UG0826
User Guide
PolarFire MIPI CSI-2 Transmitter

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

50200826. 6.0 7/21

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

1.1 Revision 6.0

The following is a summary of changes made in this revision.

- Added support of 8-lanes configuration for all the supported data types.
- Replaced Figure 13, page 10.

1.2 Revision 5.0

- MIPI CSI2 Transmitter IP is updated with several features as follows:
  - Compile Time Configuration
  - Run Time Configuration
  - Support of 4 pixels per clock
  - Added support for Data types like Raw12, Raw14, Raw16 and RGB-888 for 1, 2, and 4 lanes
- Replaced Figure 8, page 7 and Figure 18, page 13.
- Added following sections:
  - Key Features, page 2, Supported Families, page 2, MIPI CSI-2 Transmitter IP Configuration, page 2, Inputs and Outputs, page 5, Hardware Implementation for Compile Time, page 6, Hardware Implementation for Run Time, page 9, and Short Packet, page 12.
  - License, page 14, Installation Instructions, page 15, and Resource Utilization, page 16.

1.3 Revision 4.0

Revision 4.0 was published in June 2019. In this revision, the document was updated for Libero SoC PolarFire v12.1.

1.4 Revision 3.0

Revision 3.0 was published in October 2018. In this revision, the document was updated for Libero SoC PolarFire v2.3.

1.5 Revision 2.0

Revision 2.0 was published in September 2018. In this revision, RAW10 data type support is added to the IP, through out the document.

1.6 Revision 1.0

The first publication of this document.

2 Introduction

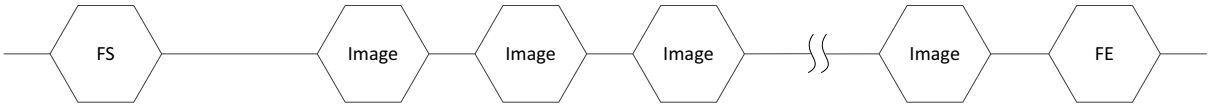
MIPI CSI-2 is a standard specification defined by Mobile Industry Processor Interface (MIPI) Alliance. The Camera Serial Interface 2 (CSI-2) specification defines an interface between a peripheral device (camera) and a host processor (baseband, application engine). This user guide describes the MIPI CSI-2 transmitter, which encodes the pixel data compliant to MIPI CSI-2 standard. The IP Core supports multi-lane (1, 2, 4, and 8 lanes) for RAW8, RAW10, RAW12, RAW14, RAW16, and RGB-888 data types.

MIPI CSI-2 transmitter operates in two modes—high-speed mode and low-power mode. In high-speed mode, MIPI CSI-2 supports the transport of image data using short and long packets. Short packets provide frame synchronization and line synchronization information. Long packet provides the pixel information. The sequence of transmitted packets is:

1. Frame start (short packet)
2. Few image data packets (long packets)
3. Frame End (short packet)

One long packet is equivalent to one image data line. The following figure shows the video data stream.

Figure 1 • **Video Data Stream**



Note:

FS: Frame Start Packet (short packet)

Image: Pixel data of image embedded in Long Packet

FE: Frame End Packet (short packet)

2.1 Key Features

- Supports Raw8, Raw10, RAW12, RAW14, RAW16, and RGB-888 data types for 1, 2, 4, and 8 lanes
- Supports 4 pixels per input clock
- Supports configuration of IP for Data type and MIPI Lanes at Compile time and Run time

2.2 Supported Families

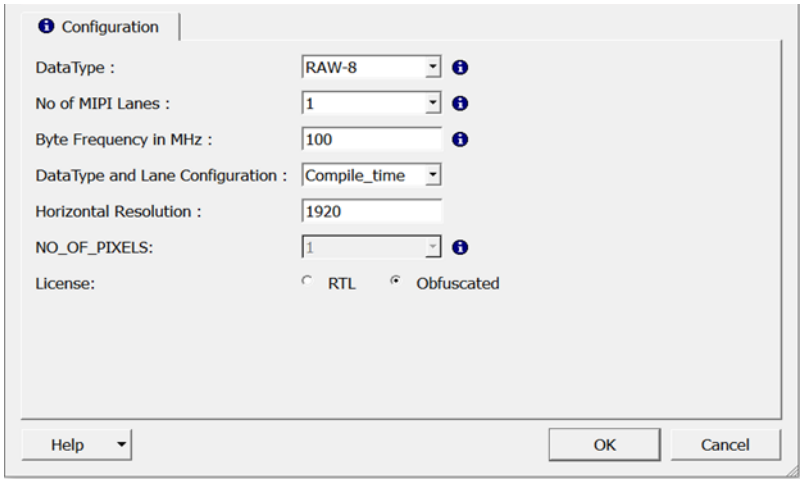
- PolarFire® SoC
- PolarFire®

2.3 MIPI CSI-2 Transmitter IP Configuration

MIPI CSI-2 Transmitter can be configured in two modes:

- Change of Data type and Number of MIPI Lanes at Compile time

Figure 2 • **MIPI CSI-2 Transmitter in Compile time Mode**

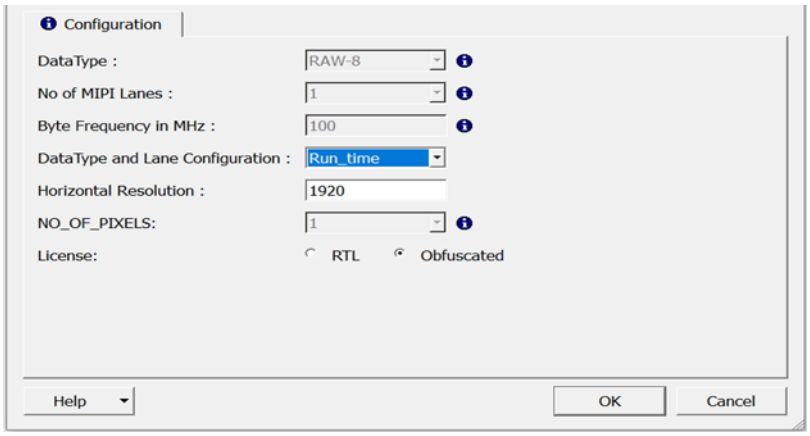


At compile time user can configure Data type, number of MIPI Lanes, Byte Frequency in MHz, and Horizontal Resolution.

Note: If the value of byte frequency has a decimal value, then consider its floor value to configure the byte frequency. For example, if the byte frequency has a decimal value of 148.2 MHz or 148.6 MHz, then configure with the floor value of 148 MHz in the IP configurator.

- Change of Data type and Number of MIPI Lanes at Run time

Figure 3 • **MIPI CSI-2 Transmitter in Run time Mode**



Data type and several MIPI Lanes can be changed dynamically by Input ports at the Run time. Horizontal Resolution can be changed at the Compile time.

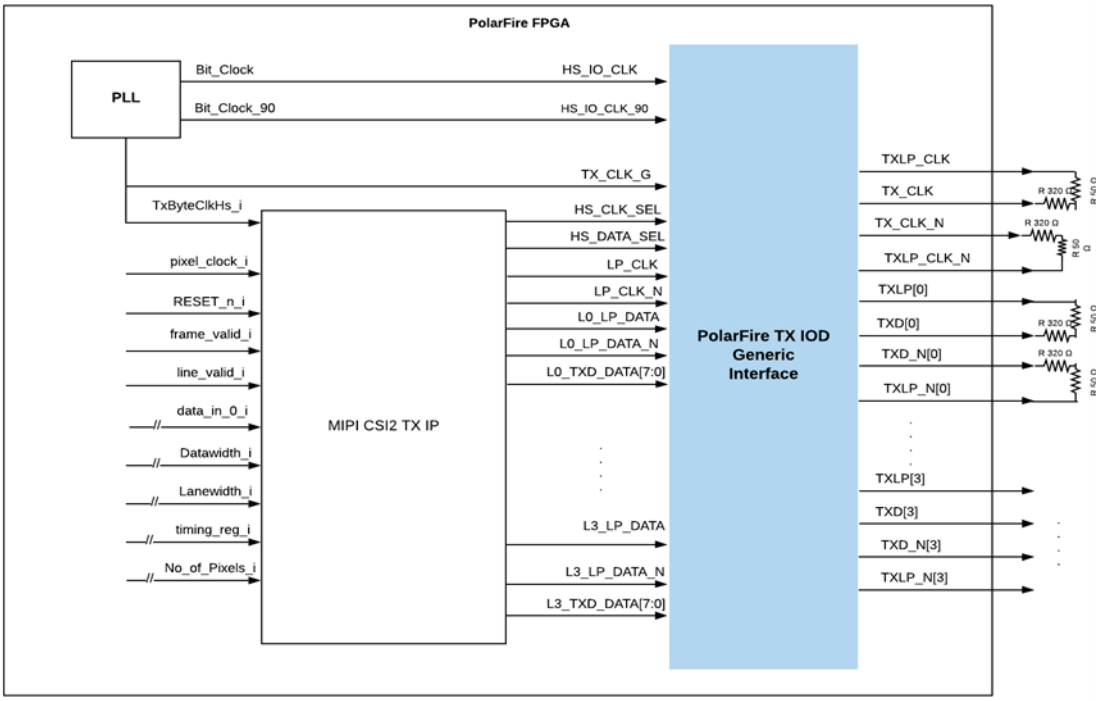
3 Hardware Implementation

3.1 Architecture of MIPI CSI-2 Transmitter

The following figure shows the MIPI CSI2 Transmitter solution that contains the MIPI CSI2 Tx IP. This IP is used in conjunction with the PolarFire MIPI IOD generic interface block and PLL. The figure shows the pin connections from the MIPI CSI2 Tx IP to the PolarFire IOG. A PLL is required to generate the TxByteClkHs_i clock (Byte clock). The input clock to the PLL is the Pixel clock. The PLL is configured to produce the TxByteClkHs_i clock, the MIPI bit clock and 90° phase shifted Bit_clock_90, whose frequency is based on the pixel clock, and the number of lanes used.

An external resistor network as shown in following figure is needed to accommodate low power (LP) and high-speed (HS) mode transitioning on the same signal pairs and to set the voltage swing to 200 mV during HS clock and data transfers.

Figure 4 • **MIPI CSI-2 Transmitter Architecture**



3.2 Inputs and Outputs

The following table lists the input and output ports of the MIPI CSI-2 Tx configuration parameters.

Table 1 • **Inputs and Outputs Ports of MIPI CSI-2 Transmitter**

Signal Name	Direction	Width	Port Valid under	Description
RESET_n_i	Input	1		Active Low Asynchronous reset signal to design.
pixel_clock_i	Input	1		Input clock with which incoming pixels are sampled
TxByteClkHs_i	Input	1		Tx Byte Clock (gearing ratio of 4). This clock must be configured such that the pixels sent on the MIPI CSI2 Interface are sampled according to it.
frame_valid_i	Input	1		Asserts High for every valid frame
line_valid_i	Input	1		Asserts High for every valid packet
data_in_0_i	Input	[31:0]	No_of_Pixels = 1 or 4	Input Pixel data
data_in_(1-3)_i	Input	[31:0]	No_of_Pixels = 4	Input Pixel data when there are 4 pixels per clock
Datawidth_i	Input	[7:0]	Run Time	Supports for RAW8, RAW10, RAW12, RAW14, RAW16, and RGB-888 data types.
Lanewidth_i	Input	[3:0]	Run Time	Supports for 1, 2, 4, and 8 lanes
timing_reg_i	Input	[11:0]	Run Time	D-PHY timing values under Run time Configuration. 11:8 - Used for address detection 7:6 - Reserved 5:0 - Timing values in no of byte clocks
No_of_Pixels_i	Input	[2:0]	Run Time	It is configured either 1 pixel per clock or 4 pixels per clock
L(0 - 3)_LP_DATA	Output	1		Low Power Data (P side)
L(0 - 3)_LP_DATA_N	Output	1		Low Power Data (N side)
LP_CLK	Output	1		Low Power Clock (P side)
LP_CLK_N	Output	1		Low Power Clock (N side)
L(0 - 3)_TXD_DATA	Output	[7:0]		Lane 0 to Lane 3 Transmit bytes
HS_CLK_SEL	Output	1		High Speed mode clock select
HS_DATA_SEL	Output	1		High Speed mode data select

3.3 Configuration Parameters

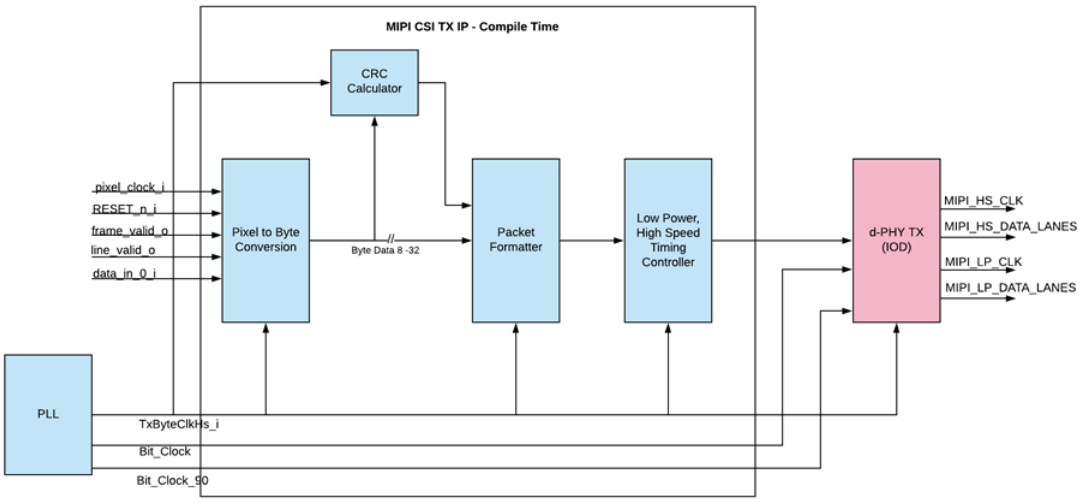
Table 2 • **Configuration Parameters**

Parameter Name	Configurator Prompt	Parameter Valid under	Description
g_DATAWIDTH	Data Type	Compile Time	Input pixel data-width. Supports for RAW8, RAW10, RAW12, RAW14, RAW16, and RGB-888
g_LANE_WIDTH	No of MIPI Lanes	Compile Time	Supports for 1, 2, 4, and 8 lanes
g_HORIZONTAL_RESOLUTION	Horizontal Resolution	Compile Time / Run Time	Active horizontal resolution
g_TX_BYTE_FREQ	Byte Frequency in MHz	Compile Time	Byte Frequency in MHz derived by calculation. Refer PLL, page 8.
g_EXECUTION_TYPE	Data type and Lane Configuration		Can be configured either in Compile Time or Run Time
g_NO_OF_PIXELS	NO_OF_PIXELS	Compile Time	The following options are available: 1 - One pixel per clock 4 - Four pixels per clock

3.4 Hardware Implementation for Compile Time

The following figure shows the block diagram of MIPI CSI-2 Transmitter in Compile Mode.

Figure 5 • **Block diagram of MIPI CSI-2 Transmitter in Compile Mode**



3.4.1 Design Description for Compile Time

This section describes the different internal modules of the MIPI CSI-2 transmitter core in compile time mode.

3.4.1.1 Pixel to Byte Conversion

This module converts the incoming pixel data to bytes based on the configured IP's number of lanes. The user is expected to transmit the pixels along with the control signals line_valid_i and frame_valid_i. It uses an internal clock crossing FIFO to convert the incoming data from pixel clock to byte clock domain. It also generates the byte enable signal, which indicates the valid byte data.

3.4.1.2 Packet Formatter

This module consists of two blocks – Header insertion and lane distribution block. On detecting the control signals (frame_valid_i), it transmits the frame start short packet, then long packet with the header inserted and the data from the pixel to byte conversion module. When vertical resolution number of packets are generated, frame end short packet is generated. It also calculates the Error Correction Code (ECC) and appends it to the packet header.

Figure 6 • **Packet Assembly for Long Packet**

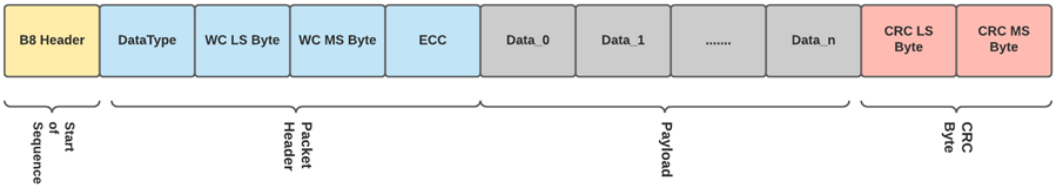


Figure 7 • **Packet Assembly for Short Packet**

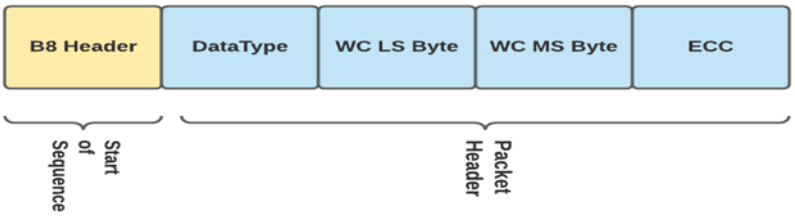
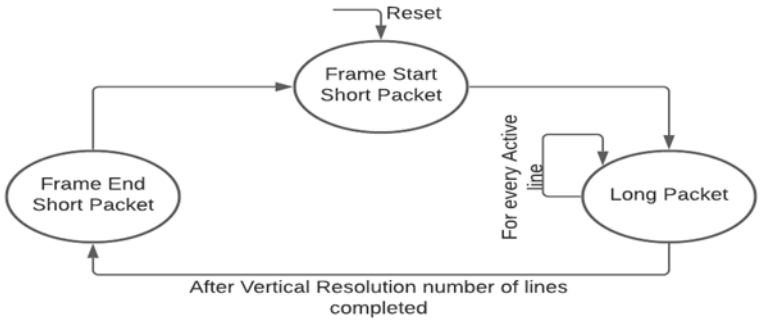


Figure 8 • **FSM Implementation of High-Speed Data Generation**



3.4.1.3 PLL

Pixel_clock_i is the input clock with which incoming pixels are sampled. A PLL is used to generate the Byte clock (TxByteClkHs_i) and bit clocks used by the MIPI DPHY block (PolarFire IOD). TxByteClkHs_i must be configured such that the output MIPI CSI2 compliant packets sent on the interface are sampled. The following equations show the relation between Pixel_clock_i and TxByteClkHs_i depending on the number of lanes configured.

TxByteClkHs_i = (Pixel_clock_i × Bits per pixel) / (Number of lanes × 8)

MIPI bit clock = 4 × TxByteClkHs_i

Two serial MIPI bit clocks are required—0° and 90° phase shifted.

MIPI CSI-2 TX IP supports RAW8, RAW10, RAW12, RAW14, RAW16, and RGB-888 data types. For RAW8 data type, one 8-bit pixel is transmitted per clock, and for RAW10 data type, one 10-bit pixel is transmitted per clock.

3.4.1.4 Low Power/High-Speed

When the MIPI packet enable is asserted, transition to high-speed mode follows the following sequence: LP-11, LP-01, LP-00, HS0/1. It indicates the HS request path and following the timing based on MIPI DPHY Specification version 1.1.
LP request, Escape mode and Turn around modes are not supported.

Figure 9 • **Clock Lane Switching between Clock Transmission and LP mode**

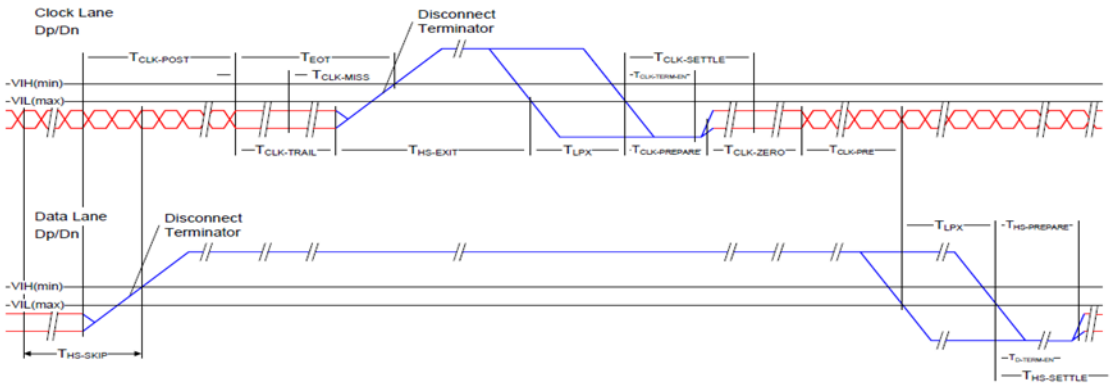
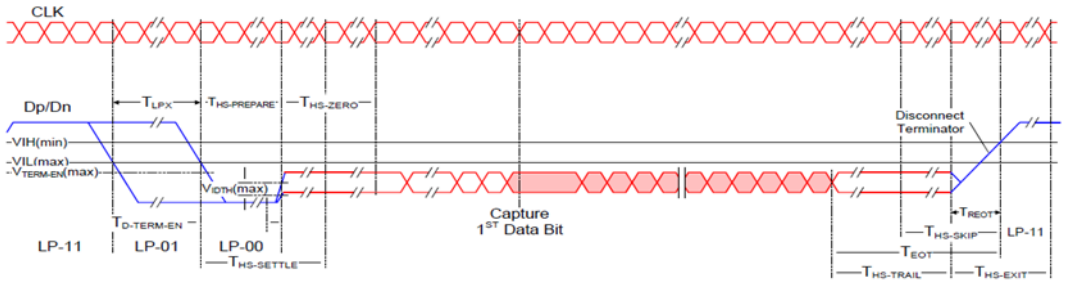


Figure 10 • **HS Data Transmission Control**



3.4.1.5 D-PHY TX

This module uses PolarFire IOD generic blocks to convert Byte data to serial data. A gearing ratio of four is used to convert the parallel data to serial data. It generates both HS and LP signals (for both clock and data). It also switches between HS and LP modes using the HS_CLK_SEL and HS_DATA_SEL signals.

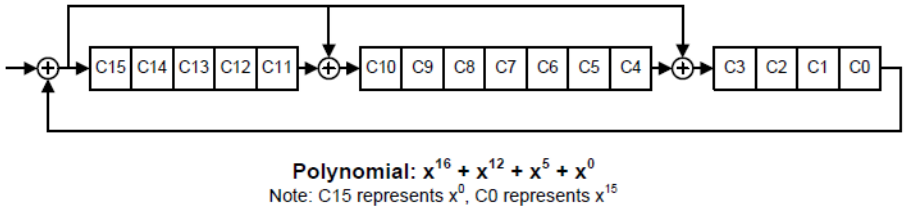
3.4.1.6 CRC Calculator

This module uses the bytes generated from the pixel to byte conversion module and calculates the 16-bit CRC for the generated bytes. This 16-bit CRC is sent to the packet formatter, which appends the value at the end of the long packet.

The checksum is realized as 16-bit CRC based on the generator polynomial:

$X^{16}+X^{12}+X^5+X^0$

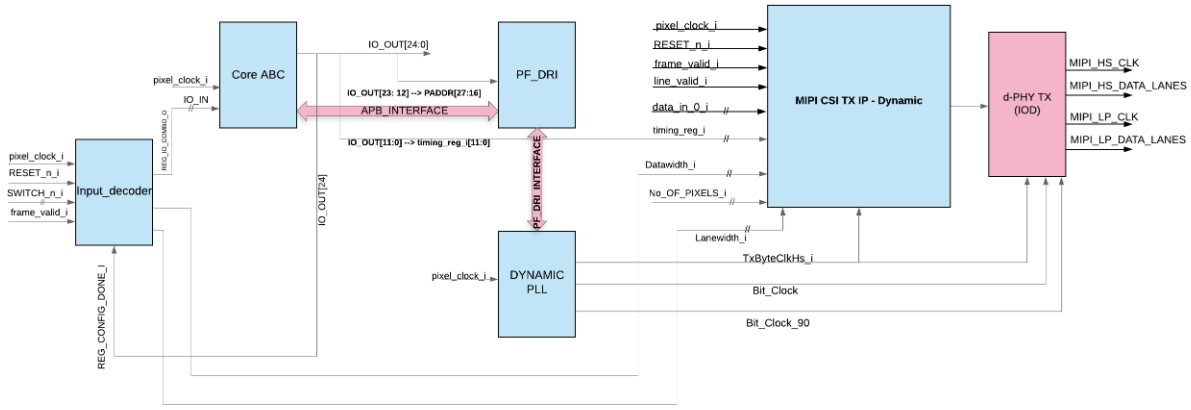
Figure 11 • **CRC implementation**



3.5 Hardware Implementation for Run Time

The following figure shows the block diagram of MIPI CSI-2 Transmitter in Dynamic Mode.

Figure 12 • **Block Diagram of MIPI CSI-2 Transmitter in Dynamic Mode**



3.5.1 Design Description for Run Time

This section describes the different internal modules of the MIPI CSI-2 transmitter core in Run time mode in addition to compile time modules.

3.5.1.1 Dynamic PLL

A PLL with DRI port is used to generate the HS_IO_clock, HS_IO_clock_90, and TXbyte_clock_i by dynamically configuring the PLL Registers through DRI (Dynamic Reconfiguration Interface). The base address of PLL are as follows:

- Base Adder for PLL_SE_0 is 0x8010000
- Base Adder for PLL_SE_1 is 0x8020000
- Base Adder for PLL_NE_0 is 0x8040000
- Base Adder for PLL_NE_1 is 0x8080000
- Base Adder for PLL_NW_0 is 0x8100000
- Base Adder for PLL_NW_1 is 0x8200000
- Base Adder for PLL_SW_0 is 0x8400000
- Base Adder for PLL_SW_1 is 0x8800000

The following are the set of PLL registers that need to be configured to achieve dynamic clocking:

- RF DIV
- FB DIV
- Post Div clock 0_1
- Post Div clock 2_3
- Phase

Figure 13 • **PLL Register Configuration for various Data types**

Switch {Switch3, Switch2, Switch1, Switch0}	Datawidth	Lanes	DL SELECT (1 Hot Encoding)	REF DIV Value in decimal	REF DIV write data	FB DIV value in Decimal	FB DIV write data	POST DIV value for HS Clock in Decimal	POST DIV write Data for HSCLK_0 and HSCLK_90	POST DIV value for Byte Clock in Decimal	POST DIV write Data-Byte Clock
1	8	1	1	4	32'h00000400	64	32'h00000040	1	32'h01000100	4	32'h04000400
2	10	1	2	2	32'h00000200	40	32'h00000028	1	32'h01000100	4	32'h04000400
3	12	1	4	2	32'h00000200	48	32'h00000030	1	32'h01000100	4	32'h04000400
4	14	1	8	4	32'h00000400	112	32'h00000070	1	32'h01000100	4	32'h04000400
5	24	1	16	2	32'h00000200	96	32'h00000060	1	32'h01000100	4	32'h04000400
6	8	2	32	4	32'h00000400	32	32'h00000020	1	32'h01000100	4	32'h04000400
7	10	2	64	2	32'h00000200	20	32'h00000014	1	32'h01000100	4	32'h04000400
8	12	2	128	2	32'h00000200	24	32'h00000018	1	32'h01000100	4	32'h04000400
9	14	2	256	2	32'h00000200	28	32'h0000001C	1	32'h01000100	4	32'h04000400
10	24	2	512	2	32'h00000200	48	32'h00000030	1	32'h01000100	4	32'h04000400
11	8	4	1024	4	32'h00000400	16	32'h00000010	1	32'h01000100	4	32'h04000400
12	10	4	2048	4	32'h00000400	20	32'h00000014	1	32'h01000100	4	32'h04000400
13	12	4	4096	4	32'h00000400	24	32'h00000018	1	32'h01000100	4	32'h04000400
14	14	4	8192	4	32'h00000400	28	32'h0000001C	1	32'h01000100	4	32'h04000400
15	24	4	16384	2	32'h00000200	24	32'h00000018	1	32'h01000100	4	32'h04000400
16	16	1	32768	2	32'h00000200	64	32'h00000040	1	32'h01000100	4	32'h04000400
17	16	2	65536	2	32'h00000200	32	32'h00000020	1	32'h01000100	4	32'h04000400
18	16	4	131072	4	32'h00000400	32	32'h00000020	1	32'h01000100	4	32'h04000400
19	8	8	262144	8	32'h00000800	16	32'h00000010	1	32'h01000100	4	32'h04000400
20	10	8	524288	8	32'h00000800	20	32'h00000014	1	32'h01000100	4	32'h04000400
21	12	8	1048576	8	32'h00000800	24	32'h00000018	1	32'h01000100	4	32'h04000400
22	14	8	2097152	8	32'h00000800	28	32'h0000001C	1	32'h01000100	4	32'h04000400
23	16	8	4194304	8	32'h00000800	32	32'h00000020	1	32'h01000100	4	32'h04000400
24	24	8	8388608	8	32'h00000800	48	32'h00000030	1	32'h01000100	4	32'h04000400

The PLL is configured from Core-ABC IP for RAW8, RAW10, RAW12, RAW14, RAW16, and RGB-888 data types for 1, 2, 4, and 8 lanes.

Note: For more information about PLL register configuration, refer to *PolarFire-device-register-map.*

3.5.1.2 Input Decoder

This module is used to generate the inputs for Core-ABC IP and MIPI CSI2 Transmitter IP. Depending on the values of input switches of the input decoder, this module will generate the output values for Core-ABC IP in a one-hot encoding manner, whereas Data_width_O and Lanewidth_O outputs for MIPI CSI2 Transmitter IP.

3.5.1.3 Core-ABC

Core-ABC is a light weight soft processor that can be used to program PLL registers, Input values to PF_DRI and MIPI CSI2 TX IP for various data types and lanes.

D-PHY timing values in dynamic mode have to be configured from Core-ABC. The following are the equations to calculate timing values.

Figure 14 • **D-PHY Timing Calculations**

```
UI            = 1000 / (g_TX_BYTE_FREQ * 8);                                          // UI= Unit Interval
BYTECLK_NS    = 1000 / g_TX_BYTE_FREQ;                                                // Byte clock in n-sec
T_LPX         = (50 / BYTECLK_NS) + 1;                                                // 50ns min
T_CLK_PREPARE = g_TX_BYTE_FREQ < 22 ? 1 : (50 / BYTECLK_NS) + 1;                      // 38-95 ns
T_CLK_ZERO    = (250 / BYTECLK_NS) + 1;                                               // CLK-PREPARE + CLK-ZERO = 300ns min
T_CLK_PRE     = 6'd1;                                                                 // 8UI min
T_CLK_POST    = ((60 + (52 * UI)) / BYTECLK_NS) + 1;                                  // 60 ns + 52*UI min
T_CLK_TRAIL   = 60 / BYTECLK_NS + 1;                                                  // 60 ns min
T_CLK_EXIT    = 100 / BYTECLK_NS + 1;                                                 // 100 ns min
T_HS_PREPARE  = ((41 + (4 * UI)) / BYTECLK_NS) + 1;                                   // 40ns +4UI to 85ns +6UI.
T_HS_ZERO     = ((145 + (10 * UI)) / BYTECLK_NS) - ((41 + (4 * UI)) / BYTECLK_NS);    // HS-PREPARE + HS-ZERO = 145ns + 10UI min
T_HS_TRAIL    = ((60 + (4 * UI)) / BYTECLK_NS) + 1;                                   // 60ns +4UI to 105+12UI
T_HS_EXIT     = 100 / BYTECLK_NS + 1;                                                 // 100ns min
```

G_TX_BYTE_FREQ has been calculated from the equation, which is shown in PLL, page 8.

Sample Core-ABC program is provided as follows for 1 Lane RAW8.

Figure 15 • **One Hot Encoding type of inputs for Core-ABC Program**

```
// CoreABC (Soft Processor) Program for Dynamic Clock Configuration

$DL_Select

    JUMP IF INPUT0 $RAW8_1lane
    JUMP IF INPUT1 $RAW10_1lane
    JUMP IF INPUT2 $RAW12_1lane
    JUMP IF INPUT3 $RAW14_1lane
    JUMP IF INPUT4 $RAW24_1lane
    JUMP IF INPUT5 $RAW8_2lane
    JUMP IF INPUT6 $RAW10_2lane
    JUMP IF INPUT7 $RAW12_2lane
    JUMP IF INPUT8 $RAW14_2lane
    JUMP IF INPUT9 $RAW24_2lane
    JUMP IF INPUT10 $RAW8_4lane
    JUMP IF INPUT11 $RAW10_4lane
    JUMP IF INPUT12 $RAW12_4lane
    JUMP IF INPUT13 $RAW14_4lane
    JUMP IF INPUT14 $RAW24_4lane
    JUMP IF INPUT15 $RAW16_1lane
    JUMP IF INPUT16 $RAW16_2lane
    JUMP IF INPUT17 $RAW16_4lane
    JUMP $DL_Select
```

Figure 16 • **Core-ABC Program 1 Lane and RAW8**

```
$RAW8_1lane
//REF DIV VALUE
    IOWRT 0x08400000
    APBWRT DAT 0 0x00008 0x00000400
// FBDIV
    APBWRT DAT 0 0x0002C 0x00000040
// phase Adjsutement of 90 degrees for clock1
    APBWRT DAT 0 0x00020 0x00000040
// post 0_1 clocks
    APBWRT DAT 0 0x00010 0x01000100
// post 2_3 clocks
    APBWRT DAT 0 0x00014 0x04000400
// Enabling Register Configuration Done
    IOWRT 0x10000000
    IOWRT 0x00000000
// ------------------- T_LPX Timing Value -- 6bit [5:0]
    IOWRT 0x105
// ------------------- T_CLK_PREPARE
    IOWRT 0x205
// ------------------- T_CLK_ZERO
    IOWRT 0x316
// ------------------- T_CLK_PRE
    IOWRT 0x401
// ------------------- T_CLK_POST
    IOWRT 0x50A
// ------------------- T_CLK_TRAIL
    IOWRT 0x606
// ------------------- T_CLK_EXIT
    IOWRT 0x709
// ------------------- T_HS_PREPARE
    IOWRT 0x805
// ------------------- T_HS_ZERO
    IOWRT 0x909
// ------------------- T_HS_TRAIL
    IOWRT 0xA06
// ------------------- T_HS_EXIT
    IOWRT 0xB09
    IOWRT 0x000

    JUMP $DL_Change_Done


// DATA-LANE CHANGE DONE
$DL_Change_Done

    NOP
    NOP
    JUMP $DL_Select

    HALT
```

3.5.1.4 PF_DRI

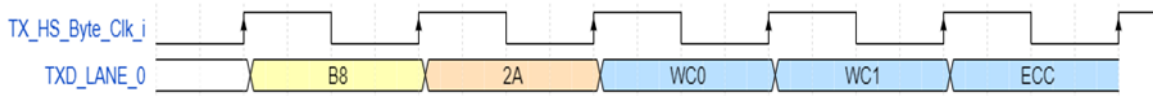
This module is used to convert APB transactions from Core-ABC into Dynamic Reconfiguration Interface (DRI) transactions. These transactions are fed into Dynamic PLL by DRI.

3.6 Timing Diagrams

3.6.1 Short Packet

The following figure shows the timing waveform of the short packet for 1 Lane RAW8 format.

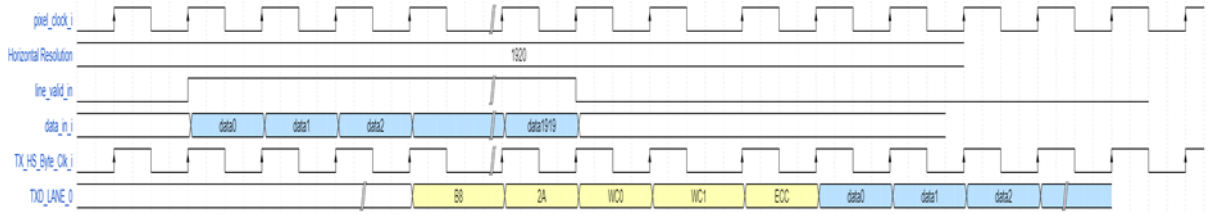
Figure 17 • **Timing diagram for short packet for 1 Lane RAW8 format**



3.6.2 Long Packet

The following figure shows the timing waveform of the long packet for 1 Lane RAW8 format.

Figure 18 • **Timing Waveform of Long Packet for 1 Lane RAW8 format**



4 License

The core is license locked for clear text RTL. The core supports the generation of Encrypted RTL for the Verilog version of the core with no license.

5 Installation Instructions

The core must be installed into Libero software. It is done automatically through the Catalog update function in Libero, or the CPZ file can be manually added using the **Add Core** catalog feature. Once the CPZ file is installed in Libero, the core can be configured, generated, and instantiated within SmartDesign for inclusion in the Libero project.

For further instructions on core installation, licensing, and general use, refer to *Libero SoC Online Help*.

6 Resource Utilization

The following table shows the resource utilization of a sample MIPI CSI-2 Transmitter Core implemented in a PolarFire FPGA (MPF300TS-1FCG1152I package) for 4 Lane RAW8 configuration in compile time mode.

Table 3 • **Resource Utilization of the MIPI CSI-2 Transmitter in compile time mode**

Element	Usage
DFFs	906
4-input LUTs	957
LSRAM	11
Math Blocks	1