MIPI DSI RX to Parallel Bridge

Reference Design

FPGA-RD-02068-1.4

November 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

Figures

Tables

1. Introduction

The Mobile Industry Processor Interface (MIPI) has become a specification standard for interfacing components in consumer mobile devices. The MIPI DSI specification provides a protocol layer interface definition, which is used to interface with displays. Normally an applications processor (AP) would directly connect to a display that has a DSI interface. See Figure 1.1.

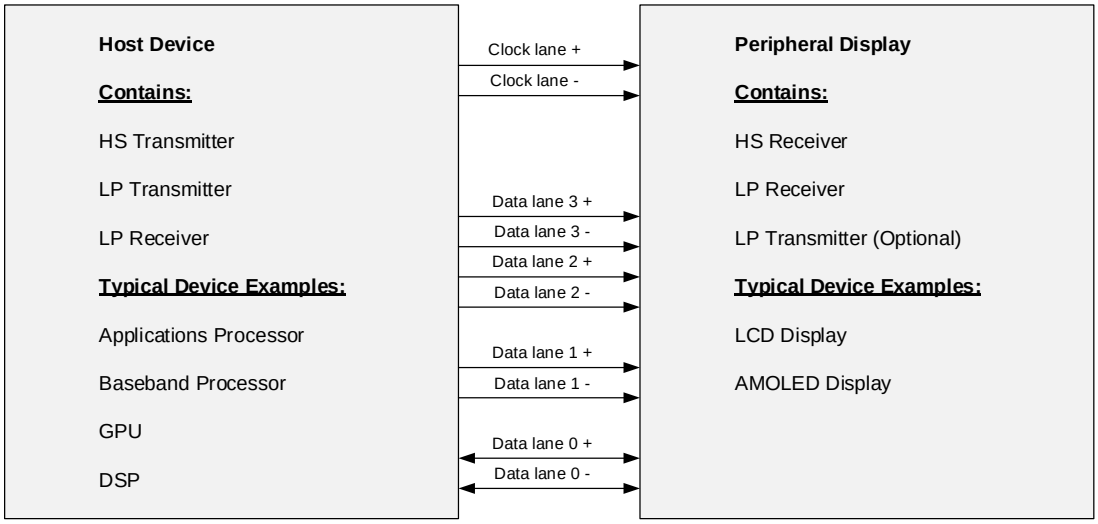

Figure 1.1. DSI Interface

Host processor devices are typically limited to only a select number of display interface types. Some processors may contain a DSI Tx interface, but others that do not may use RGB CMOS or Flat Panel Display Link (7:1 LVDS). In some cases, a user may want to interface to a MIPI DSI Host, but use a different type of display (Flat Panel Dis-play Link, Channel Link, CMOS bus interface, etc.) or extract the video data for some other purpose. The Lattice MIPI DSI RX to Parallel Bridge Reference Design allows users to extract MIPI DSI display data from a host device.

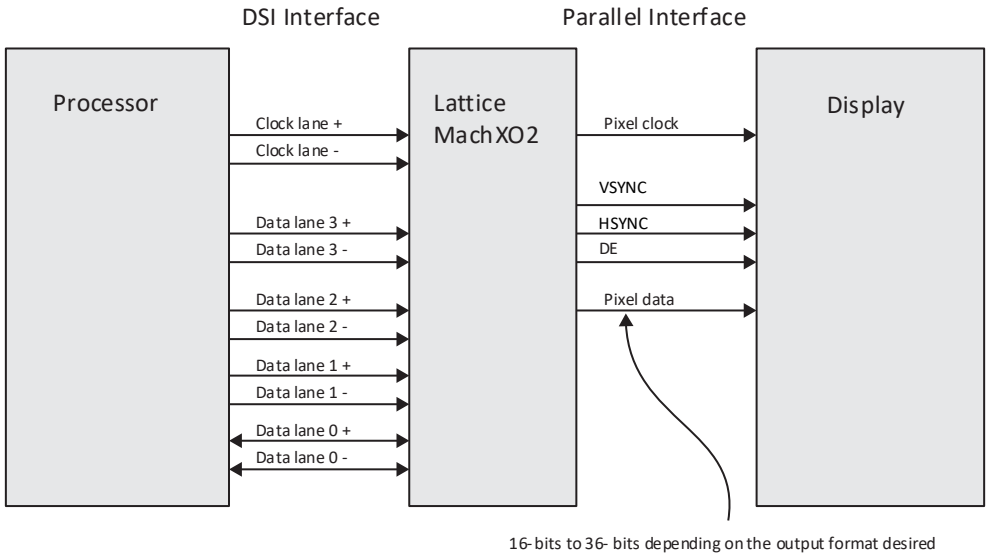

Figure 1.2. MIPI DSI RX to Parallel Bridge

2. Key Features

- Interfaces to MIPI DSI Transmitting devices
- Supports Receiving in HS (High Speed) Mode
- Supports Receiving and Transmitting in LP (Low Power) Mode
- Deserializes HS (High Speed) data up to four data lanes
- Supports all DSI compatible video formats (RGB, YCbCr and User Defined)

3. Functional Description

The MIPI DSI RX to Parallel Bridge Reference Design converts MIPI DSI clock and data streams into a standard parallel video interface. The input interface for the design utilizes the Lattice MIPI D-PHY Reference IP (RD1182, MIPI D-PHY Reference IP). It consists of an HS (High Speed) clock (DCK) and up to four HS (High Speed) data inputs (D0, D1, D2 and D3). It also consists of bidirectional LP (Low Power) signals for all clock and data pairs. These signals are connected together externally using an external resistor network as specified in the Lattice MIPI D-PHY Reference IP (RD1182, MIPI D-PHY Reference IP) documentation. The external resistor network implementation is also located in the D-PHY RX section of this document.

The output interface of the MIPI DSI Rx to Parallel Bridge Reference Design consists of a single ended CMOS interface. This output bus includes a data bus (PIXDATA), Vertical Sync (VSYNC) and Horizontal (HSYNC) indicators, a data enable (DE) indicator and a pixel clock. The pixel data bus is configurable from 16 bits to 36 bits depending on the data type parameter which is set by the user in the HDL code.

Each DSI data lane is deserilized to byte packets while in HS mode. The byte data is word and lane aligned based on the HS_Sync sequence defined in the MIPI D-PHY Specification. Once aligned, byte data packets are extracted based on packet headers in the capture_control module. Individual ports are available for the virtual channel number (VC), Data Type (DT), Word Count (WC), Error Correction Code (ECC), and the long packet checksum (checksum) based on packet header and packet footer extractions. VSYNC and HSYNC ports are also controlled by this module. VSYNC goes active ‘high’ and inactive ‘low’ when the “VSYNC Start” and “VSYNC End” short packets are seen. HSYNC goes active ‘high’ when the “HSYNC Start” short packet is seen. It can also go active ‘high’ during “VSYNC Start” or “VSYNC end” short packets for data bursting in cases where “HSYNC start” is not issued during a VSYNC start or end event. HSYNC goes inactive ‘low’ when the “HSYNC end” short packet is seen. The source code for this operation is available in the top level module and is configurable by the user if a different control is desired.

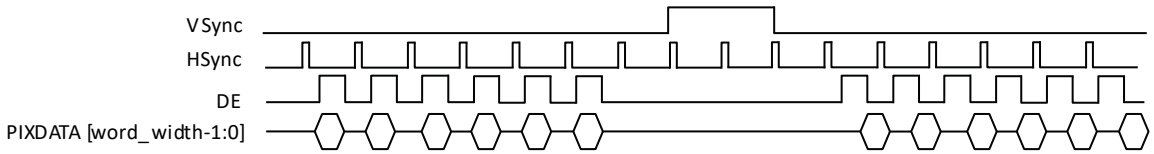

Figure 3.1. Parallel Output Bus Waveform

A byte_en signal is passed on to the parser module to indicate when a long packet is received matching the mode set in the data_type parameter at the top level. The parser module converts byte data to pixel data based on the data_type parameter. The output of the parser module is the pixel data bus (PIXDATA) and data enable (DE).

The top level design (top.v) consists of four main modules:

- DPHY_RX_INST.v – Serializes byte data using iDDRx4 gearbox primitives. Controls high impedance and bidirectional states of HS and LP signals.
- Control_Capture_Lane*.v – extracts data packet information. Controls VSYNC and HSYNC outputs.
- Parser_*.v – Converts MIPI byte packet data to pixels based on a particular data type. Controls DE and PIXDATA outputs.
- pll_*bit_*lane.v – provides byte clock and pixel clocks.

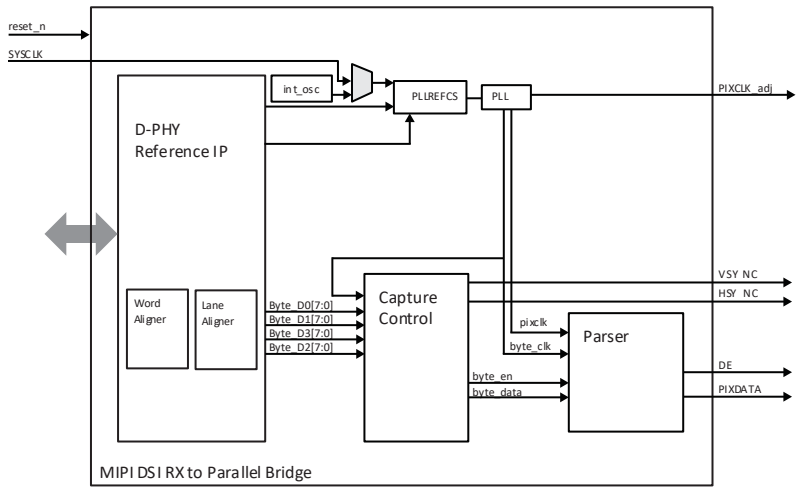

Figure 3.2. Top Level Block Diagram

To control the ports defined at the top level, `define compiler directives are used. These compiler directives can be found in compiler_directives.v.

Table 3.1. Compiler Directives Defined in compiler_directives.v:

Directive	Description
`define HS_3	Generates IO for four HS data lanes.
`define HS_2	Generates IO for three HS data lanes. Overridden if HS_3 is defined.
`define HS_1	Generates IO for two HS data lanes. Overridden if HS_3 or HS_2 is defined.
`define HS_0	Generates IO for one HS data lanes. Overridden if HS_3, HS_2, or HS_1 is defined.
`define LP_CLK	Generates IO for LP mode on clock lane.
`define LP_0	Generates IO for LP mode on data lane 0.
`define LP_1	Generates IO for LP mode on data lane 1.
`define LP_2	Generates IO for LP mode on data lane 2.
`define LP_3	Generates IO for LP mode on data lane 3.
`define word_alignment	Turns on word alignment for each data lane used.
`define lane_alignment	Turns on lane alignment for each data lane used.
`define use_sysclk	Replaces internal oscillator with SYSCLK port.
`define freerunning_clk	Removes clock mux when MIPI clock lane is in “free-running” mode.
`define crossclkfifo	Adds a cross clock FIFO between the gearbox and the aligner to convert from a non-continuous to a continuous. clock

Design parameters control other features of the design. These design parameters are located at the top of the module declaration in top.v.

Table 3.2. Top Level Module Parameters

Parameter	Options	Description
word_width	Up to 36 bits	Bus width of pixel data bus input.
byte_freq	See internal oscillator frequency options in Table 1000	Frequency of internal oscillator when used. Should be set as close to the MIPI byte clock frequency used as possibleMIPI byte clock frequency used as possible.
DT	6-bit Data Type Value	Data Type used to identify byte packets and pixel format to parse.

Top level IO ports are defined as follows for top.v. The number of IO is dependent on the compiler directives defined by compiler_directives.v.

Table 3.3. Top Level Design Port List

Signal	Direction	Description
reset_n	Input	Resets module (Active 'low')
DCK	Input	HS (High Speed) Clock
SYSCLK	Input	Clock on same domain as MIPI clock. Used to replace internal oscillator continuous clock on the same MIPI clock domain
D0	Input	HS Data lane 0
D1	Input	HS Data lane 1
D2	Input	HS Data lane 2
D3	Input	HS Data lane 3
LPCLK [1:0]	Bidirectional	LP clock lane; LPCLK[1] = P wire, LPCLK[0] = N wire
LP0 [1:0]	Bidirectional	LP data lane 0; LP0[1] = P wire, LP0[0] = N wire
LP1 [1:0]	Bidirectional	LP data lane 1; LP1[1] = P wire, LP1[0] = N wire
LP2 [1:0]	Bidirectional	LP data lane 2; LP2[1] = P wire, LP2[0] = N wire
LP3 [1:0]	Bidirectional	LP data lane 3; LP3[1] = P wire, LP3[0] = N wire
PIXCLK	Output	Parallel Pixel Clock
VSYNC	Output	Parallel Vertical Sync Indicator
HSYNC	Output	Parallel Horizontal Sync Indicator
DE	Output	Parallel Data Enable Indicator
PIXDATA[*:0]	Output	Parallel Data Bus

The top level module instantiates and connects the four main modules. The PLL module controls clocking for the entire design. The output of the PLL provides a continuously running clock (byte_clk_fr), a pixel clock (pixclk) and a pixel clock shifted by 180 degrees (PIXCLK_adj) to provide center alignment with the single data rate CMOS output signals.

The input of the PLL is dependent on the compiler directives set. By default the PLL input is driven by a clock mux. This default setup assumes that the MIPI clock lane goes in and out of HS and LP modes, which means that the clock is not continually running. As a result, a free running clock is created by using a clock mux primitive to switch between the HS clock when in HS mode and the internal oscillator while in LP mode. The byte_freq primitive defines the internal oscillator frequency. It should be set to a clock frequency as close to the MIPI byte clock frequency as possible (MIPI serial clock frequency / 4).

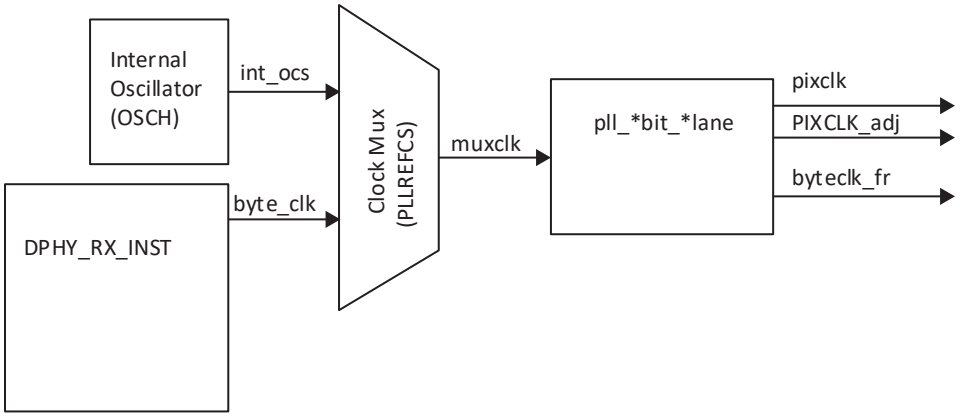

Figure 3.3. Default Clock Configuration Scheme

Table 3.4. 1000 – MachXO2 Internal Oscillator Frequency Options shown in Mhz

2.08	9.17	33.25
2.46	10.23	38
3.17	13.3	44.33
4.29	14.78	53.2
5.54	20.46	66.5
7	26.6	88.67
8.31	29.56	133

An additional option that can simplify the clocking scheme under certain conditions is to use an additional clock input that is on the same clock domain as the MIPI clock. This is often times a clock routed to the transmitting device in order to clock the DSI transmitting IP Core. The “use_sysclk” compiler direct can be used to add a SYSCLK input port. The SYSCLK input goes directly to the PLL input. The PLL can then be adjusted to derive the proper pixel and byte clock frequencies.

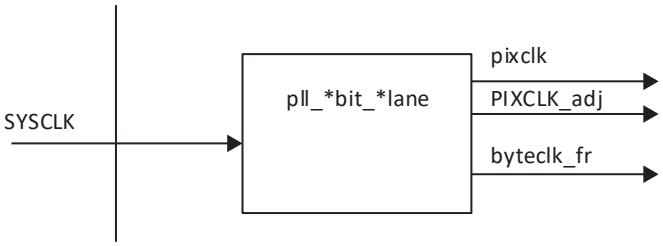

Figure 3.4. SYSCLK Input

The PLL comes preconfigured for a mux_clk, SYSCLK, or byte_clk input of ~39Mhz. Output clocks can be calculated for PIXCLK, PIXCLK_adj and byte_clk_fr (free running byte clock) using Table 3.5.

Table 3.5. Clocking for the PLL Output Ports Calculation

PLL Module Port Name	Clock Description	Clock Equation
CLKI	PLL Input	CLKI
CLKOP	PIXCLK	CLKOP = CLKI * (8 * lane_width) / word_width
CLKOS	oDDRx4 gearbox Clock (90 degree shift)	Same as CLKOP, but with static phase shift of 90 degrees
CLKOS2	Byte Clock	CLKOS2 = CLKI

The PLL is configured using IPExpress in the Lattice Diamond Software. The PLL can be adjusted and reconfigured to individual design needs by double clicking on pll_*bit_*lane.ipx in the file list. An IPExpress configuration GUI will open to adjust the PLL.

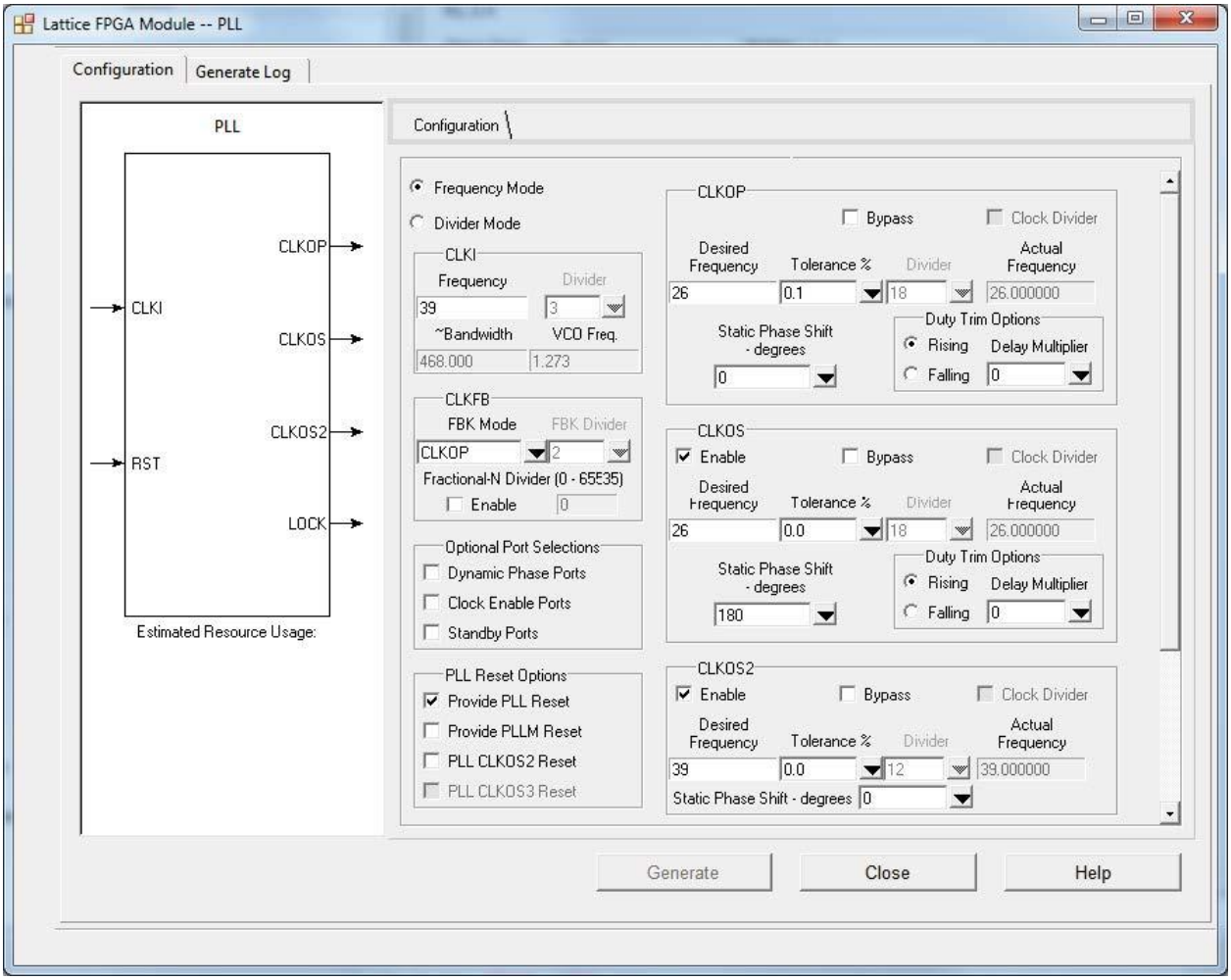

Figure 3.5. IPExpress Configuration Page for pll_24bit_2lane.ipx

The DPHY_RX_INST module deserializes HS data, controls HS termination and LP command identification. The DPHY_RX_INST module is further described in the Lattice MIPI D-PHY Reference IP (RD1182) documentation. Word alignment and lane alignment are both turned on within the module. This allows the byte data output of the module to be completely aligned to the byte clock. hs_en and term_en signals are controlled by the observation of entrance from LP mode to HS mode on the clock lane and the burst_done output port of the control_capture_* module.

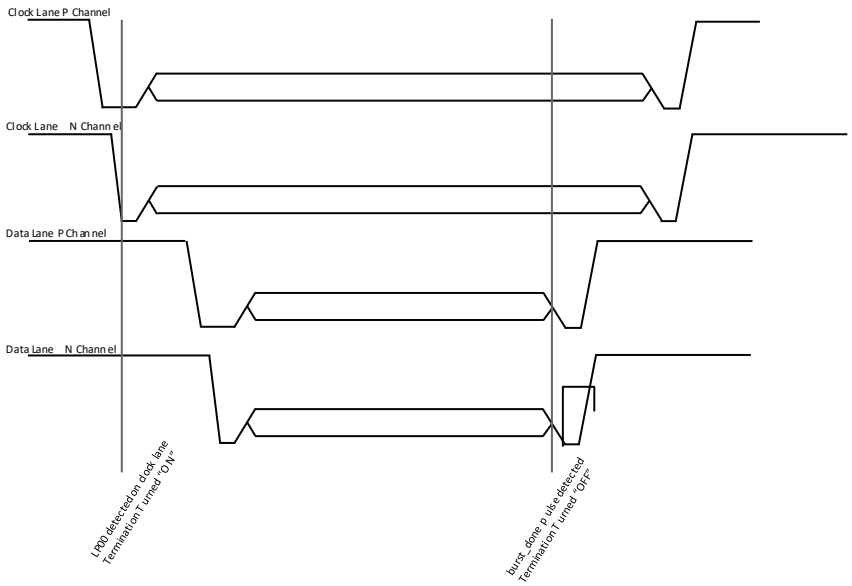

Figure 3.6. Timing Diagram

DT, WC and ECC ports are updated when PH_enable is active “high.” Chksum is updated when PF_en is active “high.” The PH_en and PF_en signals stay active for one clock cycle; the data ports retain the packet header and footer information until the next PH_en or PF_en pulse goes active. Control_Capture*Lane module also is used to identify short and long packets via long_data_en and short_data_en. Short_data_en goes active “high” for one clock cycle. Long_data_en goes active for multiple clock cycles. Both of these enables go active on the same clock cycle as PH_en. Long_data_en stays active based on the number of clock cycles identified from the word count of the packet header. The data on byte_data_out is realigned so that the first set of byte data is available when long_data_goes active ‘high.’ This realignment makes it easy to handle consecutive HS packet bursts of data. Regardless of what lane a data packet starts on within a burst, the module will always align byte 0 to byte_data_out[7:0], byte 1 to byte_data_out[15:8] (in the case of two or four lane), byte 2 to byte_data_out[23:16] (in the case of four lane) and so on. The burst_done signal is used to identify when no new short or long packets are seen on the data bus. This signal goes active for one clock cycle after the last short or long packet is seen. This can be useful for and is use in the design by default to disable termination as it is expected that each data lane will enter LP mode shortly after.

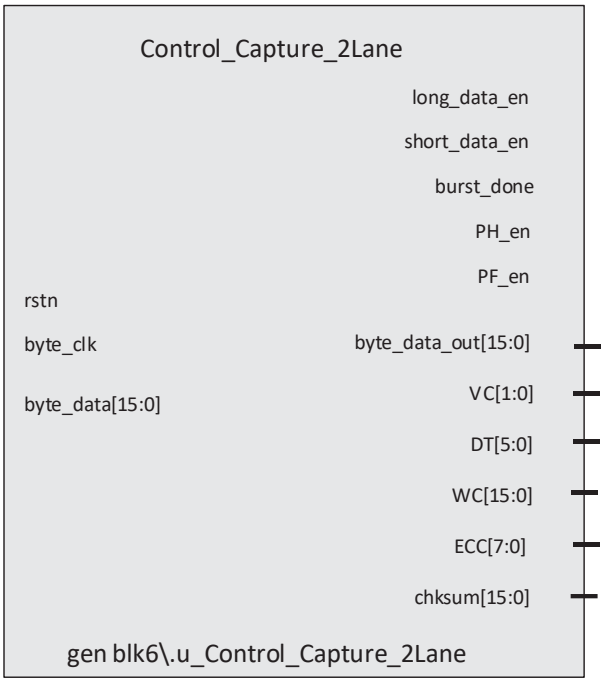

Figure 3.7. Example instantiation Diagram of Control_Capture_2lane

At the top level of the design VSYNC and HSYNC ports are controlled by comparing the short_data_en signal and the DT (data type) ports of Capture_Control_*Lane. Additionally, a byte_en signal is defined, by comparing long_data_en and DT (data type) ports, as well as the DT parameter defined at the top level. The byte_enable signal is used to identify a particular long packet desired by the user. The signal is fed to the byte_en port of the parser* module, which is a module developed specifically to convert the data type specified by the DT parameter. This module converts byte data to pixels based on the MIPI DSI specification for that data type. The output of the parser* module is the DE (data enable) signal and the PIXDATA bus which are both fed to top level ports.

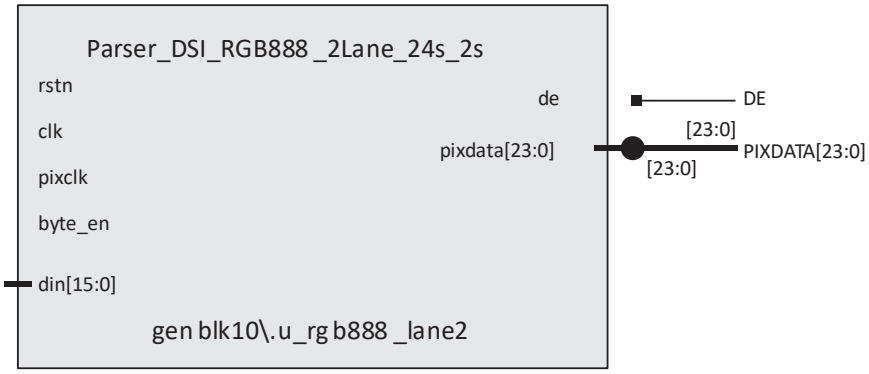

Figure 3.8. Example Instantiation Diagram of Parser_DSI_RGB888_2Lane_24s_2s

Pixel data formatting is dependent on the data type. Table 3.6. shows the format for each data type.

Table 3.6. DSI Pixel Data Order

DATA TYPE	Cycle	FORMAT
RGB121212	Any	PIXELDATA[35:0]={R[11:0], G[11:0], B[11:0]}
RGB101010	Any	PIXELDATA[29:0]={R[9:0], G[9:0], B[9:0]}
RGB888	Any	PIXELDATA[23:0]={R[7:0], G[7:0], B[7:0]}
RGB666	Any	PIXELDATA[17:0]={R[5:0], G[5:0], B[5:0]}
RGB565	Any	PIXELDATA[15:0]={R[4:0], G[5:0], B[4:0]}
YCbCr 4:2:2 24-bit	Odd Pixels	PIXELDATA[23:0]={Cb[11:0], Y[11:0]}
	Even Pixels	PIXELDATA[23:0]={Cr[11:0],Y[11:0]}
YCbCr 4:2:2 20-bit	Odd Pixels	PIXELDATA[19:0]={Cb[9:0], Y[9:0]}
	Even Pixels	PIXELDATA[19:0]={Cr[9:0],Y[9:0]}
YCbCr 4:2:2 16-bit	Odd Pixels	PIXELDATA[15:0]={Cb[7:0], Y[7:0]}
	Even Pixels	PIXELDATA[15:0]={Cr[7:0],Y[7:0]}
YCbCr 4:2:0 12-bit	Odd Lines	PIXELDATA[11:0]={Cb1[7:0], Y1[7:4]}, {Y1[3:0],Y2[7:0]}
	Even Lines	PIXELDATA[11:0]={Cr1[7:0], Y1[7:4]}, {Y1[3:0],Y2[7:0]}

4. Packaged Design

The Parallel to MIPI DSI RX Bridge Reference Design is available for Lattice MachXO2 devices. The reference design immediately available on latticesemi.com is configured for RGB888, 2-lane mode. Other designs are available through the bridge request form. The packaged design contains a Lattice Diamond project within the *\impl\ folder configured for the MachXO2 device. Verilog source is contained within the *\rtl\ folder. The Verilog test bench is contained within the tb folder. The simulation folder contains an Aldec Active-HDL project. It is recommended that users access the active HDL Simulation environment through the Lattice Diamond Software and the simulation setup script contained within the project. For details on how to access the design simulation environment see the “Functional Simulation” section of this document.

Figure 4.1. Packaged Design Directory Structure

5. Functional Simulation

The simulation environment utilizes a pattern_gen module instantiated in the testbench to generate transmit data. The pattern generated from pattern_gen.v is an actual recorded frame from a DSI host. You will see that pattern_gen module burst multiple short and long packets within an HS transfer. The DSI2Parallel_tb.v testbench utilizes serializes the data from pattern_gen and sends it to the input of the bridge design.

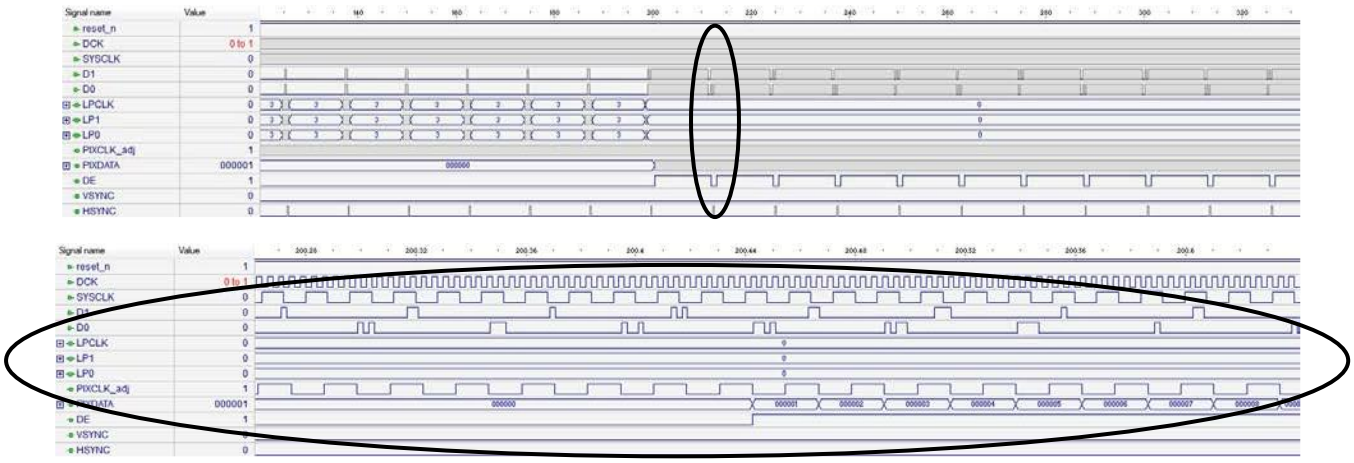

Figure 5.1. Simulation Waveforms

The simulation environment can be accessed by double clicking on the Simulation.spf script file in Lattice Diamond from the file list. Aldec ActiveHDL will then open after clicking OK to the pop-up windows. Compile the project and initialize the simulation. Add signals to the waveform viewer that are desired to be viewer and run the simulation.

6. Design Testing and Demonstration

MIPI DSI RX to Parallel Bridge Reference Design was tested using an Intrinsyc Snapdragon S4 development platform and a Lattice MachXO2-4000 FPGA in cs132bga package. The incoming DSI video stream was decoded and VSYNC, HSYNC and DE signals were analyzed on a scope. Additionally, the Lattice Reveal software capture tool was used to capture all output signals including the pixel data bus.

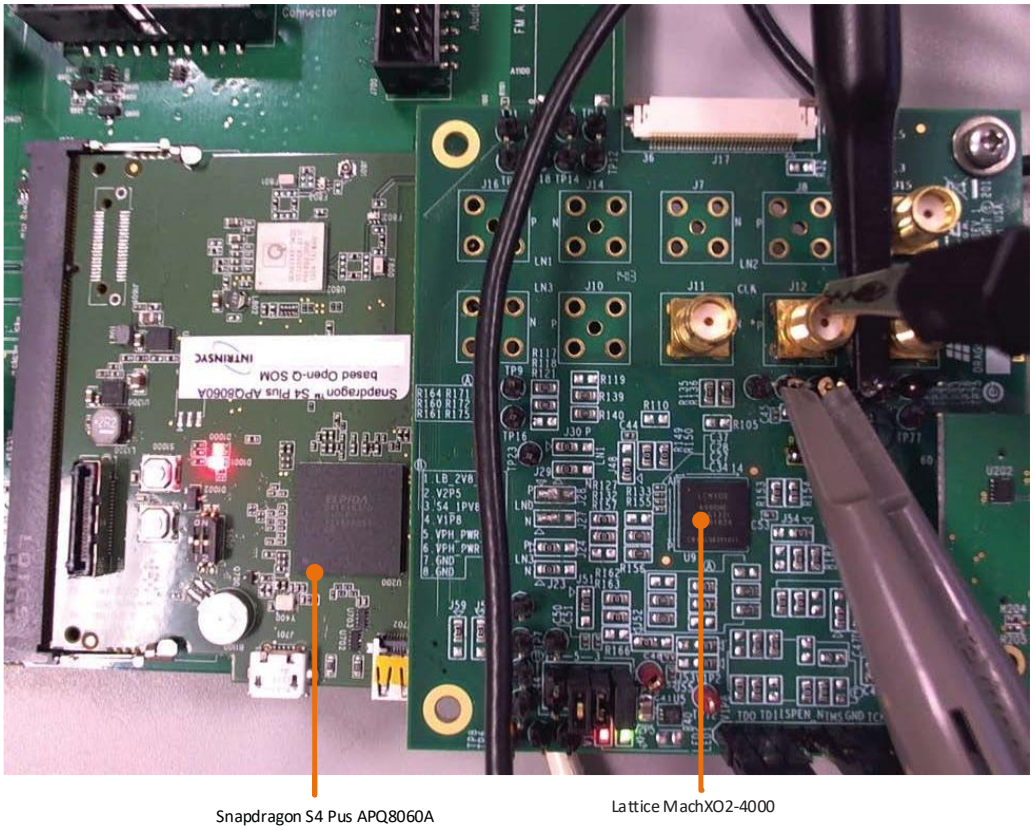

Figure 6.1. Hardware Development and Testing using a MIPI DSI Display

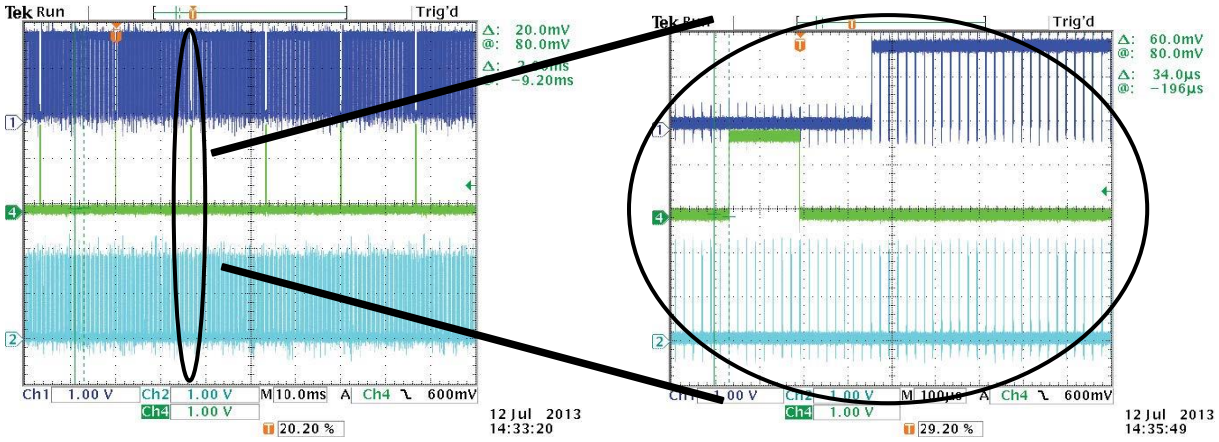

Figure 6.2. Oscilloscope Screenshot of DE (top), VSYNC (middle), and HSYNC (bottom) Output Signals

Figure 6.3. Lattice Reveal Capture of DE, VSYNC, HSYNC, and PIXDATA sampled with PIXCLK

7. External Resistor Network Implementation for D-PHY RX

As described in the Lattice MIPI D-PHY Interface Reference IP documentation (RD1182, MIPI D-PHY Reference IP), an external resistor network is needed to accommodate the LP and HS mode transitioning on the same signal pairs as well as to provide 50 ohm series termination while in HS mode. The resistor network needed for MIPI RX implementations is provided below.

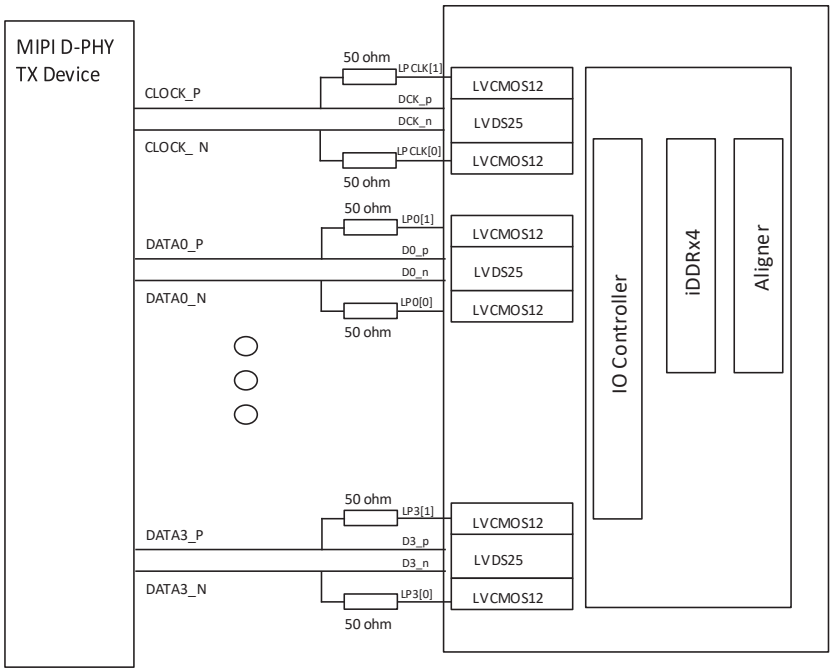

Figure 7.1. Unidirectional HS Mode and Bidirectional LP Mode Interface Implementation

8. Device Pinout and Bank Voltage Requirements

Choosing a proper pinout to interface with another D-PHY device is essential to meet functional and timing requirements.

The following are rules for choosing a proper pinout on MachXO2 devices:

- Bank 2 should be used for HS inputs (DCK, D0, D1, D2, D3) with the RX D-PHY IP since these pins utilize
- iDDRx4 gearbox primitives
- The VCCIO voltage for banks 2 should be 2.5V
- The HS input clock (DCK) for the RX D-PHY IP should use an edge clock on bank 2
- The HS data signals (D0, D1, D2, D3) for the RX D-PHY IP should only use A/B IO pairs
- LP signals (LPCLK, LP0, LP1, LP2, LP3) for RX D-PHY IP can use any other bank
- The VCCIO voltage for the bank containing LP signals (LPCLK, LP0, LP1, LP2, LP3) should be 1.2 V
- When in doubt, run the pinout through Lattice Diamond software can check for errors

With the rules mentioned above a recommend pinout is provided for the most common package chosen for this reference design. For the MachXO2 the cs132bga is the most common package. The pinouts chosen below are pin compatible with MachXO2-1200, MachXO2-2000 and MachXO2-4000 devices.

Table 8.1. Recommended RX Pinout and Package

Signal	MachXO2 1200/2000/4000 cs132bga Package	
DCK_p	Bank 2	N6
DCK_n		P6
D0_p		M11
D0_n		P12
D1_p		P8
D1_n		M8
D2_p		P2
D2_n		N2
D3_p		N3
D3_n		P4
LPCLK [1]	Bank 1	J13
LPCLK [0]		K12
LP0 [1]		K13
LP0 [0]		K14
LP1 [1]		L14
LP1 [0]		M13
LP2 [1]		M12
LP2 [0]		M14
LP3 [1]		N13
LP3 [0]		N14

Table 8.2. RX IO Timing

Device Family	Synthesis Engine	Speed Grade -4		Speed Grade -5		Speed Grade -6	
		Setup (ps)	Hold (ps)	Setup (ps)	Hold (ps)	Setup (ps)	Hold (ps)
MachXO2		197	337	213	254	222	175
MachXO3L	LSE (Lattice Synthesis Engine)			248	292	243	158
	Synplify Pro®			248	292	243	158

Table 8.3. RX Maximum Operating Frequencies by Configuration[1]

Device Family	Synthesis Engine	Configuration	Speed Grade -4 (MHz)		Speed Grade -5 (MHz)		Speed Grade -6 (MHz)	
MachXO2			PIXCLK	byte_clk	PIXCLK	byte_clk	PIXCLK	byte_clk
MachXO3L[2]	LSE	RGB888, 1-Lane	128.4	90.3	142.8	99.1	155.7	110
	Synplify Pro	RGB888, 2-Lane	131.5	77.6	123.6	89.1	143.3	110.8
		RGB888, 4-Lane	78.4	67.5	143.9	80.3	160.1	90.4
		RGB888, 2-Lane			123.6	89.1	143.3	110.8
					121.9	82.5	151.8	100.5

Notes:

1. The maximum operating frequencies were obtained by post P&R timing analysis. They do not correlate to clocking ratios (obtained from PLL clock equations) used for proper design operation.
2. The maximum operating frequencies were obtained for LCMXO3L-4300C device.

9. Resource Utilization

The resource utilization tables below represent the device usage in various configurations of the reference design. Resource utilization was performed on the design in configurations of 1, 2 and 4 data lanes. For each of these configurations LP mode on the data lanes used was turned on. In addition, HS and LP clock signals were available for each configuration. The data type for the resource utilization number was configured for RGB888. It's important to note that resource utilization numbers can change depending on the number of data lanes and the data type used.

Table 9.1. RX Resource Utilization

Device Family	Configuration	Register	LUT	EBR	PLL	Gearbox	Clock Divider
MachXO2	RGB888, 1-Lane	290	211	4	1	1	1
	RGB888, 2-Lane	623	350	5	1	2	1
	RGB888, 4-Lane[1]	1100	903	6	1	4	1
MachXO3L	RGB888, 2-Lane	561	397	5	1	2	1

Note: The "crossclkfifo" compiler directive feature was turned off in the case of RGB888, 4-lane. This allowed the design to fit in an XO2-1200; it reduced the resource utilization to under 600 slices.

References

- MIPI Alliance Specification for Display Serial Interface (DSI) V1.1
- MIPI Alliance Specification for D-PHY V1.1

Technical Support Assistance

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

Revision History

Revision 1.4, November 2019

Section	Change Summary
All	• Changed document number from RD1185 to FPGA-RD-02068. • Updated document template.
Disclaimers	Added this section.

Revision 1.3, April 2014

Section	Change Summary
Device Pinout and Bank Voltage Requirements	Updated the following tables to add support for MachXO3L device family: • Table 8.2, RX10 Timing. • Table 8.3, RX Maximum Operating Frequencies by Configuration • Table 9.1, Resource Utilization

Revision 1.2, December 2013

Section	Change Summary
Functional Description	Added DSI Pixel Data Order table in Functional Description section.

Revision 1.1, August 2013

Section	Change Summary
Device Pinout and Bank Voltage Requirements	Updated Table 8.3. title to RX Maximum Operating Frequencies by Configuration and added footnote.

Revision 1.0, August 2013

Section	Change Summary
All	Initial release.

www.latticesemi.com