Revision History

Revision	Date	Author	Comments
1.0	2019-05-24	zjliu	Initial version
1.1	2019-06-17	zjliu	加入 mipi rx to mipi tx command 屏参考设计，加入 mipi rx 回 ID 参考设计

1. Introduction

This application note describes the methods to use HME H1 FPGA device for IP mipi dsi transfer. The H1 FPGA device has two mipi dsi controllers. The function includes:

- Two mipi dsi controllers both support TX mode
- One mipi dsi controller in TX mode, the other one mipi dsi controller in RX mode
- Two mipi dsi controllers both in RX mode
- Loop back test support (need pin connected from outside)
- One PLL inside, reference clock from OSC
- Target 2560x1440 output with 2DSI
- Target 1920x1080 output with one DSI, and 1920x1080 input with the other one DSI.

This application note also includes interface guidelines. With simple settings, user can easily add the mipi dsi transfer to the design through IP wizard in Fuxi software.

2. HME H1 mipi dsi controller Overview

The HME H1 mipi dsi controller supports the following features:

- Implements all three DSI Layers (Pixel to Byte packing, Low Level Protocol, Lane Management)
- Support for Command and Video Modes
- Host and Peripheral versions
- Scalable data lane support, 1 to 4 Data Lanes
  o Optional bidirectional support on lane 0
- Supports High Speed (1.5 Gbit/s) and Low Power operation
- Support for all DSI data types and formats
- Virtual Channel support
- Supports ULPS mode
- Full Low-Level Protocol Error and Contention detection and reporting
- Supports continuous and non-continuous Clock Lane operation
- Supports multiple packets per transmission
- Support for all three Video Mode packet sequences
  o Non-Burst Mode with Sync Pulses
  o Non-Burst Mode with Sync Events
  o Burst mode
- Support for bus turnaround signaling
- Flexible packet based user interface
  o APB interface option (status and control)
  o Display Pixel Interface Core (DPI-2) option
- Optional multiport interface allows up to 4 interfaces to the DSI

- Supports PHY Protocol Interface (PPI) compatible MIPI D-PHYs
  o Delivered fully integrate and verified with target MIPI D-PHY

3. Architecture

Figure 1 shows the block diagram of mipi dsi transfer.

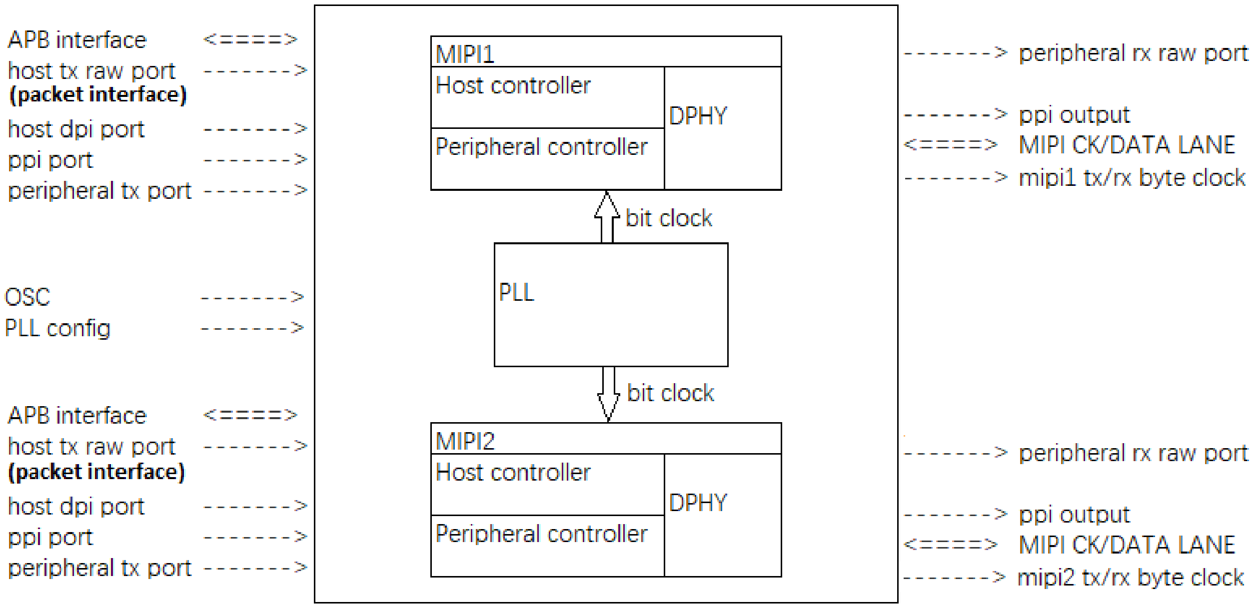


Figure 1 block diagram

Note:mipi1/2 can be used as Host controller or Peripheral controller.

4. Interface

4.1. system Interface

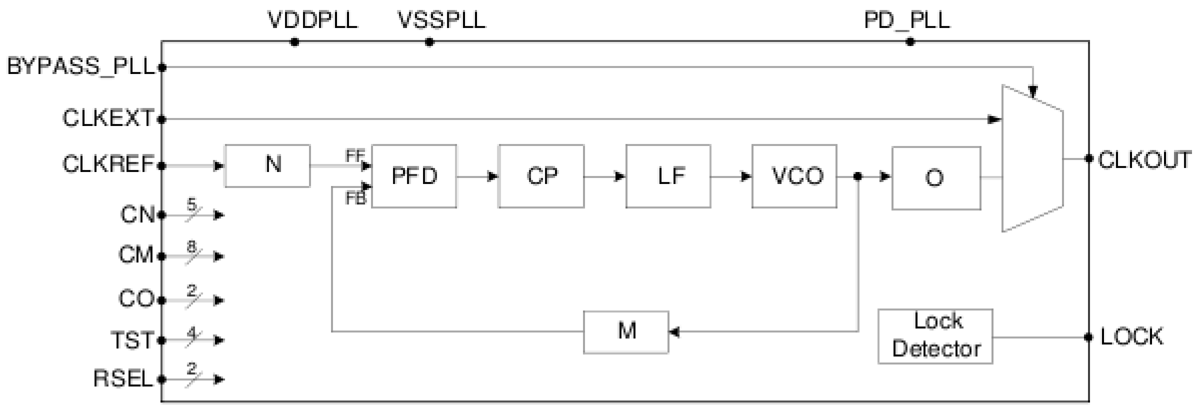
Table 1 system interface

Port_sel[1:0]	input	PPI:2’b00; Host:2’b01; Periph:2’b10; Test:2’b11
TxEscClk	input	escape clock that is provided to the DPHY
RxEscClk	input	escape clock that is provided to the DPHY
reset_n	input	System reset (active low)
reset_esc_n	input	Reset for the esc_clk domain(active low)
enable_dpi_port	input	1’b1 – DPI interface module enable
enable_raw_dsi_port	input	1’b1 – packet interface enable
clk	output	TX Byte clock out

RxByteClkHS	output	RX Byte clock from PHY
clk_periph	input	RX Byte clock in
CLKP	inout	PHY physical interface.
CLKN	inout	PHY physical interface.
DATAN0	inout	PHY physical interface.
DATAP0	inout	PHY physical interface.
DATAN1	inout	PHY physical interface.
DATAP1	inout	PHY physical interface.
DATAN2	inout	PHY physical interface.
DATAP2	inout	PHY physical interface.
DATAN3	inout	PHY physical interface.
DATAP3	inout	PHY physical interface.
CN[4:0]	output	Output to External PLL(MIPI PLL)
CM[7:0]	output	Output to External PLL(MIPI PLL)
CO[1:0]	output	Output to External PLL(MIPI PLL)
ENP_DESER	input	To override the Deserializer token detector and enable Deserializer Byte Clock and DATA. Only applicable in Test mode(default) 1’b0
LOCK	input	MIPI PLL is PLL-locked, Lock Detect output. Asserted when the PLL has achieved frequency lock.
BITCLK	input	MIPI PLL output,used by TX
PD_DPHY	input	MIPI PLL power down signal.Power Down input for D-PHY. When high, all blocks are powered down.
AUTO_PD_EN	input	Powers down inactive lanes reported by CFG_NUM_LANES input bus. 1’b0: inactive lanes are powered up and driving LP11. 1’b1: inactive lanes are powered down.
tx_dphy_rdy	output	1’b1 – the MIPI can work normally
TEST mode		
TEST_ENBL[5:0]	input	
TEST_PATTERN[31:0]	input	
D0_LB_PASS[1:0]	output	
D1_LB_PASS[1:0]	output	
D2_LB_PASS[1:0]	output	
D3_LB_PASS[1:0]	output	
D0_LB_ERR_CNT[9:0]	output	
D1_LB_ERR_CNT[9:0]	output	
D2_LB_ERR_CNT[9:0]	output	

D3_LB_ERR_CNT[9:0]	output	
D0_LB_BYTE_CNT[9:0]	output	
D1_LB_BYTE_CNT[9:0]	output	
D2_LB_BYTE_CNT[9:0]	output	
D3_LB_BYTE_CNT[9:0]	output	
D0_LB_ACTIVE[1:0]	output	
D1_LB_ACTIVE[1:0]	output	
D2_LB_ACTIVE[1:0]	output	
D3_LB_ACTIVE[1:0]	output	
D0_LB_VALID[1:0]	output	
D1_LB_VALID[1:0]	output	
D2_LB_VALID[1:0]	output	
D3_LB_VALID[1:0]	output	
CLK_LB_ACTIVE	output	
DC_TEST_OUT[9:0]	output	

The input to the mipi_pll is the input clock CLKREF signal. The input frequency ranges from 24 MHz till 200 MHz. The input divider has to be programmed such that the frequency after the input divider ranges from 24 MHz till 30 MHz. The VCO (CLKREF*M/N)maxi mum output frequency 1.5GHz. The mipi_pll has output port CLKOUT. It multiplies the input frequency by(M / (N * O)).The definition of CM(M),CN(N),CO(O) is as illustrated in Figure below.



DVR	CM[7:0]	DVR	CM[7:0]	DVR	CM[7:0]	DVR	CM[7:0]
16	111X0000	46	11001110	76	10001100	106	10101010
17	111X0001	47	11001111	77	10001101	107	10101011
18	111X0010	48	11010000	78	10001110	108	10101100
19	111X0011	49	11010001	79	10001111	109	10101101
20	111X0100	50	11010010	80	10010000	110	10101110
21	111X0101	51	11010011	81	10010001	111	10101111
22	111X0110	52	11010100	82	10010010	112	10110000
23	111X0111	53	11010101	83	10010011	113	10110001
24	111X1000	54	11010110	84	10010100	114	10110010
25	111X1001	55	11010111	85	10010101	115	10110011
26	111X1010	56	11011000	86	10010110	116	10110100
27	111X1011	57	11011001	87	10010111	117	10110101
28	111X1100	58	11011010	88	10011000	118	10110110
29	111X1101	59	11011011	89	10011001	119	10110111
30	111X1110	60	11011100	90	10011010	120	10111000
31	111X1111	61	11011101	91	10011011	121	10111001
32	11000000	62	11011110	92	10011100	122	10111010
33	11000001	63	11011111	93	10011101	123	10111011
34	11000010	64	10000000	94	10011110	124	10111100
35	11000011	65	10000001	95	10011111	125	10111101
36	11000100	66	10000010	96	10100000	126	10111110
37	11000101	67	10000011	97	10100001	127	10111111
38	11000110	68	10000100	98	10100010	128	00000000
39	11000111	69	10000101	99	10100011	129	00000001
40	11001000	70	10000110	100	10100100	130	00000010
41	11001001	71	10000111	101	10100101	131	00000011
42	11001010	72	10001000	102	10100110	132	00000100
43	11001011	73	10001001	103	10100111	133	00000101
44	11001100	74	10001010	104	10101000	134	00000110
45	11001101	75	10001011	105	10101001	135	00000111

Figure2   8-bit Feedback Divider Part 1

DVR	CM[7:0]	DVR	CM[7:0]	DVR	CM[7:0]	DVR	CM[7:0]
136	00001000	166	00100110	196	01000100	226	01100010
137	00001001	167	00100111	197	01000101	227	01100011
138	00001010	168	00101000	198	01000110	228	01100100
139	00001011	169	00101001	199	01000111	229	01100101
140	00001100	170	00101010	200	01001000	230	01100110
141	00001101	171	00101011	201	01001001	231	01100111
142	00001110	172	00101100	202	01001010	232	01101000
143	00001111	173	00101101	203	01001011	233	01101001
144	00010000	174	00101110	204	01001100	234	01101010
145	00010001	175	00101111	205	01001101	235	01101011
146	00010010	176	00110000	206	01001110	236	01101100
147	00010011	177	00110001	207	01001111	237	01101101
148	00010100	178	00110010	208	01010000	238	01101110
149	00010101	179	00110011	209	01010001	239	01101111
150	00010110	180	00110100	210	01010010	240	01110000
151	00010111	181	00110101	211	01010011	241	01110001
152	00011000	182	00110110	212	01010100	242	01110010
153	00011001	183	00110111	213	01010101	243	01110011
154	00011010	184	00111000	214	01010110	244	01110100
155	00011011	185	00111001	215	01010111	245	01110101
156	00011100	186	00111010	216	01011000	246	01110110
157	00011101	187	00111011	217	01011001	247	01110111
158	00011110	188	00111100	218	01011010	248	01111000
159	00011111	189	00111101	219	01011011	249	01111001
160	00100000	190	00111110	220	01011100	250	01111010
161	00100001	191	00111111	221	01011101	251	01111011
162	00100010	192	01000000	222	01011110	252	01111100
163	00100011	193	01000001	223	01011111	253	01111101
164	00100100	194	01000010	224	01100000	254	01111110
165	00100101	195	01000011	225	01100001	255	01111111

Figure3   8-bit Feedback Divider Part 2

N	CN[4:0]
1	11111
2	00000
3	10000
4	11000
5	11100
6	01110
7	00111
8	10011
9	01001
10	00100
11	00010
12	10001
13	01000
14	10100
15	01010
16	10101
17	11010
18	11101
19	11110
20	01111
21	10111
22	11011
23	01101
24	10110
25	01011
26	00101
27	10010
28	11001
29	01100
30	00110
31	00011
32	00001

Figure4 5-bit Input Divider

N	CO1	CO0
1	0	0
2	0	1
4	1	0
8	1	1

Figure5 2-bit Output Divider

The power down signal PD_PLL is active high, and is used to power down the PLL. Asserting the PD_PLL signal anytime will reset the PLL to its initial unlocked state. To ensure proper PLL functionality, CLKREF needs to be stable before PLL power up. The expected power on sequence for the PLL is as illustrated in Figure below. The LOCK signal rising edge could be used to determine the lock state of the PLL.

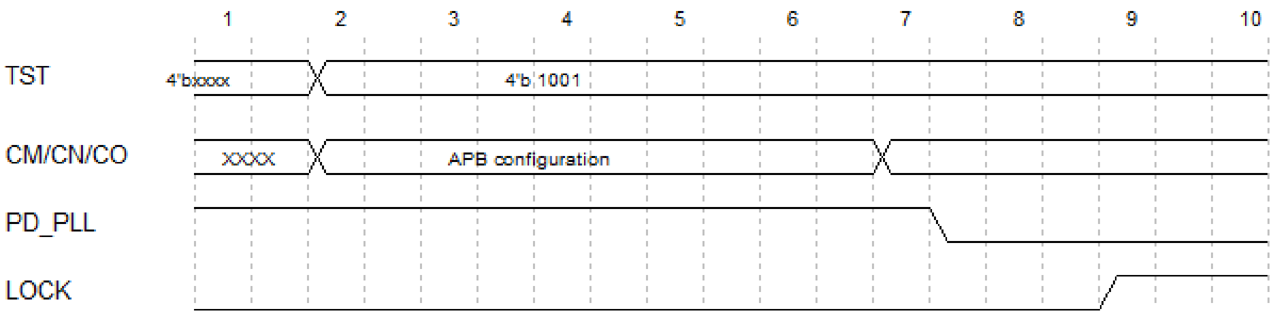


Figure6 Power on sequence

4.2. PHY Protocol Interface (PPI)

4.2.1. clock interface

Table 2 PPI clock interface

ext_TxRequestHS_lnclk	input	request HSTX mode, and submit data
Stopstate_lnclk	output	clock lane in stop state

ext_TxUlpsClk_lnclk	input	clock lane request enter ULPS mode
ext_TxUlpsExit_lnclk	input	clock lane request exit ULPS mode
UlpsActiveNot_lnclk	output	clock lane ULPS status
RxUlpsClkNot	output	
RxClkActiveHS	output	
ext_Enable_lnclk	input	

4.2.2. PPI in tx mode

This interface can transmit data, send Escape sequences, receive and transmit triggers, and detect and report D-PHY error conditions.

Table 3 PPI tx mode interface

MIPI D-PHY High Speed Interface		
ext_TxDataHS_lnx[7:0]	input	High-Speed Transmit Data for lane x. 8 bit High-Speed transmit data to the DPHY. Valid when TxReadyHS_lnx is asserted high.
ext_TxRequestHS_lnx	input	High-Speed Transmit Request for lane x. A low-to-high transition on TxRequestHS_lnx indicates to the D-PHY that that it is to initiate a Start-of-Transmission sequence. A High to-Low transition causes the D-PHY to initiate an End-of Transmission sequence. When High, the DSI Transmitter Core will drive valid data out onTxDataHS_lnx[7:0], advancing the transmit data to the next value when TxReadyHs_lnx is high.
TxReadyHS_lnx	output	High-Speed Transmit Ready for lane x. Active High indicates that the data currently on the TxDataHS_lnx[7:0] port has been accepted by the D-PHY
ext_Enable_lnx	input	Enable Lane Module D-PHY lane x.
MIPI D-PHY Escape Mode Transmit Interface		
ext_TxRequestEsc_lnx	input	Escape mode Transmit Request for lane x. Valid for Lane 0 only. Inactive and low for all other data lanes. Asserted High together with TxTriggerEsc_lnx[3:0] to request that the D-PHY enter into escape mode. The D-PHY exits escape mode.when TxRequestEsc_lnx is deasserted.
ext_TxUlpsEsc_lnx	input	Escape Mode Transmit Ultra-Low Power State lane x D-PHY. Asserted HIGH with TxRequestEsc to put the D-PHY into ULP.
ext_TxUlpsExit_lnx	input	Transmit ULP Exit Sequence lane x D-PHY. Asserted High by the controller to take the D-PHY out of

		ULP.
ext_TxTriggerEsc_lnx [3:0]	input	Escape mode Transmit Trigger 0-3 lane x D-PHY. Valid for Lane 0 only. Inactive and low for all other data lanes. This one hot encoded output to the D-PHY selects, when TxRequestEsc_lnx is asserted, which causes escape triggers to be sent across the link.
ext_TxLpdtEsc_lnx	input	Escape mode Transmit Low-Power Data lane x D-PHY. Valid for Lane 0 only. Inactive and low for all other data lanes. When asserted High together with TxRequestEsc_lnx puts the D-PHY into low power data transmit mode.
ext_TxDataEsc_lnx[7:0]	input	Escape mode Transmit Data lane x D-PHY. Valid for Lane 0 only. Inactive and low for all other data lanes. Eight bit escape mode data that is to be transmitted in low power mode.
ext_TxValidEsc_lnx	input	Escape mode Transmit Data Valid lane x D-PHY. Valid for Lane 0 only. Inactive and low for all other data lanes. Asserted High along with TxDataEsc_lnx[7:0] to indicate to the DPHY that the data on TxDataEsc_lnx[7:0] is valid.
ext_TxReadyEsc_lnx	output	Escape mode Transmit Ready lane x D-PHY. Valid for Lane 0 only. Inactive and low for all other data lanes. When asserted High by the D-PHY, TxReadyEsc_lnx indicates that the D-PHY has accepted the transmit data on TxDataEsc_lnx[7:0]
MIPI D-PHY Escape Mode Receive Interface		
RxClkEsc_ln0	output	Escape mode Receive Clock. This signal is used to transfer received data to the protocol during escape mode.
RxLpdtEsc_ln0	output	Tells the Controller that Lane 0 is in Escape mode Receive Low Power state.
RxUlpsEsc_ln0	output	
RxTriggerEsc_ln0[3:0]	output	Escape mode Receive Trigger 0-3 lane 0 D-PHY. This one hot encoded input reports that a trigger has been received trigger from across the link.
RxDataEsc_ln0[7:0]	output	Escape mode Receive Data lane 0 D-PHY. Presents the eight bit data that is received in Low Power mode.
RxValidEsc_ln0	output	Escape mode Receive Data Valid lane 0 D-PHY. Asserted High along with RxDataEsc_ln0[7:0] to indicate that the data on RxDataEsc_ln0[7:0] is valid.

MIPI D-PHY Control and Status Interface		
Stopstate_lnx	output	Lane is in Stop state D-PHY lane x.
Direction_ln0	output	Transmit/Receive Direction. Only valid for lane 0. 1'b0 = Transmitter. 1'b1 = Receiver.
UlpsActiveNot_ln0	output	
ext_TurnRequest_ln0	input	Bus Turn Around request.
ext_TurnDisable_ln0	input	Disable Turn-around. Only valid for lane 0.
ext_ForceRxmode_ln0	input	Force Lane Module Into Receive mode / Wait for Stop state. Only valid for lane 0.
ext_ForceTxStopmode_ln0	input	
MIPI D-PHY Error Interface		
ErrSotHS_ln0	output	
ErrSotSyncHS_ln0	output	
ErrEsc_ln0	output	Escape Entry Error D-PHY lane 0.
ErrSyncEsc_ln0	output	Low-Power Data Transmission Synchronization Error D-PHY lane 0.
ErrControl_lnx	output	Control Error D-PHY lane x.
ErrContentionLP0_lnx	output	LP0 Contention Error D-PHY lane x.
ErrContentionLP1_lnx	output	LP1 Contention Error D-PHY lane x.

4.2.3. PPI in rx mode

This interface can transmit data, send Escape sequences, receive and transmit triggers, and detect and report D-PHY error conditions.

Table 4 PPI rx mode interface

MIPI D-PHY High Speed Interface		
RxByteClkHS	output	High-Speed Receive Byte clock for lane x from the D-PHY
RxDataHS_lnx[7:0]	output	High-Speed Receive Data for lane x. 8-bit High-Speed receive data from the D-PHY.Input is synchronous to the RxByteClkHS_lnx input.
RxValidHS_lnx	output	High-Speed Receive data valid for lane x. Active High indicates that the data currently on the RxDataHS_lnx[7:0] port has is valid and can be registered on the next rising edge of RxByteClkHS_lnx.
RxActiveHS_lnx	output	High Speed Receive is active on lane x.
RxSyncHS_lnx	output	Sync has been detected on High Speed lane x.
clk_periph	input	clk_periph must be 1/8th the frequency that the DPHY

		data lanes operate at and is usually provided by the TX DPHY. This clock can be independent and unrelated to clk_esc.
RxSyncHS	output	RxSyncHS_lnx(RxByteClkHS clock domain) sync to clk_periph clock domain
RxActiveHS	output	RxActiveHS_ln0 (RxByteClkHS clock domain) sync to clk_periph clock domain
RxLpdtEsc	output	RxLpdtEsc_ln0 (RxByteClkHS clock domain) sync to clk_periph clock domain

4.3. DSI Host Controller interface

Figure 7 illustrates the DSI Host Controller Core structure. The DSI Host Controller Core operates on the host(transmit) side of a DSI link.

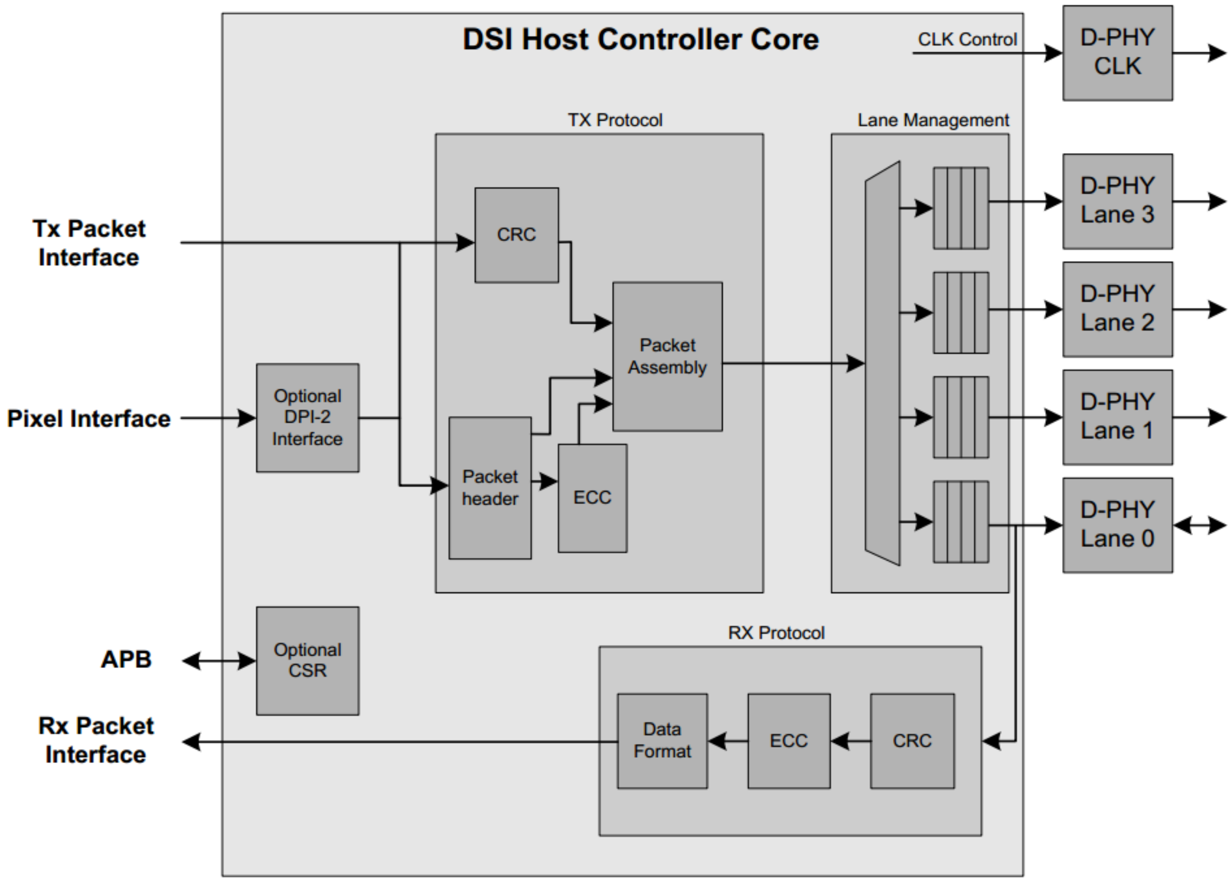


Figure 7 DSI Host Controller Core Block Diagram

The DSI Host Controller Core implements all three layers defined by the DSI Specification: Pixel to Byte Packing in the Application layer, Low Level Protocol, and Lane Management. The DSI Host Controller Core sends and receives DSI commands via the Packet Interface. The Packet Interface can be connected to a DPI-2 translator, or directly to the User’s logic.

The D-PHY interface of the DSI Host Controller Core supports up to four PHY Protocol Interface (PPI) compatible MIPI D-PHYs.

The Packet Interface is an easy-to-use data interface that accepts commands and data, and sends it over the DSI link. It supports 1 to 4 virtual channels, and the use of 1-4 D-PHY lanes. The DSI Host Controller Core takes care of all packet formatting details and transmission over the MIPI bus.

The DPI-2 Translator connects to the DSI Host Controller Core via the Packet Interface. DPI-2 masters may connect directly to the DPI-2 Translator to send commands across the DSI link.

4.3.1. APB interface

The DSI Host controller can be configured with an optional Control and Status Register (CSR) interface. The CSR provides an APB compatible interface that enables control of the controller's configuration inputs via registers accessible via the APB interface.

The port descriptions for the CSR APB interface is described below.

Table 5　DSI Host controller APB interface

pclk	Input	csr_clk is the APB pclk. All signals, except pclk_reset_n, are synchronous to pclk.
pclk_reset_n	Input	Async reset input for all logic in the pclk clock domain.
paddr[17:0]	Input	APB address. All registers are addressed on a 32 bit boundary so paddr[1:0] should always be set to 2'b00.
pwrite	Input	APB write signal, active high for write, low for read. Assert during setup phase for writes.
psel	Input	APB select signal, active high. The CSR responds with psel is asserted,and paddr contains the address of a valid register.
pwdata[31:0]	Input	APB write data.
prdata[31:0]	Output	APB read data.
pready	Output	APB pready output. Always asserted for writes, asserted during access phase for reads.
penable	Input	APB penable. Assert during access phase, this can be asserted for multiple clocks　(even though APB spec specifies only one clock).

The memory map of the TX Controller CSR APB interface is described below.

Table 6　DSI Host controller APB memory map

0x00000	R/W	[1:0]	CFG_NUM_LANES: cfg_num_lanes[1:0] setting for the Host Controller. Sets the number of active lanes that are to be used for transmitting data. 2'b00 – 1 Lane 2'b01 – 2 Lanes 2'b10 – 3 Lanes 2'b11 – 4 Lanes
0x00004	R/W	[0]	CFG_NONCONTINUOUS_CLK: cfg_noncontinuous_clk[0] setting for the Host

			Controller. Sets the TX Controller into non-continuous MIPI CLK mode. When in non-continuous clock mode, the High Speed Clk will transition into low power mode in between transmissions. 1’b0 – Continuous high speed clock 1’b1 – Non-Continuous high speed clock
0x00008	R/W	[6:0]	CFG_T_PRE: cfg_t_pre[6:0] setting for the Host Controller. Sets the number of byte clock periods (‘clk_byte’ input) that the controller will wait after enabling the clock lane for HS operation before enabling the data lanes for HS operation
0x0000c	R/W	[6:0]	CFG_T_POST: cfg_t_post[6:0] settting for the Host Controller. Sets the number of byte clock periods (‘clk_byte’ input) to wait before putting the clock lane into LP mode after the data lanes have been put into LP mode.
0x00010	R/W	[6:0]	CFG_TX_GAP: cfg_tx_gap[6:0] setting for the Host Controller. Sets the number of byte clock periods (‘clk_byte’ input) that the controller will wait after the clock lane has been put into LP mode before enabling the clock lane for HS mode again
0x00014	R/W	[0]	CFG_AUTOINSERT_EOTP: cfg_autoinsert_eotp for the Host Controller. Enables the Host Controller to automatically insert an EoTp short packet when switching from HS to LP mode. 1’b0 – eotp is not automatically inserted 1’b1 – eotp is automatically inserted
0x00018	R/W	[7:0]	CFG_EXTRA_CMDS_AFTER_EOTP: cfg_extra_comds_after_eotp setting for the Host Controller. Configures the DSI Host Controller to send extra End Of Transmission Packets after the end of a packet. The value of cfg_extra_cmd_after_eotp is the number of extra EOTP packets sent.
0x0001c	R/W	[23:0]	CFG_HTX_TO_COUNT: cfg_htx_to_count setting for the Host Controller. Host HS TX Timeout count, HS TX Timeout. Sets the value of the DSI host High Speed TX timeout count in clk_byte clock periods that once reached will

			initiate a timeout error and follow the recovery procedure documented in the DSI specification.
0x00020	R/W	[23:0]	CFG_LRX_H_TO_COUNT: cfg_lrx_h_to_count setting for the Host Controller. Host Low Power RX Timeout, LP_RX-H Timeout. Sets the value of the DSI Low Power RX timeout count in clk_byte clock periods that once reached will initiate a timeout error and follow the recovery procedure documented in the DSI specification.
0x00024	R/W	[23:0]	CFG_BTA_H_TO_COUNT: cfg_bta_h_to_count setting for the Host Controller. Host Bust Turn Around (BTA) Timout. Sets the value of the DSI Host Bus Turn Around timeout in clk_byte clock periods that once reached will initiate a timeout error.
0x00028	R/W	[18:0]	CFG_TWAKEUP: cfg_twakeup setting for the Host Controller. DPHY Twakeup timing parameter. Sets the number of clk_esc clock periods to keep a clock or data lane in Mark-1 state after exiting ULPS.The MIPI DPHY spec requires a minimum of 1ms in Mark-1 state after leaving ULPS.
0x0002c	R	[31:0]	CFG_STATUS_OUT: cfg_status_out status for the Host Controller.
0x00030			DSI_HOST_BASE_CFG_RX_ERROR_STATUS
0x00200	R/W	[15:0]	CFG_DPI_PIXEL_PAYLOAD_SIZE: cfg_dpi_pixel_payload_size setting for the Host Controller DPI-2 Interface Core if used. Maximum number of pixels that should send as one DSI packet. Recommended to be evenly divisible by the line size (in pixels)
0x00204	R/W	[15:0]	CFG_DPI_PIXEL_FIFO_SEND_LEVEL: cfg_dbi_pixel_fifo_send_level setting for the Host Controller DPI-2 Interface Core if used. In order to optimize DSI utility, the DPI bridge buffers a certain number of DPI pixels before initiating a DSI packet. This configuration port controls the level at which the DPI Host bridge begins sending pixels.
0x00208	R/W	[2:0]	CFG_DPI_INTERFACE_COLOR_CODING: cfg_dpi_interface_color_coding setting for the Host Controller DPI-2 Interface Core if used. Sets the distribution of RGB bits within the 24-bit d bus, as specified by the DPI specification. 0= RGB 16-bit Configuration 1 1= RGB 16-bit Configuration 2 2= RGB 16-bit Configuration 3

			3= RGB 18-bit Configuration 1 4= RGB 18-bit Configuration 2 5= RGB 24-bit
0x0020c	R/W	[1:0]	CFG_DPI_PIXEL_FORMAT: cfg_dpi_pixel_format setting for the Host Controller DPI-2 Interface Core if used. Sets the DSI packet type of the pixels. 0= RGB 16-bit 1= RGB 18-bit 2= RGB 18-bit loosely packed, 3= RGB 24-bit
0x00210	R/W	[0]	CFG_DPI_VSYNC_POLARITY: cfg_dpi_vsync_polarity setting for the Host Controller DPI-2 Interface Core if used. Sets polarity of dpi_vsync input, 0 – active low, 1 active high
0x00214	R/W	[0]	CFG_DPI_HSYNC_POLARITY: cfg_dpi_hsync_polarity setting for the Host Controller DPI-2 Interface Core if used. Sets Polarity of dpi_hsync input, 0 – active low, 1 – active high
0x00218	R/W	[1:0]	CFG_DPI_VIDEO_MODE: cfg_dpi_video_mode setting for the Host Controller DPI-2 Interface Core if used. Select DSI video mode that the host DPI module should generate packets for. 2'b00 – Non-Burst mode with Sync Pulses 2'b01 – Non-Burst mode with Sync Events 2'b10 – Burst mode 2'b11 – Reserved, not valid
0x0021c	R/W	[15:0]	CFG_DPI_HFP: cfg_dpi_hfp setting for the Host Controller DPI-2 Interface Core if used. Sets the DSI packet payload size, in bytes, of the horizontal front porch blanking packet.
0x00220	R/W	[15:0]	CFG_DPI_HBP: cfg_dpi_hbp setting for the Host Controller DPI-2 Interface Core if used. Sets the DSI packet payload size, in bytes, of the horizontal back porch blanking packet.
0x00224	R/W	[15:0]	CFG_DPI_HSA: cfg_dpi_hsa setting for the Host Controller DPI-2 Interface Core if used. Sets the DSI packet payload size, in bytes, of the horizontal sync width filler blanking packet.

0x00228	R/W	[0]	CFG_DPI_ENABLE_MULT_PKTS: cfg_dpi_enable_mult_pkts setting for the Host Controller DPI-2 Interface Core if used. Enable Multiple packets per video line. When enabled, cfg_dpi_pixel_payload_size must be set to exactly half the size of the video line. 0 – Video Line is sent in a single packet 1 – Video Line is sent in two packets
0x0022c	R/W	[7:0]	CFG_DPI_VBP: cfg_dpi_vbp setting for the Host Controller DPI-2 Interface Core if used. Sets the number of lines in the vertical back porch
0x00230	R/W	[7:0]	CFG_DPI_VFP: cfg_dpi_vfp setting for the Host Controller DPI-2 Interface Core if used. Sets the number of lines in the vertical front porch
0x00234	R/W	[0]	CFG_DPI_BLLP_MODE: cfg_dpi_bllp_mode setting for the Host Controller DPI-2 Interface Core if used. Optimize bllp periods to Low Power mode when possible 0 – blanking packets are sent during BLLP periods 1 – LP mode is used for BLLP periods
0x00238	R/W	[0]	CFG_DPI_USE_NULL_PKT_BLLP: cfg_dpi_use_null_pkt_bllp setting for the Host Controller DPI-2 Interface Core if used. Selects type of blanking packet to be sent during bllp region 0 - Blanking packet used in bllp region 1 - Null packet used in bllp region
0x0023c	R/W	[13:0]	CFG_DPI_VACTIVE: cfg_dpi_vactive setting for the Host Controller DPI-2 Interface Core if used. Sets the number of lines in the vertical active aread.
0x00240	R/W	[1:0]	CFG_DPI_VC: cfg_dpi_vc setting for the Host Controller DPI-2 Interface Core if used. Sets the Virtual Channel (VC) of packets that will be sent to the receive packet interface. Packets with VC not equal to this value are discarded and the "DSI VC ID Invalid" bit (bit 12) in the DSI error report is set
0x00300		[1:0]	DSI_HOST_HME_DPHY_INTFC_DPHY_M_PRG_HS_PREPARE: hstx_state_machine,TxClkEsc domain,,enter HS mode from LP11->LP01->(pre_timer,escclk)LP00->(zero timer,byteclk)HS0->send sync->start clock data->(trail timer,byteclk)->LP11

0x00304		[0]	DSI_HOST_HME_DPHY_INTFC_DPHY_MC_PRG_HS_PREPARE: hstx_state_machine,TxClkEsc domain,,enter HS mode from LP01->LP11->(pre_timer,escclk)LP00->(zero timer,byteclk)->send sync->start clock data->(trail timer,byteclk)->LP11. 为 1 时，LP00 会多 0.5 个 ESCCLK。
0x00308		[4:0]	DSI_HOST_HME_DPHY_INTFC_DPHY_M_PRG_HS_ZERO: hstx_state_machine,TxByteClkHS domain ,enter HS mode from LP01->LP11->(pre_timer,escclk)LP00->(zero timer,byteclk)->send sync->start clock data->(trail timer,byteclk)->LP11
0x0030c		[5:0]	DSI_HOST_HME_DPHY_INTFC_DPHY_MC_PRG_HS_ZERO: hstx_state_machine,TxByteClkHS domain ,enter HS mode from LP01->LP11->(pre_timer,escclk)LP00->(zero timer,byteclk)->send sync->start clock data->(trail timer,byteclk)->LP11。期间 clk 为 0
0x00310		[3:0]	DSI_HOST_HME_DPHY_INTFC_DPHY_M_PRG_HS_TRAIL: hstx_state_machine,TxByteClkHS domain ,enter HS mode from LP01->LP11->(pre_timer,escclk)LP00->(zero timer,byteclk)->send sync->start clock data->(trail timer,byteclk)->LP11
0x00314		[3:0]	DSI_HOST_HME_DPHY_INTFC_DPHY_MC_PRG_HS_TRAIL: hstx_state_machine,TxByteClkHS domain ,enter HS mode from LP01->LP11->(pre_timer,escclk)LP00->(zero timer,byteclk)->send sync->start clock data->(trail timer,byteclk)->LP11。期间 clk 为 0
0x00318		[4:0]	DSI_HOST_HME_DPHY_INTFC_PLL_CN
0x0031c		[7:0]	DSI_HOST_HME_DPHY_INTFC_PLL_CM
0x00320		[1:0]	DSI_HOST_HME_DPHY_INTFC_PLL_CO

Timing for apb interface is list below:

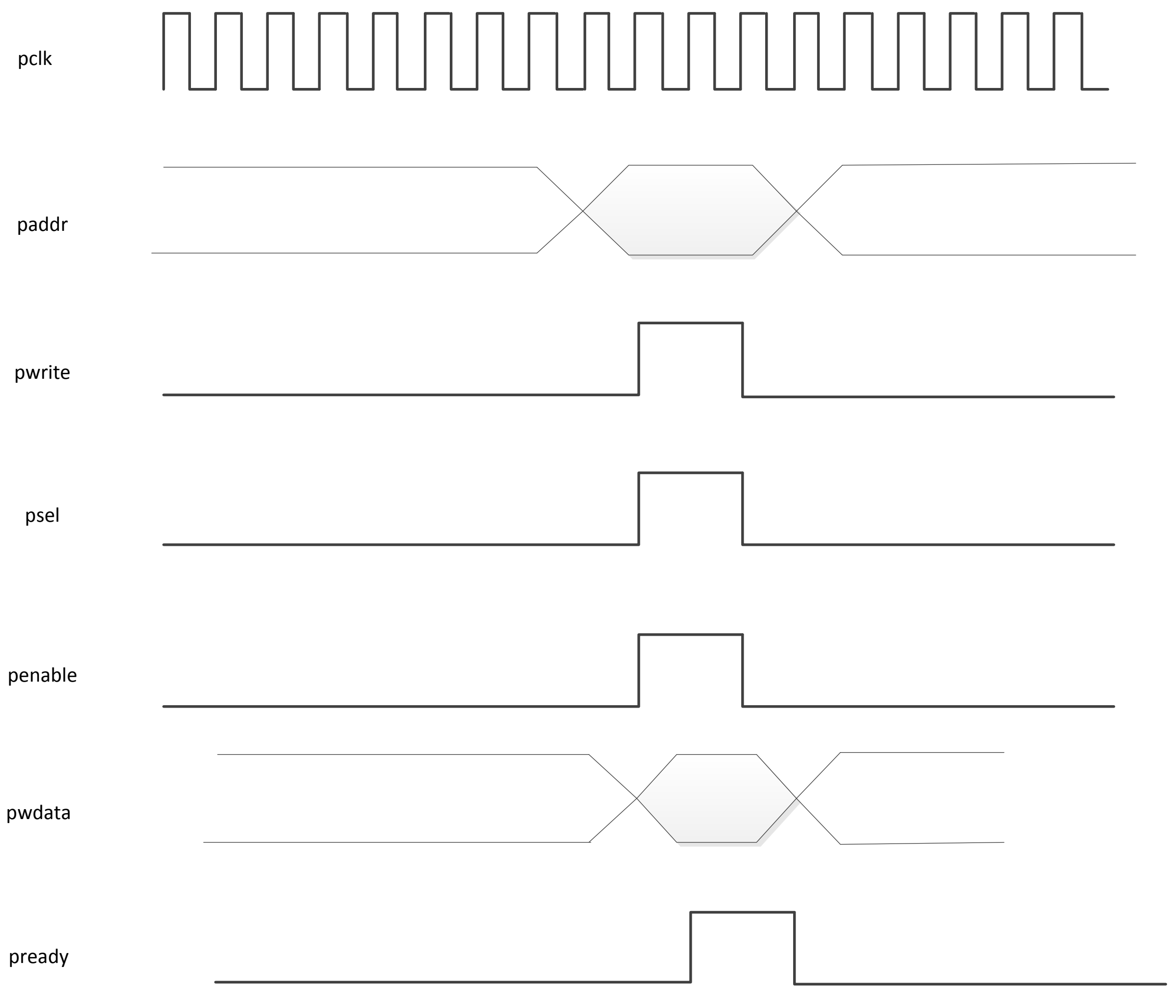


Figure 8   Timing for apb interface

4.3.2. DPI interface

The DSI Host Controller DPI-2 Interface Core provides the following features:

- Support for Type 2,3, and 4 displays
- Support for RGB 16-, 18- and 24- bit Pixel data and all alignment configurations
- Support for RGB 16, 18, 24, and 18 bit loosely packed DSI data types
- Supports DSI video modes
  - o Non-Burst Mode with Sync Pulses
  - o Non-Burst Mode with Sync Events
  - o Burst Mode
- Supports normal or inverted HSYNC and VSYNC signals
- Handles clock domain crossing from DPI-2 Pixel clock to the Host controller TX Byte clock
- Interfaces directly to the Host Controller’s DSI Packet Interface
- Comes already integrated with the DSI Host Controller

The ports on the DSI Host DPI-2 Interface core are described below.

Table 7   DSI Host controller DPI interface

dpi_pclk	Input	Pixel Clock – all other inputs are synchronous to dpi_pclk

reset_dpi_n	Input	Async reset, active low, for the dpi_pclk clock domain.
Host_dpi_vsync	Input	Vertical Sync timing signal
Host_dpi_hsync	Input	Horizontal Sync timing Signal
Host_dpi_de	Input	Data Enable signal, assertion indicates valid pixels
Host_dpi_d[23:0]	Input	Pixel data in RGB 16-, 18-, or 24-bit format
Host_dpi_sd	Input	Shut Down – Control to shutdown display (type 4 only) 1'b1= Send shutdown command. 1'b0= No effect
Host_dpi_cm	Input	Color Mode control. 1'b0== Normal Mode 1'b1== Low-color Mode
dpi_host_underrun_err	Output	During DSI Host transmission of DPI data insufficient DPI data was received. This may indicate that DPI_CLK is too slow, or that the cfg_dpi_* parameters are incorrectly set.

The dpi_pclk clock is used on the optional Host DPI-2 interface. All of the Host DPI-2 signals are synchronous to this clock. The DSI Host Controller's DPI-2 Bridge module handles transferring video data received in the dpi_pclk clock domain over to the clk_byte clock domain.

The dpi_pclk and clk_byte frequencies are related by the following formula:

clk_byte_freq >= dpi_pclk_freq * DPI_pixel_size / (8 * (cfg_num_lanes + 1))

cfg_num_lanes = the configuration port setting that selects the number of active MIPI DPHY data lanes
clk_byte_freq = frequency of clk_byte which is 1/8th the High Speed data lane rate.
dpi_pclk_freq = frequency of the dpi_pclk clock on the DPI-2 interface.
DPI_pixel_size = size of pixels, in bits, on the DPI-2 interface

Timing for host dpi interface is list below:

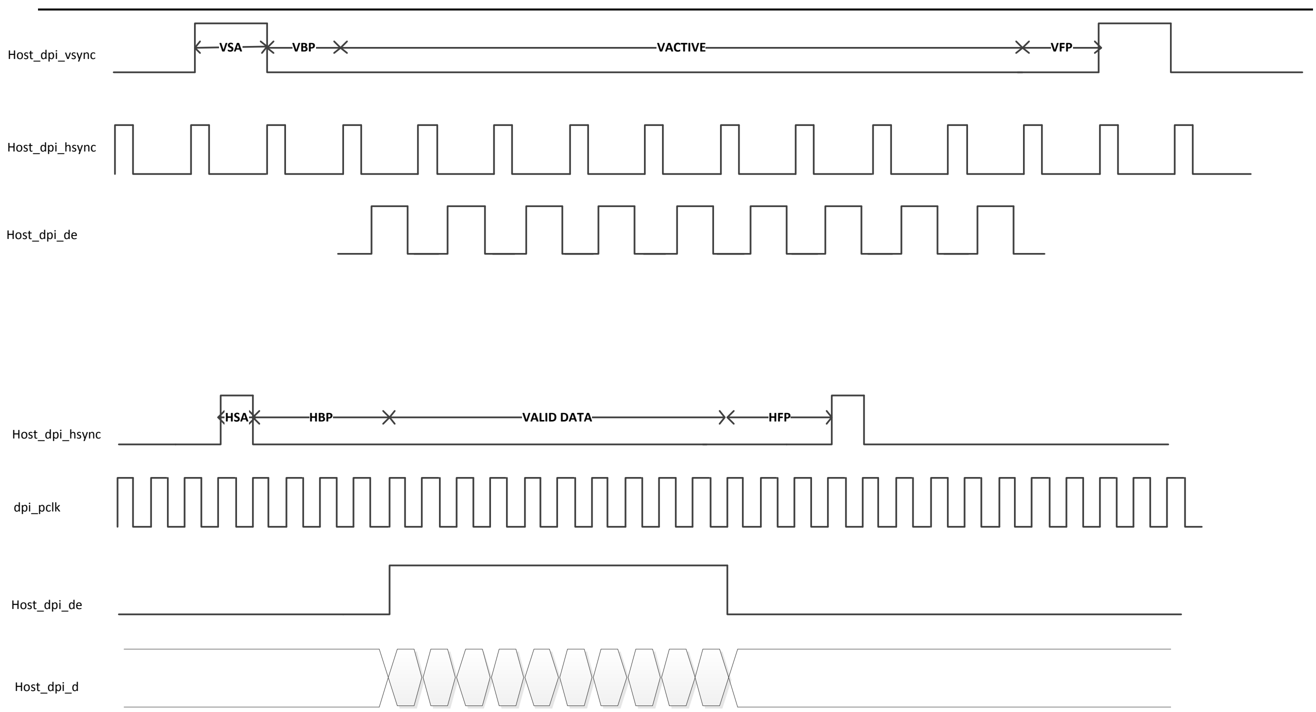


Figure 9 Timing for host dpi interface

Note: CFG_DPI_VSYNC_POLARITY:set 1, CFG_DPI_HSYNC_POLARITY:set 1

4.3.3. packet interface

The DSI Host Controller Core Packet Interface consists of Transmit, Receive, and Control & Status sections. Through these interfaces, the user application can take complete control of the DSI interface, sending all video timing, sending DSI commands, receiving DSI reads, monitoring the status of the interface and responding to error reporting.

The Transmit Packet Interface is the mechanism with which the user creates packets to send over the MIPI Interface.

For Long Packets, the user provides the Virtual Channel (VC) number, Data Type (DT), and Word Count (WC) via the tx_cmd ports to the controller. The Controller then creates a packet header and pulls the packet data from the Packet Interface and out to the D-PHY to transmit.

For Short Packets, the user provides the Virtual Channel (VC) number, Data Type (DT), and required parameters (if any) via the tx_cmd ports to the controller. The Controller then creates the short packet and sends it to the DPHY to transmit. This interface enables the user application to transmit and receive any type of DSI packet.

The Packet Interface Transmit Interface ports are listed below.

Table 8 DSI Host controller packet interface Transmit Interface

clk	output	Byte clock . The D-PHY PPI interface, the tx_cmd and tx_payload interfaces are synchronous to clk_byte. This clock can be independent and unrelated to clk_esc.
TxEscClk	Input	The clk_tx_esc clock is used by the MIPI Tx DPHY for state control and low power data transmission. The DSI Host Controller also uses clk_tx_esc for the portion of controller

		logic that interfaces to the MIPI Tx DPHY that are synchronous to clk_tx_esc. The frequency of clk_tx_esc is defined by the requirements of the MIPI interface and the MIPI Tx DPHY.
reset_n	Input	Asynchronous reset, active low. This reset applies to all logic in the clk_byte clock domain.
reset_esc_n	Input	Asynchronous reset, active low. This reset applies to all logic in the clk_esc clock domain.
Host_tx_payload[31:0]	Input	Packet data input.
Host_tx_payload_en	Output	Packet data read enable. This active high signal indicates that the controller requires a valid packet during the next clk_byte period.
Host_tx_payload_en_last	Output	Last packet read enable, active high signals last cycle of tx_payload_en.
Host_tx_cmd_data_type[5:0]	Input	Transmit packet DSI data type. It is written into the command buffer when tx_cmd_ack is asserted high.
Host_tx_cmd_vc[1:0]	Input	Transmit packet command virtual channel. It is written into the command buffer when tx_cmd_ack is asserted high.
Host_tx_cmd_byte_count[15:0]	Input	Transmit packet payload byte count. It is written into the command buffer when tx_cmd_ack is asserted high. For DSI Long packet types, tx_cmd_byte_count defines the number of bytes of packet data to pull from the tx_payload port. For DSI Short packets, the format of tx_cmd_byte_count contains any optional parameters. If the SDI Short packet type does not have any parameters, it is recommended to set tx_cmd_byte_count to all 0s.
Host_tx_cmd_req	Input	Transmit packet command request. This active high signal informs the controller that the packet command is valid. The packet command consists of the ports tx_cmd_data_type, tx_cmd_vc, and tx_cmd_byte_count. The controller will assert tx_cmd_ack when it accepts the command, after which, the user should either update port values for the next transmit packet command or deassert tx_cmd_req.
Host_tx_cmd_ack	Output	Transmit packet command request acknowledge. This active high signal indicates that the controller has accepted the TX packet request and the user logic should either submit a new request or deassert tx_cmd_req on the next rising edge of clk_byte.
Host_trigger_req	Input	Transmit trigger request. This active high signal informs the controller that the trigger number on trigger_send is valid. The controller will assert trigger_ack when it accepts the command, after which, the user should either put update trigger_ack with the values for the next transmit packet or deassert trigger_req.
Host_trigger_ack	Output	Transmit trigger request acknowledge. This active high signal

		indicates that the controller has accepted the trigger request and the user logic should either submit a new request or deassert trigger_req on the next rising edge of user_clk.
Host_trigger_send[1:0]	Input	Transmit trigger. The trigger number on trigger_send is sampled when trigger_ack is asserted high. The format of trigger_send is as follows: 1'b00 = Trigger 0 (Reset-Trigger) 1'b01 = Trigger 1 ([Reserved]) 1'b10 = Trigger 2 ([Reserved]) 1'b11 = Trigger 3 ([Reserved])
Host_tx_hs_mode	Input	Switches the DPHY into High Speed Data Transfer mode or Low Power Data Transfer mode. 1'b1 = request HS mode 1'b0 = request LP mode. The Packet interface will not acknowledge packet commands or data while switching modes.
Host_dphy_turnaround	Input	Requests bus turnaround. 1'b1 = Request reverse direction LP mode, from Host TX to Host RX. 1'b0 = No effect. This signal is ignored if the bus is already in Reverse (Host RX) direction.
Host_dphy_direction	Output	Reports the current bus direction. 1'b1 = Bus is in Reverse direction (Host RX). 1'b0 = Bus is in Forward direction (Host TX).
Host_tx_active	Output	tx_active asserts high when the Host Controller is actively transmitting data or when it has accepted a request from the user but has not yet started transmitting.
Host_hs_tx_timeout	Output	Asserts high for one clk_byte period to indicate that a High Speed transmit has timed out.
Host_lp_rx_timeout	Output	Asserts high for one clk_byte period to indicate that a Low Power RX timeout has occurred.
Host_bta_timeout	Output	
host_tx_ulps_enable[4:0]	Input	
host_tx_ulps_active[4:0]	Output	

The Receive Packet Interface returns data from the Peripheral to the user. The user is provided the Virtual Channel (VC) number, Data Type (DT), and Word Count (WC) via the l_rx_pkt_* signals. The user is presented with any returned data on the rx_payload* signals.This interface enables the user application to receive any type of DSI packet.

The Packet Interface Receive Interface ports are listed below.

Table 9 DSI Host controller packet interface Receive Interface

clk	output	Byte clock . This clock can be independent and unrelated to clk_esc.
RxEscClk	Input	The clk_rx_esc clock is use by the MIPI Tx DPHY for reverse

		low power data reception. Tx DPHY may or may not require this clock. When this clock is required, the frequency of clk_rx_esc will be defined by the Tx DPHY and MIPI DPHY interface timing requirements.
Host_rx_payload[31:0]	Output	Received packet data output. The Host Receive Packet Interface presents 4 bytes at a time. Bytes are valid in this interface according to the rx_cmd_byte_count signal, beginning with the lowest byte.
Host_rx_payload_valid	Output	Packet data valid. This active high signal indicates that the controller is presenting valid packet during the next clk_byte period.
Host_rx_payload_valid_last	Output	This data is the last of the packet, active high signals last cycle of rx_payload_valid.
Host_rx_cmd_valid	Output	Packet header data is valid on the packet header ports below when this signal is asserted.
Host_rx_cmd_vc[1:0]	Output	Packet virtual channel number, valid when rx_cmd_valid is asserted.
Host_rx_cmd_data_type[5:0]	Output	Packet data type, valid when rx_cmd_valid is asserted. See the MIPI DSI-2 specification for a definition of possible values.
Host_rx_cmd_byte_count[15:0]	Output	Packet Word Count (byte count). Contains the number of bytes of data in the received packet. Valid when rx_cmd_valid is asserted.
Host_ecc_one_bit_error	Output	Single bit error in the packet header was detected and corrected. Active high. Valid when rx_cmd_valid is high.
Host_ecc_two_bit_error	Output	Two packet header bit errors were detected and not corrected,active high. Valid when rx_cmd_valid is high.
Host_ecc_one_bit_error_pos[4:0]	Output	Position of the corrected single bit error in the packet header. Valid when ecc_one_bit_error is high.
Host_ecc_err	Output	Error detected in the ECC bits. Active high. Valid when rx_cmd_valid is high.
Host_ecc_err_pos[2:0]	Output	Position of the erroneous bit in the ECC bits, valid when ecc_err is asserted.
Host_crc_err	Output	Asserts high when the CRC calculated on the received data does not match the CRC the transmitter sent at the end of the packet.

Timing for Generic Long Write with a payload of 20 bytes, single DPHY lane, and a VC=0 is list below:

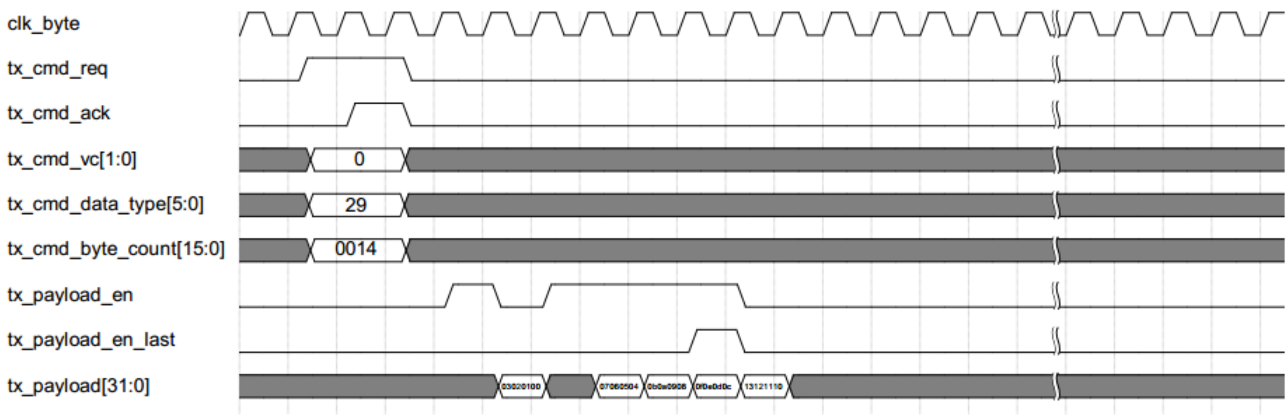


Figure 10 Generic Long Write with a payload of 20 bytes, single DPHY lane, and a VC=0

Timing for Generic Long Write with a payloads of 20 bytes and 10 bytes, VC=0 is list below:

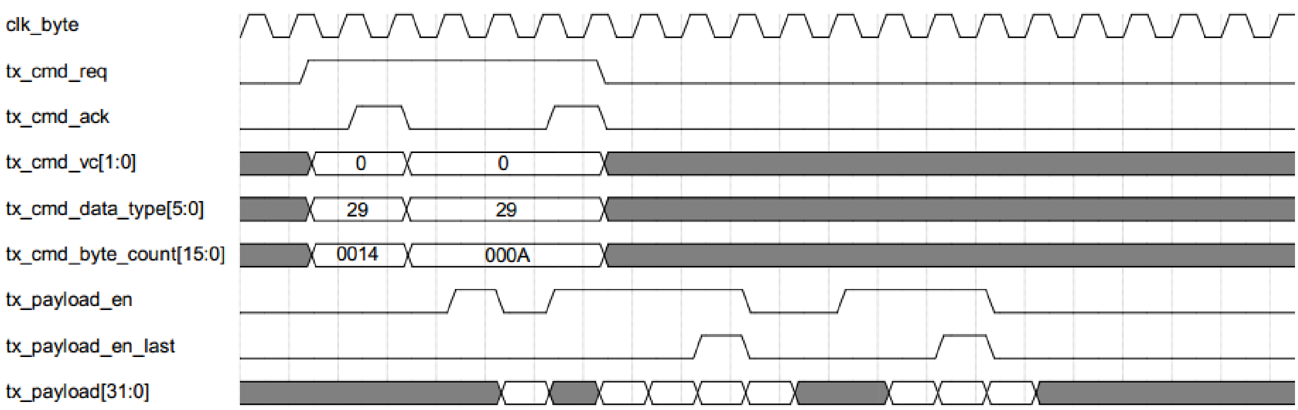


Figure 11 Generic Long Write with a payloads of 20 bytes and 10 bytes, VC=0

Timing for Sync Start, V Sync Start Packet is list below:

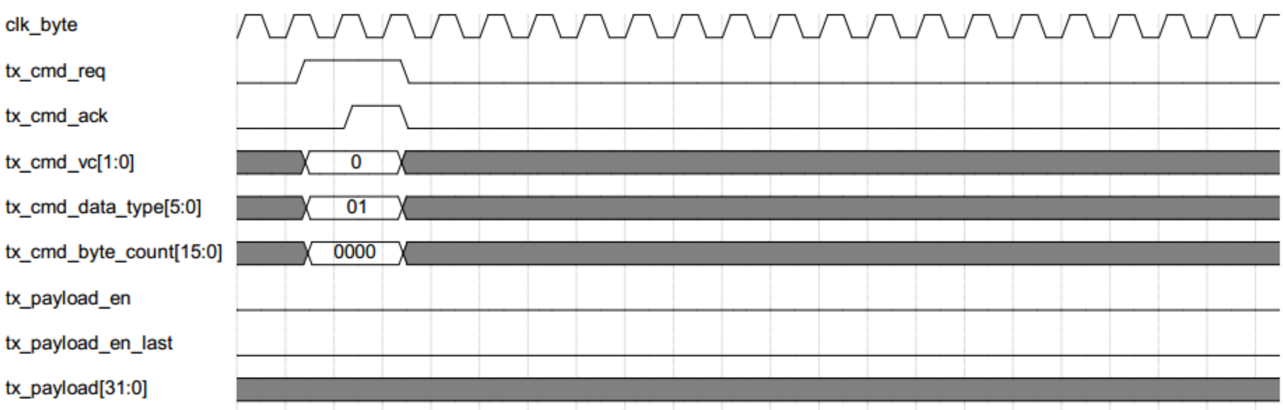


Figure 12 Sync Start, V Sync Start Packet

Timing for Generic Read with no parameters, one DPHY data lane is list below:

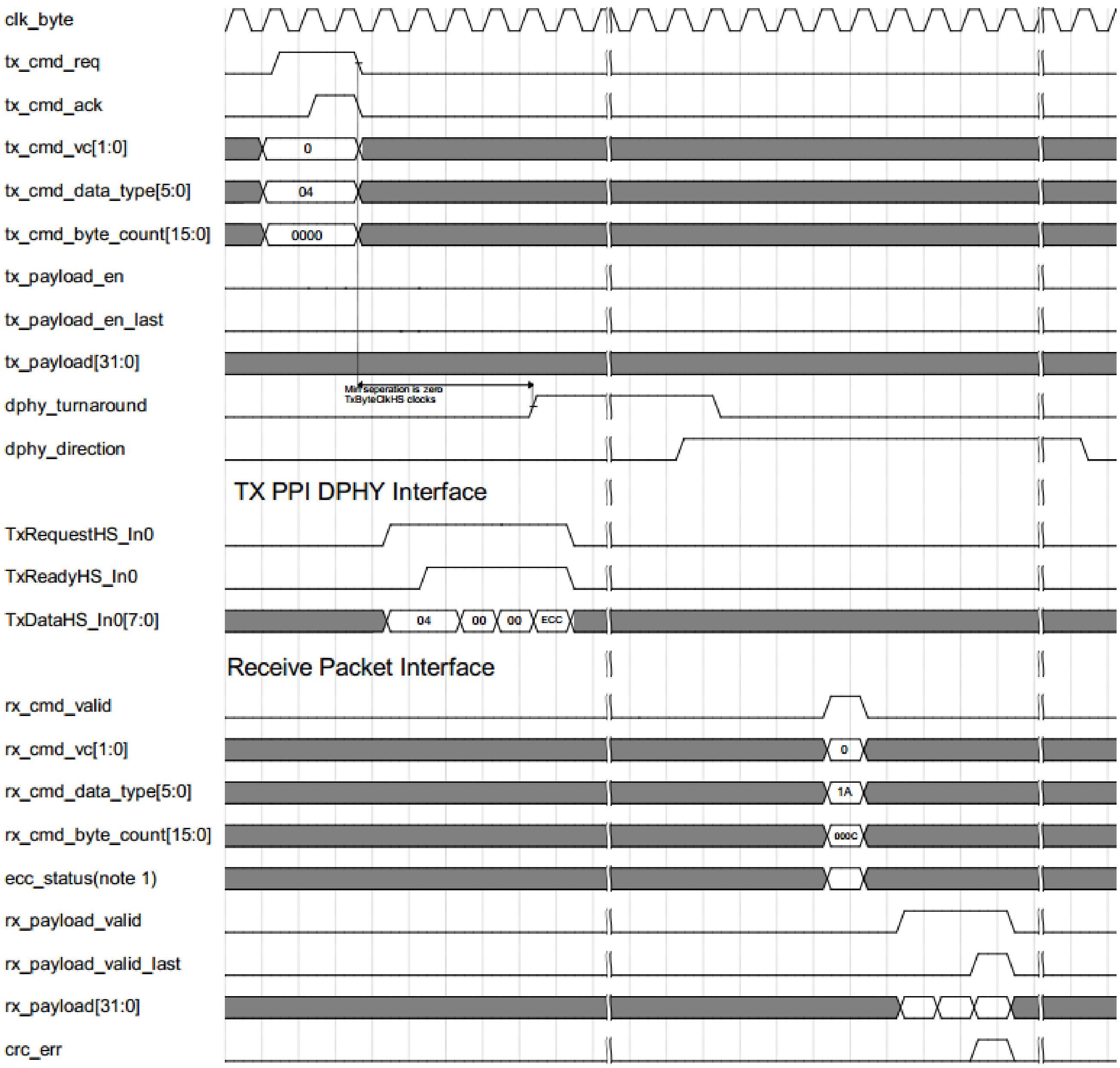


Figure 13 Generic Read with no parameters, one DPHY data lane

Note : ecc_status is the combination of ecc_one_bit_error, ecc_two_bit_error, ecc_one_bit_error_pos, ecc_err and ecc_err_pos

4.4. DSI Peripheral Controller interface

Figure 14 illustrates the DSI Peripheral Controller Core. The DSI Peripheral Controller Core operates on the peripheral (receive) side of a DSI link.

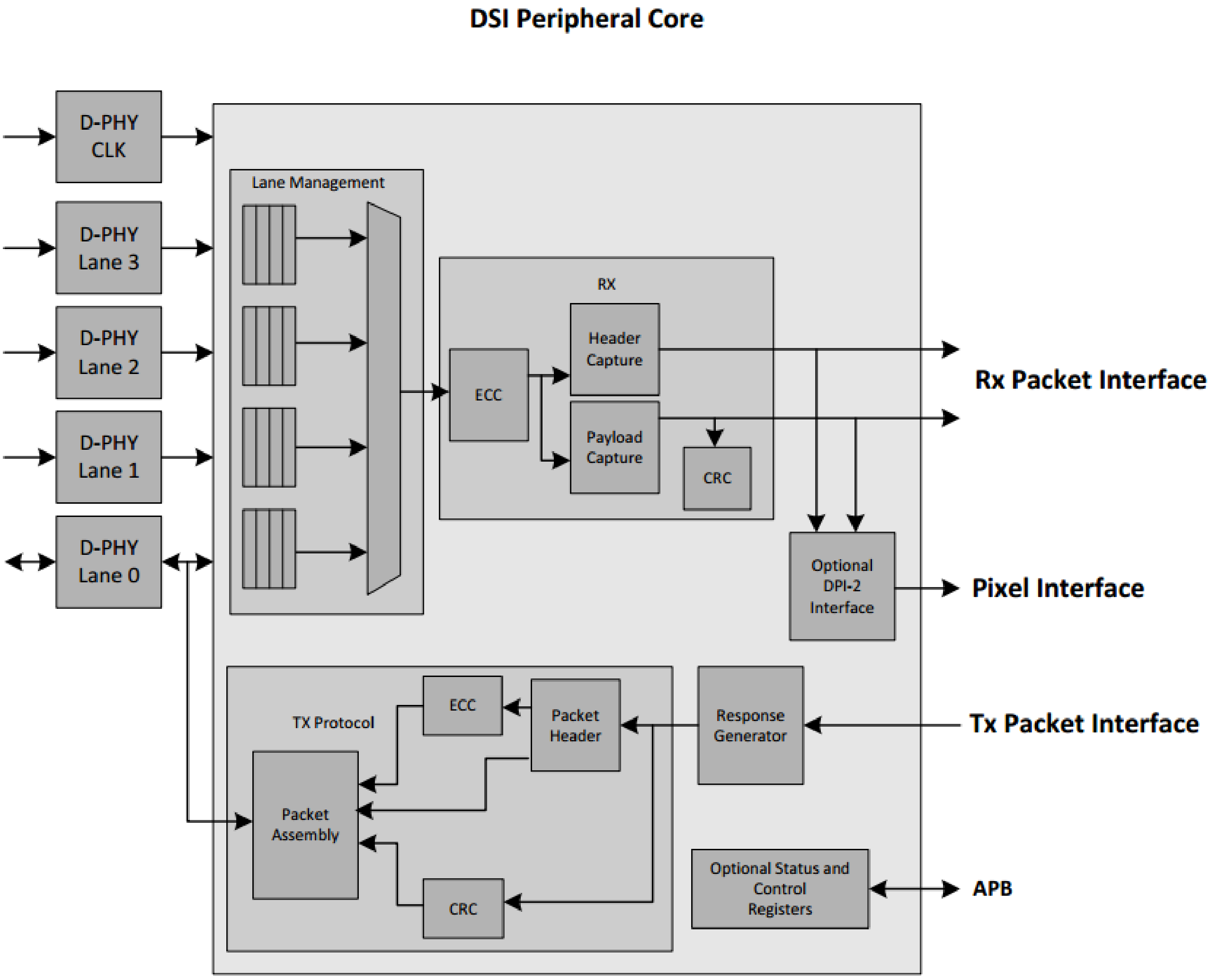


Figure 14   DSI Peripheral Controller Core Block Diagram

The DSI Peripheral Controller Core is the DSI implementation on the Display side of link. It accepts commands from the DSI Host Controller Core through 1-4 D-PHY lanes, and reassembles the DSI commands and data for the Display. A Packet Interface is supported for direct connection to the DSI Peripheral Controller Core.

4.4.1. APB interface

The DSI Peripheral controller can be configured with an optional Control and Status Register (CSR) interface. The CSR provides an APB compatible interface that enables control of the controller's configuration inputs via registers accessible via the APB interface.

The port descriptions for the CSR APB interface is described below.

Table 10   DSI Peripheral controller APB interface

pclk	Input	csr_clk is the APB pclk. All signals, except pclk_reset_n, are synchronous to pclk.
pclk_reset_n	Input	Async reset input for all logic in the pclk clock domain.
paddr[17:0]	Input	APB address. All registers are addressed on a 32 bit boundary so paddr[1:0] should always be set to 2'b00.
pwrite	Input	APB write signal, active high for write, low for read. Assert

		during setup phase for writes.
psel	Input	APB select signal, active high. The CSR responds with psel is asserted,and paddr contains the address of a valid register.
pwdata[31:0]	Input	APB write data.
prdata[31:0]	Output	APB read data.
pready	Output	APB pready output. Always asserted for writes, asserted during access phase for reads.
penable	Input	APB penable. Assert during access phase, this can be asserted for multiple clocks (even though APB spec specifies only one clock).

The memory map of the DSI Peripheral Controller CSR APB interface is described below.

Table 11 DSI Peripheral controller APB memory map

0x00400	R/W	[1:0]	CFG_NUM_LANES: cfg_num_lanes[1:0] setting for the Peripheral Controller. Sets the number of active lanes that are to be used for receiving MIPI data. 2'b00 – 1 Lane 2'b01 – 2 Lanes 2'b10 – 3 Lanes 2'b11 – 4 Lanes
0x00404	R/W	[1:0]	CFG_VC: cfg_vc setting for the Peripheral Controller. Sets the Virtual Channel (VC) of packets that will be sent to the receive packet interface. Packets with VC not equal to cfg_vc are discarded and the "DSI VC ID Invalid" bit (bit 12) in the DSI error report is set.
0x00408	R/W	[0]	CFG_DISABLE_VC_CHECK: Disables the peripheral controller from filtering out packets that have a VC not equal to cfg_vc. 1'b0 – Peripheral controller discard all packets with a VC not equal to cfg_vc setting. 1'b1 – Peripheral controller will not discard packets based on VC. Note that with this setting, the error "DSI VC ID invalid" will never assert.
0x0040c	R/W	[0]	CFG_DSI_REPRESSED_AFTER_UNRECOVER_ECC_ERR:cfg_dsi_repressed_after_unrecover_ecc_err setting for the Peripheral Controller. When set to 1'b1 the DSI Peripheral Controller will suppress any outputting of packet data on the packet interface until the current errored High Speed Transfer is complete.
0x00410	R/W	[23:0]	CFG_HRX_TO_COUNT: cfg_hrx_to_count setting for the Peripheral Controller. High Speed RX TimeOut in number of clk_periph clock periods. A value of 0x000000 disables the timeout.
0x00414	R/W	[23:0]	CFG_LTX_P_TO_COUNT: cfg_ltx_p_to_count

			setting for the Peripheral Controller. Low Power TX Timeout in number of clk_periph clock periods. A value of 0x000000 disables the timeout.
0x00418	R/W	[23:0]	CFG_BTA_P_TO_COUNT: cfg_bta_p_to_count setting for the Peripheral Controller. Bus Turn Around (BTA) timeout in number of clk_periph clock periods. A value of 0x000000 disables the timeout.
0x0041c	R/W	[0]	CFG_CRC_ERR_ASSERTS_INVALID_TX_LENGTH_ERR:cfg_crc_err_asserts_invalid_tx_length_err setting for the Peripheral Controller. Enables CRC error detection to set invalid tx length error flag. 0 – CRC error does not set tx length error flag 1 – CRC error does set tx length error flag
0x00420	R/W	[0]	CFG_ALLOW_READBACK_AFTER_MISSING_BTA_ERR:cfg_allow_readback_after_missing_bta_err setting for the Peripheral Controller. Allow read response even if last non-eotp packet received was not a read command. 0 – enables normal DSI behavior where last packet before a BTA must be a read cmd or else read response is not allowed and a protocol violation is logged. 1 – enables non-standard DSI behavior where the last packet received before a BTA can be either a read or write and the controller will still allow a read response packet to be sent. Protocol error is logged even in this case.
0x00424	R/W	[0]	CFG_DISABLE_RLPDT_CRC:cfg_disable_rlpdt_crc setting for the Peripheral Controller. Disables CRC generation in Reverse Low Power Data Transmission.When asserted high, the peripheral controller will not calculate CRC over the payload data, instead inserting 0x00 into the CRC fields as per the MIPI DSI specification.
0x00428	R/W	[0]	CFG_DISABLE_EOTP: cfg_disable_eotp setting for the Peripheral Controller. Disables EOTP packet support in the Peripheral Controller. 1’b0 – Peripheral Controller requires the Host to send EOTP packets at the end of every High Speed burst. 1’b1 – Peripheral Controller does not require the Host to send EOTP packets at the end of every High Speed Burst.
0x0042c	R/W	[0]	CFG_ENABLE_AUTOCLEAR_STATUS_REG: cfg_enable_autoclear_status_reg setting for the Peripheral Controller. Select whether status port

			reads clear accumulated dsi error report bits in the Peripheral Controller. The Error Report is a short packet of data type 0x02 that the Peripheral Control will send in response to a BTA if any of the error bits are set. 1'b0 – The Error Report Bits are only cleared after the Peripheral Controller send the Error Report packet to the Host Controller 1'b1 – The Error Report bits are cleared when either the Error Report packet is sent to the Host or when the user application reads the peripheral status port.
0x00430	R	[31:0]	CFG_STATUS_OUT: cfg_status_out status for the Peripheral Controller.
0x00680		[5:0]	ClkEsc domain ,THS-SETTLE
0x00684		[5:0]	ClkEsc domain,TCLK-SETTLE

Timing for apb interface is list below:

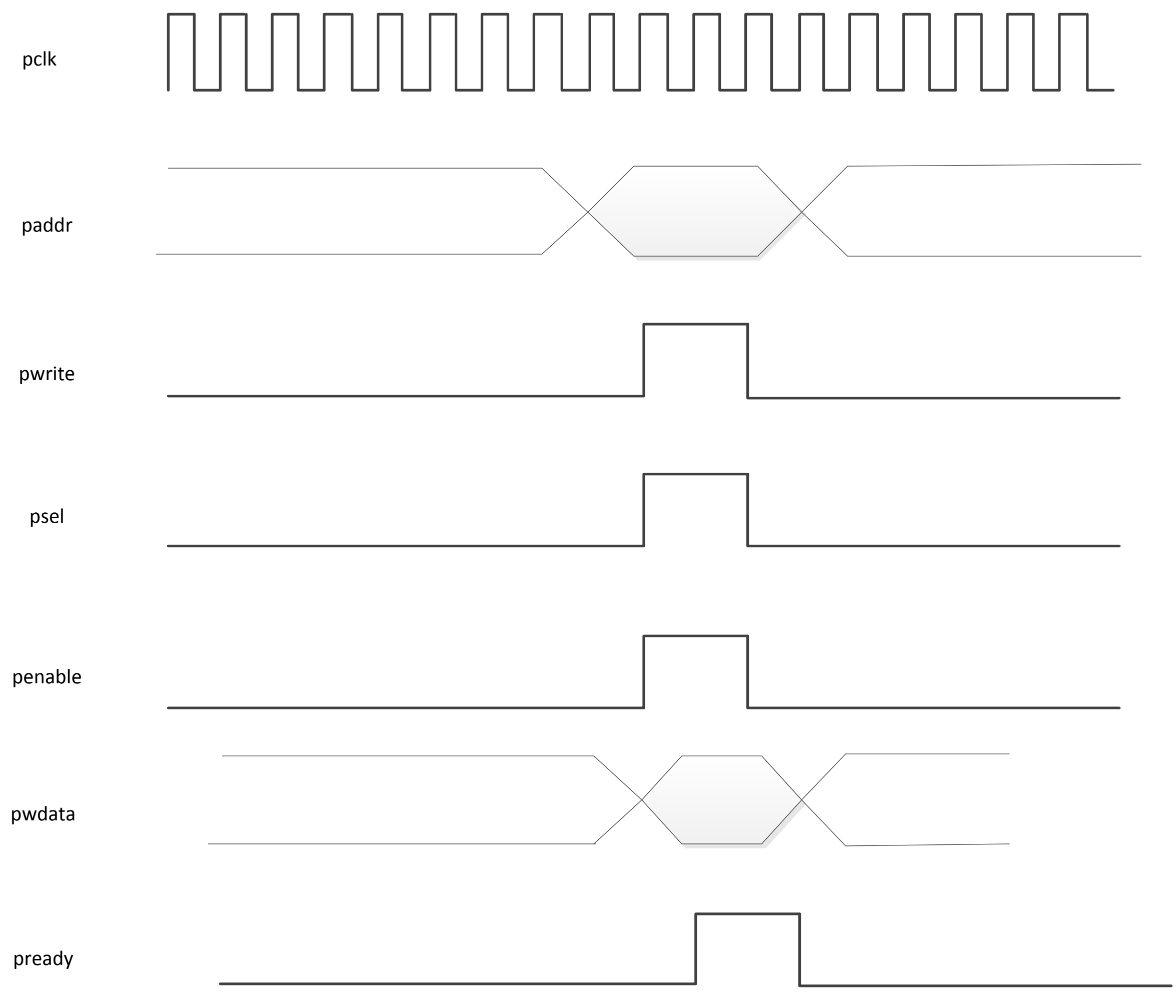


Figure 15 Timing for apb interface

4.4.2. packet interface

The DSI Peripheral Controller Core Packet Interface consists of Transmit, Receive, and Control & Status sections.Through these interfaces, the user application can take complete control of the DSI interface, receiving all video timing, receiving DSI commands, responding to DSI commands, monitoring the status of the interface, and error reporting.

The Transmit Packet Interface is the mechanism with which the user creates packets to return data to the Host over the MIPI Interface.

For Long Packets, the user provides the Virtual Channel (VC) number, Data Type (DT), and Word Count (WC) via the tx_cmd ports to the controller. The Controller then creates a packet header and pulls the packet data from the Packet Interface and outputs to the D-PHY to transmit.

For Short Packets, the user provides the Virtual Channel (VC) number, Data Type (DT), and required parameters (if any) via the tx_cmd ports to the controller. The Controller then creates the short packet and sends it to the DPHY to transmit.This interface enables the user application to transmit and receive any type of DSI packet.

The Packet Interface Transmit Interface ports are listed below.

Table 13 DSI Peripheral controller packet interface Transmit Interface

clk_periph	Input	TX Byte clock input. The tx_cmd and tx_payload interfaces are synchronous to clk_periph. clk_periph must be 1/8th the frequency that the DPHY data lanes operate at and is usually provided by the TX DPHY. This clock can be independent and unrelated to clk_esc.
TxEscClk	Input	The TxClkEsc is used by the MIPI RX DPHY and the DSI RX Controller to transmit Low Power data in the reverse direction to the Transmitter. The frequency of TxClkEsc is determined by system requirements and MIPI DPHY timing requirements. TxClkEsc is asynchronous to any other clock.
reset_n	Input	Asynchronous reset, active low. This reset applies to all logic in the Controller Core that uses clk_periph.
reset_esc_n	Input	Asynchronous reset, active low. This reset applies to all logic in the clk_esc clock domain.
periph_tx_payload[31:0]	Input	Packet data input. Tx_payload is expected to be valid one clock after tx_payload_en is asserted.
periph_tx_payload_en	Output	Packet data read enable. This active high signal indicates that the controller requires a valid packet during the next periph_clk period.
periph_tx_payload_en_last	Output	Last packet read enable, active high signal indicates last cycle of tx_payload_en.
periph_tx_cmd_data_type [5:0]	Input	Transmit packet DSI data type. It is written into the command buffer on the next rising edge of clk_periph when tx_cmd_ack is asserted high.
periph_tx_cmd_vc[1:0]	Input	Transmit packet command virtual channel. It is written into the command buffer on the next rising edge of clk_periph when tx_cmd_ack is asserted high.
periph_tx_cmd_byte_count[15:0]	Input	Transmit packet payload byte count. It is written into the command buffer on the next rising edge of clk_periph when tx_cmd_ack is asserted high.

		For DSI Long packet types, tx_cmd_byte_count defines the number of bytes of packet data to pull from the tx_payload port. For DSI Short packets, the format of tx_cmd_byte_count contains any optional parameters. If the SDI Short packet type does not have any parameters, it is recommended to set tx_cmd_byte_count to all 0s.
periph_tx_cmd_req	Input	Transmit packet command request. This active high signal informs the controller that the packet command is valid. The packet command consists of the ports tx_cmd_data_type, tx_cmd_vc, and tx_cmd_byte_count. The controller will assert tx_cmd_ack when it accepts the command, after which, the user should either update command port values for the next transmit packet command or deassert tx_cmd_req.
periph_tx_cmd_ack	Output	Transmit packet command request acknowledge. This active high signal indicates that the controller has accepted the TX packet request and the user logic should either submit a new request or deassert tx_cmd_req on the next rising edge of clk.
periph_trigger_req	Input	Transmit trigger request. This active high signal informs the controller that the trigger number on trigger_send is valid. The controller will assert trigger_ack when it accepts the command, after which, the user should either put update trigger_ack with the values for the next transmit packet or deassert trigger_req.
periph_trigger_ack	Output	Transmit trigger request acknowledge. This active high signal indicates that the controller has accepted the trigger request and the user logic should either submit a new request or deassert trigger_req on the next rising edge of user_clk.
periph_trigger_send[1:0]	Input	Transmit trigger. The trigger number on trigger_send is sampled when trigger_ack is asserted high. The format of trigger_send is as follows: 1'b00 = Trigger 0 (Reset-Trigger) 1'b01 = Trigger 1 (Tearing Effect) 1'b10 = Trigger 2 (Peripheral Acknowledge) 1'b11 = Trigger 3 ([Reserved])
periph_tx_timeout_error	Output	Asserts high for one clk_periph period when a low power tx timeout has occurred. This indicates that the current transmission by the peripheral controller to the host controller was not successful and that it is unknown how much, if any, of the transmitted packet was received by the Host end.
hs_rx_timeout	Output	Asserts for one clk_periph when the high speed timeout counter has reached cfg_hrx_to_count[23:0]
lp_tx_timeout	Output	Asserts for one clk_periph when the low power tx timeout counter has reached cfg_ltx_to_count[23:0]
periph_te_enable	Input	Tearing Effect Enable, active high. When set to 1'b1, the peripheral controller will wait for te_event_in to assert to 1'b1

		after back to back BTAs have been received without any other traffic in between. Once te_event_in is asserted the peripheral controller will send the TE trigger (01011101 first bit to last bit order) to the host, perform a BTA back to the host and assert te_ack for one clk_periph.
periph_te_rdy	Output	Tearing Event Ready. Asserts to 1'b1 when the peripheral controller is in control of the MIPI interface and ready to accept a te_event_in assertion to 1'b1. It is not necessary to wait for te_rdy = 1'b1 before asserting te_event_in.
periph_te_ack	Output	Tearing Event acknowledge. Asserts to 1'b1 for one clk_periph clock to acknowledge te_event_in and to signal that the controller will send a TE Trigger to the host
periph_te_event_in	Input	Display Tearing Event input, active high. Assert to 1'b1 to indicate a Tearing Event has occurred. Hold at 1'b1 until te_ack asserts upon which te_event_in should be deasserted on the next clk_periph rising edge.
periph_te_fail	Input	

The Receive Packet Interface is where the user at the Peripheral receives commands and data from the Host. The user is provided the Virtual Channel (VC) number, Data Type (DT), and Word Count (WC) via the l_rx_pkt_* signals. The user is presented with any sent data on the rx_payload* signals. This interface enables the user application to receive any type of DSI packet.

The Packet Interface Receive Interface ports are listed below.

Table 14 DSI Peripheral controller packet interface Receive Interface

clk_periph	Input	TX Byte clock input. The tx_cmd and tx_payload interfaces are synchronous to clk_periph. clk_periph must be 1/8th the frequency that the DPHY data lanes operate at and is usually provided by the TX DPHY. This clock can be independent and unrelated to clk_esc.
RxEscClk	Input	The RxClkEsc is used by the MIPI RX DPHY to receive MIPI DPHY low power signaling and Forward Low Power Data.The frequency of RxClkEsc is determined by the system requirements and MIPI DPHY timing requirements for the RX DPHY. RxClkEsc is asynchronous to all other clocks.
periph_rx_payload[31:0]	Output	Received packet data output. The Peripheral Receive Packet Interface presents 4 bytes at a time. Bytes are valid in this interface according to the rx_cmd_byte_count signal.
periph_rx_payload_valid	Output	Packet data valid. This active high signal indicates that the controller is presenting valid packet during the next byte_clk period.
periph_rx_payload_valid_last	Output	This data is the last of the packet, active high signals last cycle of rx_payload_valid.
periph_rx_cmd_valid	Output	Packet header data is valid on the packet header ports below when this signal is asserted.
periph_rx_cmd [23:0]	Output	Packet Command, valid when rx_cmd_valid is asserted.

		[23:08] word count (wc) - byte count of payload [07:06] virtual channel number (vc) [05:00] packet data type – See the MIPI DSI-2 specification for a definition of possible values.
periph_rx_trigger[3:0]	Output	Received Escape Trigger from the Peripheral DPHY (RX). Value is one hot and represents one of 4 possible triggers. Refer to DPHY documentation for exact mapping of DPHY triggers to one hot value
periph_rx_trigger_valid	Output	Received Escape Trigger Valid. Asserts for one clk_periph to indicate that the value on rx_trigger[3:0] is valid. Active high.
periph_ecc_one_bit_err	Output	Single bit error in the packet header was detected and corrected.Active high. Valid when rx_cmd_valid is high.
periph_ecc_two_bit_err	Output	Two packet header bit errors were detected and not corrected, active high. Valid when rx_cmd_valid is high.
periph_ecc_one_bit_err_pos[4:0]	Output	Position of the corrected single bit error in the packet header. Valid when ecc_one_bit_error is high.
periph_ecc_err	Output	Error detected in the ECC bits. Active high. Valid when rx_cmd_valid is high
periph_ecc_err_pos [2:0]	Output	Position of the erroneous bit in the ECC bits, valid when ecc_err is asserted.
periph_crc_err	Output	Asserts high when the CRC calculated on the received data does not match the CRC the transmitter sent at the end of the packet. crc_err is valid when rx_payload_valid_last asserts.
periph_dphy_direction	Output	Reports the current bus direction. 1'b0 = Bus is in Reverse direction (Peripheral is TX). 1'b1 = Bus is in Forward direction (Peripheral is RX).
periph_bta_timeout	Output	Peripheral BTA timeout. Asserts when the bta timeout counter has reached a count equal to the value set via cfg_bta_p_to_count[23:0]
periph_rx_ulps_active[4:0]	Output	Receive ULPS is active. Each bit represents a data lane and the clock lane. A '1' indicates the associated clock lane or data lane is in ULPS,'0' indicates not in ULPS [0] – clock lane [1] – data lane 0 [2] – data lane 1 [3] – data lane 2 [4] – data lane 3
periph_rx_ulps_mark_active[4:0]	Output	Receive ULPS is in mark state, about to exit ULPS. Each bit represents a data lane and the clock lane. A '1' indicates the associated clock lane or data lane is in Mark state and will soon leave ULPS, '0' indicates not in ULPS or Mark [0] – clock lane [1] – data lane 0 [2] – data lane 1 [3] – data lane 2 [4] – data lane 3

Timing for Generic Long Write packet with a payload of 8 bytes, single DPHY lane, and Virtual Channel=0 is list below:

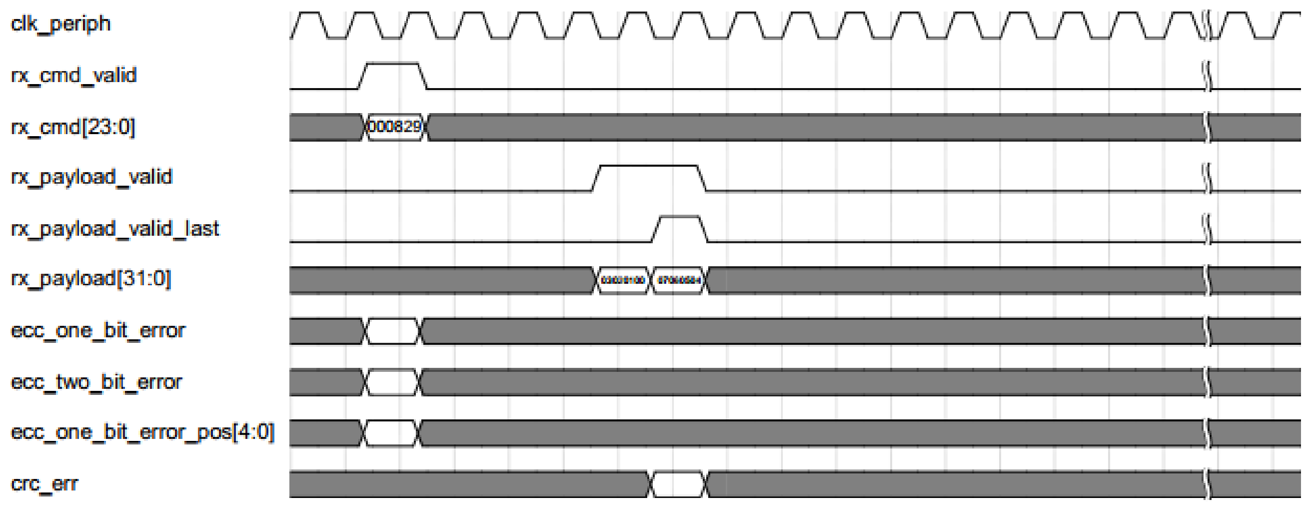


Figure16 Generic Long Write packet with a payload of 8 bytes, single DPHY lane, and Virtual Channel=0

Timing for Short Packet Receive With Data Type Of CM Off is list below:

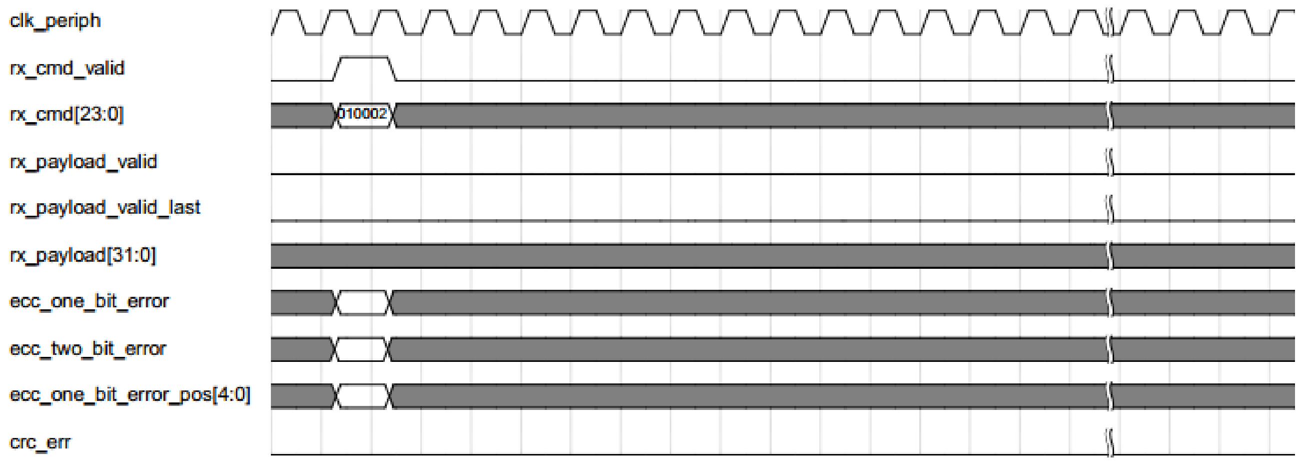


Figure17 Short Packet Receive With Data Type Of CM Off

Timing for Generic Read packet and read response of 12 bytes is list below:

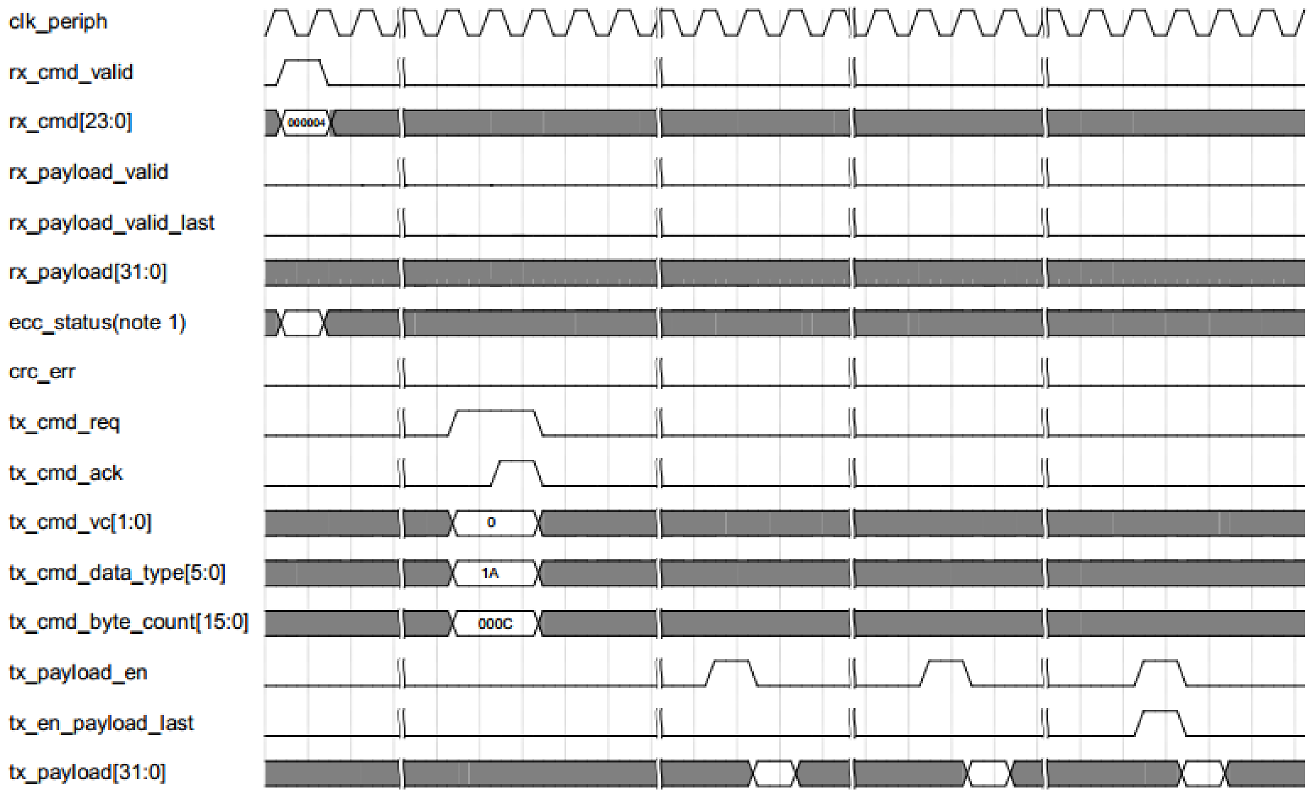


Figure18 Generic Read packet and read response of 12 bytes

Note : ecc_status represets the ecc status signals ecc_one_bit_error,ecc_two_bit_error,ecc_one_bit_error_pos,ecc_err and ecc_err_pos.

5. Typical mipi Application Example

本章节包含 4 个参考设计：RGB to mipi tx 参考设计，mipi rx to mipi tx 参考设计，mipi rx to mipi tx command 屏参考设计，mipi rx 回 ID 参考设计，可以帮助用户更好的熟悉 H1 mipi DSI controller 的使用方法。

5.1. RGB to mipi tx 参考设计

这个参考设计主要功能是：将fpga内部产生的rgb pattern信号转化成mipi信号发送出去。使用mipi DSI controller实现一路MIPI发送，最后接屏显示，下面是功能框图。

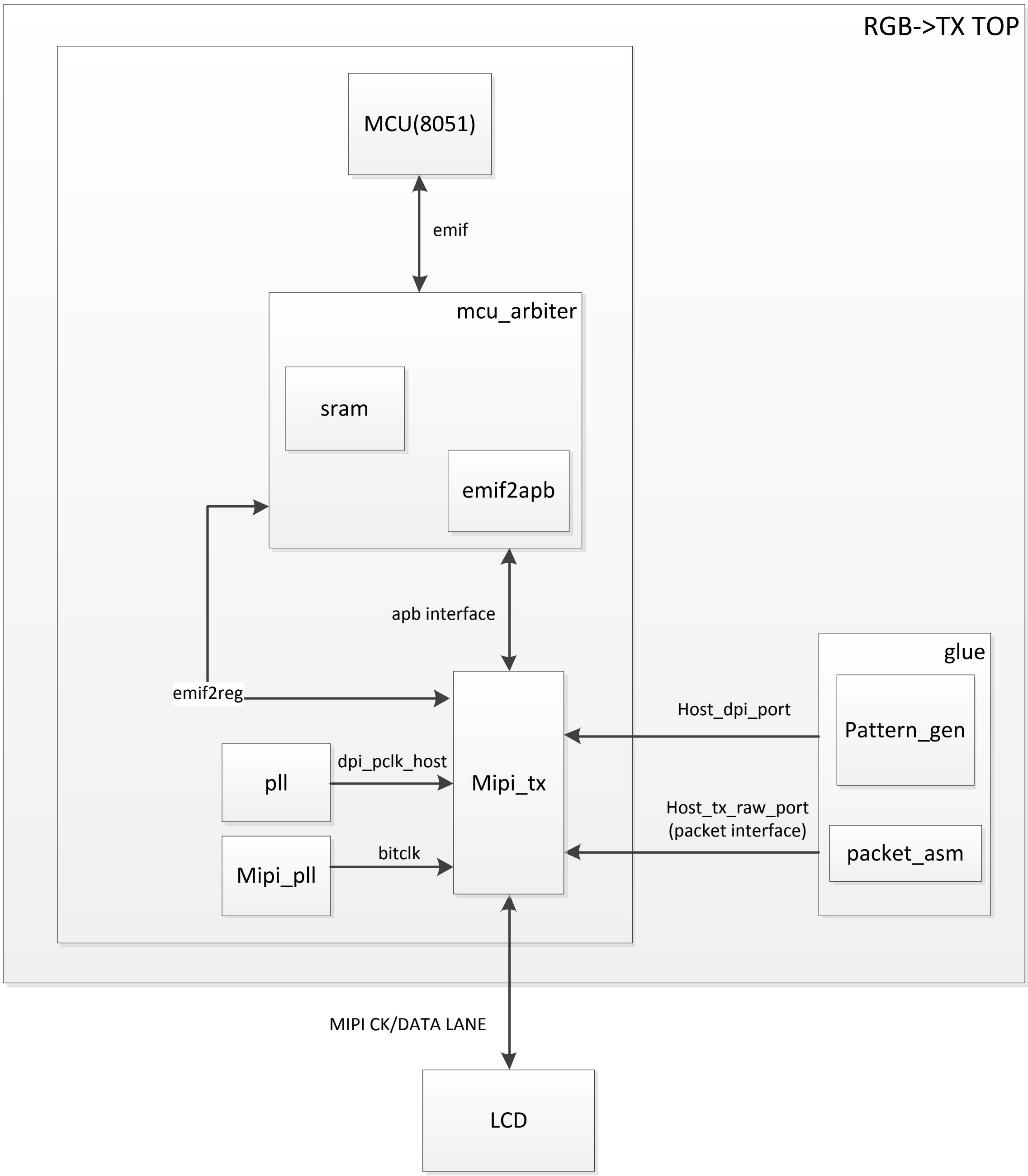


Figure19　RGB to mipi tx 参考设计功能框图

主要由 8 个模块组成：MCU, mcu_arbiter,sram,emif2apb, pll, mipi_pll, mipi_tx ,glue.

■ MCU

用户可以通过 Keil 软件编程,生成的 HEX 文件作为 MCU(8051) sram 的初始化文件。主要功能包括对屏幕的复位(rstn_lcd)，对 mipi DSI controller 的复位(rstn_mipi)，初始化完成后视频数据复位信号的释放(reset_dpi_n)；对 mipi DSI Host controller 控制和状态寄存器进行配置；自定义 8051 的扩展寄存器，用户可以使用这些寄存器，在 fp 完成需要的功能，对屏寄存器进行初始化配置。

8051 GPIO		
port0o[0]	reset_dpi_n	所有的初始化工作完成
port0o[1]	pstart	屏幕初始化命令开始信号
port0o[2]	rstn_mipi	mipi DSI controller 的复位

port0o[7]	rstn_lcd	屏幕的复位
port0i[3]	packet_finish	屏幕初始化命令结束信号
port0i[4]	tx_dphy_rdy	mipi tx bitclk 准备好

- mcu_arbiter

对MCU emif接口进行仲裁，分为三路，一路为emif2apb,用来对mipi DSI Host controller控制和状态寄存器进行配置。一路为emif2sram,使用fp内部sram作为8051的程序和数据存储空间。一路为emif2reg,8051的扩展寄存器，用户可以自定义这些寄存器，在fp完成需要的功能。

地址			
0x8000~0x8320	mipi DSI Host controller寄存器		包括dpi分辨率寄存器，bitclk频率设置寄存器
0x8800	8051的扩展寄存器	cmd_fifo	屏幕初始化cmd和data fifo
0x8804		mipi_sel	0：对mipi1寄存器进行配置，1：对mipi2寄存器进行配置
0x8805		func_set	func_set[0]　control sel; 1= enable func_set[1]　hs mode sel; 1= hs mode func_set[2]　pfifo rstn; low active

- sram

8051程序和数据存储空间。

- emif2apb

将 8051 emif 接口转换为 apb 接口，对 mipi DSI Host controller 控制和状态寄存器进行配置。

- pll

时钟输入 83M，用户可以修改 2 路时钟输出频率。dpi_pclk_host 即 mipi tx pixel clock，时钟频率必须 <=4/(3*8)th bitclk；esc_clk 作为 mipi tx lp mode 的时钟。

- mipi_pll

用来产生 mipi tx hs mode bitclk， tx bitclk (8 the byte clock frequency TxByteClkHS_s)=clkref*M/(N*O)，其中 clkref 为 83M，用户通过在 8051 中配置 host_pll_cn, host_pll_cm, host_pll_co 寄存器，改变 bitclk 时钟频率。具体配置参考章节 4.1。用户还需要配置 host_fifo_level 寄存器，用来平衡 dpi_pclk_host 时钟域和 TxByteClkHS_s 时钟域之间 fifo,保证 fifo 不空不满，具体寄存器含义参考章节 4.3.1。

- mipi_tx

将 dpi 信号转换成 mipi 信号，具体协议参考章节 4.3.2。

- glue

用户可以编辑 glue 模块，其中 packet_asm 模块用来对屏幕进行初始化；pattern_gen 模块产生 rgb 格式的 colorbar，然后送给 mipi tx 模块 dpi interface。

用户可以通过Keil软件编程，按照以下顺序对mipi DSI controller和屏进行相应的设置。

1. 对屏幕进行复位(rstn_lcd=0)，对mipi DSI controller进行复位(rstn_mipi=0)。
2. 对mipi DSI Host controller进行配置，mipi_sel=1（选择mipi2作为mipi tx），用户可以通过修改这些寄存器从而对mipi 发送端分辨率，时钟tx bitclk进行控制。tx bitclk (8 the byte clock frequency

TxByteClkHS_s)=clkref*M/(N*O)，其中clkref为83M，用户可以配置host_pll_cn, host_pll_cm, host_pll_co寄存器从而改变bitclk时钟频率。具体配置参考章节4.1。用户还需要配置host_fifo_level寄存器，用来平衡dpi_pclk_host时钟域和TxByteClkHS_s时钟域之间fifo,保证fifo不空不满，具体寄存器含义参考章节4.3.1。

```
//8051 to tx mipi apb
U32 xdata host_num_lanes   _at_ 0x8000;
U32 xdata host_noctn_clk   _at_ 0x8004;
U32 xdata host_t_pre     _at_ 0x8008;
U32 xdata host_t_post    _at_ 0x800c;
U32 xdata host_tx_gap    _at_ 0x8010;
U32 xdata host_auto_eotp   _at_ 0x8014;
U32 xdata host_ext_cmd     _at_ 0x8018;
U32 xdata host_hstx_timer  _at_ 0x801c;
U32 xdata host_lpdt_timer  _at_ 0x8020;
U32 xdata host_bta_timer   _at_ 0x8024;
U32 xdata host_twakeup     _at_ 0x8028;
U32 xdata host_status_ro   _at_ 0x802c;
U32 xdata host_error_ro    _at_ 0x8030;
U32 xdata host_line_size   _at_ 0x8200;
U32 xdata host_fifo_level  _at_ 0x8204;
U32 xdata host_color_code  _at_ 0x8208;
U32 xdata host_rbg_fmt     _at_ 0x820c;
U32 xdata host_vs_pol    _at_ 0x8210;
U32 xdata host_hs_pol    _at_ 0x8214;
U32 xdata host_video_mode  _at_ 0x8218;
U32 xdata host_hfp       _at_ 0x821c;
U32 xdata host_hbp       _at_ 0x8220;
U32 xdata host_hsa       _at_ 0x8224;
U32 xdata host_en_mult_pkts  _at_ 0x8228;
U32 xdata host_vbp       _at_ 0x822c;
U32 xdata host_vfp       _at_ 0x8230;
U32 xdata host_bllp_mode   _at_ 0x8234;
U32 xdata host_en_null_pkt   _at_ 0x8238;
U32 xdata host_vactive     _at_ 0x823c;
U32 xdata host_vc      _at_ 0x8240;
U32 xdata host_phy_d_pre   _at_ 0x8300;
U32 xdata host_phy_clk_pre   _at_ 0x8304;
U32 xdata host_phy_d_zero  _at_ 0x8308;
U32 xdata host_phy_clk_zero  _at_ 0x830c;
U32 xdata host_phy_d_trail   _at_ 0x8310;
U32 xdata host_phy_clk_trail   _at_ 0x8314;
U32 xdata host_pll_cn    _at_ 0x8318;
U32 xdata host_pll_cm    _at_ 0x831c;
U32 xdata host_pll_co    _at_ 0x8320;
```

```
//8051 to tx mipi apb
U32 cfg_data[37]={
  0x3,   //host_num_lanes
  0x0,   //host_noctn_clk
  0x64,
  0x21,
  0x1e,
  0x1,   //host_auto_eotp
  0x0,
  0x0,
  0x0,
  0x0,
  0xc8,
  0x438, //host_line_size 宽度
  0x89,  //host_fifo_level
  0x5,   //host_color_code 5= RGB 24-bit
  0x3,   //host_rbg_fmt 0= RGB 16-bit,1= RGB 18-bit,2= RGB 18-bit loosely packed,3= RGB 24-bit
  0x1,   //host_vs_pol  Sets polarity of dpi_vsync input, 0 active low, 1 active high
  0x1,   //host_hs_pol  Sets Polarity of dpi_hsync input, 0 active low, 1 active high
  0x0,   //host_video_mode 0=Non-Burst mode with Sync Pulses,1=Non-Burst mode with Sync Events,2=Burst mode
  0x14,  //host_hfp  宽前肩
  0x16,  //host_hbp  宽后肩
  0xa,   //host_hsa  宽同步
  0x0,   //host_en_mult_pkts 0=Video Line is sent in a single packet,1=Video Line is sent in two packets
  0x1,   //host_vbp  高后肩
  0x14,  //host_vfp  高前肩
  0x1,   //host_bllp_mode 0=blanking packets are sent during BLLP periods,1=LP mode is used for BLLP periods
  0x0,   //host_en_null_pkt 0=Blanking packet used in bllp region,1=Null packet used in bllp region
  0x780, //host_vactive  高度
  0x0,   //host_vc
  0x1,
  0x0,
  0x9,
  0x3c,
  0xd,
  0xd,
  0x10,  //host_pll_cn
  0xff,  //host_pll_cm
  0x0    //host_pll_co
  };
```

3. 解除对屏幕的复位(rstn_lcd=1)，解除对 mipi DSI controller 的复位(rstn_mipi=1),等待 bitclk 准备好(tx_dphy_rdy=1)。用户可以对屏幕进行初始化配置：mipi_lp_cmd_send(U8 cmd_set,UINT16 cmd_length ,UINT16 num ,U8 *buf , U8 long_cmd)，cmd_set 初始化命令，cmd_length 初始化命令参数个数，buf 初始化参数，long_cmd 长包或短包。

```
//panel initial use cmd29

 mydelay(1000);
 mipi_lp_cmd_send(0x29, 2 ,984 ,&pinf_cfg_data[0],1);

 cmd_d[1]=0x11;
 mipi_lp_cmd_send(0x29, 2 ,2,cmd_d,1);

 mydelay(3000);//180ms

 cmd_d[1]=0x29;
 mipi_lp_cmd_send(0x29, 2 ,2,cmd_d,1);
```

4. 所有的初始化工作完成，设置 reset_dpi_n=1,用户可以编辑 glue 模块，pattern_gen 模块产生 rgb 格式的 colorbar，然后送给 mipi tx 模块 dpi interface。

5.2. mipi rx to mipi tx 参考设计

这个参考设计主要功能是：使用H1 fpga 内部mipi DSI controller来实现一路MIPI接收，一路MIPI发送，最后接屏显示，下面是功能框图。

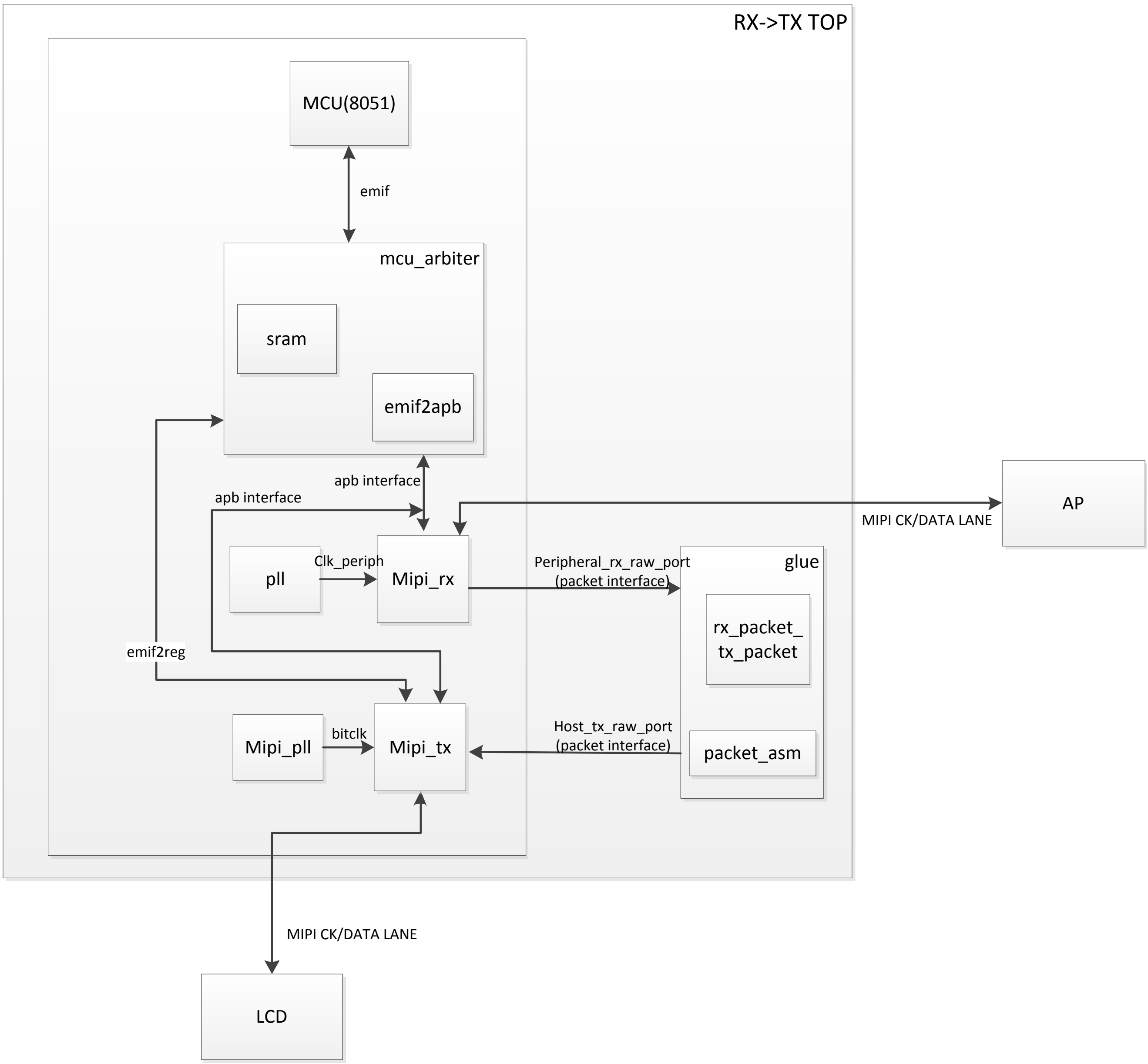


Figure20 mipi rx to mipi tx 参考设计功能框图

主要由 9 个模块组成：MCU, mcu_arbiter,sram,emif2apb, pll, mipi_pll,mipi_rx, mipi_tx ,glue.

- MCU

用户可以通过 Keil 软件编程,生成的 HEX 文件作为 MCU(8051) sram 的初始化文件。主要功能包括对屏幕的复位(rstn_lcd)，对 mipi DSI controller 的复位(rstn_mipi)，初始化完成后视频数据复位信号的释放(reset_dpi_n)；对 mipi DSI Host controller 和 mipi DSI Peripheral controller 控制和状态寄存器进行配置；自定义 8051 的扩展寄存器，用户可以使用这些寄存器，在 fp 完成需要的功能，对屏寄存器进行初始化配置。

8051 GPIO		
port0o[0]	reset_dpi_n	所有的初始化工作完成
port0o[1]	pstart	屏幕初始化命令开始信号
port0o[2]	rstn_mipi	mipi DSI controller 的复位
port0o[7]	rstn_lcd	屏幕的复位
port0i[3]	packet_finish	屏幕初始化命令结束信号
port0i[4]	tx_dphy_rdy	mipi tx bitclk 准备好

- mcu_arbiter

对MCU emif接口进行仲裁，分为三路，一路为emif2apb,用来对mipi DSI Host controller 和mipi DSI Peripheral controller控制和状态寄存器进行配置。一路为emif2sram,使用fp内部sram作为8051的程序和数据存储空间。一路为emif2reg,8051的扩展寄存器，用户可以自定义这些寄存器，在fp完成需要的功能。

地址			
0x8000~0x8320	mipi DSI Host controller寄存器		包括dpi分辨率寄存器，bitclk频率设置寄存器
0x8400~0x8684	mipi DSI Peripheral controller寄存器		
0x8800	8051的扩展寄存器	cmd_fifo	屏幕初始化cmd和data fifo
0x8804		mipi_sel	0：对mipi1寄存器进行配置，1：对mipi2寄存器进行配置
0x8805		func_set	func_set[0] control sel; 1= enable func_set[1] hs mode sel; 1= hs mode func_set[2] pfifo rstn; low active

- sram

8051程序和数据存储空间。

- emif2apb

将 8051 emif 接口转换为 apb 接口，对 mipi DSI Host controller 和 mipi DSI Peripheral controller 控制和状态寄存器进行配置。

- pll

时钟输入 77M，用户可以修改三路时钟输出频率。clk_periph 即 mipi rx hs mode byte clock，时钟频率必须高于 AP mipi 数据线 hs mode 速率的 1/8 倍；esc_clk 作为 mipi tx lp mode 的时钟；esc_clk_rx 作为 mipi rx lp mode 的时钟(必须高于 AP mipi lp mode 速率) 。

- mipi_pll

用来产生 mipi tx hs mode bitclk，必须高于 AP mipi 数据线 hs mode 速率。tx bitclk (8 the byte clock frequency TxByteClkHS_s)=clkref*M/(N*O)，其中 clkref 为 77M，用户通过在 8051 中配置 host_pll_cn, host_pll_cm, host_pll_co 寄存器，改变 bitclk 时钟频率。具体配置参考章节 4.1。

- mipi_rx

将 mipi 信号解析为 packet interface，具体协议参考章节 4.4.2。

■ mipi_tx

将 packet interface 信号转换成 mipi 信号，具体协议参考章节 4.3.3。

■ glue

用户可以编辑 glue 模块，其中 packet_asm 模块用来对屏幕进行初始化；rx_packet_tx_packet 模块接收 mipi rx 模块产生的 packet interface(peripheral_rx_raw_port) 数据，进行处理，然后送给 mipi tx 模块 packet interface (Host_tx_raw_port)。**用户产生 mipi tx 模块 packet interface 逻辑时需要注意：需要发送多个 command 时，Host_tx_cmd_req 最好按照 figure11 的时序产生，不要在 command 之间拉下来，否则 mipi lane 很容易进入 lp mode,再次从 lp mode 进入 hs mode,传输效率会降低。**

用户可以通过Keil软件编程，按照以下顺序对mipi DSI controller和屏进行相应的设置。

1. 对屏幕进行复位(rstn_lcd=0)，对mipi DSI controller进行复位(rstn_mipi=0)。
2. 对mipi DSI Host controller进行配置，mipi_sel=1（选择mipi2作为mipi tx），用户可以通过修改这些寄存器从而对mipi 发送端时钟tx bitclk进行控制。tx bitclk (8 the byte clock frequency TxByteClkHS_s)=clkref*M/(N*O)，其中clkref为77M，用户可以配置host_pll_cn, host_pll_cm, host_pll_co寄存器从而改变bitclk时钟频率。具体配置参考章节4.1。

```
//8051 to tx mipi apb
U32 xdata host_num_lanes  _at_ 0x8000;
U32 xdata host_noctn_clk  _at_ 0x8004;
U32 xdata host_t_pre     _at_ 0x8008;
U32 xdata host_t_post    _at_ 0x800c;
U32 xdata host_tx_gap    _at_ 0x8010;
U32 xdata host_auto_eotp  _at_ 0x8014;
U32 xdata host_ext_cmd    _at_ 0x8018;
U32 xdata host_hstx_timer _at_ 0x801c;
U32 xdata host_lpdt_timer _at_ 0x8020;
U32 xdata host_bta_timer  _at_ 0x8024;
U32 xdata host_twakeup    _at_ 0x8028;
U32 xdata host_status_ro  _at_ 0x802c;
U32 xdata host_error_ro   _at_ 0x8030;
U32 xdata host_line_size  _at_ 0x8200;
U32 xdata host_fifo_level _at_ 0x8204;
U32 xdata host_color_code _at_ 0x8208;
U32 xdata host_rbg_fmt    _at_ 0x820c;
U32 xdata host_vs_pol    _at_ 0x8210;
U32 xdata host_hs_pol    _at_ 0x8214;
U32 xdata host_video_mode _at_ 0x8218;
U32 xdata host_hfp       _at_ 0x821c;
U32 xdata host_hbp       _at_ 0x8220;
U32 xdata host_hsa       _at_ 0x8224;
U32 xdata host_en_mult_pkts _at_ 0x8228;
U32 xdata host_vbp       _at_ 0x822c;
U32 xdata host_vfp       _at_ 0x8230;
U32 xdata host_bllp_mode  _at_ 0x8234;
U32 xdata host_en_null_pkt  _at_ 0x8238;
U32 xdata host_vactive    _at_ 0x823c;
U32 xdata host_vc      _at_ 0x8240;
U32 xdata host_phy_d_pre  _at_ 0x8300;
U32 xdata host_phy_clk_pre  _at_ 0x8304;
U32 xdata host_phy_d_zero _at_ 0x8308;
U32 xdata host_phy_clk_zero _at_ 0x830c;
U32 xdata host_phy_d_trail  _at_ 0x8310;
U32 xdata host_phy_clk_trail  _at_ 0x8314;
U32 xdata host_pll_cn    _at_ 0x8318;
U32 xdata host_pll_cm    _at_ 0x831c;
U32 xdata host_pll_co    _at_ 0x8320;
```

```
//8051 to tx mipi apb
U32 cfg_data[37]={
  0x3,   //host_num_lanes
  0x0,   //host_noctn_clk
  0x64,
  0x21,
  0x1e,
  0x1,   //host_auto_eotp
  0x0,
  0x0,
  0x0,
  0x0,
  0xc8,
  0x438, //host_line_size 宽度
  0x89,  //host_fifo_level
  0x5,   //host_color_code 5= RGB 24-bit
  0x3,   //host_rbg_fmt 0= RGB 16-bit,1= RGB 18-bit,2= RGB 18-bit loosely packed,3= RGB 24-bit
  0x1,   //host_vs_pol  Sets polarity of dpi_vsync input, 0 active low, 1 active high
  0x1,   //host_hs_pol  Sets Polarity of dpi_hsync input, 0 active low, 1 active high
  0x0,   //host_video_mode 0=Non-Burst mode with Sync Pulses,1=Non-Burst mode with Sync Events,2=Burst mode
  0x14,  //host_hfp  宽前肩
  0x16,  //host_hbp  宽后肩
  0xa,   //host_hsa  宽同步
  0x0,   //host_en_mult_pkts 0=Video Line is sent in a single packet,1=Video Line is sent in two packets
  0x1,   //host_vbp  高后肩
  0x14,  //host_vfp  高前肩
  0x1,   //host_bllp_mode 0=blanking packets are sent during BLLP periods,1=LP mode is used for BLLP periods
  0x0,   //host_en_null_pkt 0=Blanking packet used in bllp region,1=Null packet used in bllp region
  0x780, //host_vactive  高度
  0x0,   //host_vc
  0x1,
  0x0,
  0x9,
  0x3c,
  0xd,
  0xd,
  0x10,  //host_pll_cn
  0xff,  //host_pll_cm
  0x0    //host_pll_co
  };
```

3. 对mipi DSI Peripheral controller进行配置，mipi_sel=0（选择mipi1作为mipi rx）。

```
//8051 to rx mipi apb
U32 xdata periph_lanes          _at_ 0x8400;
U32 xdata periph_vc           _at_ 0x8404;
U32 xdata periph_vc_check       _at_ 0x8408;
U32 xdata periph_ecc_err         _at_ 0x840c;
U32 xdata periph_hrx            _at_ 0x8410;
U32 xdata periph_ltx            _at_ 0x8414;
U32 xdata periph_bta            _at_ 0x8418;
U32 xdata periph_crc_err         _at_ 0x841c;
U32 xdata periph_bta_err         _at_ 0x8420;
U32 xdata periph_dis_rlpdt       _at_ 0x8424;
U32 xdata periph_dis_eotp       _at_ 0x8428;
U32 xdata periph_clr_status      _at_ 0x842c;
U32 xdata periph_m_settle       _at_ 0x8680;
U32 xdata periph_mc_settle       _at_ 0x8684;
```

```
//8051 to rx mipi apb
U32 peri_cfg_data[37]={
  0x3,   //periph_lanes
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,
  0x0,   //periph_dis_eotp
  0x0,
  0x1,
  0x0 };
```

4. 解除对屏幕的复位(rstn_lcd=1)，解除对 mipi DSI controller 的复位(rstn_mipi=1),等待 bitclk 准备好(tx_dphy_rdy=1)。用户可以对屏幕进行初始化配置：mipi_lp_cmd_send(U8 cmd_set,UINT16 cmd_length ,UINT16 num ,U8 *buf , U8 long_cmd)，cmd_set 初始化命令，cmd_length 初始化命令参数个数，buf 初始化参数，long_cmd 长包或短包。

```
//panel initial use cmd29

  mydelay(1000);
  mipi_lp_cmd_send(0x29, 2 ,984 ,&pinf_cfg_data[0],1);

  cmd_d[1]=0x11;
  mipi_lp_cmd_send(0x29, 2 ,2,cmd_d,1);

  mydelay(3000);//180ms

  cmd_d[1]=0x29;
  mipi_lp_cmd_send(0x29, 2 ,2,cmd_d,1);
```

5. 所有的初始化工作完成，设置 reset_dpi_n=1,用户可以编辑 glue 模块， rx_packet_tx_packet 模块接收 mipi rx 模块产生的 packet interface(peripheral_rx_raw_port) 数据，进行处理，然后送给 mipi tx 模块 packet interface (Host_tx_raw_port)。

5.3. mipi rx to mipi tx command 屏参考设计

这个参考设计主要功能是：使用H1 fpga 内部mipi DSI controller来实现一路MIPI接收，一路MIPI发送，最后接屏显示，下面是功能框图。

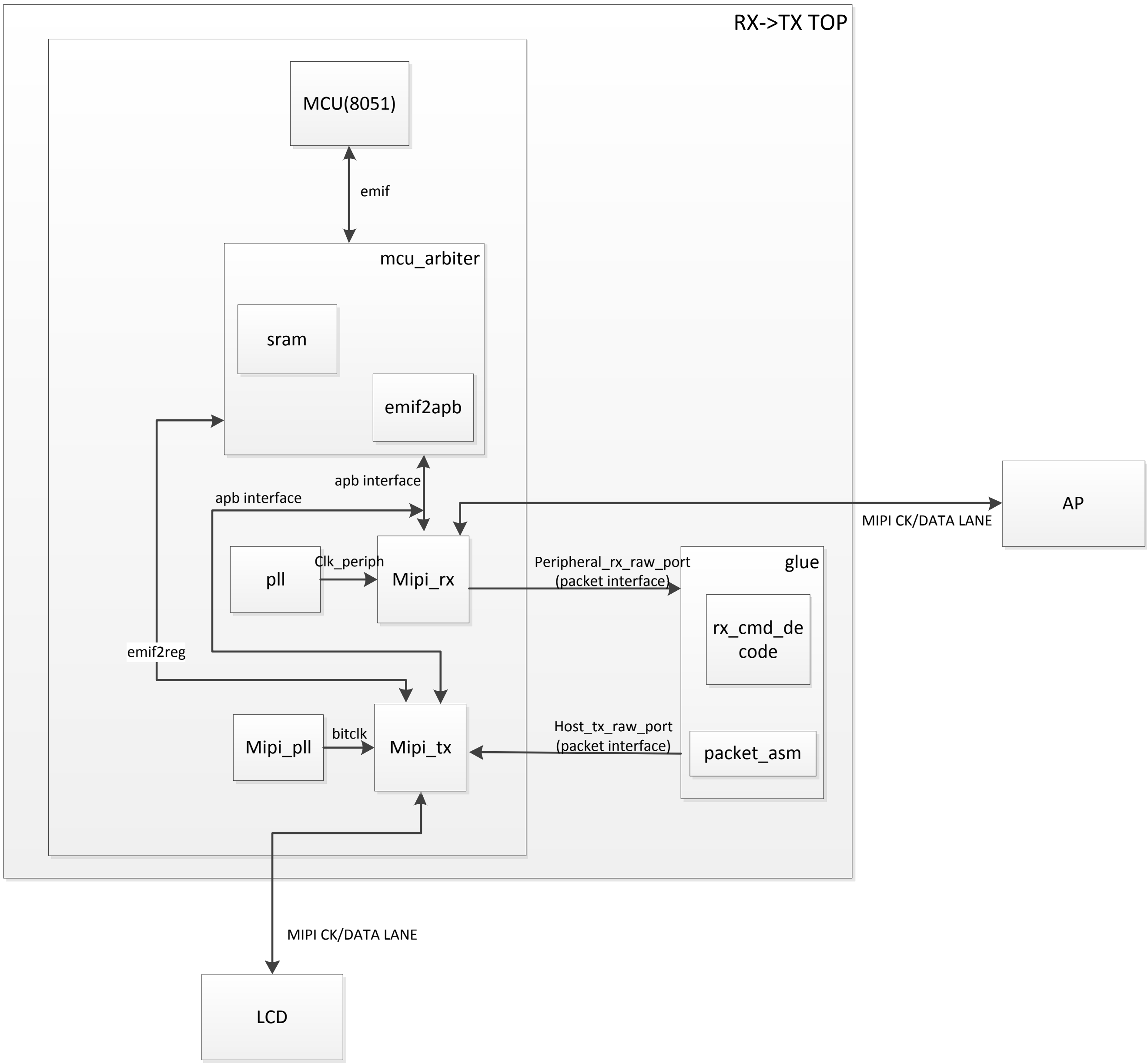


Figure21　mipi rx to mipi tx command屏参考设计功能框图

主要由 9 个模块组成：MCU, mcu_arbiter,sram,emif2apb, pll, mipi_pll,mipi_rx, mipi_tx ,glue.

- MCU

用户可以通过 Keil 软件编程,生成的 HEX 文件作为 MCU(8051) sram 的初始化文件。主要功能包括对屏幕的复位(rstn_lcd)，对 mipi DSI controller 的复位(rstn_mipi)，初始化完成后视频数据复位信号的释放(fp_cmd_enter(1))；对 mipi DSI Host controller 和 mipi DSI Peripheral controller 控制和状态寄存器进行配置；自定义 8051 的扩展寄存器，用户可以使用这些寄存器，在 fp 完成需要的功能，对屏寄存器进行初始化配置。

8051 GPIO		
port0o[1]	pstart	屏幕初始化命令开始信号
port0o[2]	rstn_mipi	mipi DSI controller 的复位
port0o[7]	rstn_lcd	屏幕的复位
port0i[3]	packet_finish	屏幕初始化命令结束信号

port0i[4]	tx_dphy_rdy	mipi tx bitclk 准备好

- mcu_arbiter

对MCU emif接口进行仲裁，分为三路，一路为emif2apb,用来对mipi DSI Host controller 和mipi DSI Peripheral controller控制和状态寄存器进行配置。一路为emif2sram,使用fp内部sram作为8051的程序和数据存储空间。一路为emif2reg,8051的扩展寄存器，用户可以自定义这些寄存器，在fp完成需要的功能。

地址			
0x8000~0x8320	mipi DSI Host controller寄存器		包括dpi分辨率寄存器，bitclk频率设置寄存器
0x8400~0x8684	mipi DSI Peripheral controller寄存器		
0x8800	8051的扩展寄存器	dnum_l	屏幕初始化packet interface wc低字节
0x8801		dnum_h	屏幕初始化packet interface wc高字节
0x8802		dset	屏幕初始化packet interface payload
0x8803		cmd	屏幕初始化packet interface cmd
0x8804		mipi_sel	0：对mipi1寄存器进行配置，1：对mipi2寄存器进行配置
0x8805		func_set	func_set[0] control sel; 1= enable func_set[1] hs mode sel; 1= hs mode func_set[2] pfifo rstn; low active func_set[3] pfifo wr; rising edge func_set[4] fp_cmd_sel; 0= mcu cmd tx, 1= fp cmd tx

- sram

8051程序和数据存储空间。

- emif2apb

将 8051 emif 接口转换为 apb 接口，对 mipi DSI Host controller 和 mipi DSI Peripheral controller 控制和状态寄存器进行配置。

- pll

时钟输入 83M，用户可以修改三路时钟输出频率。clk_periph 即 mipi rx hs mode byte clock，时钟频率必须高于 AP mipi 数据线 hs mode 速率的 1/8 倍；esc_clk 作为 mipi tx lp mode 的时钟；esc_clk_rx 作为 mipi rx lp mode 的时钟(必须高于 AP mipi lp mode 速率) 。

- mipi_pll

用来产生 mipi tx hs mode bitclk，必须高于 AP mipi 数据线 hs mode 速率。tx bitclk (8 the byte clock frequency TxByteClkHS_s)=clkref*M/(N*O)，其中 clkref 为 83M，用户通过在 8051 中配置 host_pll_cn, host_pll_cm, host_pll_co 寄存器，改变 bitclk 时钟频率。具体配置参考章节 4.1。

- mipi_rx

将 mipi 信号解析为 packet interface，具体协议参考章节 4.4.2。

- mipi_tx

将 packet interface 信号转换成 mipi 信号，具体协议参考章节 4.3.3。

- **glue**

  用户可以编辑 glue 模块，其中 packet_asm 模块用来对屏幕进行初始化；rx_cmd_decode 模块接收 mipi rx 模块产生的 packet interface(peripheral_rx_raw_port) 数据，进行处理，然后送给 mipi tx 模块 packet interface (Host_tx_raw_port)。**用户产生 mipi tx 模块 packet interface 逻辑时需要注意：需要发送多个 command 时，Host_tx_cmd_req 最好按照 figure11 的时序产生，不要在 command 之间拉下来，否则 mipi lane 很容易进入 lp mode,再次从 lp mode 进入 hs mode,传输效率会降低。**

用户可以通过Keil软件编程，按照以下顺序对mipi DSI controller和屏进行相应的设置。

1. 对屏幕进行复位(rstn_lcd=0)，对mipi DSI controller进行复位(rstn_mipi=0)。
2. 对mipi DSI Host controller进行配置，mipi_sel=1（选择mipi2作为mipi tx），用户可以通过修改这些寄存器从而对mipi 发送端时钟tx bitclk进行控制。tx bitclk (8 the byte clock frequency TxByteClkHS_s)=clkref*M/(N*O)，其中clkref为83M，用户可以配置host_pll_cn, host_pll_cm, host_pll_co寄存器从而改变bitclk时钟频率。具体配置参考章节4.1。

```
//8051 to tx mipi apb
U32 xdata host_num_lanes  _at_ 0x8000;
U32 xdata host_noctn_clk  _at_ 0x8004;
U32 xdata host_t_pre    _at_ 0x8008;
U32 xdata host_t_post   _at_ 0x800c;
U32 xdata host_tx_gap   _at_ 0x8010;
U32 xdata host_auto_eotp  _at_ 0x8014;
U32 xdata host_ext_cmd    _at_ 0x8018;
U32 xdata host_hstx_timer _at_ 0x801c;
U32 xdata host_lpdt_timer _at_ 0x8020;
U32 xdata host_bta_timer  _at_ 0x8024;
U32 xdata host_twakeup    _at_ 0x8028;
U32 xdata host_status_ro  _at_ 0x802c;
U32 xdata host_error_ro   _at_ 0x8030;
U32 xdata host_line_size  _at_ 0x8200;
U32 xdata host_fifo_level _at_ 0x8204;
U32 xdata host_color_code _at_ 0x8208;
U32 xdata host_rbg_fmt    _at_ 0x820c;
U32 xdata host_vs_pol   _at_ 0x8210;
U32 xdata host_hs_pol   _at_ 0x8214;
U32 xdata host_video_mode _at_ 0x8218;
U32 xdata host_hfp      _at_ 0x821c;
U32 xdata host_hbp      _at_ 0x8220;
U32 xdata host_hsa      _at_ 0x8224;
U32 xdata host_en_mult_pkts _at_ 0x8228;
U32 xdata host_vbp      _at_ 0x822c;
U32 xdata host_vfp      _at_ 0x8230;
U32 xdata host_bllp_mode  _at_ 0x8234;
U32 xdata host_en_null_pkt  _at_ 0x8238;
U32 xdata host_vactive    _at_ 0x823c;
U32 xdata host_vc     _at_ 0x8240;
U32 xdata host_phy_d_pre  _at_ 0x8300;
U32 xdata host_phy_clk_pre  _at_ 0x8304;
U32 xdata host_phy_d_zero _at_ 0x8308;
U32 xdata host_phy_clk_zero _at_ 0x830c;
U32 xdata host_phy_d_trail  _at_ 0x8310;
U32 xdata host_phy_clk_trail  _at_ 0x8314;
U32 xdata host_pll_cn   _at_ 0x8318;
U32 xdata host_pll_cm   _at_ 0x831c;
U32 xdata host_pll_co   _at_ 0x8320;
```

```
//8051 to tx mipi apb
U32 cfg_data[37]={
  0x3,   //host_num_lanes
  0x0,   //host_noctn_clk
  0x64,
  0x21,
  0x1e,
  0x1,   //host_auto_eotp
  0x0,
  0x0,
  0x0,
  0x0,
  0xc8,
  0x438, //host_line_size 宽度
  0xb9,  //host_fifo_level
  0x5,   //host_color_code 5= RGB 24-bit
  0x3,   //host_rbg_fmt 0= RGB 16-bit,1= RGB 18-bit,2= RGB 18-bit loosely packed,3= RGB 24-bit
  0x1,   //host_vs_pol  Sets polarity of dpi_vsync input, 0 active low, 1 active high
  0x1,   //host_hs_pol  Sets Polarity of dpi_hsync input, 0 active low, 1 active high
  0x0,   //host_video_mode 0=Non-Burst mode with Sync Pulses,1=Non-Burst mode with Sync Events,2=Burst mode
  0x14,  //host_hfp  宽前肩
  0x16,  //host_hbp  宽后肩
  0xa,   //host_hsa  宽同步
  0x0,   //host_en_mult_pkts 0=Video Line is sent in a single packet,1=Video Line is sent in two packets
  0x1,   //host_vbp  高后肩
  0x14,  //host_vfp  高前肩
  0x1,   //host_bllp_mode 0=blanking packets are sent during BLLP periods,1=LP mode is used for BLLP periods
  0x0,   //host_en_null_pkt 0=Blanking packet used in bllp region,1=Null packet used in bllp region
  0x780, //host_vactive  高度
  0x0,   //host_vc
  0x1,
  0x0,
  0x19,  //host_phy_d_zero
  0x3c,
  0xd,
  0xd,
  0x10,  //host_pll_cn
  0xc9,  //host_pll_cm
  0x0    //host_pll_co
  };
```

3. 对mipi DSI Peripheral controller进行配置，mipi_sel=0（选择mipi1作为mipi rx）。

```
//8051 to rx mipi apb
U32 xdata periph_lanes        _at_ 0x8400;
U32 xdata periph_vc          _at_ 0x8404;
U32 xdata periph_vc_check     _at_ 0x8408;
U32 xdata periph_ecc_err       _at_ 0x840c;
U32 xdata periph_hrx          _at_ 0x8410;
U32 xdata periph_ltx          _at_ 0x8414;
U32 xdata periph_bta          _at_ 0x8418;
U32 xdata periph_crc_err       _at_ 0x841c;
U32 xdata periph_bta_err       _at_ 0x8420;
U32 xdata periph_dis_rlpdt     _at_ 0x8424;
U32 xdata periph_dis_eotp     _at_ 0x8428;
U32 xdata periph_clr_status    _at_ 0x842c;
U32 xdata periph_m_settle     _at_ 0x8680;
U32 xdata periph_mc_settle     _at_ 0x8684;
```

```
//8051 to rx mipi apb
 U32 peri_cfg_data[37]={
 0x3,    //periph_lanes
 0x0,    //cfg_vc
 0x0,    //dis_vc_check
 0x0,    //ecc error
 0x0,    //hrx timer
 0x0,    //ltx timer
 0x0,    //bta timer
 0x0,    //tx length err
 0x0,    //bta err
 0x0,    //dis_rlpdt crc
 0x1,    //periph_dis_eotp
 0x1,    //periph_clr_status
 0x3,    //periph_m_settle
 0x1     //periph_mc_settle
 };
```

4. 解除对屏幕的复位(rstn_lcd=1)，解除对 mipi DSI controller 的复位(rstn_mipi=1),等待 bitclk 准备好(tx_dphy_rdy=1)。用户可以对屏幕进行初始化配置：mipi_lp_cmd_send(U8 cmd_set,UINT16 cmd_length ,U8 *buf , U8 long_cmd)，cmd_set 初始化命令，cmd_length 初始化命令参数个数，buf 初始化参数，long_cmd 长包或短包。

```
panel_init();

  cmd_d[0]=0x29;
mipi_lp_cmd_send(0x5, 0x1 ,cmd_d,0);//0x05 no parameter, 0x29 is not parameter

  cmd_d[0]=0x11;
mipi_lp_cmd_send(0x5, 0x1 ,cmd_d,0);
```

5. 所有的初始化工作完成，设置 fp_cmd_sel=1(fp_cmd_enter(1)),用户可以编辑 glue 模块，rx_cmd_decode 模块接收 mipi rx 模块产生的 packet interface(peripheral_rx_raw_port) 数据，进行处理，然后送给 mipi tx 模块 packet interface (Host_tx_raw_port)。

5.4. mipi rx 回 ID 参考设计

这个参考设计主要功能是：使用mipi DSI controller实现一路MIPI接收，当检测到读ID命令时，回复ID给AP。下面是功能框图。

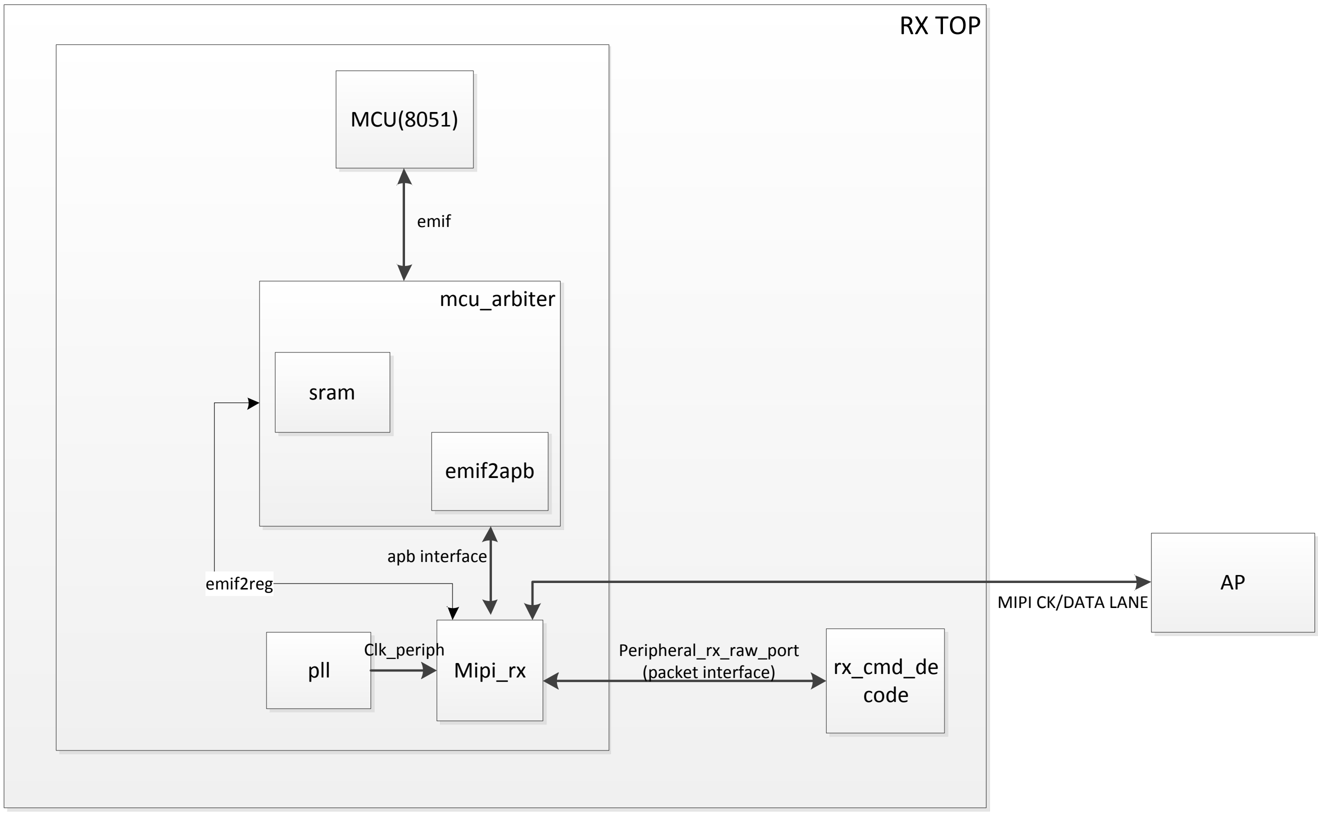


Figure22 mipi rx 回ID 参考设计功能框图

主要由 7 个模块组成：MCU, mcu_arbiter,sram,emif2apb, pll,mipi_rx , rx_cmd_decode.

- MCU

  用户可以通过 Keil 软件编程,生成的 HEX 文件作为 MCU(8051) sram 的初始化文件。主要功能包括对 mipi DSI controller 的复位(rstn_mipi)，对 mipi DSI Peripheral controller 控制和状态寄存器进行配置；自定义 8051 的扩展寄存器，用户可以使用这些寄存器，在 fp 完成需要的功能。

- mcu_arbiter

  对MCU emif接口进行仲裁，分为三路，一路为emif2apb,用来对mipi DSI Peripheral controller控制和状态寄存器进行配置。一路为emif2sram,使用fp内部sram作为8051的程序和数据存储空间。一路为emif2reg,8051的扩展寄存器，用户可以自定义这些寄存器，在fp完成需要的功能。

地址			
0x8400~0x8684	mipi DSI Peripheral controller寄存器		
0x8804	8051的扩展寄存器	mipi_sel	0：对mipi1寄存器进行配置，1：对mipi2寄存器进行配置
0x8805		func_set	func_set[0]　control sel; 1= enable
0x8806		rd_id_cmd	Read id 命令cmd
0x8807		rd_id_param1	Read id 命令parameter1
0x8808		rd_id_param2	Read id 命令parameter2
0x8809		id_return_cmd	需要回复id命令cmd
0x880a		id_return_param1	需要回复id命令parameter1

0x880b		id_return_param2	需要回复id命令parameter2

- sram

  8051程序和数据存储空间。

- emif2apb

  将 8051 emif 接口转换为 apb 接口，对 mipi DSI Peripheral controller 控制和状态寄存器进行配置。

- pll

  时钟输入 83M，用户可以修改三路时钟输出频率。clk_periph 即 mipi rx hs mode byte clock，时钟频率必须高于 AP mipi 数据线 hs mode 速率的 1/8 倍；esc_clk 作为 mipi tx lp mode 的时钟；esc_clk_rx 作为 mipi rx lp mode 的时钟(必须高于 AP mipi lp mode 速率) 。

- mipi_rx

  将 mipi 信号解析为 packet interface，具体协议参考章节 4.4.2。

- rx_cmd_decode

  用户可以编辑 rx_cmd_decode 模块,接收 mipi rx 模块产生的 packet interface(peripheral_rx_raw_port)数据 ，当检测到读 ID 命令时，回复 ID 给 AP。

用户可以通过Keil软件编程，按照以下顺序对mipi DSI controller进行相应的设置。

1. 对mipi DSI controller进行复位(rstn_mipi=0)。
2. 对8051的扩展寄存器进行赋值

   rd_id_cmd=0x06;rd_id_param1=0x0a;rd_id_param2=0x00;id_return_cmd=0x21;

   id_return_param1=0x9e; id_return_param2=0x00;
3. 对mipi DSI Peripheral controller进行配置，mipi_sel=0（选择mipi1作为mipi rx）。

```
//8051 to rx mipi apb
U32 xdata periph_lanes        _at_ 0x8400;
U32 xdata periph_vc           _at_ 0x8404;
U32 xdata periph_vc_check     _at_ 0x8408;
U32 xdata periph_ecc_err      _at_ 0x840c;
U32 xdata periph_hrx          _at_ 0x8410;
U32 xdata periph_ltx          _at_ 0x8414;
U32 xdata periph_bta          _at_ 0x8418;
U32 xdata periph_crc_err      _at_ 0x841c;
U32 xdata periph_bta_err      _at_ 0x8420;
U32 xdata periph_dis_rlpdt    _at_ 0x8424;
U32 xdata periph_dis_eotp     _at_ 0x8428;
U32 xdata periph_clr_status   _at_ 0x842c;
U32 xdata periph_m_settle     _at_ 0x8680;
U32 xdata periph_mc_settle    _at_ 0x8684;
```

```
//8051 to rx mipi apb
 U32 peri_cfg_data[37]={
 0x3,    //periph_lanes
 0x0,    //cfg_vc
 0x0,    //dis_vc_check
 0x0,    //ecc error
 0x0,    //hrx timer
 0x0,    //ltx timer
 0x0,    //bta timer
 0x0,    //tx length err
 0x0,    //bta err
 0x0,    //dis_rlpdt crc
 0x1,    //periph_dis_eotp
 0x1,    //periph_clr_status
 0x3,    //periph_m_settle
 0x1     //periph_mc_settle
 };
```

4. 解除对 mipi DSI controller 的复位(rstn_mipi=1).