0	
3	1	1	0	0	0	
2	0	1	0	0	0	
1	—	1	0	0	0	
0	—	1	0	0	0	

		Protection Bits			
V	H	P3	P2	P1	P0
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

3.3. trim_ctrl

In many cases, the data comes out from rx_sublvds includes unnecessary data and discarded by the downstream devices. This module enables to trim the edge data based on the given parameters. In case that USE_I2C is enabled, the amount of trimming data can be changed through I²C register update. The following parameters are taken as input data:

- top_trim_i[5:0] – Number of top lines to be trimmed. The first line is always trimmed by rx_lvds so that the minimum value must be 1.
- v_active_i[11:0] – Number of active lines to be sent to TX module.
- left_trim_unit_i[5:0] – Number of unit pixels to be trimmed. One unit is pixels equal to RX lane count in case of RX Gear 8 and 2x of RX lane count in case of RX Gear 16.
- left_trim_lane_i[3:0] – Number of pixels to be trimmed after unit trimming. Total pixels to be trimmed is (left_trim_unit_i) x (RX lane count) x (RX Gear / 8) + (left_trim_lane_i).
- h_active_unit_i[9:0] – Number of unit pixels to be sent to TX module. One unit is equal to RX lane count in case of RX Gear 8 and 2x of RX lane count in case of RX Gear 16.

Figure 3.4 shows the global timing example of trim_ctrl. In this case, the first and last lines are trimmed by trim_ctrl (note that the original first line is already trimmed by rx_sublvds).

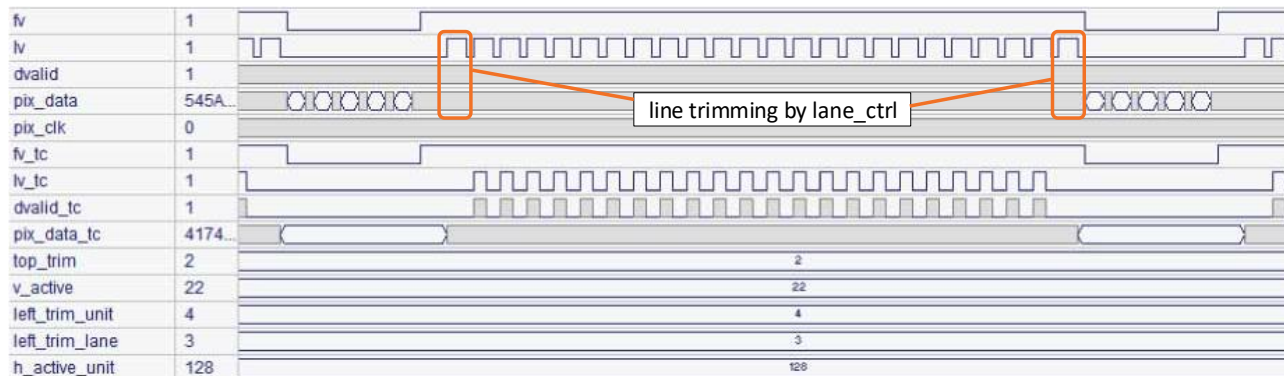


Figure 3.4. trim_ctrl Global Timing (RAW10, 10 Lanes)

Figure 3.5 shows the close-up of the above focusing the beginning of the line. In this case, the pixel counts to be trimmed is $4 (\text{left_trim_unit}) \times 10 (\text{RX lane count}) + 3 (\text{left_trim_lane}) = 43$. That means 4 unit data are trimmed from the beginning and 3 pixel data (LSB 30 bits) are trimmed from the fifth unit data (pix_data_A). The rest of the fifth unit data are shifted down towards LSB and LSB 30 bits of the next data (pix_data_B) are placed at the MSB 30 bits. The result is the first output data from lane_ctrl (pix_data_tc_A). This means

$$\text{pix_data_tc_A}[99:0] = \{\text{pix_data_B}[29:0], \text{pix_data_A}[99:30]\}.$$

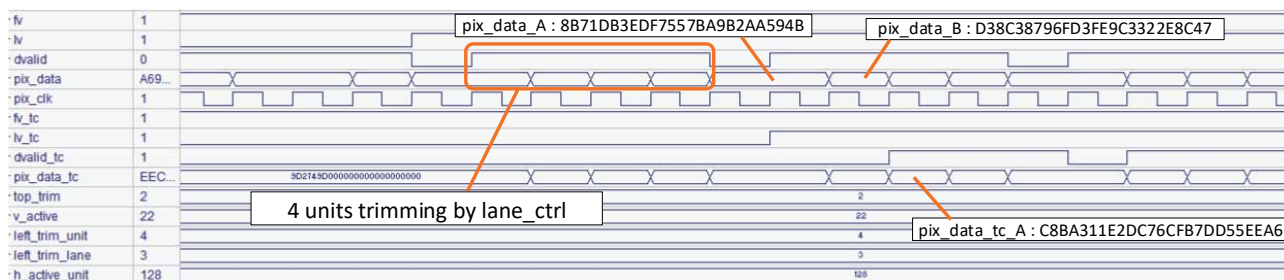


Figure 3.5. Trimming in the Beginning of the Line (RAW10, 10 Lanes)

The number of lines and pixels to be trimmed can be set in the unit of 1 line or 1 pixel as well as number of active lines to be cropped. However, there exist some limitations on the number of pixels to be cropped. Pixel cropping has to be in the unit of RX lane count as a parameter h_active_unit. In addition to this, the number of active pixels must be a multiple of 4 in case of RAW10 and a multiple of 2 in case of RAW12 according to CSI-2 spec. Moreover, due to the FIFO data width in rx_sublvds, another restriction has to be applied. Table 3.2 shows the unit value (granularity) of h_active_unit for all cases considering all of limitations mentioned above. In most cases, cropping with finer granularity is possible through WC setting. This WC is fed to lane_ctrl as the second stage cropping.

Example: RAW10, 10 RX lane, 2 TX lane with cropping 1936 pixels from 2240 active pixel input

The unit value of h_active_unit for RAW10, 10 RX lane is 16.

Since $1936 / (10 \text{ lanes} \times 16 \text{ units}) = 12.1$, which is rounded up to 13 as the nearest greater integer,

set $\text{h_active_unit} = 16 \times 13 = 208$ to cover 1936 pixels.

$\text{WC} = 1936 \times 5/4 = 2420$.

Table 3.2. Granularity of h_active_unit and WC

Data Type	RX Lane Count	RX Gear	TX Lane Count	TX Gear	Unit Value of h_active_unit	Granularity of WC
RAW10	4	8	1	16	4 (= 16 pixels)	10 (= 8 pixels)
			2	16	4 (= 16 pixels)	20 (= 16 pixels)
			4	8	4 (= 16 pixels)	20 (= 16 pixels)
		16	2	16	2 (= 16 pixels)	20 (= 16 pixels)
			4	16	4 (= 32 pixels)	40 (= 32 pixels)
			8	16	8 (= 48 pixels)	10 (= 8 pixels)
	6	8	2	16	8 (= 48 pixels)	20 (= 16 pixels)
			4	8	8 (= 48 pixels)	20 (= 16 pixels)
			8	16	4 (= 32 pixels)	20 (= 16 pixels)
	8	8	4	16	4 (= 32 pixels)	40 (= 32 pixels)
			2	16	16 (= 160 pixels)	20 (= 16 pixels)
			4	16	16 (= 160 pixels)	40 (= 32 pixels)
RAW12	4	8	1	16	2 (= 8 pixels)	6 (= 4 pixels)
			2	16	2 (= 8 pixels)	12 (= 8 pixels)
			4	8	2 (= 8 pixels)	12 (= 8 pixels)
		16	2	16	2 (= 16 pixels)	12 (= 8 pixels)
			4	16	2 (= 16 pixels)	24 (= 16 pixels)
			8	16	4 (= 24 pixels)	6 (= 4 pixels)
	6	8	2	16	4 (= 24 pixels)	12 (= 8 pixels)
			4	8	4 (= 24 pixels)	12 (= 8 pixels)
			8	16	2 (= 16 pixels)	12 (= 8 pixels)
	8	8	4	16	2 (= 16 pixels)	24 (= 16 pixels)
			2	16	8 (= 80 pixels)	12 (= 8 pixels)
			4	16	8 (= 80 pixels)	24 (= 16 pixels)

Note: In the case of RX Gear 16, pixel count per one h_active_unit is doubled compared to RX Gear 8.

3.4. pixel2byte

This module must be created for RX channel according to data type, the number of RX lanes, RX Gear, and others. [Figure 3.6](#) shows an example of IP interface settings in Clarity for the pixel2byte IP. You can use the sbx file (p2b/p2b.sbx) included in the sample project and re-configure according to your needs. Refer to [Pixel-to-Byte Converter IP User Guide \(FPGA-IPUG-02026\)](#) for details.

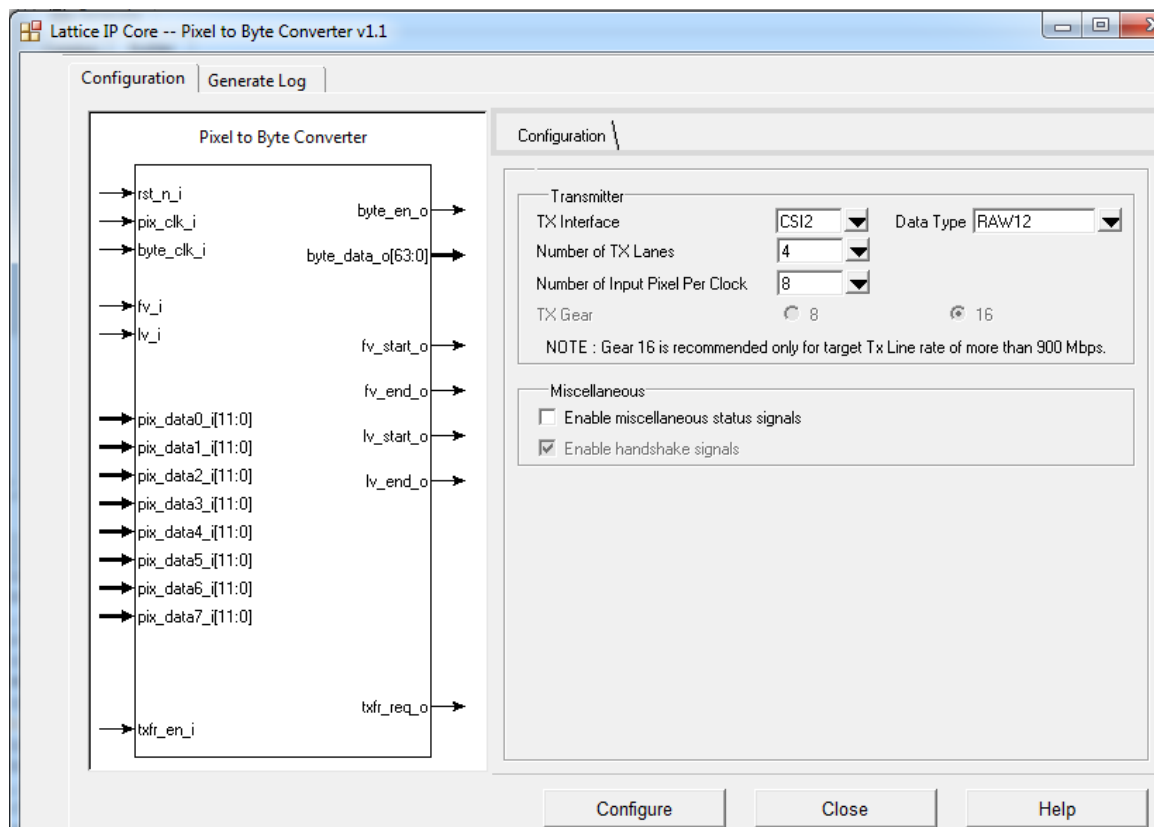


Figure 3.6. pixel2byte IP Creation in Clarity Designer

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

- TX Interface – Select CSI-2.
- Data Type – Select RAW10 or RAW12. Others are not supported by RX SubLVDS IP.
- Number of TX Lanes – Always set 4.
- Number of Input Pixel Per Clock – Set the value equal to (RX lane count) x (RX Gear / 8).
- TX Gear – automatically set in case of Number of Pixel Per Clock = 6, 8, or 10. 8 or 16 can be selected in case of Number of Input Pixel Per Clock = 4 (refer to [Table 1.1](#)).
- Enable miscellaneous status signals – unchecked.

In case that you generate this IP from scratch, it is recommended to set the design name to p2b and the module name to pix2byte so that you do not need to modify the instance name of this IP in the top-level design as well as the simulation setup file. Otherwise, you have to modify the names accordingly.

This module receives pixel data (pix_data_tc) from trim_ctrl along with fv_tc, lv_tc, and dvalid_tc (frame valid, line valid and data valid) and re-organizes the data to form the byte data via FIFO according to the data type specified. FIFO is used as a data buffer as well as a clock domain bridge. Pixel data are written to FIFO in pixel clock domain and read in byte clock domain. Byte clock is provided from TX D-PHY module. [Figure 3.7](#) shows the global timing of pixel2byte in the case of 4-lane with RX Gear 16.

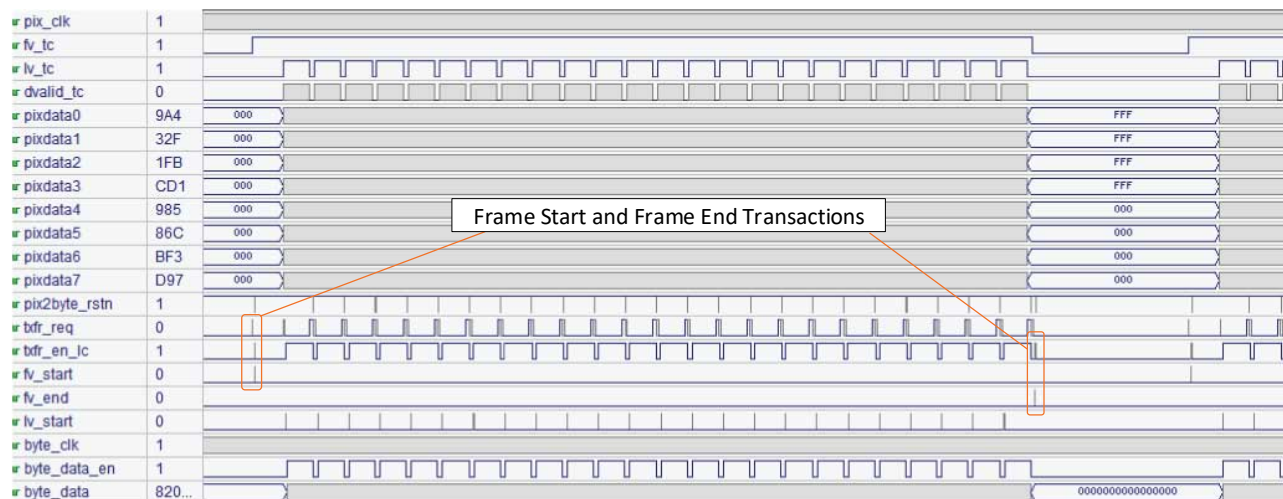


Figure 3.7. Global Timing of pixel2byte

Figure 3.8 shows line transactions. pixel2byte asserts txfr_req after receiving the valid line data and start sending byte based data following lv_start assertion. At the end of one-line valid data, txfr_req is asserted again. This assertion is redundant and masked by lane_ctrl and does not affect the system operation. pix2byte_rstn is a signal that comes from TX D-PHY to reset the internal FIFO of pixel2byte. This signal must be asserted before the new line data comes in to pixel2byte. Otherwise, the FIFO data is corrupted and results in data mismatch in simulation. If this happens, there can be four workarounds:

- Increase the blanking time.
- Use HS_ONLY mode of TX D-PHY IP if the current mode is HS_LP.
- Trim the right edge pixels by reducing the values of h_active_unit/WC if acceptable.
- Trim the left edge pixels by increasing the values of left_trim_unit/left_trim_lane if acceptable.

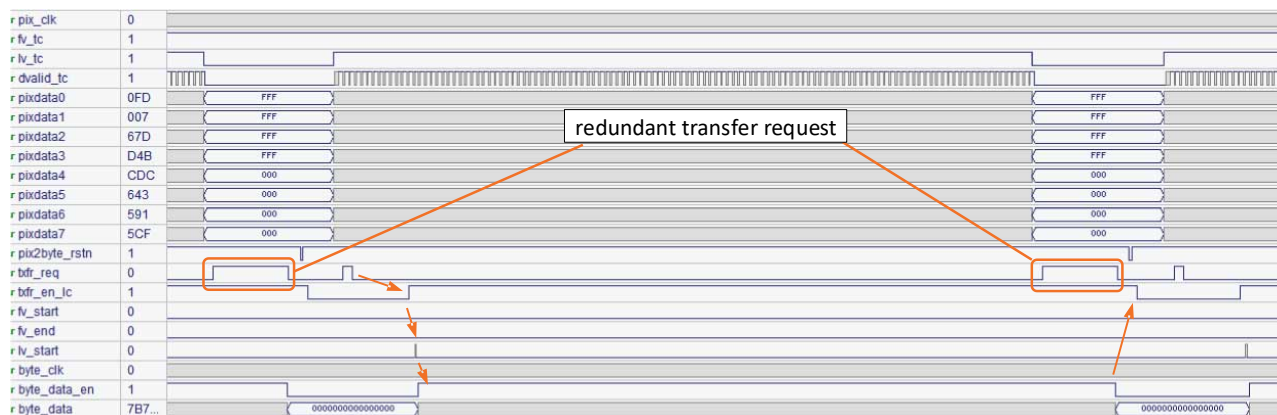


Figure 3.8. Line Transactions of pixel2byte

3.5. lane_ctrl

This module resides between pixel2byte and tx_phy. It has two major functions:

- communication control between pixel2byte and tx_dphy
- bus width and lane assignment conversion between pixel2byte and tx_dphy

3.5.1. Communication Control

pixel2byte and tx_dphy have handshake signals to request and grant data transfer from pixel2byte to tx_dphy, but current version of IP cannot handle a certain condition (Frame End event happens just after the valid data transmission of the last line) when those signals are directly connected. lane_ctrl holds the assertion of sp_en when txfr_en is high and waits for txfr_req_lc assertion until txfr_en goes low. Figure 3.9 shows the global timing of lane_ctrl in case of RAW12, RX lane count = 4 with RX Gear 16, TX lane count = 2 with TX Gear 16. In this case, byte data that comes from pixel2byte is 64 bits and active data input of TX D-PHY is 32 bits. Therefore hs_byte_clk (= 2x byte_clk) is used to match the input and output bandwidths. As described in the pixel2byte section, txfr_req assertion around the end of valid data transmission is masked by this module and is not transmitted to tx_dphy. This module also handles the clock domain crossing in case that hs_byte_clk is used. A compiler directive of HS_BYTE_CLK is automatically defined in the top-level RTL when hs_byte_clk is required.

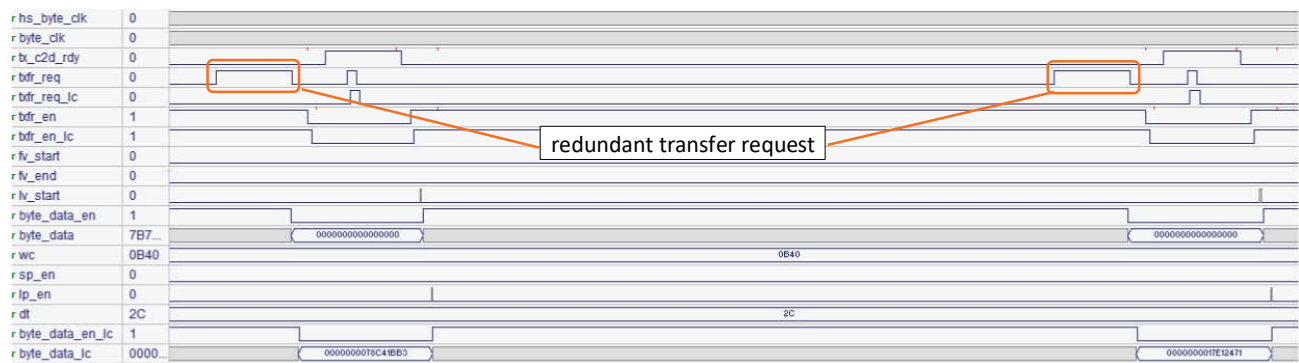


Figure 3.9. Global Timing of lane_ctrl

Figure 3.10 shows the handshake flow from trim_ctrl to tx_dphy to send Frame Start short packet. fv_tc from trim_ctrl lets pixel2byte asserts txfr_req. Then txfr_req is forwarded as txfr_req_lc when tx_c2d_rdy = 1, which means tx_dphy is read to begin a new HS transmission. After tx_dphy goes in to HS mode, txfr_en goes 1 and that was transferred as txfr_en_lc to pixel2byte. That lets pixel2byte asserts fv_start and lane_ctrl asserts sp_en, which results in short packet transmission by tx_dphy.

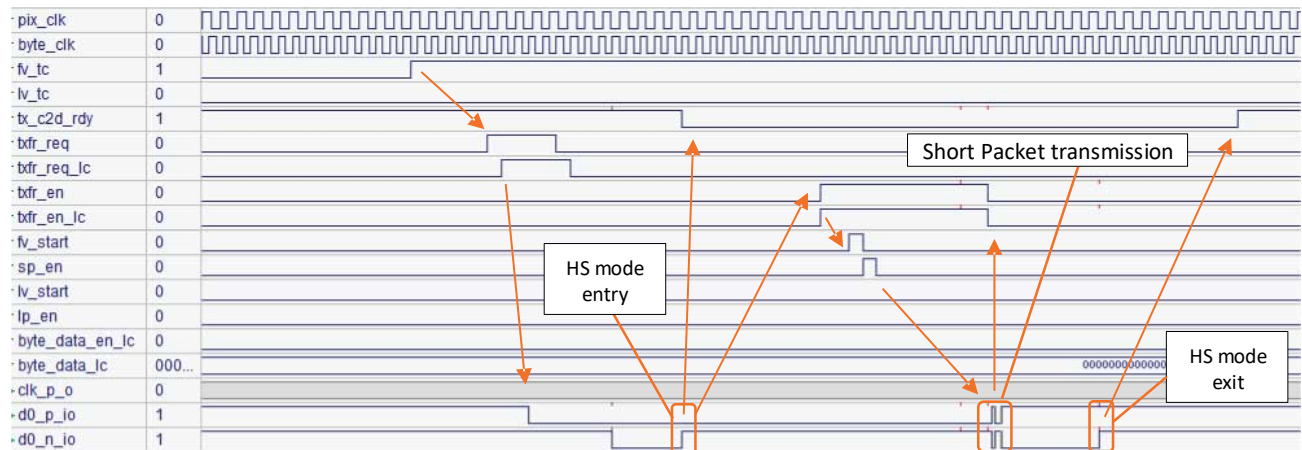


Figure 3.10. Handshake to Transfer Short Packet

Figure 3.11 shows an example of the handshake for Long Packet transmission. As mentioned earlier, txfr_req while txfr_en = 1 is masked by lane_ctrl and is not transferred to tx_dphy.

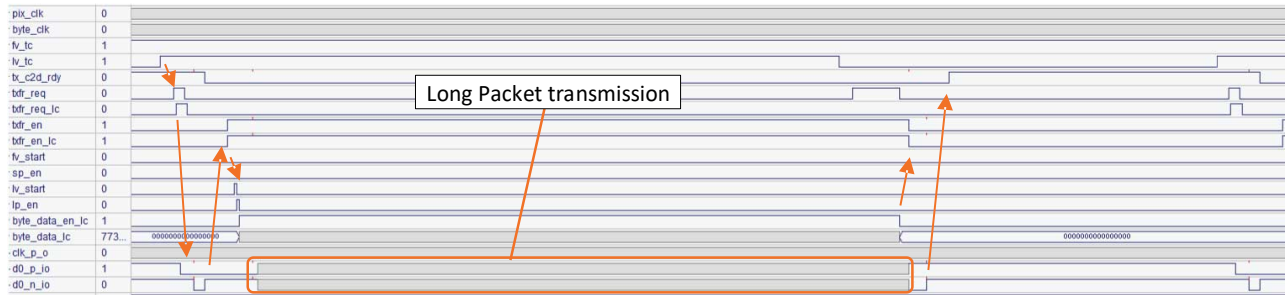


Figure 3.11. Handshake to Transfer Long Packet

3.5.2. Bus Width and Byte Data Assignment Conversion

Since pixel2byte configuration is always with 4 TX lanes, bus width and/or Byte data assignment conversion is necessary when TX lane count is 1 or 2. Figure 3.12 shows the example in case of RAW12, RX lane count = 4 with RX Gear 16, TX lane count = 2 with TX Gear 16. 64-bit data are stored into the internal FIFO using byte_clk and read back using hs_byte_clk in every other cycles, then data re-ordering is performed to assign each byte data to appropriate byte location within 64-bit data interface connected to tx_dphy.

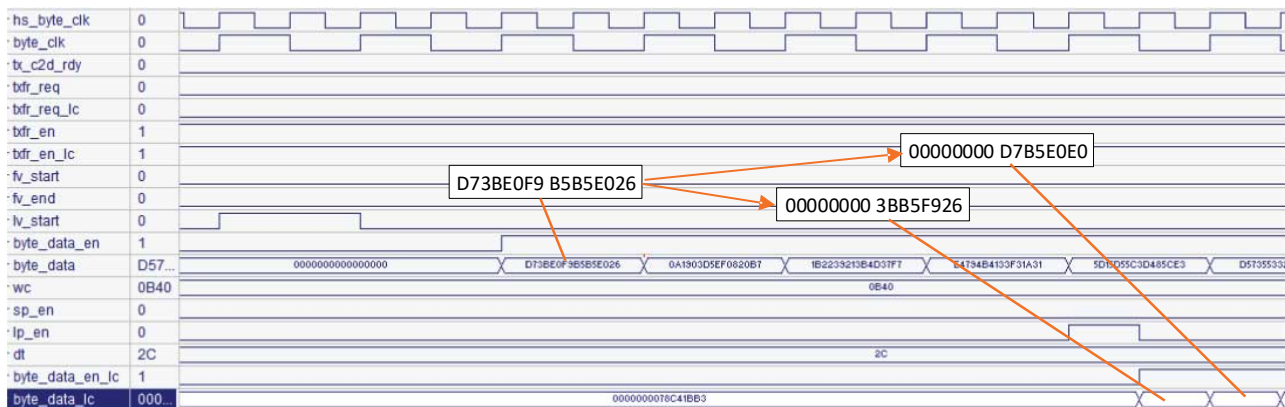


Figure 3.12. Byte Data Assignment Conversion by lane_ctrl

Table 3.3 shows the byte data allocation in all cases. In case hs_byte_clk is used, 32/64-bit data are split after the readout from FIFO and sent to tx_dphy alternately (shown as first out[63:0] and second out[63:0] in the table).

Table 3.3. Byte Data Reallocation

TX Lane Count	RX Lane Count	RX Gear	Byte Data Reassignments
4	all	all	out[63:0] = in[63:0]
2	4	8	out[63:0] = {32'd0, in[55:48], in[23:16], in[39:32], in[7:0]}
	6	8	
	4	16	First out[63:0] = {32'd0, in[55:48], in[23:16], in[39:32], in[7:0]} Second out[63:0] = {32'd0, in[63:56], in[31:24], in[47:40], in[15:8]}
	8	8	
1	4	16	First out[63:0] = {48'd0, in[23:16], in[7:0]}
	6	16	Second out[63:0] = {48'd0, in[55:48], in[39:32]}

3.6. tx_dphy

This module must be created for TX channel according to the number of TX lanes, TX Line Rate, and others. Figure 3.13 shows an example of IP interface settings in Clarity for the TX D-PHY IP. You can use the sbx file (tx/tx.sbx) included in the sample project and re-configure according to your needs. Refer to [CSI-2/DSI D-PHY Transmitter Submodule IP User Guide \(FPGA-IPUG-02024\)](#) for details.

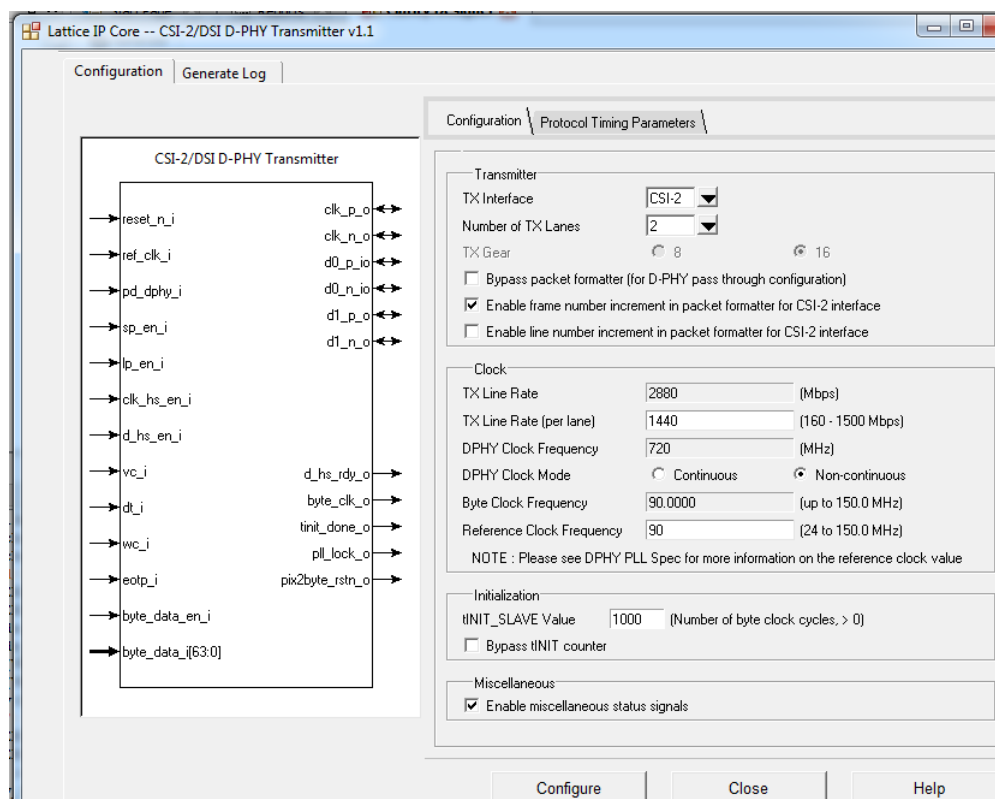


Figure 3.13. tx_dphy IP Creation in Clarity Designer

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

- TX Interface – Select CSI-2.
- Number of TX Lanes – Set according to TX lane configuration. Must match NUM_TX_LANE_* setting.
- TX Gear – Automatically set according to TX Line Rate.
- Bypass packet formatter – Must be disabled (unchecked).
- Enable frame number increment – Either is fine.
- Enable line number increment – Either is fine (unused).
- TX Line Rate (per lane) – Use the value specified in the Excel sheet.
- DPHY Clock Mode – Set according to TCX channel configuration. Continuous is recommended when the length of the horizontal blanking period is unknown.
- Reference Clock Frequency – Set the appropriate value which can be obtained by pix_clk or GPLL output. This clock frequency must be in one of the following ranges:
 - 24-30 MHz
 - 48-60 MHz
 - 72-90 MHz
 - 96-150 MHz
- tINIT_SLAVE Value – 1000 (default) is recommended.
- Bypass tINIT counter – Disabled (unchecked) is recommended.
- Enable miscellaneous status signals – must be set to enabled (checked).
- Protocol Timing Parameters tab – Default values are recommended.

This module takes the byte data and outputs CSI-2 data after serialization in CSI-2 High Speed mode. In case that you generate this IP from scratch, it is recommended to set the design name to tx and module name to tx_dphy so that you do not need to modify the instance name of this IP in the top level design as well as simulation setup file. Otherwise, you need to modify the names accordingly.

TX Line Rate is derived from the following equation:

$$TX_lane_bandwidth = \frac{RX_lane_bandwidth * number_of_RX_lane}{number_of_TX_lane}$$

Example #1: RAW10 RX with 10-lane at RX lane bandwidth = 600 Mbps with 4 TX lanes

TX lane bandwidth = (600 x 10) / 4 = 1500 Mbps.

Example #2: RAW12 RX with 4-lane at RX lane bandwidth = 1200 Mbps with 4 TX lanes

TX lane bandwidth = (1200 x 4) / 4 = 1200 Mbps.

3.7. int_gpll

You must create GPLL module to generate the reference clock when pix_clk frequency is not in the ranges shown below. You can use the sbx file (int_gpll/int_gpll.sbx) included in the sample project and re-configure according to your needs. In case you generate this IP from scratch, it is recommended to set the design name to int_gpll so that you do not need to modify the instance name of this IP in the top level design as well as simulation setup file. Otherwise, you need to modify the names accordingly. CLKOP output must be within one of the following frequency ranges:

- 24-30 MHz
- 48-60 MHz
- 72-90 MHz
- 96-150 MHz

You have to define USE_GPLL directive in synthesis_directives.v.

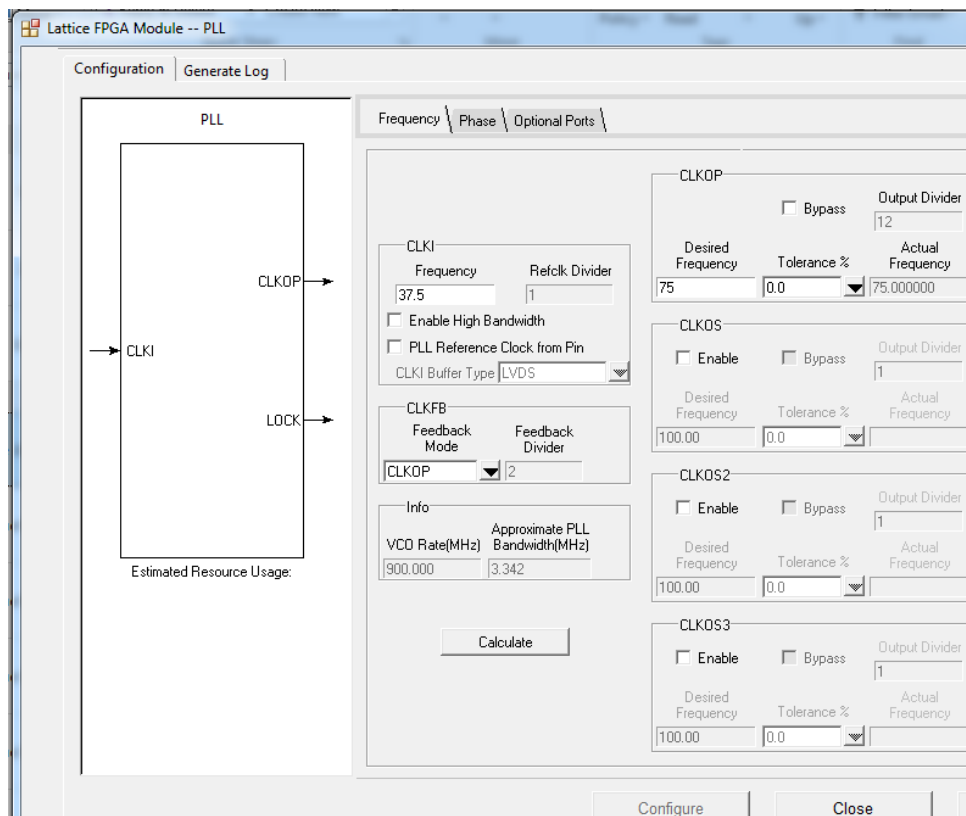


Figure 3.14. GPLL IP Creation

3.8. i2c_slave

This module is instantiated when USE_I2C is defined and enables you to change parameters on the fly through I²C connections. I²C Hard IP is instantiated and used as an I²C slave device. You can use the sbx file (i2c_s/i2c_s.sbx) included in the sample project and re-configure. In case that you generate this IP from scratch, it is recommended to set the design name to i2c_s so that you do not need to modify the instance name of this IP in the top level design as well as simulation setup file. Otherwise, you need to modify the names accordingly. There exist two I²C Hard IP modules in CrossLink and I2C0 is used in this IP as shown in Figure 3.15. You have to change the setting if the other IP (I2C1) is used. In this RD the internal oscillator is used to create a system clock for this module. You must change the System Bus clock frequency setting in case you use a different clock.

One typical usage of this module is changing parameters during the system development and/or debugging stages so that you can try many cases without updating FPGA bitfiles as long as I²C registers cover what you like to change. After that you can recreate a bitfile using the finalized parameters without USE_I2C directive for the product.

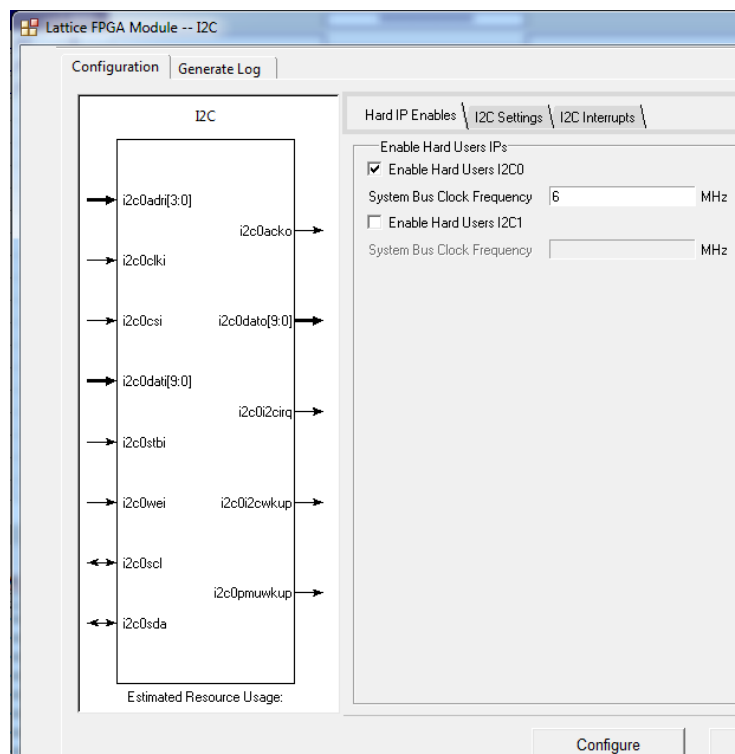


Figure 3.15. I²C IP Creation #1

Figure 3.16 shows basic settings of I²C IP. You can change the settings according to own needs, but the following have to be enforced:

- FIFO Mode must be disabled (unchecked)
- Address Match must be enabled (checked)

MSB 5 bits of I²C slave address can be set here. In the case of I2C0, LSB 2 bits is fixed to 2'b10.

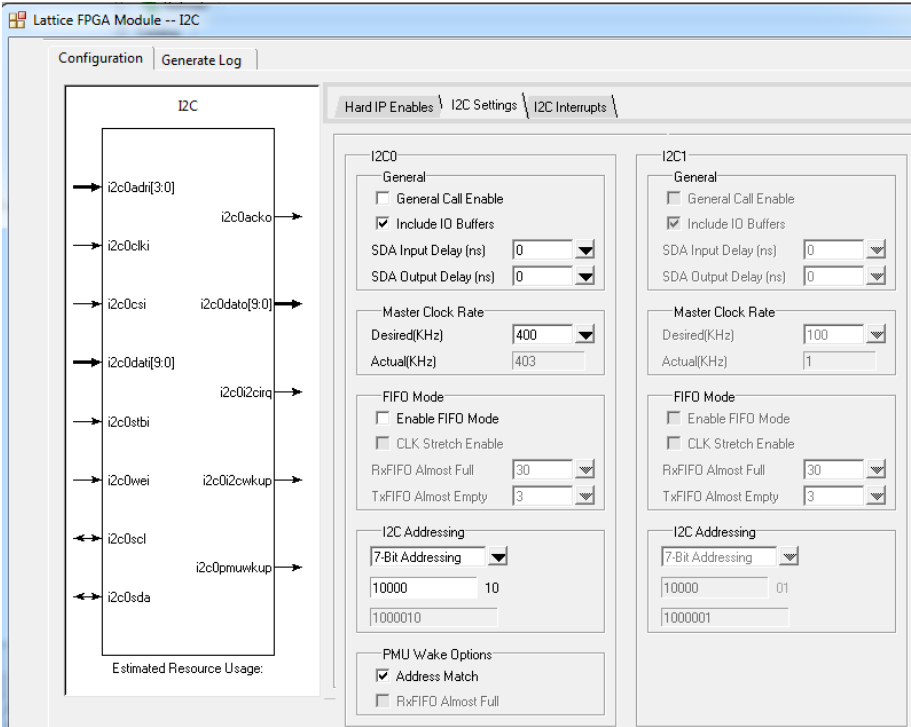


Figure 3.16. I²C IP Creation #2

Figure 3.17 shows interrupt settings. At least Tx/Rx Ready must be enabled (checked).

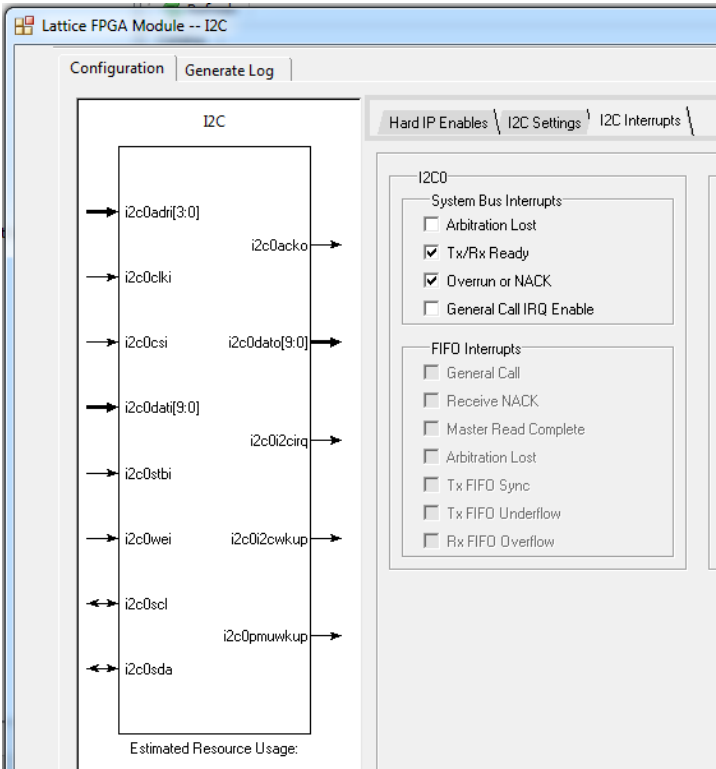


Figure 3.17. I²C IP Creation #3

This module is equipped with parameter registers of 4-bit address area of I²C sub-address shown in Table 3.4.

Table 3.4. I²C Slave Register Map

Sub Address	Name	Bits	Description
0	SW Reset	[0]	Software reset register. When this is active, all modules except for i2c_slave are in reset condition. Active low.
1	TOP_TRIM	[5:0]	Top edge trim register. The value must be 6'd1 – 6'd63.
2	V_ACTIVE_LSB	[7:0]	Vertical active line register. The value must be 12'd1 – 12'd4095.
3	V_ACTIVE_MSB	[11:8]	
4	LEFT_TRIM_UNIT	[5:0]	Left edge trim register by the unit. The value must be 6'd0 – 6'd63.
5	LEFT_TRIM_LANE	[3:0]	Left edge trim register within the lanes. The value must be 4'd0 – 4'd9.
6	H_ACTIVE_UNIT_LSB	[7:0]	Horizontal Active Pixel register by the unit. The value must be 10'd1 – 10'd1023.
7	H_ACTIVE_UNIT_MSB	[9:8]	
8	WC_LSB	[7:0]	Word count register. The value must be 16'd1 – 16'd65535.
9	WC_MSB	[15:8]	
10	VC	[1:0]	Virtual channel ID register. The value must be 2'd0 – 2'd3.
11	V_TOTAL_LSB	[7:0]	Total line count register used in Sensor Slave Mode. The value must be 12'd10 – 12'd4095.
12	V_TOTAL_MSB	[11:8]	
13	H_TOTAL_LSB	[7:0]	Total horizontal cycle count register used in Sensor Slave Mode. The value must be 12'd10 – 12'd4095.
14	H_TOTAL_MSB	[11:8]	
15	XHS_LENGTH	[7:0]	XHS pulse length register. Used in Sensor Slave Mode. The value must be 8'd1 - 8'd255.

Note: All registers less than 8-bit data width are aligned to LSB of 8-bit data area.

All registers are set to the default values specified by corresponding directives defined in `synthesis_directives.v`.

Software reset works as the system reset (`reset_n_i`) for all modules other than `i2c_slave`, therefore you can assert this while updating other I²C registers, then release Software reset upon completing the register update to avoid an unexpected operation during register update. Refer to the [Simulation Directives](#) and [trim_ctrl](#) sections for register details.

4. Design and File Modification by User

This RD is based on version 1.1/1.2 of the SubLVDS Image Sensor Receiver Submodule IP, version 1.1/1.2 of the Pixel-to-Byte Converter IP, and version 1.1/1.2 of the CSI-2/DSI D-PHY Transmitter Submodule IP. Due to the limitation of these IPs, some modifications are required depending on user configuration in addition to two directive files (synthesis_directives.v, simulation_directives.v).

4.1. Top-level RTL

You have to change IP instance names if you generate IPs created in Clarity Designer with different instance names in the sample project.

If you use I²C Slave module with a clock other than the internal oscillator clock, you have to disable the OSCI (internal oscillator) instantiation and connect an appropriate clock to clk_i port of i2c_slave.

4.2. Pixel-to-Byte IP

After creating Pixel-to-Byte IP in Clarity Designer, you must modify the top level RTL (pix2byte.v if you keep the same name as sample design) as shown below:

Add a following signal as input shown below:

```
input wire dvalid_i,
```

Modify the port name around line 132 as shown below:

```
.dvalid_i (lv_i), → .dvalid_i (dvalid_i),
```

4.3. TX D-PHY IP

After creating TX D-PHY IP in Clarity Designer, you must add an output signal *c2d_rdy_o* since this signal exists internally, but not brought out to the port.

In tx_dphy.v (assuming you have TX D-PHY IP with the instance name *tx_dphy*), add those two lines:

```
output          c2d_rdy_o,      // additional output, around line 77, after output  d_hs_rdy_o,
```

```
.c2d_rdy_o      (c2d_rdy_o      ),    // added to a submodule, around line 113, after  .d_hs_rdy_o      (d_hs_rdy_o      ),
```

In tx_dphy_dphy_tx.v, add one line and modify one line:

```
output          c2d_rdy_o,      // additional output, around line 60, after output  d_hs_rdy_o,
```

```
.c2d_ready_o    (c2d_rdy_o      ),    // fill in the port name c2d_rdy_o, around line 206 (currently empty)
```

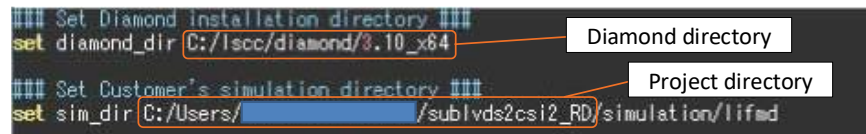
In tx_dphy_tx_global_operation.v, modify one line:

```
.c2d_ready_o    (c2d_ready_o      ), // fill in the port name c2d_ready_o, around line 67 (currently empty)
```

5. Design Simulation

The script file (sublvds2csi2_fsm.do) and testbench files are provided to run the functional simulation by Active HDL. Active HDL 10.5 bundled with Diamond 3.11 is not recommended since it might cause some simulation issues. If you follow the naming recommendations regarding design names and instance names when RX SubLVDS, Pixel-to-Byte, TX D-PHY, GPLL, and I²C IPs are created by Clarity Designer, the following are the only changes required in the script file:

- Diamond installation directory path
- User project directory



```

### Set Diamond installation directory ###
set diamond_dir C:/Iscc/diamond/3.10_x64

### Set Customer's simulation directory ###
set sim_dir C:/Users/[redacted]/sublvds2csi2_RD/simulation/lifed

```

Figure 5.1. Script File Modification

You need to modify simulation_directives.v according to your configuration (refer to [Simulation Directives](#) for details). By executing the script in Active HDL, compilation and simulation are executed automatically. The testbench takes all data comparison between the expected data and output data from the RD. It shows the following statements while running and doing data comparison:

...

```

# KERNEL: ### unit_cnt = 203, after lane_no = 1, payload_cnt = 2426, payload_LSBs = ba
# KERNEL: h_lane_on = 1, pixel_cnt = 1626
# KERNEL: unit_cnt = 203, lane_no = 2, payload_cnt = 2427, payload[11:4] = e0 by rx_dat[1626] = e09
# KERNEL: h_lane_on = 1, pixel_cnt = 1627
# KERNEL: unit_cnt = 203, lane_no = 3, payload_cnt = 2428, payload[11:4] = 2a by rx_dat[1627] = 2a5
# KERNEL: ### unit_cnt = 203, after lane_no = 3, payload_cnt = 2429, payload_LSBs = 59
# KERNEL: [128155564470][CSI-2_CHK] Frame 3, Line 23, Byte Count 1867 - 1868, payload data = 90 42 --- Data matches : 90 42
# KERNEL: [128156120150][CSI-2_CHK] Frame 3, Line 23, Byte Count 1869 - 1870, payload data = f7 20 --- Data matches : f7 20
# KERNEL: [128156675350][CSI-2_CHK] Frame 3, Line 23, Byte Count 1871 - 1872, payload data = 68 73 --- Data matches : 68 73
# KERNEL: h_lane_on = 1, pixel_cnt = 1628
# KERNEL: unit_cnt = 203, lane_no = 0, payload_cnt = 2430, payload[11:4] = 1f by rx_dat[1628] = 1f3
# KERNEL: h_lane_on = 1, pixel_cnt = 1629
# KERNEL: unit_cnt = 203, lane_no = 1, payload_cnt = 2431, payload[11:4] = 2f by rx_dat[1629] = 2f7
# KERNEL: ### unit_cnt = 203, after lane_no = 1, payload_cnt = 2432, payload_LSBs = 73
# KERNEL: h_lane_on = 1, pixel_cnt = 1630
# KERNEL: unit_cnt = 203, lane_no = 2, payload_cnt = 2433, payload[11:4] = 91 by rx_dat[1630] = 919
# KERNEL: h_lane_on = 1, pixel_cnt = 1631
# KERNEL: unit_cnt = 203, lane_no = 3, payload_cnt = 2434, payload[11:4] = 25 by rx_dat[1631] = 25f
# KERNEL: ### unit_cnt = 203, after lane_no = 3, payload_cnt = 2435, payload_LSBs = f9
# KERNEL: [128157231030][CSI-2_CHK] Frame 3, Line 23, Byte Count 1873 - 1874, payload data = bb c0 --- Data matches : bb c0
# KERNEL: [128157786710][CSI-2_CHK] Frame 3, Line 23, Byte Count 1875 - 1876, payload data = 09 13 --- Data matches : 09 13
# KERNEL: [128158342390][CSI-2_CHK] Frame 3, Line 23, Byte Count 1877 - 1878, payload data = b1 07 --- Data matches : b1 07
# KERNEL: h_lane_on = 1, pixel_cnt = 1632
# KERNEL: unit_cnt = 204, lane_no = 0, payload_cnt = 2436, payload[11:4] = 87 by rx_dat[1632] = 87f
# KERNEL: h_lane_on = 1, pixel_cnt = 1633
# KERNEL: unit_cnt = 204, lane_no = 1, payload_cnt = 2437, payload[11:4] = 84 by rx_dat[1633] = 847
# KERNEL: ### unit_cnt = 204, after lane_no = 1, payload_cnt = 2438, payload_LSBs = 7f

```

```
# KERNEL: h_lane_on = 1, pixel_cnt = 1634
# KERNEL: unit_cnt = 204, lane_no = 2, payload_cnt = 2439, payload[11:4] = 20 by rx_dat[1634] = 205
...
```

Simulation halts when data comparison fails.

When the simulation is completed, the following statements are displayed:

```
# KERNEL: [128422808150][CSI-2_CHK] Frame 3, Line 23, Byte Count 2829 - 2830, payload data = d5 dc --- Data matches : d5 dc
# KERNEL: [128423363350][CSI-2_CHK] Frame 3, Line 23, Byte Count 2831 - 2832, payload data = 8d 89 --- Data matches : 8d 89
# KERNEL: [128423919030][CSI-2_CHK] CRC = a8c5 --- OK
# KERNEL:      129235754700 : Transmitting rear blanking lines
# KERNEL:
# KERNEL: [129518451190][CSI-2_CHK] Short Packet detected : Data type = 01 --- Frame End
# KERNEL: [129519006870][CSI-2_CHK] Header1 = 00
# KERNEL:      132092649900 : End of frame :      2
# KERNEL:
# KERNEL: Total payload bytes checked were 195408 = 3 Frames x 2832 bytes (1888 pixels) x 23 lines : all matched !!!!!
# KERNEL: Simulation Succeeded !!!!!
# KERNEL: TEST END!!!
```

The above shows results after running three frames.

Figure 5.2 shows the global timing of 4-lane RX and 2-lane TX. In this case, the first and the last lines are trimmed. Note that the first line is always trimmed by sublvds_rx.

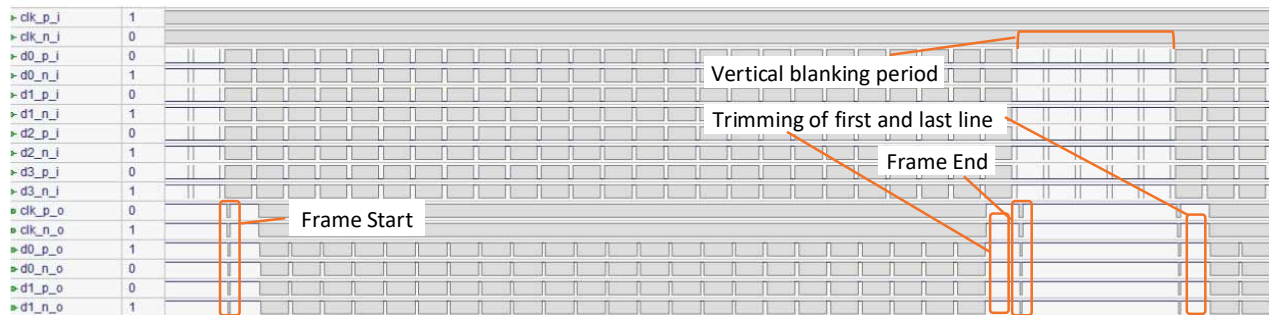


Figure 5.2. Global Timing of 4-Lane RX and 2-Lane TX

Figure 5.3 shows the close-up of the above waveform of one line period. Non-continuous clock mode is selected in TX D-PHY in this case so that CSI-2 TX clock lane goes into LP (Low Power) mode after the data lanes go into LP mode, then comes back to HS (High Speed) mode followed by the data lane transition to HS mode.

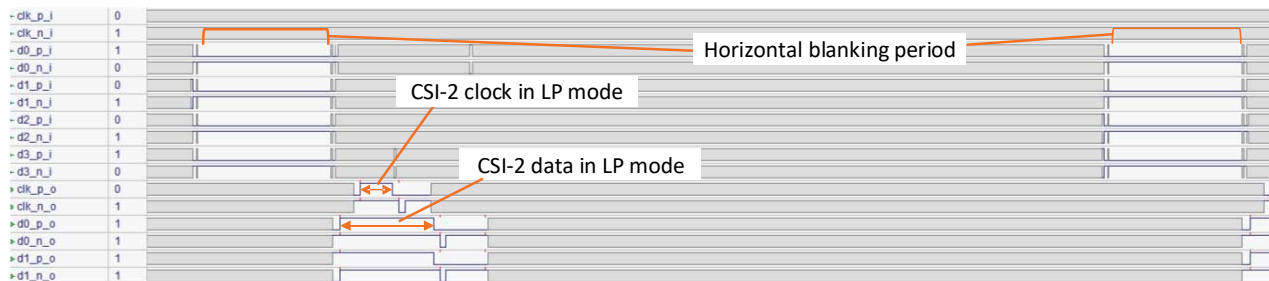


Figure 5.3. Global Timing with Sensor Slave Mode

Figure 5.4 shows the global timing of RAW10, 10-lane RX to 4-lane TX with Sensor Slave Mode. FPGA generates xvs_o and xhs_o as vertical and horizontal sync signals.

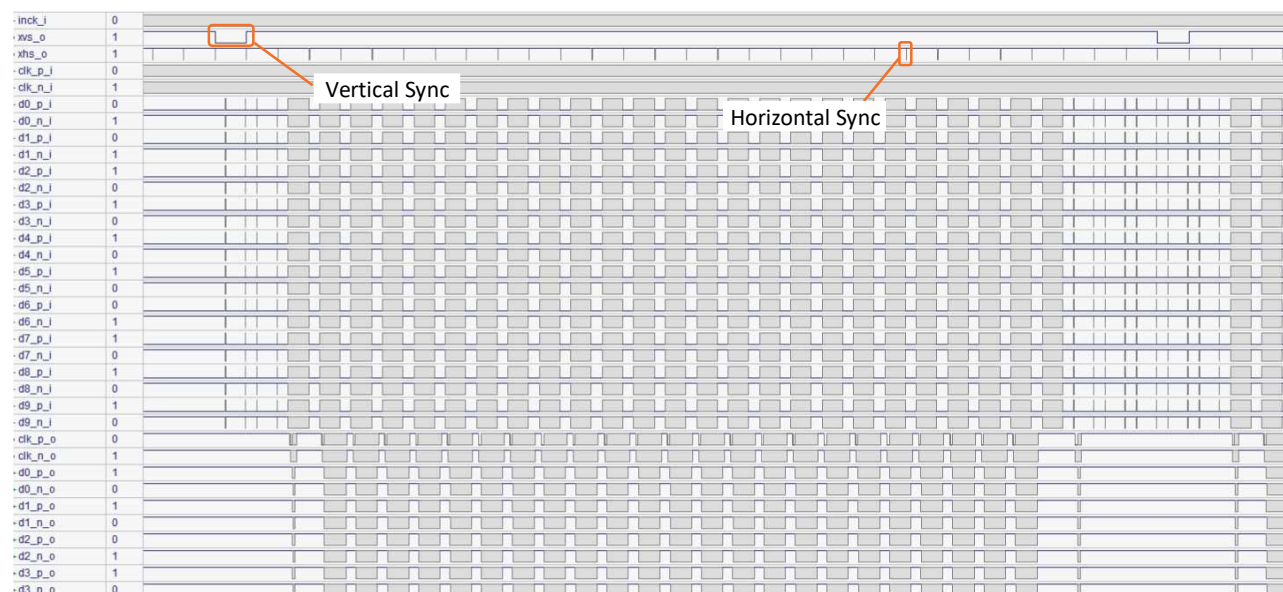


Figure 5.4. Global Timing with Sensor Slave Mode

6. Known Limitations

The following are the limitations of this reference design:

- The first line is always trimmed by rx_sublvds.
- Granularity of CSI-2 output video data is coarser than CSI-2 specification for both RAW10 and RAW12 as described in trim_ctrl section.

7. Design Package and Project Setup

SubLVDS to MIPI CSI-2 Image Sensor Bridge Reference Design for CrossLink is available on www.latticesemi.com. Figure 7.1 shows the directory structure. The design is targeted for LIF_MD6000-6KMG80I. `synthesis_directives.v` and `simulation_directives.v` are set to configure an example shown below:

- RX – RAW10 10 lanes, Gear 8, I²C enabled
- TX – 2 lanes with Gear 16

You can modify the directives for own configuration.

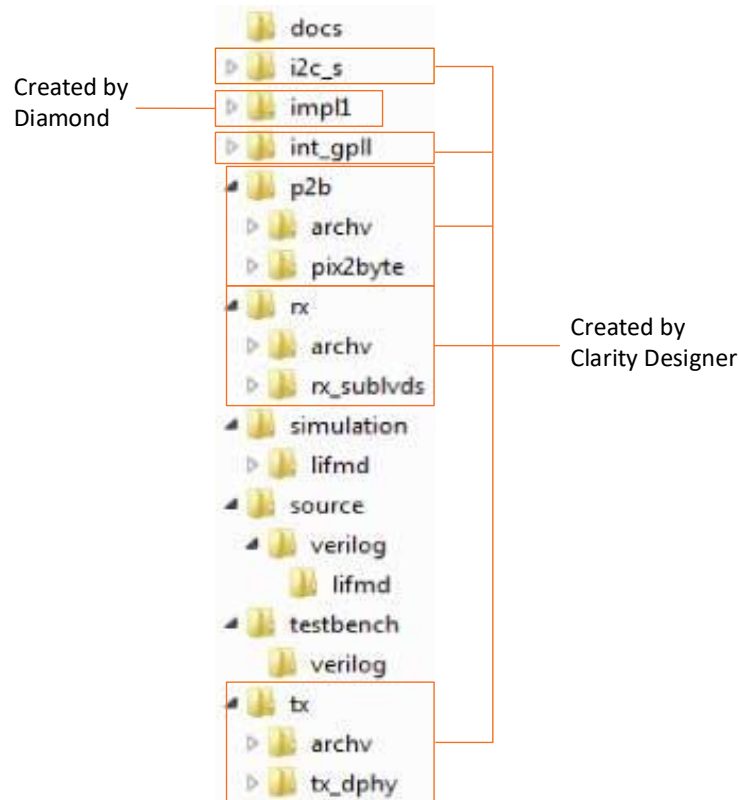


Figure 7.1. Directory Structure

Folders `i2c_s`, `int_gpll`, `p2b`, `rx`, and `tx` are created by Clarity Designer for corresponding IPs. Figure 7.2 shows design files used in the Diamond project. Clarity Designer creates five `.sbx` files.

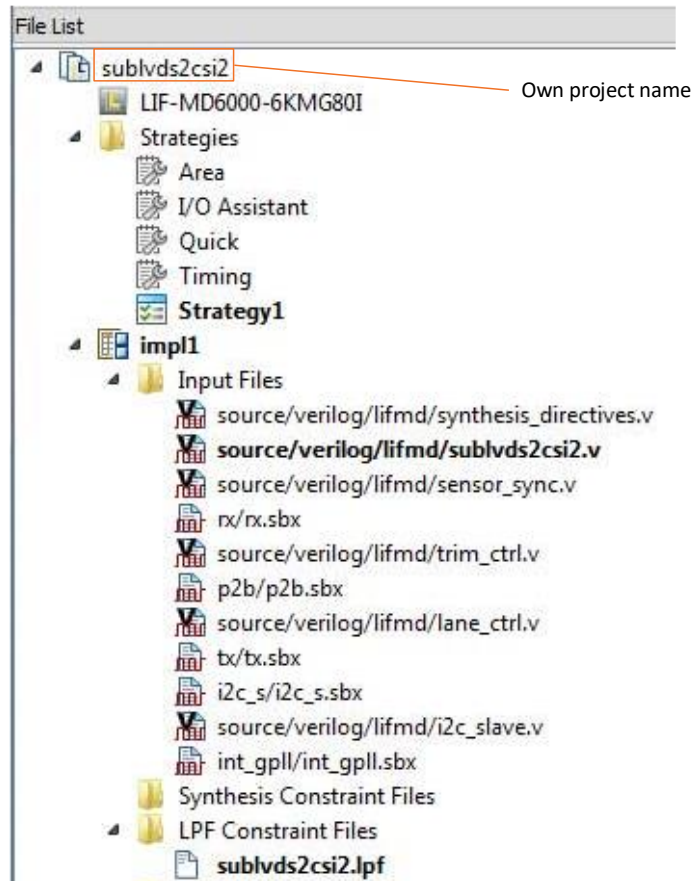


Figure 7.2. Project Files

8. Resource Utilization

Resource utilization depends on the configurations. Table 8.1 shows resource utilization examples under certain configurations. Actual usage may vary.

Table 8.1. Resource Utilization Examples

Configuration	LUT %	FF %	EBR	I/O
RAW12 RX 4 lanes, Gear 8 to TX 2 lanes without I ² C, Sensor Master Mode	24	18	6	17
RAW12 RX 4 lanes, Gear 8 to TX 1 lanes without I ² C, Sensor Master Mode	23	19	8	15
RAW12 RX 4 lanes, Gear 16 to TX 2 lanes with I ² C, Sensor Master Mode	51	33	16	19
RAW12 RX 4 lanes, Gear 16 to TX 2 lanes without I ² C, Sensor Master Mode	40	29	16	17
RAW10 RX 6 lanes, Gear 8 to TX 2 lanes without I ² C, Sensor Master Mode	28	19	8	21
RAW10 RX 6 lanes, Gear 8 to TX 1 lane without I ² C, Sensor Master Mode	27	20	10	19
RAW12 RX 8 lanes, Gear 8 to TX 4 lanes without I ² C, Sensor Master Mode	39	27	12	29
RAW12 RX 8 lanes, Gear 8 to TX 2 lanes without I ² C, Sensor Master Mode	36	28	16	25
RAW10 RX 10 lanes, Gear 8 to TX 4 lanes with I ² C, Sensor Slave Mode	66	32	12	38
RAW10 RX 10 lanes, Gear 8 to TX 2 lanes with I ² C, Sensor Slave Mode	63	33	16	34

References

- MIPI® Alliance Specification for D-PHY Version 1.1
- MIPI® Alliance Specification for Camera Serial Interface 2 (CSI-2) Version 1.1
- [SubLVDS Image Sensor Receiver Submodule IP User Guide \(FPGA-IPUG-02023\)](#)
- [Pixel-to-Byte Converter IP User Guide \(FPGA-IPUG-02026\)](#)
- [CSI-2/DSI D-PHY Transmitter Submodule IP User Guide \(FPGA-IPUG-02024\)](#)
- [Advanced CrossLink I²C Hardened IP Reference Guide \(FPGA-TN-02020\)](#)

For more information on the CrossLink FPGA device, visit
<http://www.latticesemi.com/Products/FPGAandCPLD/CrossLink>.

For complete information on Lattice Diamond Project-Based Environment, Design Flow, Implementation Flow, Tasks, and Simulation Flow, see the Lattice Diamond User Guide.

Technical Support Assistance

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

Revision History

Revision 1.1, September 2019

Section	Change Summary
Supported Device and IP	Newly added to support CrossLinkPlus.

Revision 1.0, June 2019

Section	Change Summary
All	Initial release.



www.latticesemi.com