6. Low Power MAC Transmit Block.

6.1 Design Transmit MAC Block Diagram.

Design micro-architecture of Transmit MAC Path.

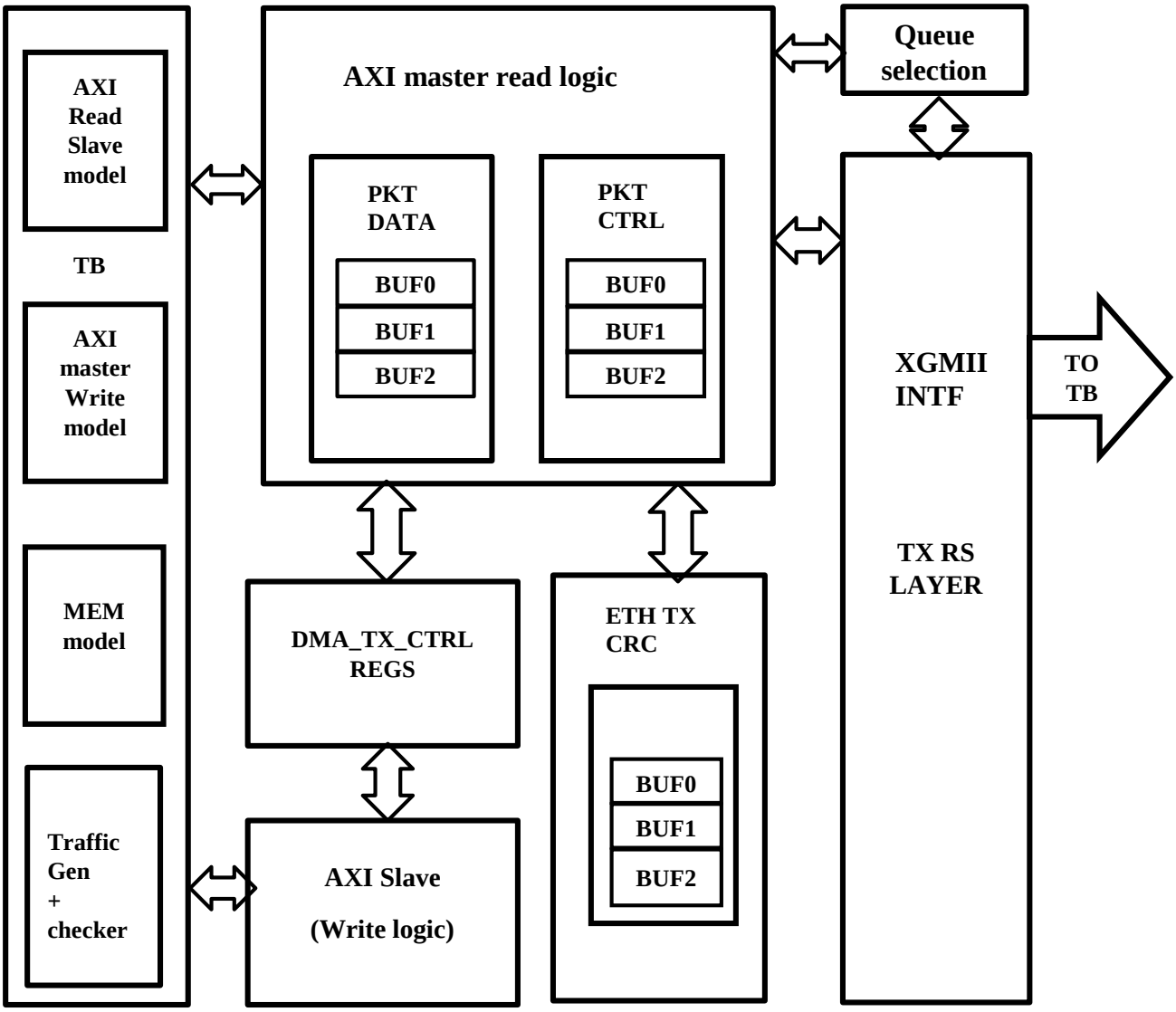


Figure 1.Transmit MAC Block Diagram

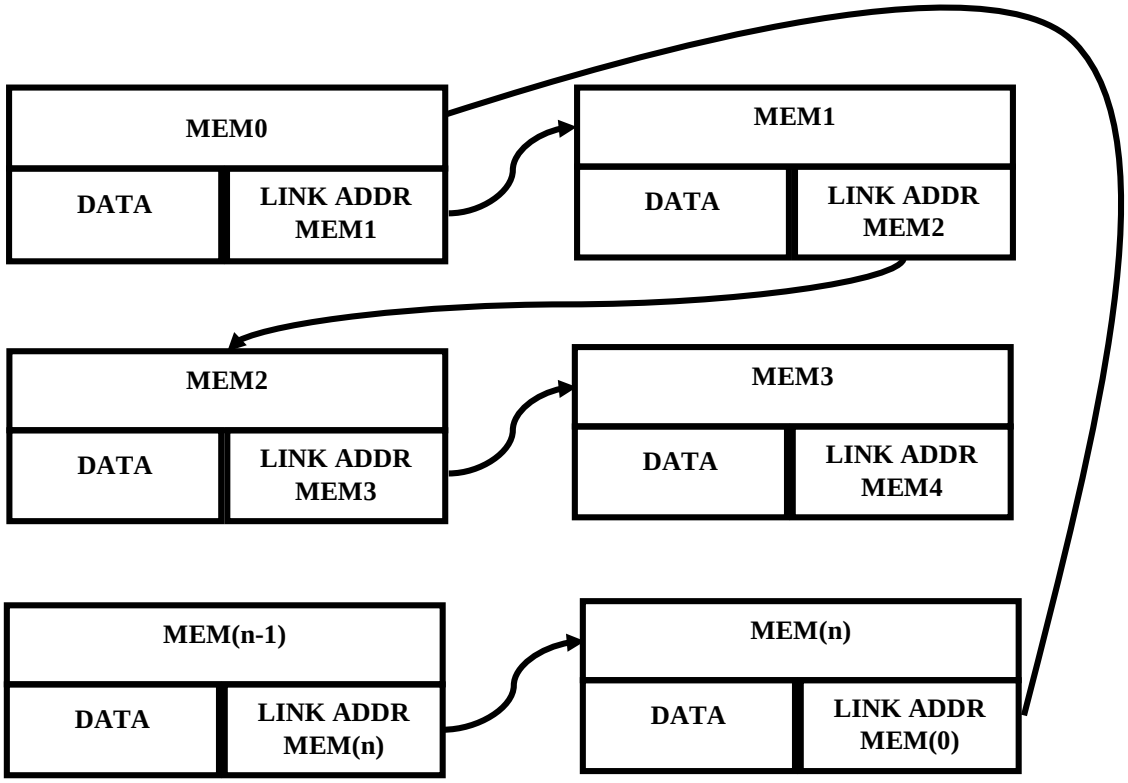
Figure shows the main design block diagram of TX Ethernet MAC block. All the main design sub modules as coded in RTL are listed below.

1. dma_controller_tx
2. AXI_slave
3. AXI_master (includes packet buffer and Ctrl header block)
4. Eth_tx_crc
5. Queue_selection
6. rs_layer
7. eth_tx (eth_tx.sv, wrapper containing all TX sub modules)
8. eth_core (top level file containing tx)

Before going over each submodule in detail, it is essential to understand the how Ethernet frame fragments are stored in memory

6.2 Ethernet frame structure in memory.

Frame segments are stored in memory in circular linked list fashion as shown below ---- the view of the memory map is shown in figure.



MEM_STRUCTURE OF FRAME/PKT FRAGMENT

Figure 2. Ethernet Frame Memory Storage

Last frame fragment's link address wraps around to point to first fragment's address. Thus, completing the circular link list. Data field in link structure is 64 bit wide and 'link addr' field is 64 bit wide although only 32 bits are used for addressing, ensuring a simpler design of AXI read master with 64-bit wide interface. Link address space is 32 bit wide and data field is 64 bits wide.

6.3 Functional flow of TX design

Testbench configures the TX block's DMA registers in dma_controller_tx module using AXI_slave module in RTL. Once the DMA registers are programmed by testbench, AXI_master modules initiates transfer of frame fragments from memory. As the frame fragments are transferred by AXI_master and stored into Pkt buffers, eth_crc_tx module calculates CRC for each frame and accumulates in internal CRC hold buffer. AXI_master also stores the control information of fragments into separate control buffer. AXI uses three channels for accumulating packets, with each channel having its own packet data, control buffers and CRC buffers. After CRC calculation, Queue_selection module selects one channel among the three available channels based on pre-programmed weights. RS_layer module begins transmission of Ethernet frames in XGMII interface for the winning channel as decided by queue_selection block.

6.4 Registers in TX MAC.

Table 1.TX MAC registers

Number	**REGISTER ADDRESS**	**REGISTER NAME**	**DESCRIPTION**
1	0xFFFF_0008	Clink_reg_0 (64 bits)	DMA frame address register 0.
2	0xFFFF_0010	Clink_reg_1 (64 bits)	DMA frame address register 1.
3	0xFFFF_0018	Clink_reg_2 (64 bits)	DMA frame address register 2.
4	0xFFFF_0020	Clink_reg_3 (64 bits)	DMA frame address register 3.
5	0xFFFF_0028	Clink_reg_4 (64 bits)	DMA frame address register 4.
6	0xFFFF_0030	Clink_reg_5 (64 bits)	DMA frame address register 5.
7	0xFFFF_0038	Clink_reg_6 (64 bits)	DMA frame address register 6.
8	0xFFFF_0040	Clink_reg_7 (64 bits)	DMA frame address register 7.
9	0xFFFF_0048	Clink_reg_ptr_8 (64 bits)	DMA frame address register 8.
10	0xFFFF_0050	Clink_reg_ptr_9 (64 bits)	DMA frame address register 9.
11	0xFFFF_0058	Clink_reg_ptr_10 (64 bits)	DMA frame address register 10.
12	0xFFFF_0060	Clink_reg_ptr_11 (64 bits)	DMA frame address register 11.
13	0xFFFF_0068	Clink_reg_ptr_12 (64 bits)	DMA frame address register 12.
14	0xFFFF_0070	Clink_reg_ptr_13 (64 bits)	DMA frame address register 13.
15	0xFFFF_0078	Clink_reg_ptr_14 (64 bits)	DMA frame address register 14.
16	0xFFFF_0080	Clink_reg_ptr_15 (64 bits)	DMA frame address register 15.

TX MAC contains 16 programmable Frame head pointer registers.

DUT as programmable frame header field registers. Frame header fields and control fields are to be programmed in those registers. Register is 64 bit wide. The generic structure of bit fields of register is given below

Table 2.Clink register

Clink register structure	
Ctrl field {Reserved+Last bytes vld+QoS}	**Frame Header address ptr MEM_ADDR0**
16bit+8bit +8 bit	32 bit

Register contains Control field and Frame Head Pointer Address field. Each of them is 32 bit wide. Head Pointer Address field needs to be programmed with first frame fragments address in memory, which in MEM0 (refer to memory map figure). Control field is further split into 8 bit QoS field, 8 bit Last byte valid and 16 bit of reserved space. Last byte Valid field contains 8 bits, with each bit representing the validity of data byte in a 64 bit data that is transferred from external memory by AXI master. Last byte valid holds significance only for last fragment of Ethernet frame transferred from memory by AXI master. QoS and reserved space are left for future use.

6.5 Interfaces in Tx MAC Block.

Table 3. AXI_clks interface

INTERFACE NAME				
AXI_clks				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_rtl	from_tb	
clk	1	input	output	Top level system clock.
rst	1	input	output	Top level active low system reset.

Table 4. AXI read address channel interface

INTERFACE NAME				
AXI_rd_addr_ch				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		Slave_if	Master_if	
ARID	4	input	output	AXI 4 read address channel identification number.
ARADDR	32	input	output	AXI 4 read address channel memory address field.
ARLEN	4	input	output	Burst size of AXI transaction.
ARSIZE	3	input	output	Number of bursts in AXI read transaction.
ARBURST	2	input	output	AXI Burst type and size information.
ARLOCK	2	input	output	Lock field not used in current design.

ARCACHE	4	input	output	Cache field not supported in design.
ARPORT	3	input	output	Protection is not supported in design.
ARQOS	1	input	output	Quality of service not supported in design.
ARREGION	1	input	output	Region identifier not supported in design.
ARUSER	1	input	output	User signal not used in design.
ARVALID	1	input	output	Read address valid in AXI read transaction.
ARREADY	1	output	input	Slave's ready signal to accept AXI read address.

Table 5.AXI read data channel interface

INTERFACE NAME				
AXI_rd_data_ch				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		master_if	slave_if	
RID	4	input	output	Read ID tag for transaction identification.
RDATA	64	input	output	Read data signal for transaction.
RRESP	2	input	output	Read response signal for transaction.
RLAST	1	input	output	Signal indication for last read transfer in transaction.
RUSER	1	input	output	Not used in current design.
RVALID	1	input	output	Read data valid indication for read transaction.
RREADY	1	output	input	Read ready to accept current transaction.

Table 6.AXI write data channel interface

INTERFACE NAME				
AXI_wr_data_ch				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		slave_if	master_if	
WID	4	input	output	Write ID tag used in write transaction to identify write channel.
WDATA	64	input	output	Write data for write transaction.
WLAST	1	input	output	Last transaction indication for write transaction.
WVALID	1	input	output	Write valid to indicate availability of write transaction signals.
WREADY	1	output	input	Write ready to indicate slave's acceptance of write transaction.

Table 7.AXI write address channel interface

INTERFACE NAME				
AXI_wr_addr_ch				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		Slave_if	Master_if	
AWID	4	input	output	Write address ID to indicate corresponding write address channel for AXI transaction.
AWADDR	32	input	output	Write address field to transfer memory write address for corresponding transfer.
AWLEN	4	input	output	Burst length that gives the exact of numbers bursts involved in current transfer.
AWSIZE	3	input	output	Gives information of size of each transfer in a burst.
AWBURST	2	input	output	Burst type associated with a transfer.
AWLOCK	2	input	output	Lock is not supported in design.
AWCACHE	4	input	output	Cache memory type not supported in design.
AWPROT	3	input	output	Protection type is not used in design.
AWVALID	1	input	output	Channel signals write valid for current transaction.
AWREADY	1	output	input	Write address valid indicates that slave is ready to accept signals for current transaction.

Table 8. AXI write response channel interface

INTERFACE NAME				
AXI_wr_resp_ch				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		master_if	slave_if	
BID	4	input	output	Response ID tag for write response.
BRESP	2	input	output	Response indicate field for write transaction.
BUSER	1	input	output	Not supported in current design.
BVALID	1	input	output	Write response valid to indicate response on channel is valid.
BREADY	1	output	input	Write ready indicates that the master can accept write response.

Table 9.XGMII transmit interface

INTERFACE NAME

tx_xgmii				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_tb	from_rtl	
TXC	4	input	output	XGMII transaction lane control bits. (4-lanes=4bits)
TXD	32	input	output	XGMII transaction byte data lanes (4-lanes= 32 bits)
TXCLK	1	input	output	Serial clock used for data transmission.

Table 10.FIFO Memory interface

INTERFACE NAME				
MEMIF				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_fifo	from_mem	
f0_waddr	4	input	output	FIFO memory write address.
f0_wdata	32	input	output	FIFO memory write data.
f0_write	1	input	output	FIFO memory write enable (1-write).
f0_raddr	4	input	output	FIFO memory read address.
f0_rdata	32	output	input	FIFO memory read data.

Table 11.CRC FIFO memory interface

INTERFACE NAME				
MEMIF_CRC				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_fifo	from_mem	
f0_waddr	4	input	output	CRC FIFO memory write address.
f0_wdata	32	input	output	CRC FIFO memory write data.
f0_write	1	input	output	CRC FIFO write enable.
f0_raddr	4	input	output	CRC FIFO read address.
f0_rdata	32	output	input	CRC FIFO read data.

Table 12. Packet data FIFO memory interface

INTERFACE NAME				
MEMIF_PKTD				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_fifo	from_mem	
f0_waddr	6	input	output	PKT DATA FIFO memory write address.
f0_wdata	64	input	output	PKT DATA FIFO memory write data.
f0_write	1	input	output	PKT DATA FIFO memory write enable.
f0_raddr	6	input	output	PKT DATA FIFO memory read address.
f0_rdata	64	output	input	PKT DATA FIFO memory read data.

Table 13. Packet control FIFO memory interface

INTERFACE NAME				
MEMIF_PKTC				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_fifo	from_mem	
f0_waddr	6	input	output	PKT CTRL FIFO memory write address.
f0_wdata	16	input	output	PKT CTRL FIFO memory write data.
f0_write	1	input	output	PKT CTRL FIFO memory write enable.
f0_raddr	6	input	output	PKT CTRL FIFO memory read address.
f0_rdata	16	output	input	PKT CTRL FIFO memory read data.

Table 14. AXI Slave write address FIFO memory interface

INTERFACE NAME				
MEMIF_SWCHADDR				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_fifo	from_mem	
f0_waddr	5	input	output	Slave write address channel memory FIFO write address.
f0_wdata	36	input	output	Slave write address channel memory FIFO write data.
f0_write	1	input	output	Slave write address channel memory FIFO write enable.
f0_raddr	5	input	output	Slave write address channel memory FIFO read address.
f0_rdata	36	output	input	Slave write address channel memory FIFO read data.

Table 15.AXI Slave write response FIFO memory interface

INTERFACE NAME				
MEMIF_SWCHRSP				
NETS	*BIT WIDTH*	*MODPORTS*		*SIGNAL DESCRIPTION*
		to_fifo	from_mem	
f0_waddr	5	input	output	Slave write response channel memory FIFO write address.
f0_wdata	6	input	output	Slave write response channel memory FIFO write data.
f0_write	1	input	output	Slave write response channel memory FIFO write enable.
f0_raddr	5	input	output	Slave write response channel memory FIFO read address.
f0_rdata	6	output	input	Slave write response channel memory FIFO read data.

6.6 DMA_controller TX_sub module.

Dma_contoller_tx module contains all the programmable registers in TX MAC core. Its contains stack of 16 Clink register. As the registers get programmed, AXI_master reads out values from these registers and starts initiating AXI read transactions to memory to fetch Ethernet frame from memory. This module essentially acts like a circular fifo of frame head pointer register. Software keeps polling for vacancy of register space and programs Clink register in dma_controller_tx though AXI slave as per systems need.

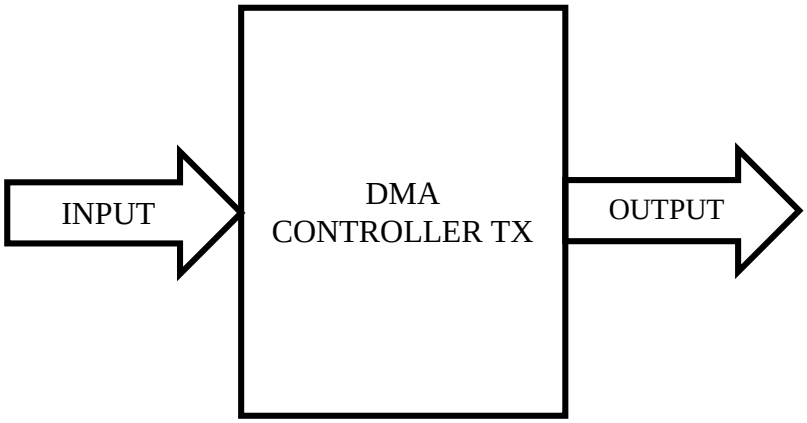


Figure 3. Dma_controller_tx block diagram.

Given below are inputs, outputs and interfaces to dma_controller_tx block.

Table 16. Dma_controller_tx Input Signals

INPUTS to dma_controller_tx			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
slave_addr	32	AXI_slave	Register address sent from axi slave for programming.
Slave_data	64	AXI_slave	Register data send from axi slave for register programming.
Wr_en	1	AXI_slave	Write enable for register programming from axi slave.
Rd_en	1	AXI_master	Read enable for register read which comes from axi master.
Rd_addr	32	AXI_master	Register address that is to be read which is sent from axi master.

Table 17. Dma_controller_tx output signals

OUTPUTS from dma_controller_tx			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
haddr	64	AXI_master	Frame header address output to AXI master from dma_controller_reg.
Slave_data	64	AXI_master	Register data of DMA registers.
Stack_empty	1	AXI_master	Empty flag indicating none of DMA registers are programmed.
Stack_full	1	AXI_master	Full flag indicating all the DMA registers are programmed.

Table 18. Dma_controller_tx block interfaces

INTERFACE to dma_controller_tx	DESCRIPTION
AXI_clks.to_rtl	Input interface to DMA controller.

6.7 AXI_slave.(write receive logic) sub module

AXI slave in TX MAC core is primarily responsible for configuration of registers in TX path. AXI write module in test bench programs the TX MAC core as per the requirement of the test case. Slave as write address channel and write data channel along with write response channel. AXI slave receives the corresponding data along with an address from test bench and programs it in TX MAC register. AXI Slave as three independent channels namely write address channel, read address channel and write response channel. All the three channels follow AXI3 specification with regard to implementation. Each of channel as an independent state machine running to implement the AXI3 protocol. AXI slave read channel receives address from AXI master model in testbench and stores in separate write address buffer. Similarly write data channel receives data using write data channel state machine. Once corresponding address and data are available in buffers, AXI slave programs the dma_controller_tx registers using given address and data information. If address received by AXI slave is a valid one, OKAY response is queued by slave in response channel for transmission to AXI master model in testbench. AXI response channel then transmits the corresponding response to read transaction via response channel to indicate the status of read transaction. Memories for all fifo buffers are placed outside the RTL to support efficient synthesis.

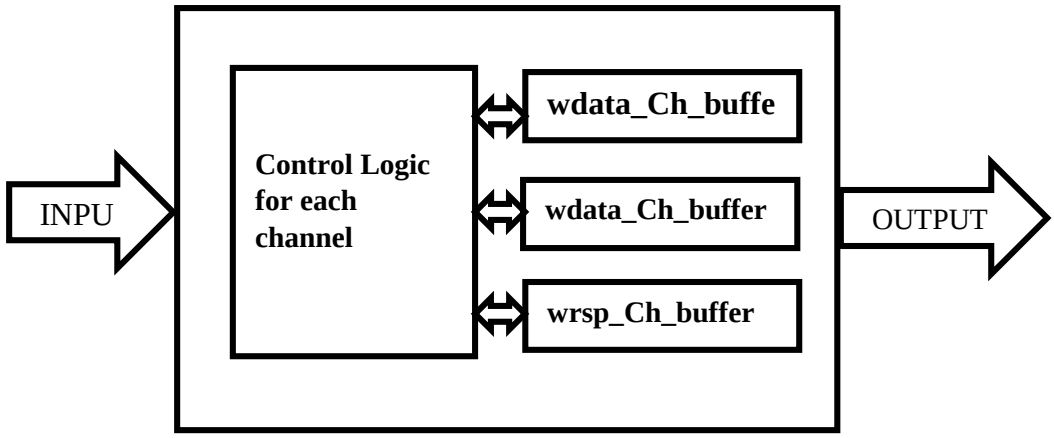


Figure 4. AXI Slave block diagram

Table 19. AXI_slave block output signals

OUTPUTS from AXI_slave			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
reg_write_addr_	32	Dma_controller_tx	DMA Register address sent from AXI slave to dma_controller_tx block for programming.
reg_write_data	64	Dma_controller_tx	DMA Register data sent from AXI slave to dma_controller_tx block.
Wr_en	1	Dma_controller_tx	DMA register write enable from AXI slave to dma_controller_tx block.

Table 20. AXI_slave block interfaces

INTERFACE to AXI_slave	INTERFACE DESCRIPTION
AXI_clks.to_rtl	Clk interface from top level
AXI_wr_addr_ch.slave_if	AXI write address channel slave interface
AXI_wr_data_ch.slave_if	AXI write data channel slave interface
AXI_wr_resp_ch.slave_if	AXI write response channel slave interface
MEMIF_SWCHADDR.from_fifo	Slave write address channel FIFO memory interface
MEMIF_SWCHDATA.from_fifo	Slave write address channel FIFO memory interface
MEMIF_SWCHRSP.from_fifo	Slave response channel FIFO memory interface

6.8 AXI master (read channel logic).

AXI master read module uses the head frame pointer address and initiates a read transaction to corresponding memory controllers AXI slave in Test bench. Once the dma_tx_controllers registers are programmed, AXI_master reads out the clink register from dma_tx_controller module and send it out via AXI read address channel to AXI slave model in test bench. AXI slave in testbench responds with burst of two read transactions over the AXI read data channel. During the first cycle of burst, 64 bit Ethernet frame fragment is sent and second read burst is used to send the 32 bit link address for next valid Ethernet frame fragment. Fragment data corresponding to read request is stored in the separate packet data buffer. Link address corresponding to read request is used to fetch next {data+linkaddr} information. The link address is stored in a 1-deep FIFO, until the time its served by channel selector. Frame transmission is complete when link address wraps around to head frame pointer address. AXI master also stores custom control word associated with start, middle and end of frame indicators to assist downstream blocks with their processing.

Due to the latency associated with fetching next read transaction from memory as a result of linked list structure. AXI master read as 3 read channels fetching 3 separate transmit frames from memory to improve system latency between two frames transmitted out of TX path. Three channel structure also improves AXI read master efficiency.

Figure below shows the AXI_master design block diagram in detail. AXI address and data channel logic implement AXI3 protocol with respective design logic to handle 3 independent address and data channels. AXI read burst is supported to reduce design power consumption. Channel selector logic is simple Round robin arbiter block. Rest of design consists of three category of buffers to move data in Ethernet TX MAC core. Channel Packet data buffer is used to store 64 bit data Ethernet data fragments. Channel control header buffer stores corresponding control information for each fragment that is stored in Packet data buffer.

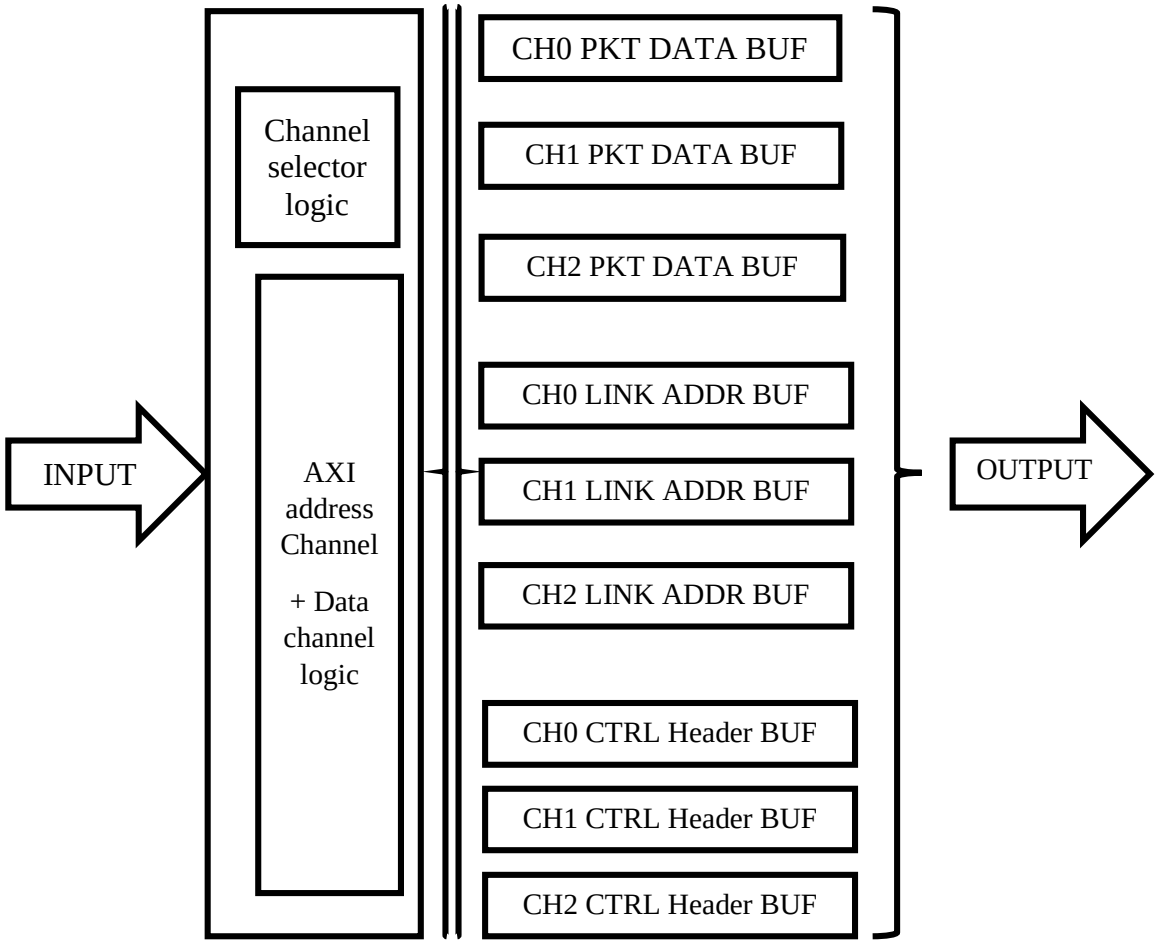


Figure 5. AXI_master design block diagram

When AXI master reads out the head address of new frame from DMA circular registers, channel selector allocates a free channel using RR arbitration among the available free channels. Once a channel gets allocated to the particular frame, it's bound to that particular frame till the entire frame is fetched from memory. All channels that are allocated for frame transmission, compete among the rest of channel to complete frame fetch from memory. All the 3 channels are given a fair share of bandwidth under enough system traffic. New frame register won't be read out of DMA registers table if all three channels are busy with previous frame transmission. Data fetched from each channel is stored in corresponding data channel buffers. Control headers associated with a particular frame is pushed into separate control buffers. Control header associated with each fragment is used by RS layer to enable smooth serial transmission of Ethernet frame. Link address fetched during transmission is used by the channel to complete rest of frame fetch. Three channel system **reduces inter-frame transmission latency**.

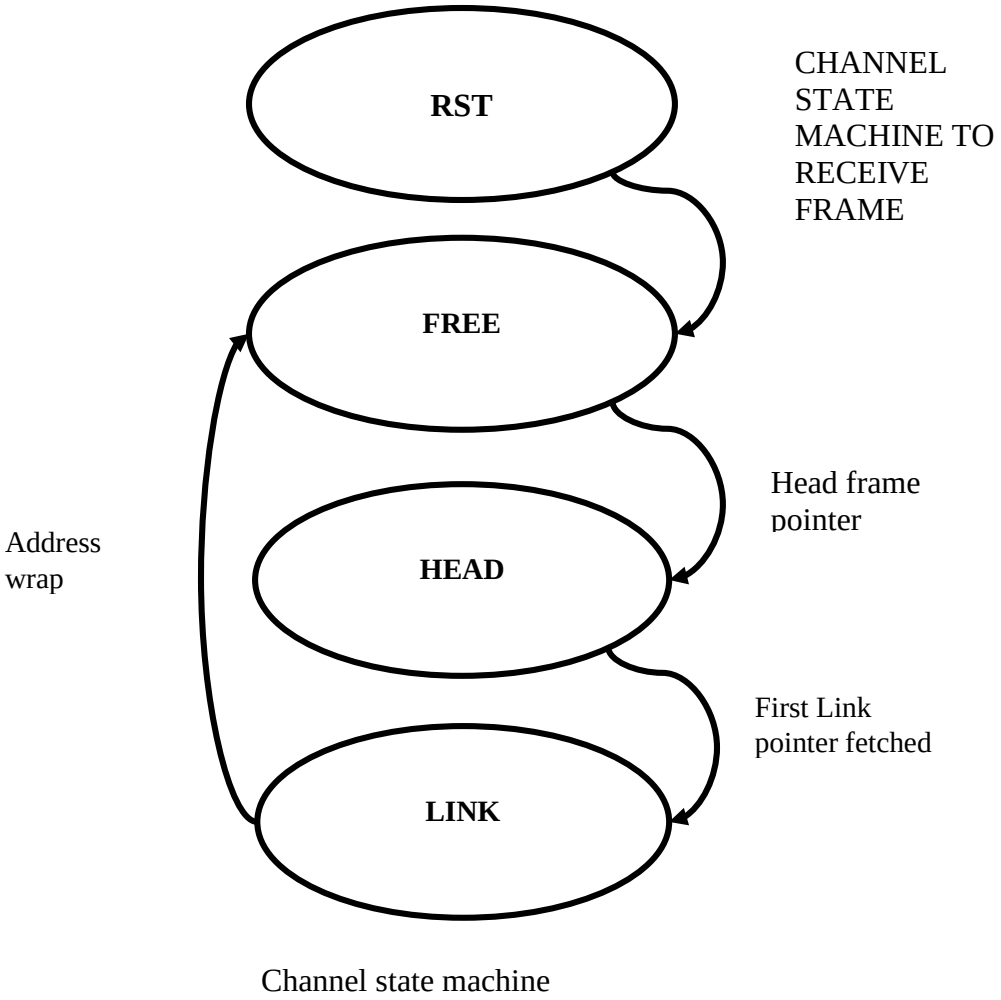


Figure 6. AXI_master Channel State machine

Above figure shows the channel state machine for 3 channels that operator in AXI_master for fetching Ethernet frame from memory. All the channels start in reset state and move to FREE state once reset is done. After the channel selector logic assigns a head pointer to selected channel, channel state machine moves into HEAD state, indicating head frame pointer state. Then, when link address is received by AXI master for that particular channel, channel state machines moves to LINK address part of frame transmission. Finally, when link address wrap to head pointer address, frame transmission is complete and corresponding channel is moved into FREE state, ready to be assigned for a new frame transmission by channel selector logic. Three such state machines exists for three separate channels in the system.

Table 21. AXI_master block inputs

INPUTS to AXI_master			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
Pfifo_pop_0	1	Rs_layer	Packet data FIFO instance 0 pop from transmit state machine.
Pfifo_pop_1	1	Rs_layer	Packet data FIFO instance 1 pop from transmit state machine.
Pfifo_pop_2	1	Rs_layer	Packet data FIFO instance 2 pop from transmit state machine.
haddr	64	Dma_controller_tx	Frame head pointer address from dma_controller_tx module.
Main_ptr_empty	1	Dam_controller_tx	Flag to indicate none of dma register are programmed.
Pcfifo_pop_0	1	Rs_layer	Packet control word FIFO instance 0 pop from transmit state machine.
Pcfifo_pop_1	1	Rs_layer	Packet control word FIFO instance 1 pop from transmit state machine.
Pcfifo_pop_2	1	Rs_layer	Packet control word FIFO instance 2 pop from transmit state machine.

Table 22. AXI_master block outputs

OUTPUTS from AXI_master			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
Pfifo_push_0	1	Eth_tx_crc	Packet data FIFO instance 0 push to eth_tx_crc block for enabling CRC calculation.
Pfifo_push_1	1	Eth_tx_crc	Packet data FIFO instance 1 push to eth_tx_crc block for enabling CRC calculation.
Pfifo_push_2	1	Eth_tx_crc	Packet data FIFO instance 2 push to eth_tx_crc block for enabling CRC calculation.
Pfifo_datain_0	64	Eth_tx_crc	Packet data FIFO instance 0 datain input to eth_tx_crc block for parallel CRC calculation.
Pfifo_datain_1	64	Eth_tx_crc	Packet data FIFO instance 1 datain input to eth_tx_crc block for parallel CRC calculation.
Pfifo_datain_2	64	Eth_tx_crc	Packet data FIFO instance 2 datain input to eth_tx_crc block for parallel CRC calculation.
Pcfifo_datain_0	16	Eth_tx_crc	Packet control FIFO instance 0 datain input to eth_tx_crc block for parallel CRC calculation.
Pcfifo_datain_1	16	Eth_tx_crc	Packet control FIFO instance 1 datain input to eth_tx_crc block for parallel CRC calculation.

Pcfifo_datain_2	16	Eth_tx_crc	Packet control FIFO instance 2 datain input to eth_tx_crc block for parallel CRC calculation.
Pfifo_dataout_0	64	Rs_layer	Packet data FIFO instance 0 datain input to rs_layer block transmit state machine.
Pfifo_dataout_1	64	Rs_layer	Packet data FIFO instance 1 datain input to rs_layer block transmit state machine.
Pfifo_dataout_2	64	Rs_layer	Packet data FIFO instance 2 datain input to rs_layer block transmit state machine.
Pcfifo_dataout_0	16	Rs_layer	Packet control FIFO instance 0 datain input to rs_layer block transmit state machine.
Pcfifo_dataout_1	16	Rs_layer	Packet control FIFO instance 1 datain input to rs_layer block transmit state machine.
Pcfifo_dataout_2	16	Rs_layer	Packet control FIFO instance 2 datain input to rs_layer block transmit state machine.

Table 23. AXI_master block interfaces

INTERFACE to AXI_master	SIGNAL DESCRIPTION
AXI_clks.to_rtl	Global Clk and reset interface
AXI_rd_addr_ch.master_if	AXI read address channel master interface
AXI_rd_data_ch.master_if	AXI read data channel master interface
MEMIF_PKTD.from_fifo (3 instances)	Packet data FIFO memory interface
MEMIF_PKTC.from_fifo (3 instances)	Packet control FIFO memory interface

6.9 ETH_TX_CRC sub module:

As data is pushed into data frame buffer, a corresponding parallel pipeline is used to compute CRC. CRC_block module (figure 16) is a store and accumulate unit. CRC is computed for each frame fragment in a frame until the end of the frame. CRC is stored in a separate CRC buffer (refer figure 15) after CRC is computed for the entire frame. Three such CRC pipelines are part of the design to handle corresponding three read data channels of AXI master module.(figure 15)

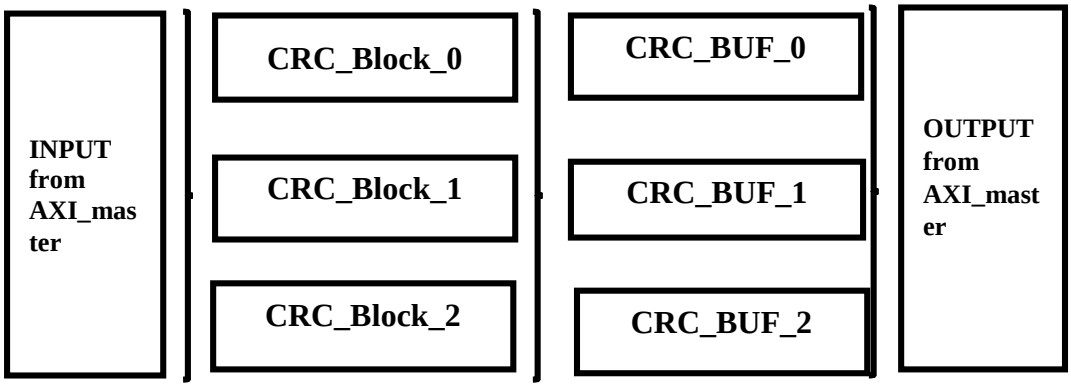


Figure 7. ETH_TX_CRC design block diagram

Eth_tx_crc (figure 15) is the top level CRC compute block for transmit MAC Core. It contains three CRC_block and CRC buffer instances to compute and accumulate CRC for Ethernet frames. Computed CRC for frames are pushed into respective CRC buffer modules after CRC computation. Figure 16 below shows the CRC_block design block diagram. Frame data and control word are simultaneously pushed into CRC_block when they stored in their respective buffers by AXI_master module (figure 13). CRC_block then uses control word information along with fragment data to compute the CRC for the frame. Once the end of frame is encountered by CRC_block, it finishes the CRC computation and stores the CRC in corresponding CRC buffer. Its takes 3 extra cycles of latency to compute CRC from the time data and control are pushed into eth_tx_crc module. So if entire frame is available in say nth cycle in packet data buffer, then CRC for that frame will be available in n+3 clock cycle. Buffer occupancy in CRC buffers activates the downstream blocks to enable successful completion of Ethernet frame out of XGMII interface.

CRC block's design block diagram is in the figure below. CRC block module contains various CRC combinational blocks depending on data width of input data fragment for which CRC is calculated. CRC8 modules computes CRC for data with of one byte.

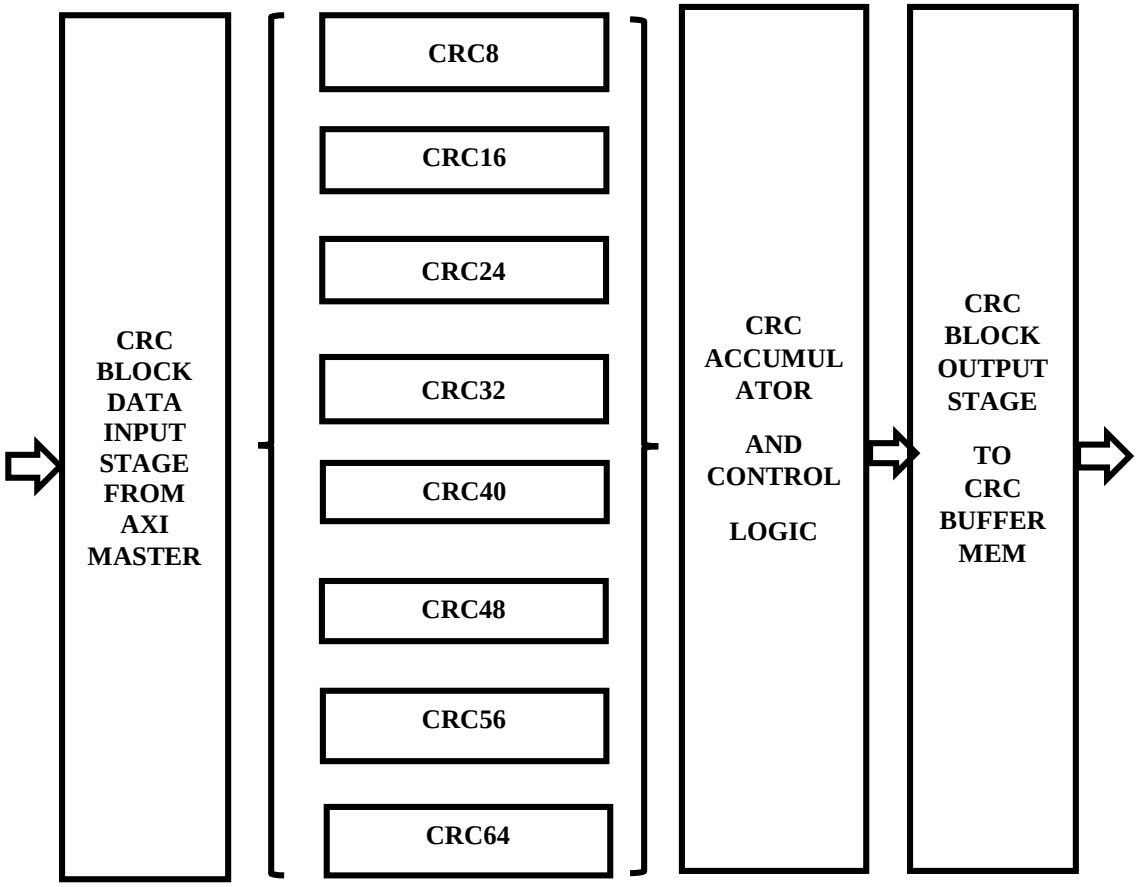


Figure 8. CRC_block design block diagram

Similarly, CRC16 computes CRC for 16 bit data and so on. Control word contain Last Byte Valid field which is used by CRC_block module to calculate the final fragments CRC before finishing the CRC calculation for that particular frame.

Table 24. ETH_TX_CRC block inputs

INPUTS to eth_tx_crc			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
Dready_pkt0	1	AXI_master	Packet ready indication to CRC block instance 0 from data packet FIFO instance 0.
Dready_pkt1	1	AXI_master	Packet ready indication to CRC block instance 1. from data packet FIFO instance 1.
Dready_pkt2	1	AXI_master	Packet ready indication to CRC block instance 2 from data packet FIFO instance 2.
Datain_pkt0	64	AXI_master	Packet datain to CRC block instance 0 from data packet FIFO instance 0.
Datain_pkt1	64	AXI_master	Packet datain to CRC block instance 1 from data packet FIFO instance 1.
Datain_pkt2	64	AXI_master	Packet datain to CRC block instance 2 from data packet FIFO instance 2.
Ctrl_wd_pkt0	8	AXI_master	Packet control word to CRC block instance 0 from control packet FIFO instance 0.
Ctrl_wd_pkt1	8	AXI_master	Packet control word to CRC block instance 1 from control packet FIFO instance 1.
Ctrl_wd_pkt2	8	AXI_master	Packet control word to CRC block instance 2 from control packet FIFO instance 2.
Bvalid_pkt0	8	AXI_master	Byte valid to CRC block instance 0 from control packet FIFO instance 0.
Bvalid_pkt1	8	AXI_master	Byte valid to CRC block instance 1 from control packet FIFO instance 1.
Bvalid_pkt2	8	AXI_master	Byte valid to CRC block instance 2 from control packet FIFO instance 2.
Crcfifo0_pull	1	Queue_ selection	CRC fifo instance 0 pull request from queue selection block.
Crcfifo1_pull	1	Queue_ selection	CRC fifo instance 1 pull request from queue selection block.
Crcfifo2_pull	1	Queue_ selection	CRC fifo instance 2 pull request from queue selection block.

Table 25.ETH_TX_CRC block outputs

OUTPUTS from eth_tx_crc			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION

Crcfifo0_empty	1	Queue_selection	Empty indication from CRC FIFO instance 0.
Crcfifo1_empty	1	Queue_selection	Empty indication from CRC FIFO instance 1.
Crcfifo2_empty	1	Queue_selection	Empty indication from CRC FIFO instance 2.
Crcfifo0_dataout	32	Queue_selection	CRC out from CRC FIFO instance 0.
Crcfifo1_dataout	32	Queue_selection	CRC out from CRC FIFO instance 1.
Crcfifo2_dataout	32	Queue_selection	CRC out from CRC FIFO instance 2.

Table 26.ETH_TX_CRC block interfaces

INTERFACE to eth_tx_crc	**INTERFACE DESCRIPTION**
AXI_clks.to_rtl	Global clock and active low reset interface.
MEMIF_CRC.from_fifo (3 instances)	CRC memory FIFO instance.

Table 27. CRC_block block inputs

INPUTS to CRC_block			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
dready	1	Eth_tx_crc	Data ready from corresponding packet FIFO instance
datain	64	Eth_tx_crc	Datain from corresponding packet FIFO instance
Ctrl_wd	8	Eth_tx_crc	Control word from corresponding packet FIFO instance
bvalid	8	Eth_tx_crc	Byte valid from corresponding packet FIFO instance

Table 28. CRC_block block output

OUTPUTS from CRC_block			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
Crc_vld_2d	1	Eth_tx_crc	CRC valid indication from corresponding CRC block instance
Crc_out	32	Eth_tx_crc	CRC data out from corresponding CRC block instance

Table 29. CRC_block block interfaces

INTERFACE to CRC_block	**INTERFACE DESCRIPTION**
AXI_clks.to_rtl	Global clock and active low reset input interface

Table 30.CRC combinational block inputs

INPUTS to CRC32_DN (N=data width)			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
data	N	CRC_block	Packet data input to combinational CRC32_DN blocks

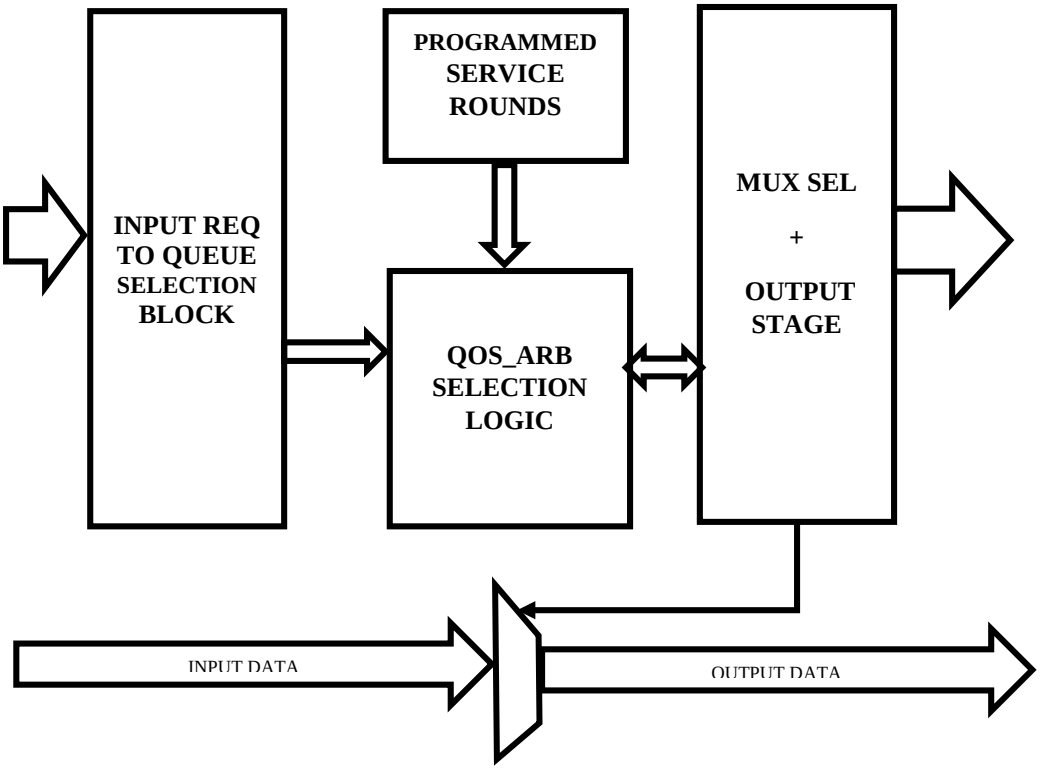
crc	32	CRC_block	CRC data input to combinational CRC32_DN blocks

Table 31. CRC combinational block outputs

OUTPUTS from CRC32_DN(N=data width)			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
Crc_nxt	32	Crc_nxt	CRC output from combinational CRC32_DN blocks

6.10 Queue_selection sub module.

Weighted priority arbiter is going to arbitrate the output CRC queues based on pre-programmed weights for respective queues. Selected CRC queue with corresponding packet data and ctrl header is pushed out into RS transmit module. CRC buffer occupancy acts as a request for the arbiter. Figure below shows the queue_selection design block diagram.



QUEUE SELECTION BLOCK DIAGRAM

Figure 9. Queue selection design block diagram

6.11 QOS_ARB sub block

QOS_ARB block consist of programmed weights/service round registers, QOS_arb module with priority encoders and output select mux for winning queue. CRC buffer occupancy acts as input requests for QOS_arb block arbiter. System as 3 queues, hence 3 bit request vector is the input to queue selection and QoS_arb block. Then the arbiter as to select a queue based on received request respecting the programmed queue. Queue selection logic consists of counters keeping track of serviced grants for a particular arbitration round. Once the programmed number of grants are serviced for a given arbitration round, counters are reset. Based on serviced count for different counter, winning counter is determined. Winning counter is determined on following rules. Rule1: If the highest priority queue's counter value does not exceed its programmed weight, then that particular counter wins the arbitration. In case it if had exhausted its quota for current round, queue with next high priority is selected and process is repeated so on. Winning counter is associated with a particular priority encoder, which decides the selection order of queue to be granted for that particular arbitration cycle. Each arbitration process takes one clock cycle to finish. Each arbitration round is decided by sum of programmed counter weights. Default programmed weights are 4:2:1 for crcfifo0, crcfifo1 and crcfifo2 respectively. Default weights was used in system level test scenario in this project. Queue_selection block is also responsible for sending start_transmit signal to rs_layer module to enable start of transmission in TX MAC Core.

Table 32. Queue_selection block inputs

INPUTS to queue_selection			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
arb_nxt	1	Rs_layer	Next arbitration cycle indication from rs layer transmit state machine
Pfifo_datain0	64	AXI_master	Packet data FIFO instance 0 datain signal
Pfifo_datain1	64	AXI_master	Packet data FIFO instance 1 datain signal
Pfifo_datain2	64	AXI_master	Packet data FIFO instance 2 datain signal
Crcfifo0_dataout	32	Eth_tx_crc	CRC data FIFO instance 0 dataout signal
Crcfifo1_dataout	32	Eth_tx_crc	CRC data FIFO instance 1 dataout signal
Crcfifo2_dataout	32	Eth_tx_crc	CRC data FIFO instance 2 dataout signal
Pfifo_datain_ctrl_0	16	AXI_master	Packet control FIFO instance 0 datain signal
Pfifo_datain_ctrl_1	16	AXI_master	Packet control FIFO instance 1 datain signal
Pfifo_datain_ctrl_2	16	AXI_master	Packet control FIFO instance 2 datain signal
Crcfifo0_empty	1	Eth_tx_crc	CRC FIFO instance 0 empty indication
Crcfifo1_empty	1	Eth_tx_crc	CRC FIFO instance 1 empty indication
Crcfifo2_empty	1	Eth_tx_crc	CRC FIFO instance 2 empty indication
Pfifo_pop	1	Rs_layer	Packet data FIFO pop indication from rs layer transmit state machine

Crcfifo_pop	1	Rs_layer	CRC data FIFO pop indication from rs layer transmit state machine

Table 33. Queue_selection block outputs

OUTPUTS from queue_selection			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
Crcfifo0_pull	1	Eth_tx_crc	CRC data FIFO instance 0 pull indication from queue selection block
Crcfifo1_pull	1	Eth_tx_crc	CRC data FIFO instance 1 pull indication from queue selection block
Crcfifo2_pull	1	Eth_tx_crc	CRC data FIFO instance 2 pull indication from queue selection block
Pfifo_pop_0	1	AXI_master	Packet data FIFO instance 0 pop indication from queue selection block
Pfifo_pop_1	1	AXI_master	Packet data FIFO instance 1 pop indication from queue selection block
Pfifo_pop_2	1	AXI_master	Packet data FIFO instance 2 pop indication from queue selection block
Pcfifo_pop_0	1	AXI_master	Packet control FIFO instance 0 pop indication from queue selection block
Pcfifo_pop_1	1	AXI_master	Packet control FIFO instance 1 pop indication from queue selection block
Pcfifo_pop_2	1	AXI_master	Packet control FIFO instance 2 pop indication from queue selection block
Pfifo_datain	64	Rs_layer	Multiplexed output of selected packet data FIFO's data by queue selection block
Pfifo_datain_ctrl	16	Rs_layer	Multiplexed output of selected packet control FIFO's data by queue selection block
Crcfifo_dataout	32	Rs_layer	Multiplexed output of selected CRC FIFO's data by queue selection block
Start_transmit	1	Rs_layer	Start transmit indication from queue selection block to rs layer transmit state machine

Table 34. Queue_selection block interfaces

INTERFACE to queue_selection	
AXI_clks.to_rtl	Global clocks and active low reset interface

Table 35. QOS_ARB block inputs

INPUTS to QOS_ARB			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION

arb_nxt	1	Rs_layer	Next arbitration cycle indication from rs layer transmit state machine
req	3	Queue_selection	Request signal to arbiter

Table 36. QOS_ARB block outputs

OUTPUTS from QOS_ARB			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
gnt	3	Queue_selection	Granted queue output signal

Table 37. QOS_ARB block interfaces

INTERFACE to QOS_ARB	**INTERFACE DESCRIPTION**
AXI_clks.to_rtl	Global clock and active low reset input interface

6.12 RS transmit sub module: reconciliation layer transmit side

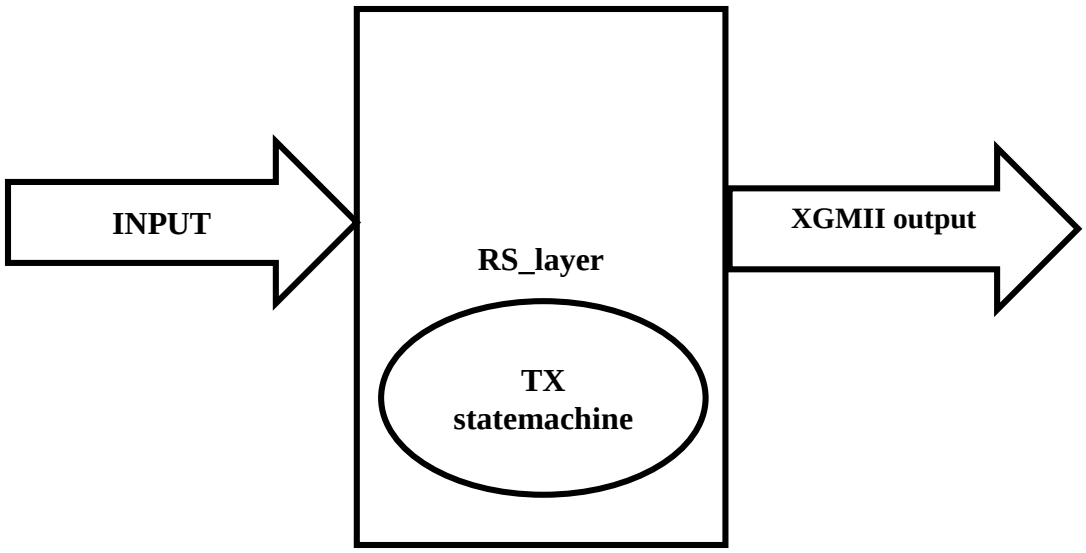


Figure 10. RS layer design block diagram

RS_layer module contains a transmit state machine to streamline the transmission of 10G Ethernet frames using XGMII interface. Data is transferred as stream of bits across XGMII interface as per IEEE 802.03-ae2012 specification. XGMII interface contains 32 data stream bits and 4 control lane bits (1-bit control word for each byte of data lane).

Table 38. XGMII lane transmission

XGMII LANE based transmission (32 bit data line)			
LANE0	LANE1	LANE2	LANE3
TXD[0:7]	TXD[8:15]	TXD[16:23]	TXD[24:31]
TXC[0]	TXC[1]	TXC[2]	TXC[3]

RS_layer gets start_transmit indication, granted queue's fragment data and control information from queue_selection block. Transmit state machine starts the transmission process once the start_transmit is asserted.

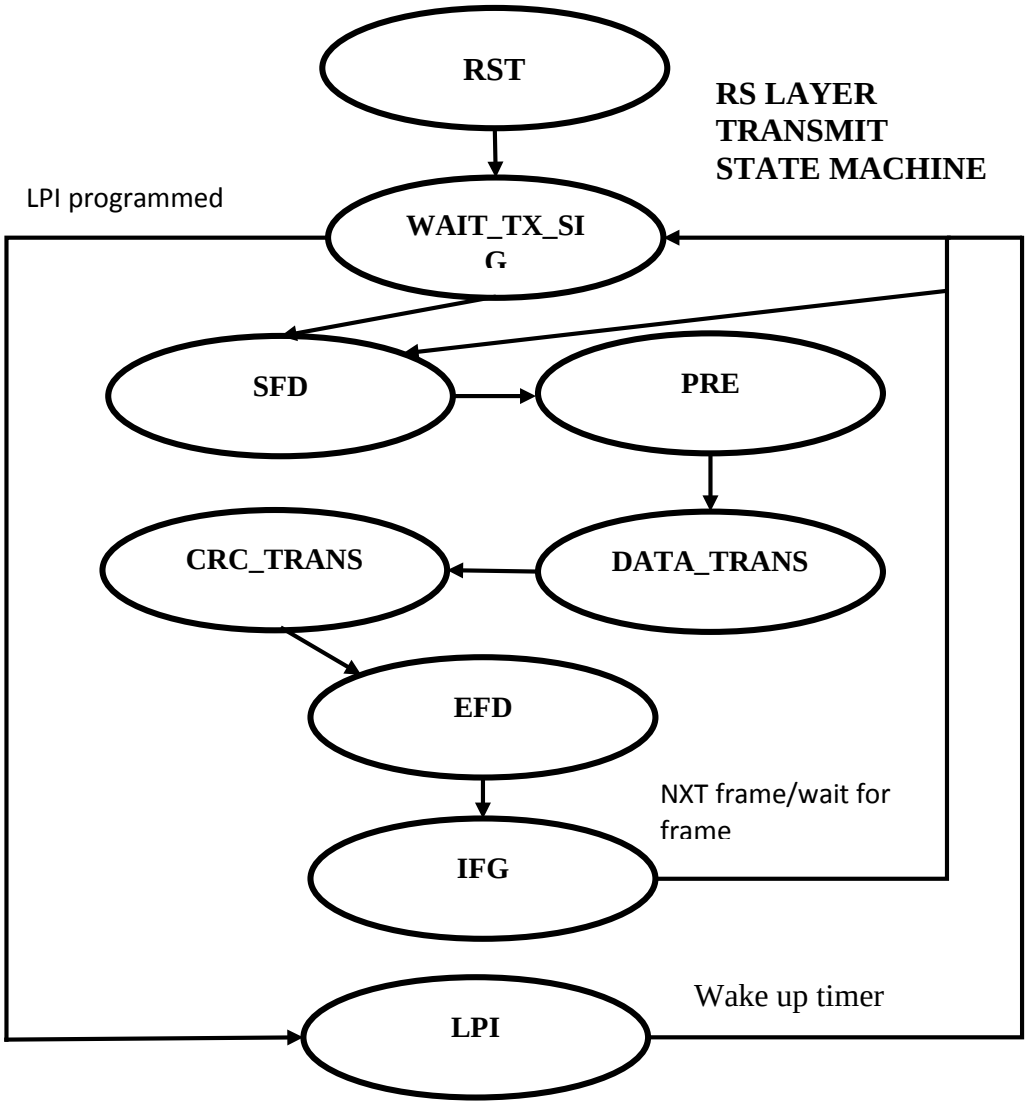


Figure 11. RS transmit state machine

Transmit state machine starts on RST state and moves into WAIT_TX_SIG state after reset is done. On receiving start_transmit from queue_selection block, rs_layer module transitions into SFD (start of frame delimiter) state, transmitting “FB” – start of frame delimiter byte in XGMII data stream. Rs_layer also stores the 64 bit Ethernet frame fragment data into hold barrel shifter. After transmitting SFD, state machine moves into PRE permeable state to transmit preamble data control stream. State machine then transitions to DATA_TRANS state, transmitting data through XGMII stream till the stop control word is encountered in fragment control word stored in packet control buffer. Various buffers are read out in accordance to data requirement in transmit state machine. Corresponding buffer control signals are sent to AXI_master block from rs_layer block. After finishing the data fragment segments, transmit state machine sends out the CRC of corresponding Ethernet frame in CRC_TRANS state. End of frame delimiter is transmitted on completion of frame CRC transmission in EFD state. Transmit state machine enters IFG(Inter frame gap) state after EFD transmission. IDC(Idle deficit count) algorithm was used to implement the inter frame gap between Ethernet frames, which will be explained in a later section. Inter frame gap basically sends idles data bytes (8’h07) in lanes with control bit set for corresponding lane in TXC. State machine transits to WAIT_TX_SIG or SFD based on the availability of next frame. If next frame is available immediately then transmit state machine moves to SFD and starts transmitting the next frame. Transmit state machine moves to WAIT_TX_SIG if no frames are available for transmission. In WAIT_TX_SIG state, transmit state machine keeps transmitting idles data control stream in XGMII data stream. RTL also transmits TX_CLK signal, which is the serial clock used by PHY layer to transmit the serial stream across physical medium to receiver side. Dual data rate transmission occurs in XGMII interface, as in data gets transmitted along both the edges of clock. For simplicity TX_CLK is generated in RTL while in a real system it will be generated by system PLL. TX_CLK is exactly half the AXI_clk in the system. AXI_clk runs at 312.5 Mhz, hence TX_CLK will run at 156.25 Mhz. XGMII interface will meet the data rate of 10Gbps at TX_CLK frequency of 156.25 Mhz.

System supports a special LPI (Low power idling) state according to 802.3 specification. In low power idling, low power data control stream is sent for 128 clock cycles and then the transmit clock is shut off till the system receives wakeup signal. Switching off of serial clock results in huge power saving as downstream PHY layers are completely shutoff. Figure below shows the transmit LPI start machine. LPI state starts in reset state and comes out of reset to move into WAIT_LPI state once reset is de-asserted. On indication of LPI from system which occurs due to non occupancy of CRC buffers or software programmable register, LPI state machine transitions into GCLK state after 128 clock cycle of low power (8’h06) indication to receive side. In GCLK state, serial clock to XGMII is gated. For the system to move out of GCLK state it takes another 128 clock cycles of wakeuptime before the system moves into WAIT_LPI (which is out of LPI).

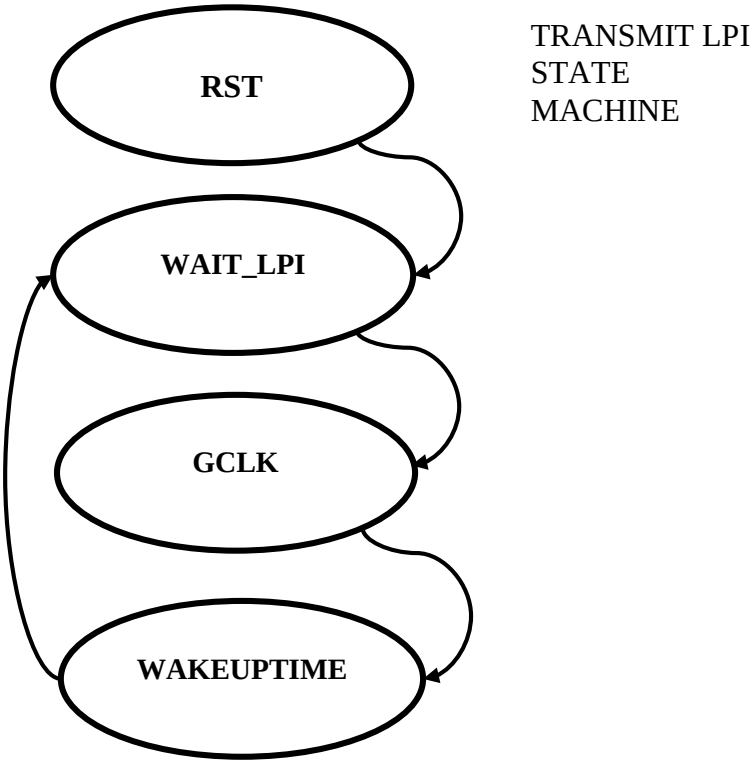


Figure 12. LPI state machine

Table 39. RS_layer block inputs

INPUTS to rs_layer			
NAME	BIT WIDTH	SOURCE BLOCK	SIGNAL DESCRIPTION
Start_transmit	1	Queue_selection	Start of transmission indication to transmit state machine from queue_selection module
lpi	1	Eth_tx	Low power idling set signal from eth_tx
Pkt_data	64	Queue_selection	Selected packet data FIFO's data from queue_selection module
Pkt_ctrl	8	Queue_selection	Selected packet control FIFO's data from queue_selection module
Pkt_crc	32	Queue_selection	Selected CRC FIFO's data from queue_selection module
bvalidin	8	Queue_selection	Selected byte valid data from queue_selection module

Pkt_empty	1	Queue_selection	All FIFO's empty indication from queue_selection module

Table 40. RS_layer block outputs

OUTPUTS from rs_layer			
NAME	BIT WIDTH	TARGET BLOCK	SIGNAL DESCRIPTION
Pop_pkt	1	Queue_selection	Pop indication to data packet FIFO after the transmission of packet data.
Pop_crc	1	Queue_selection	Pop indication to CRC FIFO after the transmission of packet data.
Nxt_buf	1	Queue_selection	Next arbitration indication to queue_selection module after the transmission of current frame.

Table 41. RS_layer block interfaces

INTERFACE to rs_layer	**INTERFACE DESCRIPTION**
AXI_clks.to_rtl	Global clock and active low reset interface
tx_xgmii.from_rtl	TX XGMII interface output interface

6.13 Idle deficit count (IDC) Algorithm.

Idle deficit counter algorithm improves transmission efficiency of serial Ethernet MAC. In traditional 10G Ethernet frame transmission requires 96bit / 8 octets of idle transmission to give leeway to physical layer for electrical reasons.

- Basic strategy is to vary number of idles from 9 to 15 during frame transmission.
- IDC algorithm sends idles (IPG in table given below) in accordance with current frame length boundary (i.e finish lane of current frame transmission) and previous history of deficit counter.
- IDC uses deficit counter to maintain history of idles inserted and deleted during transmission. At the end of each frame transmission, deficit counter is updated.
- Algorithm relies on averaging number of idle transmitted over a window of time. Due to variation in frame size of Ethernet frames over a large window of time, an average of 12 octets of idles (96 bits) are transmitted during that window of time.

Table 42.Idle count table according to IDC algorithm

IDC idle count logic table.								
FRM_LEN% 4 (bytes)	IPG (Idle count)	IDC=0 (prev transmission)	IPG (Idle count)	IDC=1 (prev transmission)	IPG (Idle count)	IDC=2 (prev transmission)	IPG (Idle count)	IDC =3 (prev transmission)

0	12	0	12	1	12	2	12	3
1	11	1	11	2	11	3	15	0
2	10	2	10	3	14	0	14	1
3	9	3	13	0	13	1	13	2

FRM_LEN or lane boundary is finish lane of current frame under transmission.
IDC Prev- Idle deficit counter which hold the number of idles that were not transmitted for previous frame.
IPG – Interframe gap (number of idles that is to be inserted for current frame under transmission).
IDC value inside the table is updated IDC value once the current frame transmission is over.

For example, if MAC starts with the IDC of 0 and receives a frame which finishes on lane 2(FRM_LEN%4 =2). Then, number of idles for current transmission is 10 and IDC is updated to 2. In essence, IDC algorithm decrements 2 idles from current frame's IPG and keeps history of it in idle deficit counter.
IDC=2 after first frame.
For calculating next frames IPG, IDC algorithm uses previous IDC count value and new frame's FRM_LEN%4 value.
Say, new frame FRM_LEN%4 = 3, then updated IDC=1 and IPG=13 (refer table).
Correspondingly, various IPG values are listed in table for different IDC and (FRM_LEN%4) values.

6.14 Ethernet TX Testbench architecture:

- The testbench is entirely coded in System Verilog.
- Entire Ethernet frame is generated in a constrained random environment with the ability to control all the header and control parameters of Ethernet frame.
- Constructed Ethernet frame is split into chunks of 64-bit fragments and stored in randomly allocated address from 32-bit address space. Circular linked list structure is created by test bench and stored in associative array structures, mimicking the free pool memory model.
- Ethernet frame's CRC is calculated and stored in associative array in test bench to enable frame check with received frame from DUT.
- Once the circular link list is built in memory, head pointer + various control headers associated with generated Ethernet frame is queued to be programmed through test bench AXI write master model.
- As and when Test bench's AXI slave read model receives the read request from DUT AXI read master, slave responds with corresponding data + link address information over two read burst cycles. Test bench also ensures that corresponding address location associated with the word is freed up after AXI master read from DUT.
- A state machine in test bench checks for protocol violation of received data XGMII interface and accumulates the data if all the control code matched the expected specification. Once the entire frame is pushed out by DUT, it is checked with the expected frame in test bench.

6.15 RTL simulation results.

RTL simulations below are shown for one particular packet from the time its generated in testbench and programmed into clink registers in dma_controller_tx module to time it exits from the system at XGMII interface.

6.15.1 Frame generated in testbench.

Raw Ethernet Frame generated from testbench.

```
-=-=-=-=-=-=-=-=-=-=RAW PKT-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
9b9bb6077db0f04990134d03af1f670311f4f9976aa61f89498312638ea5a1586034dd09afa78
36f5764cdf46a679122dd1f5ce64ba380d1f8b1c5035767e2419cce0672a0b3df5cb56d1f4840
e8
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Raw Ethernet Frame generated from testbench after CRC calculation.

```
-=-=-=-=-=-=-=-=-=-=STR RAW PKT CRC-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
9b9bb6077db0f04990134d03af1f670311f4f9976aa61f89498312638ea5a1586034dd09afa78
36f5764cdf46a679122dd1f5ce64ba380d1f8b1c5035767e2419cce0672a0b3df5cb56d1f4840
e88c40f9bc
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Ethernet frame displayed in a user friendly manner.

```
=-=-=-=-=-=-=-=-=-=-=ETHERNET FRAME=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

DA:9b::9b::b6::7::7d::b0

SA:f0::49::90::13::4d::3

ETHER_TYPE:af1f

PAYLOAD:
67:3:11:f4:f9:97:6a:a6:1f:89:49:83:12:63:8e:a5:a1:58:60:34:dd:9:af:a7:83:6f:5
7:64:cd:f4:6a:67:91:22:dd:1f:5c:e6:4b:a3:80:d1:f8:b1:c5:3:57:67:e2:41:9c:ce:6
:72:a0:b3:df:5c:b5:6d:1f:48:40:e8:

CRC:8c40f9bc

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Ethernet Frame is then stored in a memory model to enable transmission to DUT. Frame is stored in a circular linked list manner in memory.

```
-=-=-=-=-=-=-=-=-=-=MEM DISP-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
mem_location: ff0aa010 :: data_stored: 9b9bb6077db0f049 (first 8 bytes data)

mem_location: ff0aa018 :: data_stored: 00000000ffb5d240 (next8bytes linkaddr)
```

```
mem_location: ff1eb490 :: data_stored: b56d1f4840e80000

mem_location: ff1eb498 :: data_stored: 00000000ff0aa010

mem_location: ff205a90 :: data_stored: 498312638ea5a158

mem_location: ff205a98 :: data_stored: 00000000ff363120

mem_location: ff271ce0 :: data_stored: dd1f5ce64ba380d1

mem_location: ff271ce8 :: data_stored: 00000000ff3d3350

mem_location: ff363120 :: data_stored: 6034dd09afa7836f

mem_location: ff363128 :: data_stored: 00000000ffdc5d80

mem_location: ff3d3350 :: data_stored: f8b1c5035767e241

mem_location: ff3d3358 :: data_stored: 00000000ff4aabb0

mem_location: ff4aabb0 :: data_stored: 9cce0672a0b3df5c

mem_location: ff4aabb8 :: data_stored: 00000000ff1eb490

mem_location: ff6aad90 :: data_stored: 11f4f9976aa61f89

mem_location: ff6aad98 :: data_stored: 00000000ff205a90

mem_location: ffb5d240 :: data_stored: 90134d03af1f6703

mem_location: ffb5d248 :: data_stored: 00000000ff6aad90

mem_location: ffdc5d80 :: data_stored: 5764cdf46a679122

mem_location: ffdc5d88 :: data_stored: 00000000ff271ce0

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

Ethernet frame head pointer and control word for corresponding frame are shown below, which are ready to be programmed into DUT.

```
-=-=-=-=-=-=-=-=-=-=-=-LINK HDR-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=

link_hdr head_ptr:ff0aa010 Ctrl_data:000afc01

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

ETH_TB::axi_waddr:0ffff0008 axi_wdata:ff0aa010000afc01
```

6.15.2 Programming of dma_controller_tx received by AXI_slave in DUT from testbench.

Waveform Description:

6.15.2.1 AXI slave write address channel.

Testbench sends AWADDR (write address) for five packets 32'hFFFF0008, 32'hFFFF0010, 32'hFFFF0018, 32'hFFFF0020, 32'hFFFF0028 along with burst,size and burst length of write transaction with AWVALID (write valid signal which indicates all the data in channel are valid) to axi_slave write receive logic in DUT. DUT's AXI slave responds with AWREADY (ready signal to indicate acceptance of all address channel signals for processing) in very next clock cycle as shown in waveform below (Figure 21).

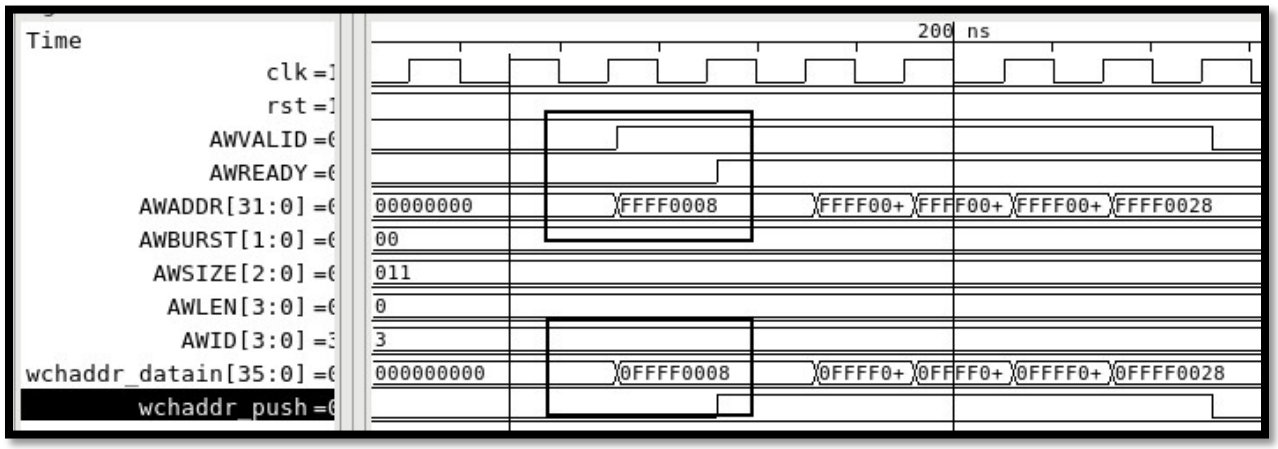


Figure 13. AXI Slave write address channel

AXI slave stores the received address in address store buffer as shown in waveform (Figure 21). Wchaddr_datain and wchaddrpush are corresponding buffer store signals in Figure 21.

Waveform description for AXI slave write data channel. (Figure 22). Signal WDATA indicates the 64 bit data that comes into AXI slave write data channel. WID represents the channel ID of write data channel. WVALID and WREADY are two-way handshake performed by AXI write data channel.

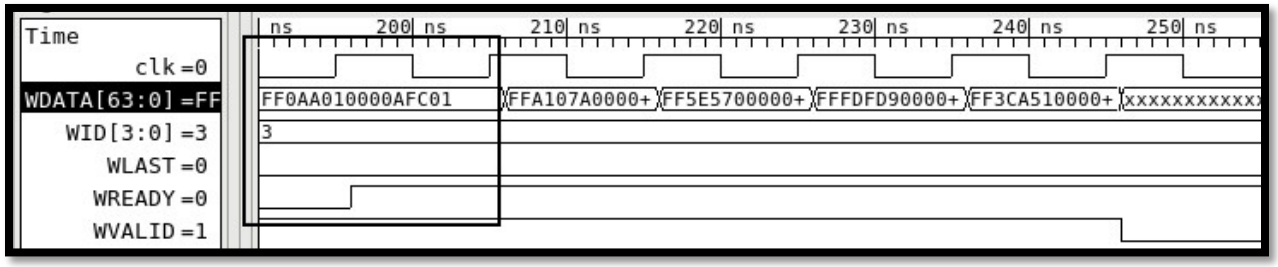


Figure 14. AXI slave write data channel

Testbench sends the `axi_wdata:ff0aa010000afc01` (Figure 22) in AXI_slave's data channel. Testbench asserts WVALID with corresponding WDATA (Figure 22). DUT responds with WREADY to accept the data send via AXI write data channel. Received data is send by slave into dma_controller_tx for programming DUT registers, which is shown in a later waveform figure 25.

6.15.2.2 AXI_slave write response channel.

AXI_slave module sends OKAY (2'b00) response to every register that is programmed in DUT by testbench using write response channel. (Figure 23). Once the registered are programmed by AXI_slave in DUT, corresponding write response is sent to testbench in case of successful write operation. Error response is sent if write operation fails.

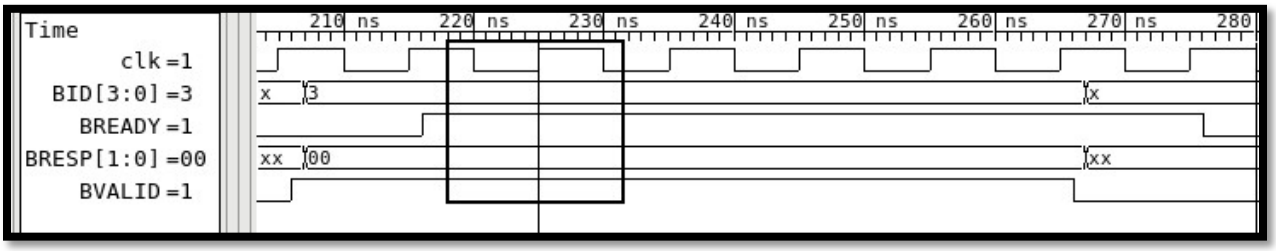


Figure 15. AXI slave write response channel

6.15.2.3 Register programming in Dma_controller_tx module.

Waveform in figure 24 show the register programming in Dma_controller_tx module via AXI_slave in RTL. After the data is written by testbench to DUT's AXI_slave write data interface, AXI_slave sends the register address 32'hFFFF0008 (reg_write_addr signal - refer waveform figure 24) and register data 64'hFF0AA010000AFC01 (reg_write_data signal - refer waveform figure 24) to Dma_controller_tx module.

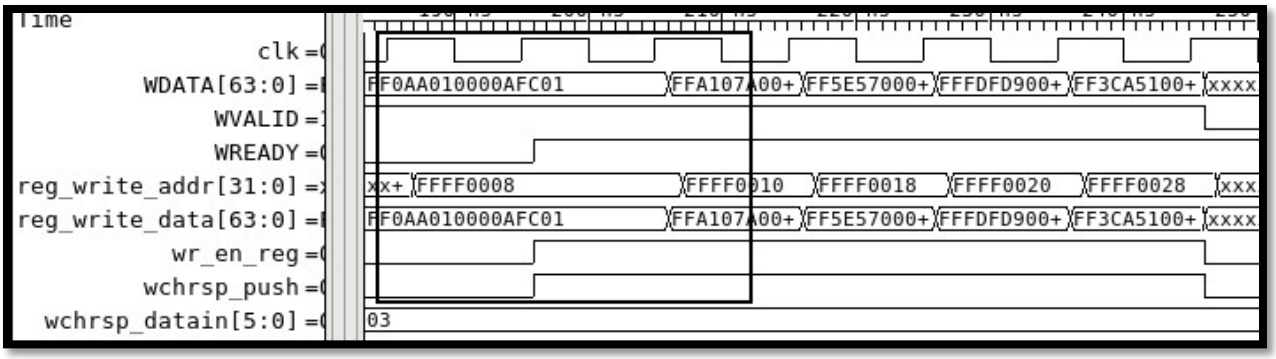


Figure 16. Dma_controller_tx register programming from AXI_slave

reg_write_addr and **reg_write_data** are programmed in dma_controller_tx module as shown in figure 25. Reg0 (clink_reg0) in RTL is programmed with head frame pointer + control header (64'hFF0AA010000AFC01) words as shown in waveform figure 25. Similarly, rest of the

registers reg_1(clink_reg1), reg_2 (clink_reg2), reg_3 (clink_reg3), reg_4 (clink_reg4) are programmed as shown in the Figure 25

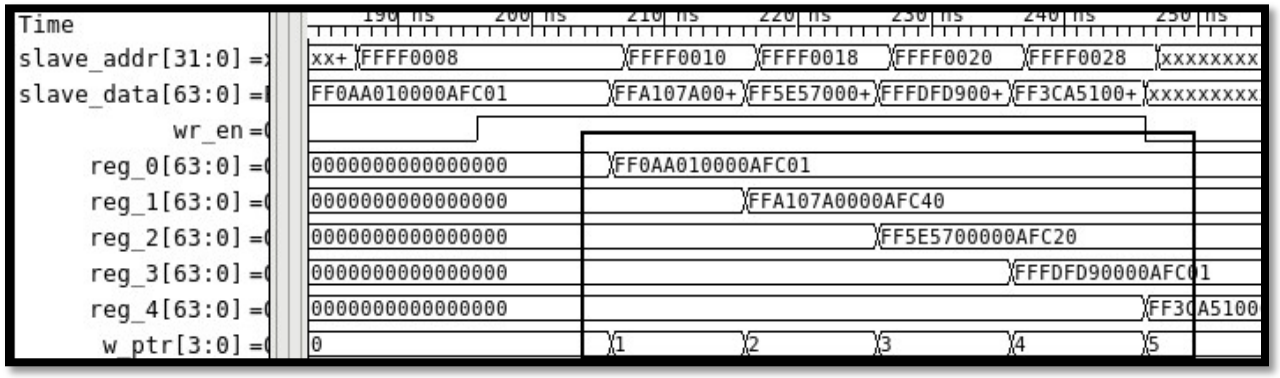


Figure 17. Programmed registers in Dma_controller_tx

6.15.3 AXI master read.

AXI master reads out the programmed frame header address in Dma_controller_reg module and starts reading out data fragment from memory and stores it in packet buffer till the link address reaches the head pointer (circular link list).

Dma_controller_tx register is read out (rd=1, refer figure 26) by axi_master module and assigned to a free AXI channel among the available channels. Arbiter in AXI master selects the channel to be served among requesting channels (ch_req_arb= 3'b111 and ch_gnt_d=3'b001). In this case, the first free channel is selected by arbiter and the head pointer is assigned to it. Then, the AXI master module start's the frame read from memory via the AXI master read interface. Waveform in figure 26 shows the ch_gnt_d=3'b001 ,which means the first channel is selected and assigned the head pointer frame address and control word (haddr=64'hFF0AA010000AFC01, in which first 32 bits are frame head address and next 32 bits are control word associated with the frame). Nxtaddr_to_axi (32'hFF0AA010) is sent to axi_master read logic to initiate the read transaction from testbench (Figure 26).

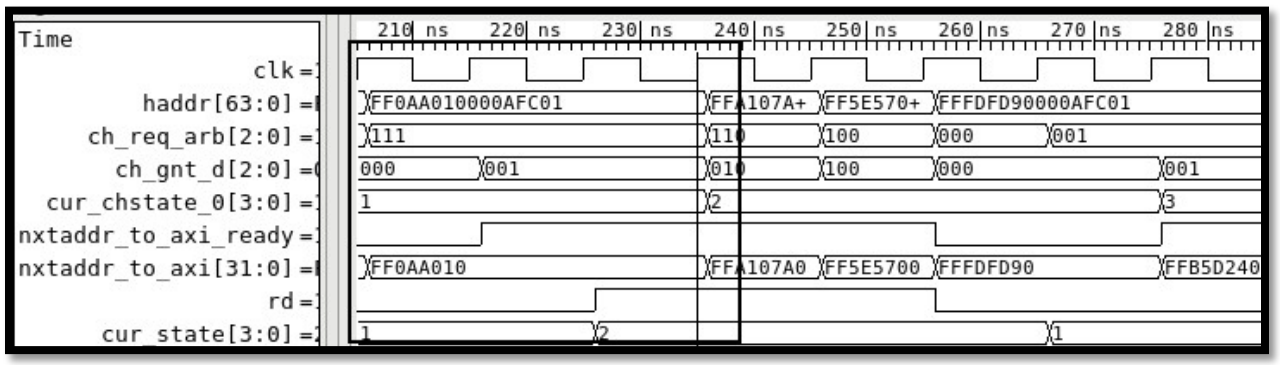


Figure 18.AXI_master register read from Dma_controller_tx block

6.15.3.1 Channel state machine waveform description.

Waveform below (figure 27) shows the haddr0_d=32'hFF0AA010 being flopped by register corresponding to channel 0. Cur_chstate_0, cur_chstate_1and cur_chstate_2 are cur states of Channel state machines described in design architecture specification section.

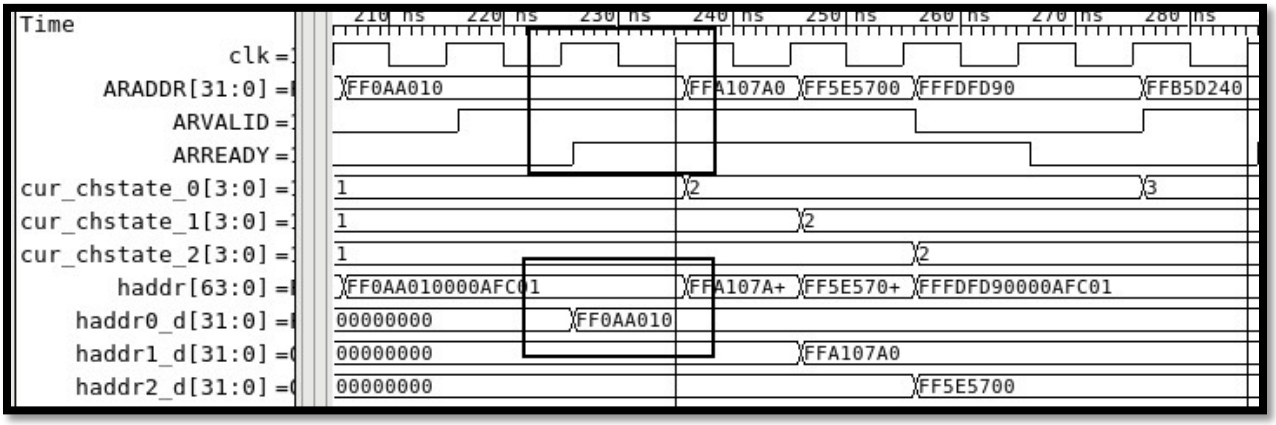


Figure 19. AXI_master channel head pointer allocation

6.15.3.2 AXI_read_addr_channel Waveform description (Figure 28)

AXI read address channel initiates the read transaction by sending frame head pointer address (32'hFF0AA01) in read address channel's ARADDR signal. ARID is set to 4'h0 (indicating channel 0) and ARBURST to 1 (support for bursting). Test bench will respond to this stimuli and start sending read transactions out in AXI read data channel.

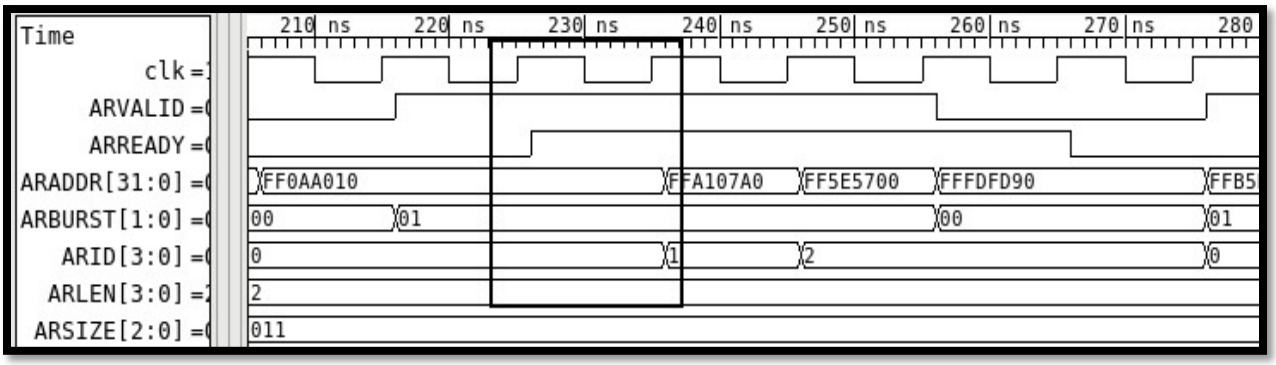


Figure 20.AXI master AXI read address channel

6.15.3.3 AXI read data channel waveform description. (Figure 29)

AXI read data channel receives a burst of read data from AXI slave model module in testbench. First 8 bytes sent by the test bench is Ethernet frame fragment data and next 8 bytes contains the link address of which 4 bytes contains the actual address used in the system. RLAST is asserted with the transaction when the last transaction in the burst is sent. One can also see the RID (read address channel id) changing after the first read burst is completed. System gives equal priority to all the three AXI read channel contained in the system. In this waveform, first 8 bytes of first frame which is RDATA=64'h 9B9BB6077DB0F049 and corresponding link address associated with that fragments which is RDATA=64'h00000000FFB5D240 are sent during the first read transaction.

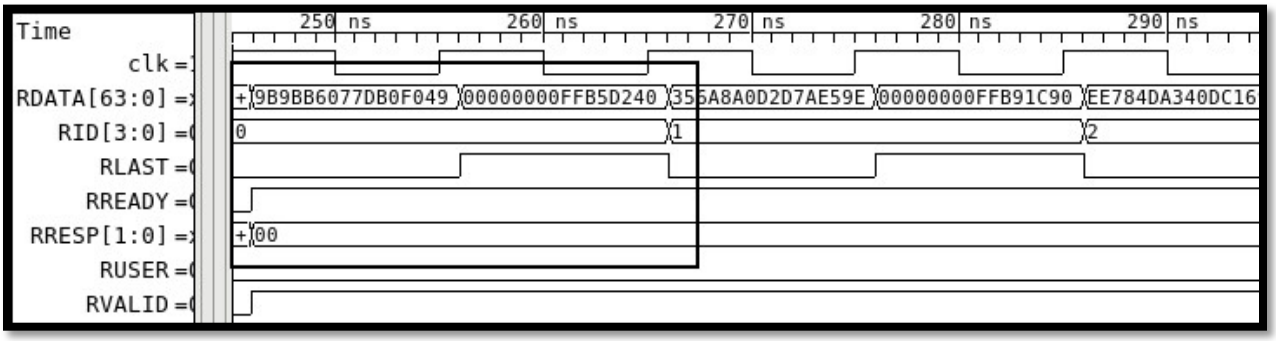


Figure 21.AXI read data channel in AXI master

6.15.3.4 Memory freeing up in testbench memory model as it read out via AXI read transaction.

```
255000 tb_axi_slave_model::==========Memory freed @ location:ff0aa010=
265000 tb_axi_slave_model::==========Memory freed @ location:ff0aa018=
325000 tb_axi_slave_model::==========Memory freed @ location:ffb5d240=
335000 tb_axi_slave_model::==========Memory freed @ location:ffb5d248=
395000 tb_axi_slave_model::==========Memory freed @ location:ff6aad90=
405000 tb_axi_slave_model::==========Memory freed @ location:ff6aad98=
465000 tb_axi_slave_model::==========Memory freed @ location:ff205a90=
475000 tb_axi_slave_model::==========Memory freed @ location:ff205a98=
535000 tb_axi_slave_model::==========Memory freed @ location:ff363120=
545000 tb_axi_slave_model::==========Memory freed @ location:ff363128=
605000 tb_axi_slave_model::==========Memory freed @ location:ffdc5d80=
615000 tb_axi_slave_model::==========Memory freed @ location:ffdc5d88=
675000 tb_axi_slave_model::==========Memory freed @ location:ff271ce0=
685000 tb_axi_slave_model::==========Memory freed @ location:ff271ce8=
745000 tb_axi_slave_model::==========Memory freed @ location:ff3d3350=
755000 tb_axi_slave_model::==========Memory freed @ location:ff3d3358=
815000 tb_axi_slave_model::==========Memory freed @ location:ff4aabb0=
825000 tb_axi_slave_model::==========Memory freed @ location:ff4aabb8=
885000 tb_axi_slave_model::==========Memory freed @ location:ff1eb490=
895000 tb_axi_slave_model::==========Memory freed @ location:ff1eb498=
```

6.15.3.5 AXI_master frame fragment data storage in packet buffer.

Data that is received from AXI_master read transaction is stored in corresponding packet data buffers associated with each channel. Pfifo_datain0 and pfifo_pushin0 are asserted in the

waveform to store the Ethernet frame fragment data 64'h 9B9BB6077DB0F049 associated with the first Ethernet frame that is read from user memory via AXI master read channel as shown in Figure 30.

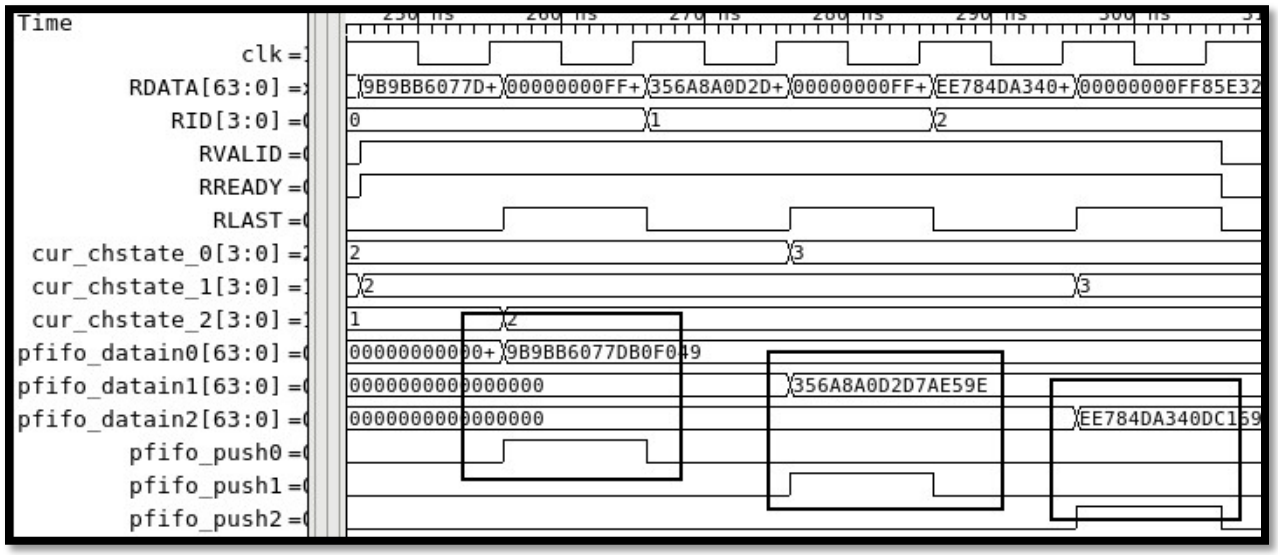


Figure 22.AXI master Ethernet frame data storage

6.15.3.6 AXI_master frame fragment link address storage in link addr buffer.

Link address that is received during the AXI_master read burst, which in this case is RDATA= 32'hFFB5D240 is stored in link address buffer as shown in figure 31. Link_datain_0 and link_push_0 are corresponding signals to link buffer as shown in the waveform. Once the link address is received by AXI read module, the channel state machine cur_chstate_0 changes to value 3, which corresponds to LINK state for the frame that is being fetched from memory. Stored link address is sent via AXI read address channel to test bench memory, once the associated channel wins the channel arbitration.

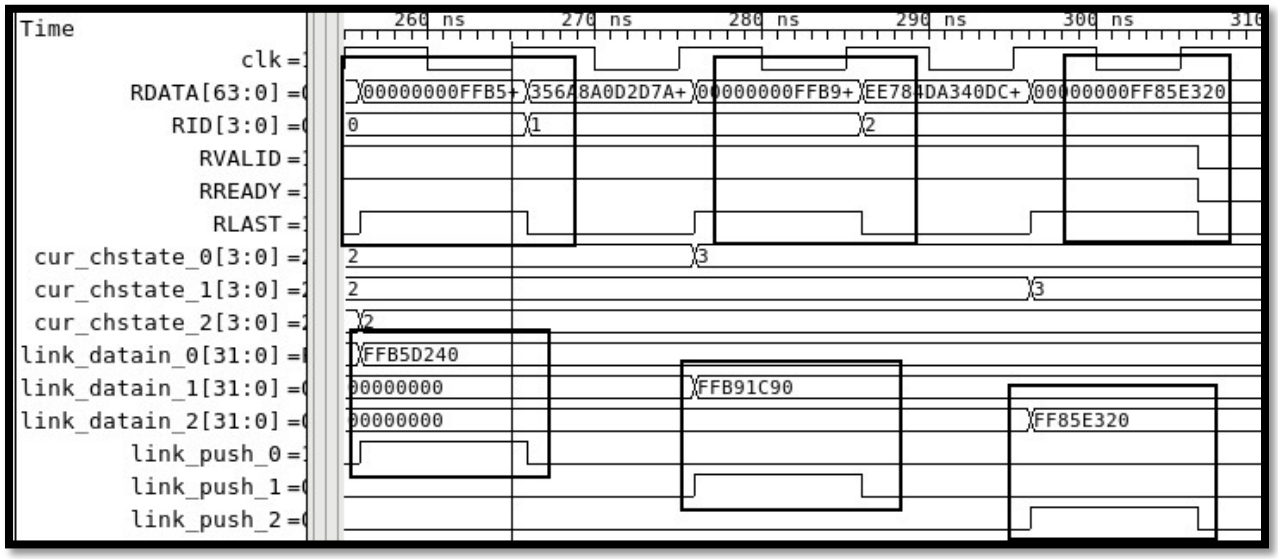


Figure 23. AXI master memory link address storage

6.15.3.7 AXI_master frame fragment link address wraps to head pointer.

Waveform figure 32 shows the link address of first Ethernet frame getting wrapped (link_wrap signal going high) around after completing transfer of all the Ethernet frame fragments associated with a particular frame via AXI read transactions. On link address wrap, link address is not stored in link address buffer and corresponding channel associated with the cur_chstate_0 changes state from 3 to 1, which frees up the channel to start transmitting new Ethernet frame.

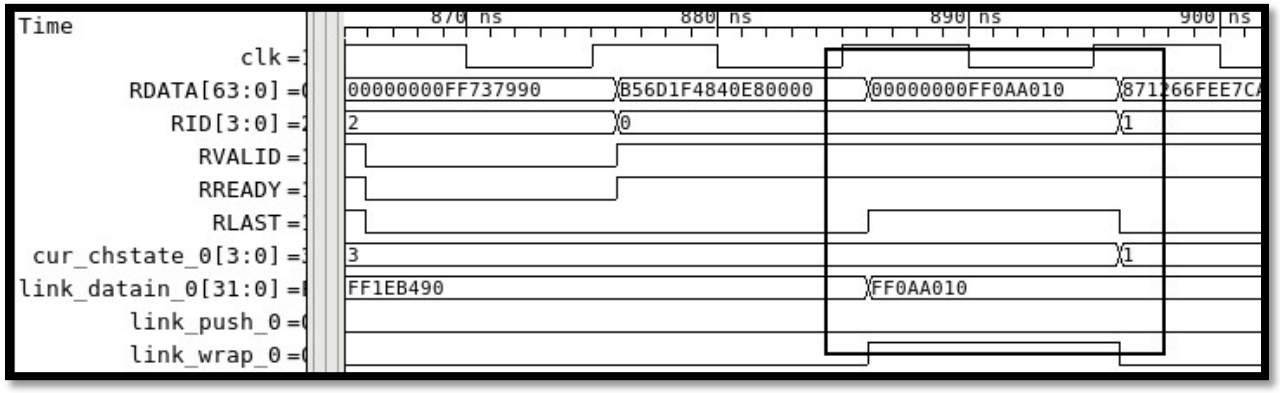


Figure 24.AXI master link address wrap to head pointer

6.15.3.8 AXI_master frame fragments control word storage in packet control buffers.

For each frame fragment that is stored in packet data buffer, a corresponding control word is stored in packet control buffer to assist downstream blocks in processing Ethernet frame fragment information. Waveform (figure 33) shows the {lastbytevalid, control code byte} stored in buffer (pcfifo_datain_0) for various Ethernet frame fragments. Data control word -09 represents start of frame fragment, 0B- middle segment of frame fragment, 0D – end of frame fragment.

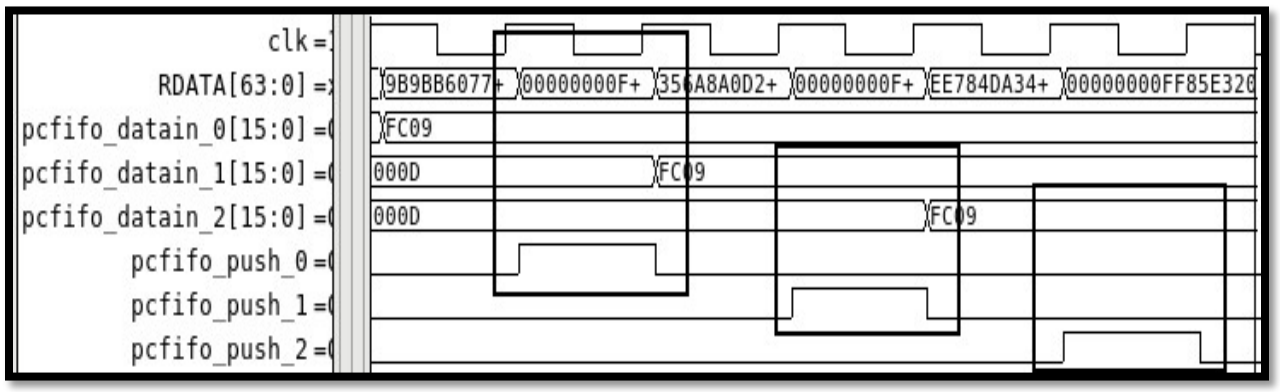


Figure 25. AXI master packet control word storage

6.15.4 CRC calculation

6.15.4.1 CRC calculation in eth_tx_crc block.

Eth_tx_crc is the major CRC calculation block in Transmit MAC. Figure 34 shows the results for CRC calculation of frame fragments in three channels associated with frame transmission. On completion of CRC calculation, resulting CRC is stored in a separate CRC buffers associated with each of three channels. Three instances of CRC_block is used in eth_tx_crc to calculate CRC of Ethernet frame fragments arriving at three channel associated with frame transmission.

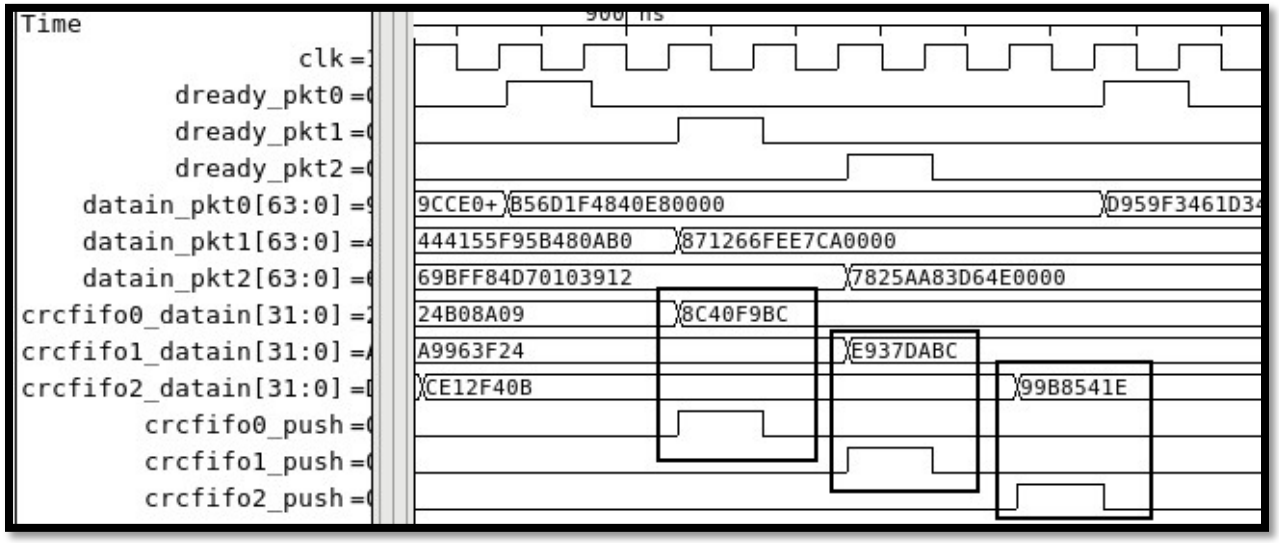


Figure 26. CRC calculation in Eth_Tx_Crc block

Frame fragments associated with the channels go into three separate CRC_blocks via datain_pkt0, datain_pkt1 and datain_pkt2 signals. Computed CRC is sent out by CRC_block and stored in CRC buffers. CRC buffer 0 stores 32'h8C40F9BC, which is the CRC for the first Ethernet frame. Crcfifo0_datain= 32'h8C40F9BC and crcfifo0_push is input to CRC buffer 0.

6.15.4.2 CRC_block - CRC calculation.

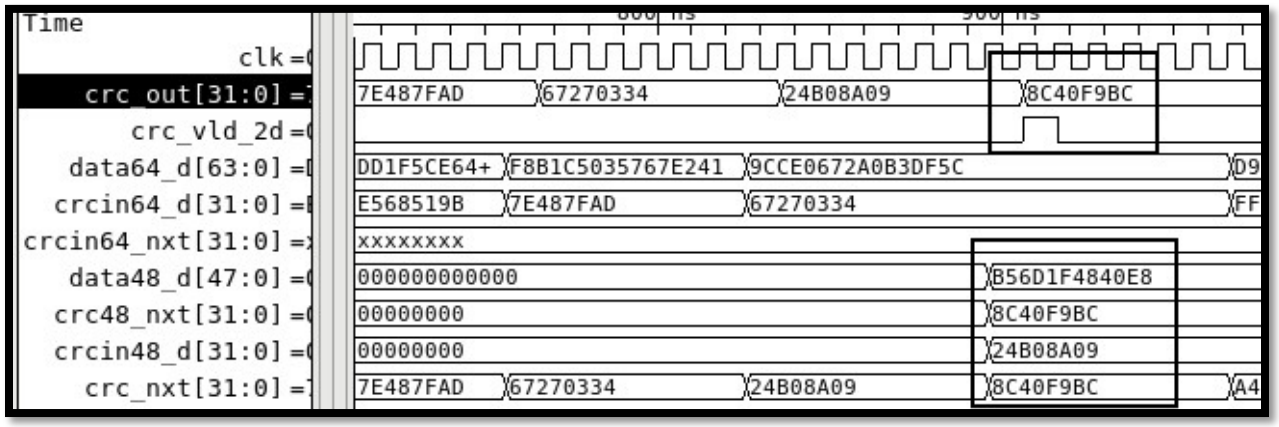


Figure 27.CRC calculation in CRC_block instance 0

CRC_block is the sub module in eth_tx_crc which calculates the CRC for each individual channel that is associated with packet transmission. Two sub blocks CRC64 and CRC48 are used for CRC calculation for this particular Ethernet frame. Figure 35 shows the results of CRC_block instance 0 in eth_tx_crc. CRC_nxt = 32'h8C40F9BC is the CRC for first frame, calculated at crc_vld_2d=1.CRC_block as a latency of 3 clock cycle for CRC calculation.

6.15.5 Channel selection to transmit Ethernet frame via transmit rs_layer.

6.15.5.1 Queue_selection block - simulation waveform.

Buffer_sel (Figure 36) shows the selected buffer that is ready to be transmitted by transmit rs_layer module. In this case the first buffer selected is that of channel 0, as it requests first with no other channel requesting for transmission across rs_layer. CRC buffer occupancy is start of arbitration. Figure 36 also shows the req and gnt vector which are part of sub module QoS_arb inside the queue selection module. QoS_arb is the arbiter module performing the required arbitration and enabling the transmission of Ethernet frames

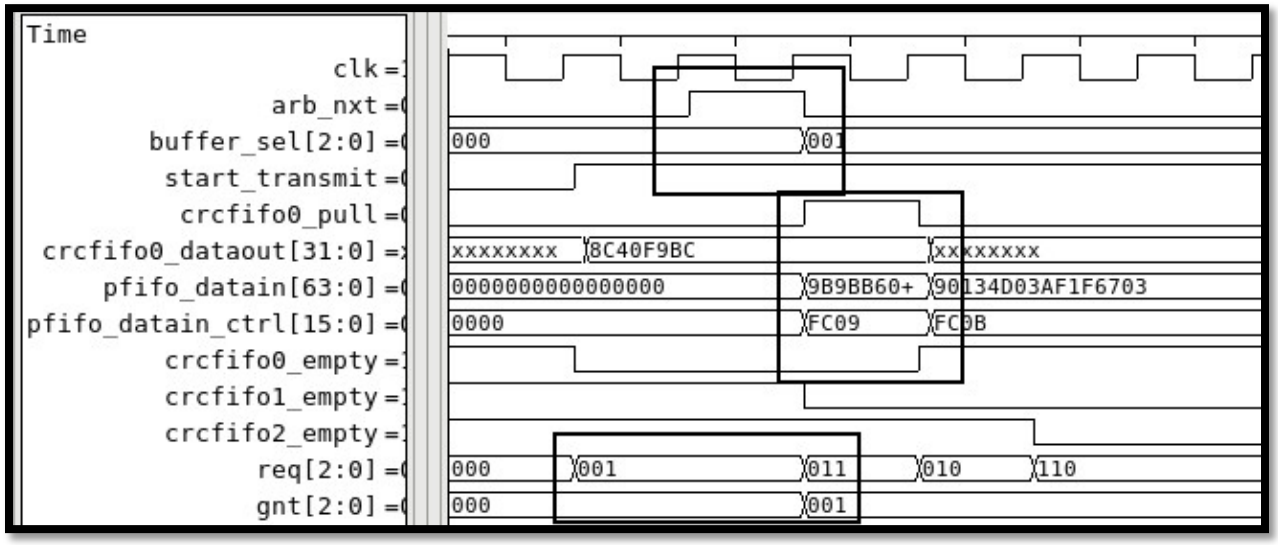


Figure 28. Queue_selection block selecting transmit queue

Verification of queue_selection arbitration at system level is possible only when RTL supports configurable buffer buildup registers, which is not implemented in present RTL and is meant to be implemented and tested in future scheme of things.

6.15.6 RS_layer transmit side.

RS_layer module uses a transmit state machine to send out data stream out of xgmii interface.TXD – 31 bit data stream, TXC- 4 bit byte wise control character.RS layer pulls out data from Packet buffer and CRC buffers of winning queue as determined queue_selection block.

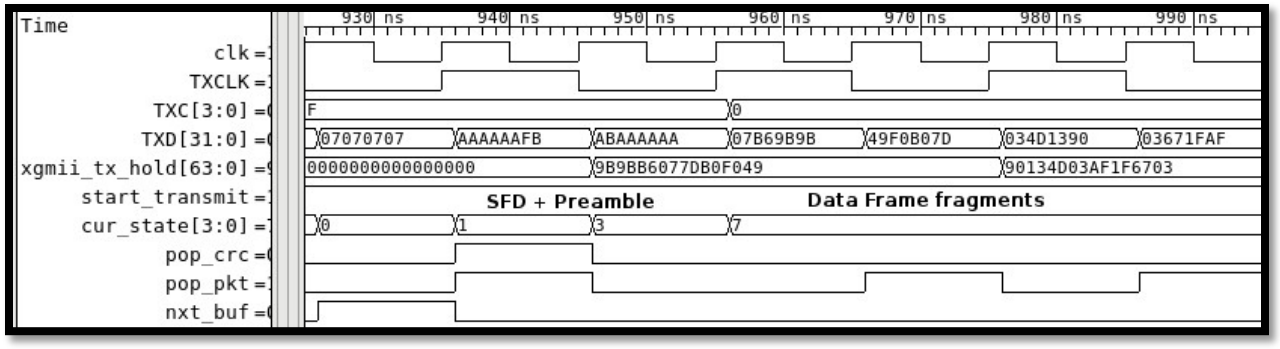


Figure 29. RS_layer Start of Transmission

For Tx state machine cur_state = 1, means SFD (start of delimiter) transmission (32'hAAAAAAFB). Cur_state= 3 is preamble transmission (32'hABAAAAAA). From state 7 onwards, frame fragment data is transmitted. Waveform in Figure 37 shows the transmission of start of frame (SFD – "FB" in 32'hAAAAAAFB), preamble and Ethernet frame fragments in 32 bit chunks TXD = {32'h49F0B070,32'h034D1390,32'h03671FAF}. Note in XGMII TXD and TXC are little endian byte order and big endian lane order.

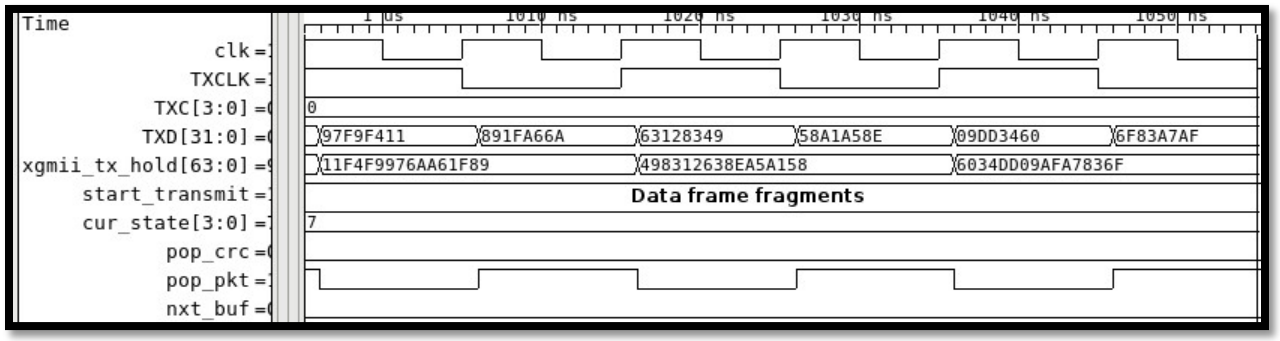


Figure 30. Ethernet Data Frame transmission continued

Waveform figure 38 shows the continued transmission of Ethernet frame fragment. Data fragments { 32'h97F9F411,32'h891FA66A,32'h64128349,32'h58A1A58E,32'h89DD3460, 32'h6F83A7AF}

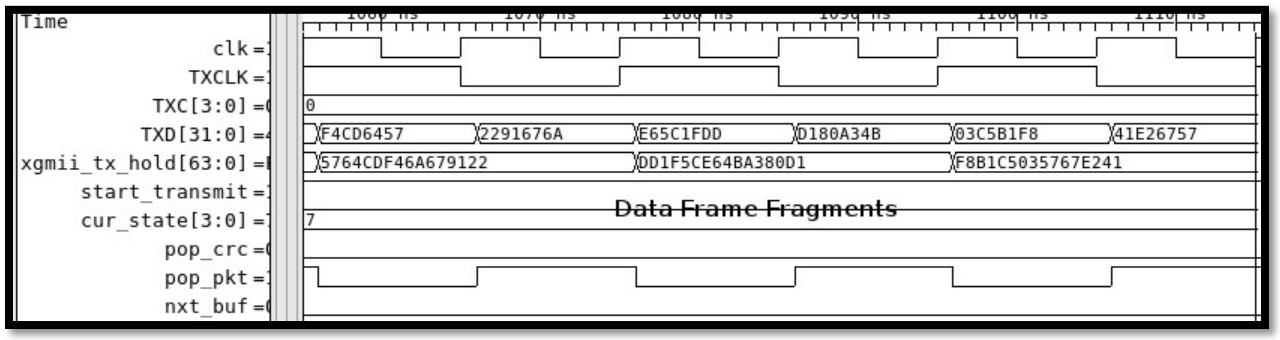


Figure 31. Ethernet Data Frame transmission continued

Waveform figure 39 shows the continued transmission of Ethernet frame fragment. Data fragments {32'hF4CD6457,32'h2291676A,32'hE65C1FDD,32'h0180A34B,32'h83C5B1F8, 32'h41E26757} are transmitted out in TXD signal.

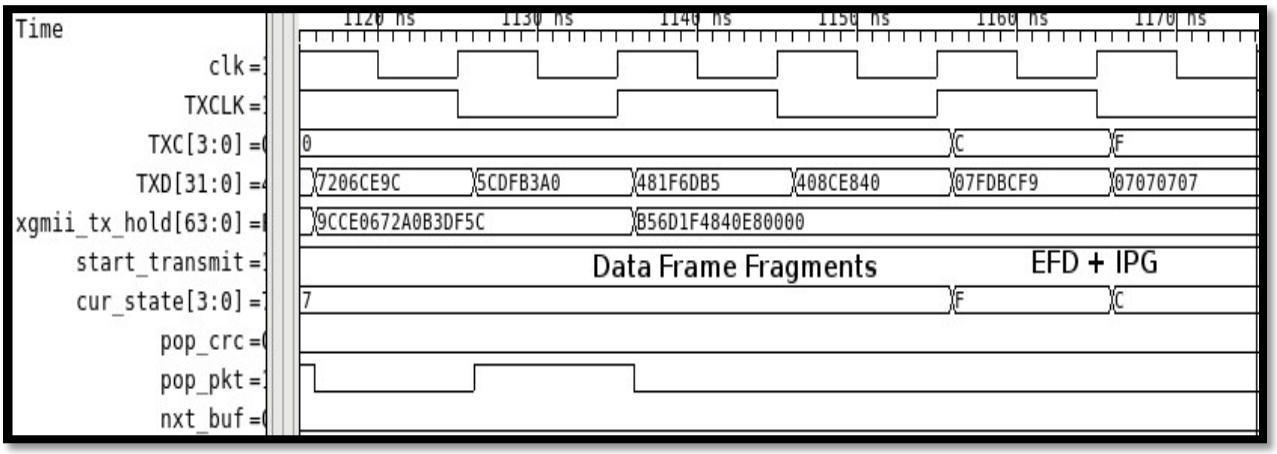


Figure 32. Ethernet Frame transmission EFD and IPG

Waveform figure 40 shows the continued transmission of Ethernet frame fragment. Data fragments {32'h7206CE9C,32'h5CDFB3A0,32'h481F6D85,32'h408CE840} are transmitted to PHY layer in figure 40.

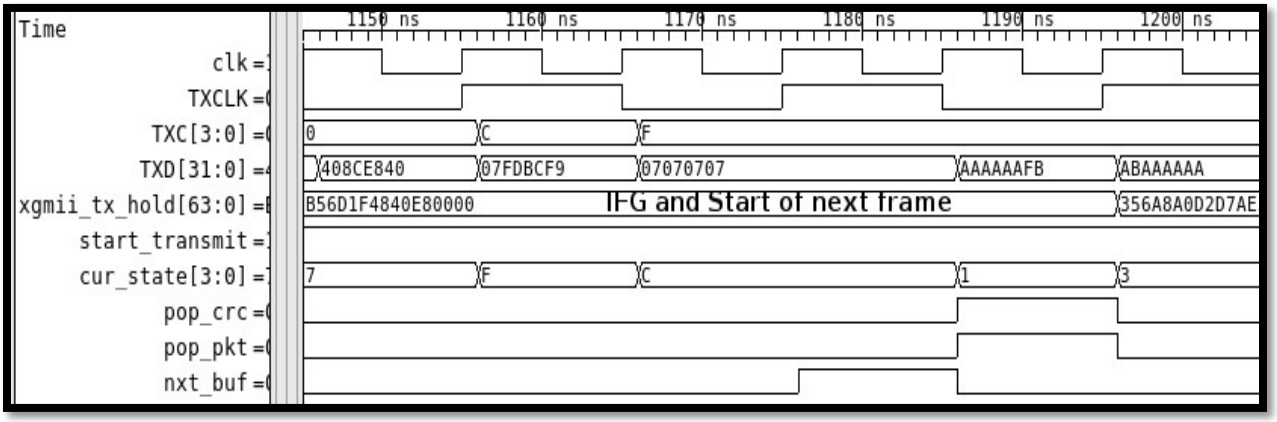


Figure 33. Ethernet Frame Inter Frame Gap

Waveform figure 41 shows the termination of current Ethernet frame fragment and idle transmission (inter frame gap). End of frame delimiter (EFD-8'hFD) in last 32 bits {32'h07FDBCF9} with idles (8'h07) . But instead of 12 octets of interfame gap only 9 octets of idles are inserted, due to implementation of IDC(idle deficit counter algorithm)- 3 idles of deficit is created to improve transmission efficiency. Figure 42 show the transmission of entire Ethernet frame in single simulation waveform transmitting SFD, PRE, DATA, EFD and IPG.

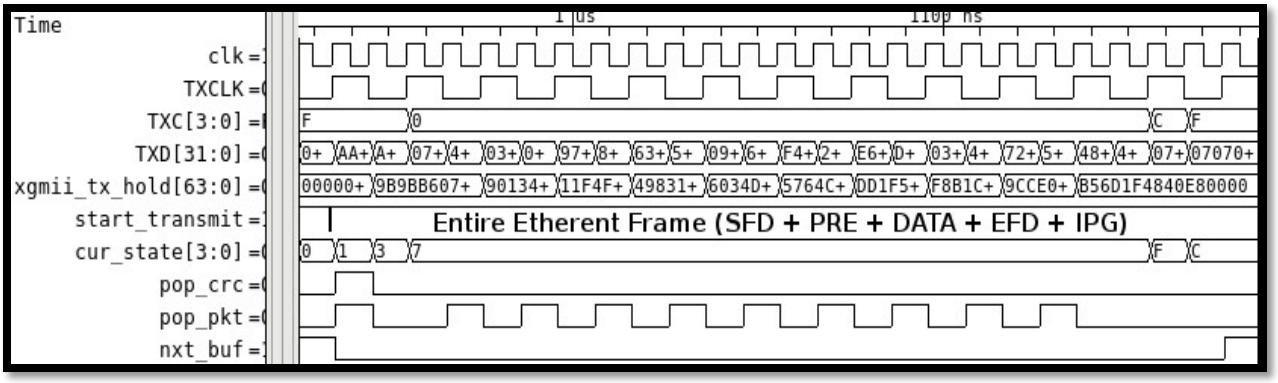


Figure 34.Complete Ethernet Frame transmitted through TX

6.15.6.1 Testbench print statements for matched first Ethernet frame.

```
------------------------------------------------------------------------------
!!!!INFO Ethernet frame matched
rec_frame:
9b9bb6077db0f04990134d03af1f670311f4f9976aa61f89498312638ea5a1586034dd09afa78
36f5764cdf46a679122dd1f5ce64ba380d1f8b1c5035767e2419cce0672a0b3df5cb56d1f4840
e88c40f9bc
exp_frame:
9b9bb6077db0f04990134d03af1f670311f4f9976aa61f89498312638ea5a1586034dd09afa78
36f5764cdf46a679122dd1f5ce64ba380d1f8b1c5035767e2419cce0672a0b3df5cb56d1f4840
e88c40f9bc

------------------------------------------------------------------------------
------------------------------------------------------------------------------
!!!!INFO Success!!!Frame matched #00000000
------------------------------------------------------------------------------
```

6.15.7 LPI mode during overall simulation.

In the current design the system is auto configured to enter LPI mode in case of non-occupancy of CRC buffers. Empty CRC buffers prompt the system to enter into Lower idling mode (LPI), signaling **TXD = 32'h06060606** and TXC=4'hF for atleast 128 clock cycles after which serial transmission clock (TXCLK) is gated as shown in figure 43.

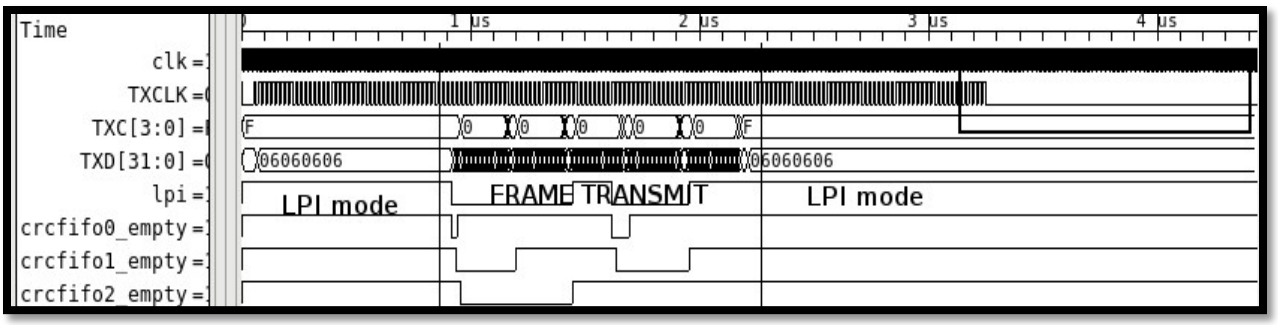


Figure 35.LPI mode assertion in TX path

Figure 44 shows the simulation waveform where the LPI is deliberately toggled to check the wakeuptimer logic in LPI state machine. After 128 clock cycle of wakeuptime(wakeuptimer done indication), system resumes normal operation and starts sending idles with TXCLK transmission. TXD data stream goes from 32'h06060606 to 32'h07070707 again to 32'h06060606 to 32'h07070707 after which frame transmission resumes.

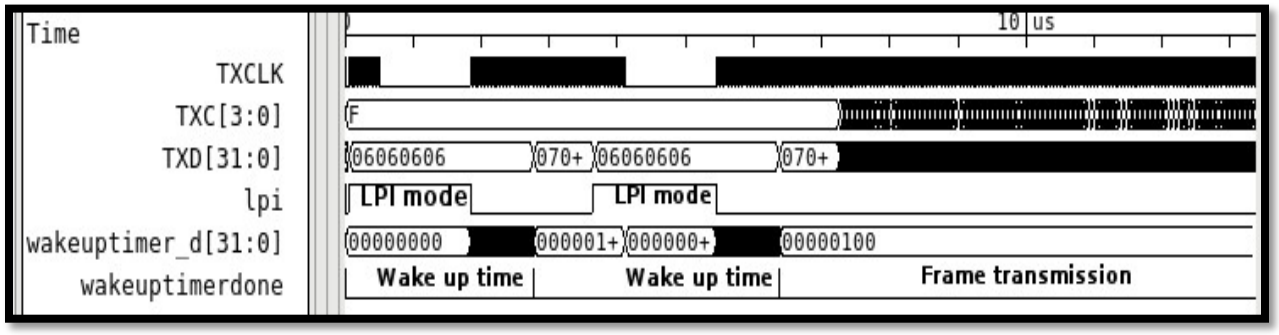


Figure 36. Wakeuptimer logic in LPI mode

6.16 Synthesis Results.

Synthesis without clock gate insertion.

AXI clock in the RTL needs to synthesized at 312.5Mhz in order to meet 10G line rate at XGMII interface. TX_CLK is exactly half of AXI clock, so TX_CLK runs at 156.25Mhz providing dual data rate transmission of XGMII signals.

Table 43. synthesis results without clock gate

Without CLK gate				
TECH	clock freq(MHz)	slack	Area (unit cells)	Dynamic Power(mW)
180 nm	333.33	0.46	1348620.00	189.36
240 nm	321.54	0.04	57250.00	351.82
45nm	312.5	0.7	110370.91	26.09

	Library	Clock frequency (Mhz)	slack	Area	Dynamic Power (mW)
Best Performance	45nm	666.66	0.03	113668.21	55.00

240nm technology consumes highest power followed by 180nm and 45nm as shown in table 45. Best performance was achieved with 45nm technology at 666.66 Mhz with 55 mW power for TX data path.

Synthesis with clock gate insertion.

Clock gate is only inserted for registers with bitwidth >= 16 bits, for efficient use of clock gate in synthesis. Clock gate brings down dynamic power significantly but at the cost of reduced frequency.

Table 44. COMPARITIVE SYNTHESIS RESULTS

COMPARITIVE SYNTHESIS RESULTS							
TECH		**Without clock gate**		**With clock gate**			
	clock freq(MHz)	Area (unit cells)	Dynamic Power(mW)	Clock Freq(MHz)	Area(unit cells)	Dynamic Power (mW)	Power reduction %
180 nm	333.33	1348620.00	189.36	303.33	1268835.00	38.36	79
250 nm	321.54	57250.00	351.82	222.22	48496.00	48.65	86
45nm	312.5	110370.91	26.09	289.85	102959.26	5.24	79

Power results with Primetime PX.

Used SAIF file generated during simulation run to estimate power in Primetime PX.(note results are not accurate as only 0.1% of RTL signals got cross annotated in PTPX map file). Also Power modelling cells were not available for accurate power estimation.

Table 45. PrimeTime PX power results

	CLK (Mhz)	SAIF				Vector Free			
		Internal (mW)	Switching (mW)	Leak (nW)	Total (mW)	Internal (mW)	Switching (mW)	Leak (nW)	Total (mW)
180 nm	312.5	21.4	1.51	214	22.9	23.3	2.63	214	25.9
240 nm	312.5	35.7	3.99	0	39.7	38.4	6.13	0	44.6
45 nm	312.5	2.93	0.123	79100	3.13	3.17	0.203	79100	3.45
180 nm	303.3	8.19	0.906	216	9.1	8.37	1.25	216	9.62
240 nm	222.22	10.2	1.48	0	11.7	10.4	1.68	0	12.1
45 nm	289.85	1.13	0.0667	76300	1.27	1.13	0.0795	76300	1.28

6.17 Gate level simulation Results. (Without clock gate)

Results of gate level simulation in NCVerilog using 240nm library. Simulation gate delays are observed for signal TXC and TXD signals of XGMII interface in Figure 45 and 46, which shows simulation results for gate level simulation. Figure 45 shows the initial SFD, Preamble and first data packet of Ethernet frame while figure 46 shows the complete Ethernet frame from start to end.

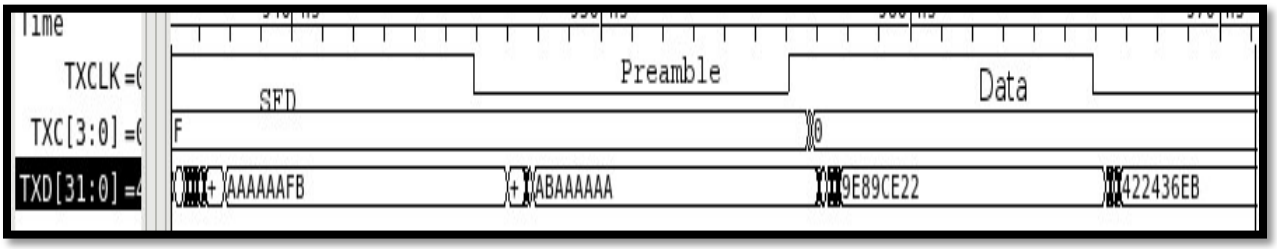


Figure 37. Gate delayed output of Ethernet frame in gate sims

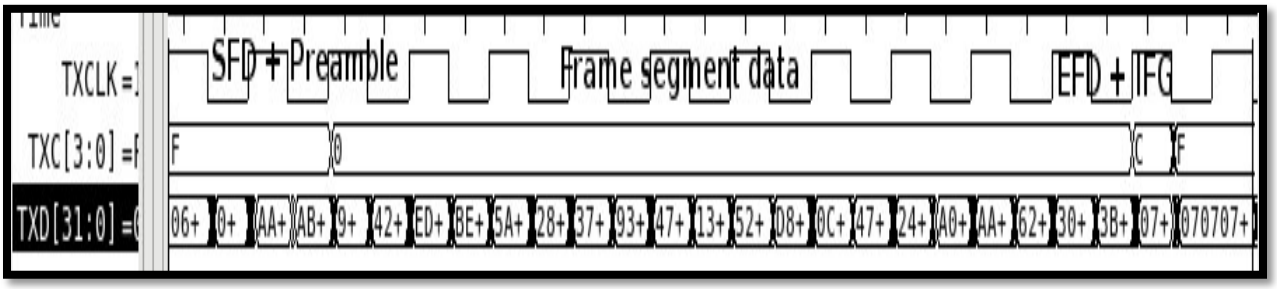


Figure 38. Complete Ethernet frame in gate sims

Testbench Print statements for gate level simulation.

Print statements below are results of gate level simulation run in NC Verilog.

```
--------------------------------------------------------------------------------
!!!!INFO Ethernet frame matched
rec_frame:
22ce899eeb3624425199a9edf0099abe3f2ab25ae6ce82284855ab371ffa99930966d047c5550
31325ff1d52672702d89cd5570c73a91647ba6f3224090fcaa0d54a0faaba71cc62887f7a308f
dc073b1476
exp_frame:
22ce899eeb3624425199a9edf0099abe3f2ab25ae6ce82284855ab371ffa99930966d047c5550
31325ff1d52672702d89cd5570c73a91647ba6f3224090fcaa0d54a0faaba71cc62887f7a308f
dc073b1476

--------------------------------------------------------------------------------
```

6.18 Gate level simulation Results. (With clock gate)

Figure 47 shows the gate level simulation results for clock gate inserted netlist in 250 nm technology library. Signal \tx_core/axi_master.\clk_gate_hdr2_d_reg[last_bvalid].ENCLK is clock enable and main_gate.Z is the output of gated clock from Integrated clock gated cell.

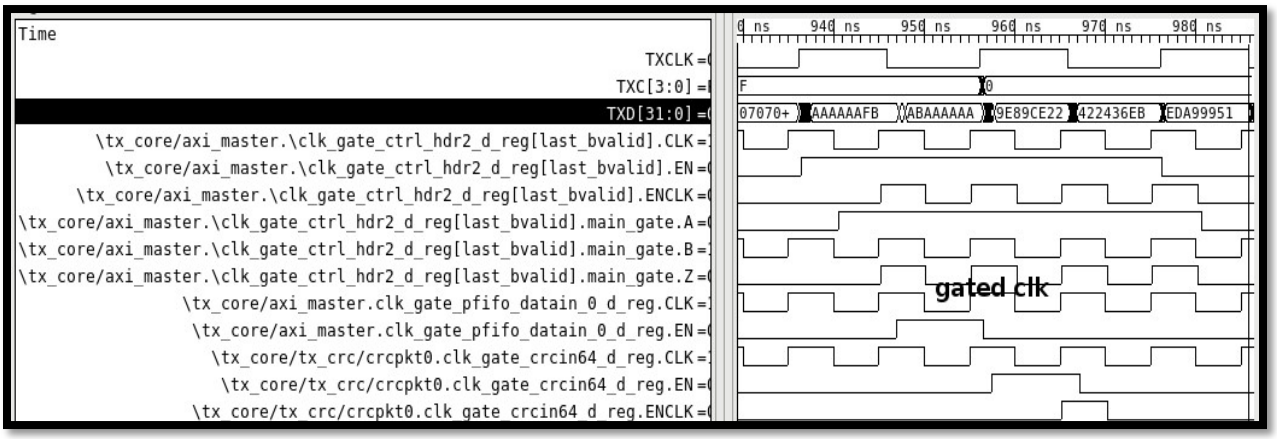


Figure 39. Clock gate in action in gate level sims

6.19 Layout Power estimation.

Tabular column below lists the power estimate obtained after place and route of design in cadence encounter. Around 58% and 80% of power reduction were obtained with clock gate in 180 nm and 45 nm technology respectively.

Power estimated with gate level simulation VCD file in SOC encounter.

Lib	**clk freq(Mhz)**	**Layout**				
WITHOUT CLK GATE						
		Internal(mW)	Switching (mW)	Leak(mW)	Total(mW)	% reduction power
180nm	333.33	376.9	174.3	0.003	551.20	
45nm	312.50	57.2	28.48	0.750	86.44	
WITH CLK GATE						
Lib	**clk freq(Mhz)**		**Layout**			
		Internal(mW)	Switching (mW)	Leak(nW)	Total(mW)	
180nm	333.33	147.1	84.56	0.004	231.70	57.96%
45nm	312.50	10.0	7.40	0.712	18.10	79.05%

Power pie chart across the module hierarchy.

180nm without clock gate and with clock gate

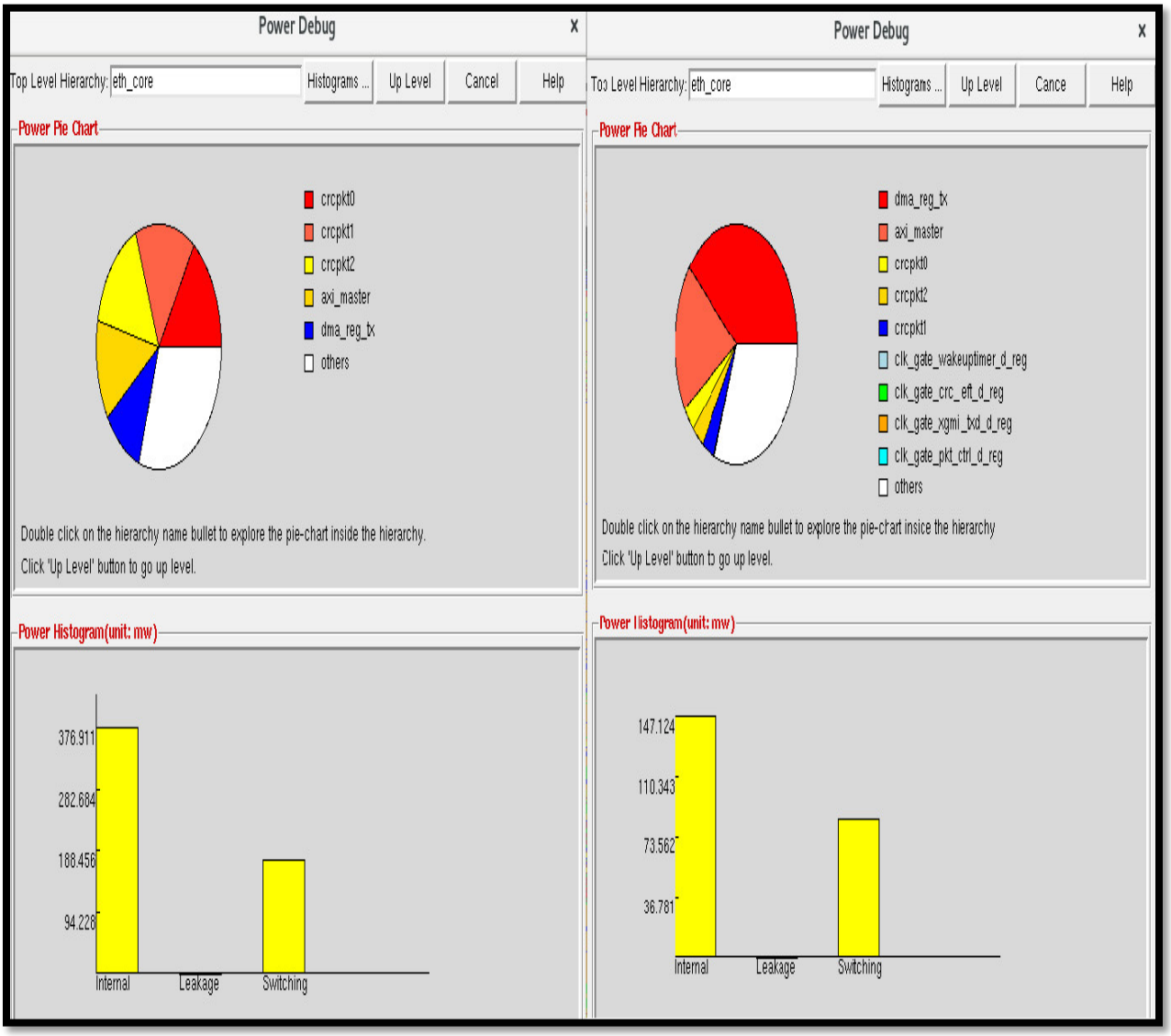


Figure 40. Power pie charts for 180nm

Figure 48 shows the power pie chart and histogram showing the variation of power dissipation across sub modules in TX data path for 180nm place and routed design in Cadence encounter for clock gated (right) and normal (left) design. Modules crcpkt0,crcpkt1 and crcpkt2 power are significantly reduced in clock gated design. While the power of AXI_master module increases in clock gated design. Overall power which is the sum of Internal, leakage and switching power get reduced by 58% from original design to clock gated design.

45nm without clock gate and with clock gate.

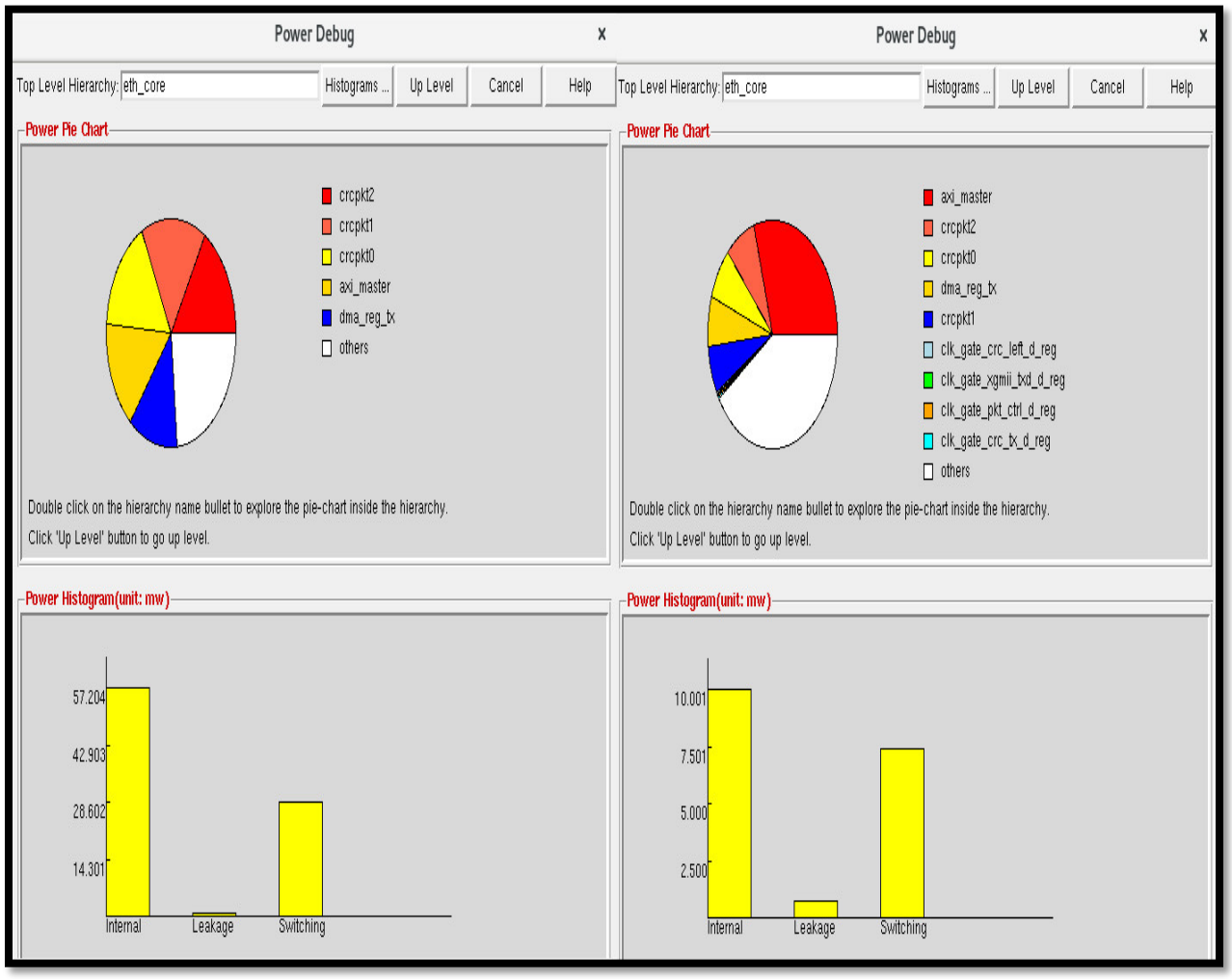


Figure 41.Power Pie chart for 45 nm

Figure 49 shows the power pie chart and histogram showing the variation of power dissipation across sub modules in TX data path for 45nm place and routed design in Cadence encounter for clock gated (right) and normal (left) design. Modules crcpkt0,crcpkt1 and crcpkt2 power are significantly reduced in clock gated design. While the power of AXI_master module increases in clock gated design. Overall power which is the sum of Internal, leakage and switching power get reduced by 80% from original design to clock gated design.

Layout results.

P&R 180nm design without clock gate.

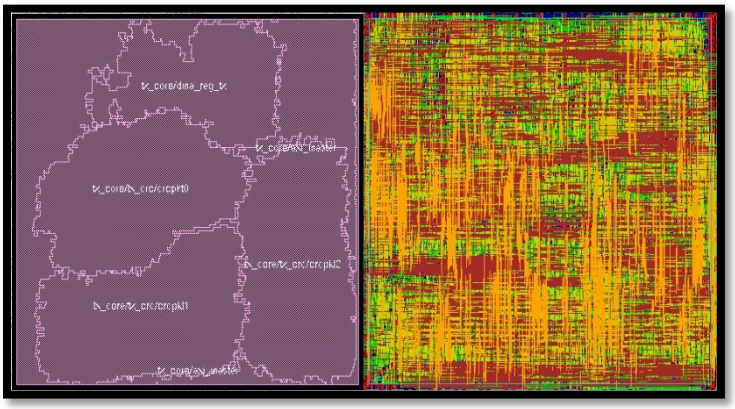


Figure 42. P&R 180nm without clock gate

Figure 50 shows the floorplan (left) and standard cell view (right) for 180 nm design without clock gate insertion. Some of statistics related to floor plan are given below.

********** Analyze Floorplan **********
Die Area(um^2) : 2163987.00
Core Area(um^2) : 1935019.00
Chip Density (Counting Std Cells and MACROs and IOs): 62.729%
Core Density (Counting Std Cells and MACROs): 70.152%
Average utilization : 126.659%
Number of instance(s) : 22783
Number of Macro(s) : 0
Number of IO Pin(s) : 1264
Number of Power Domain(s) : 0
********** Estimation Results **********
Total wire length. : 1.6685e+06
Congestion (H). : 0.007%
Congestion (V). : 0.870%
****

P&R 180nm clock gated design

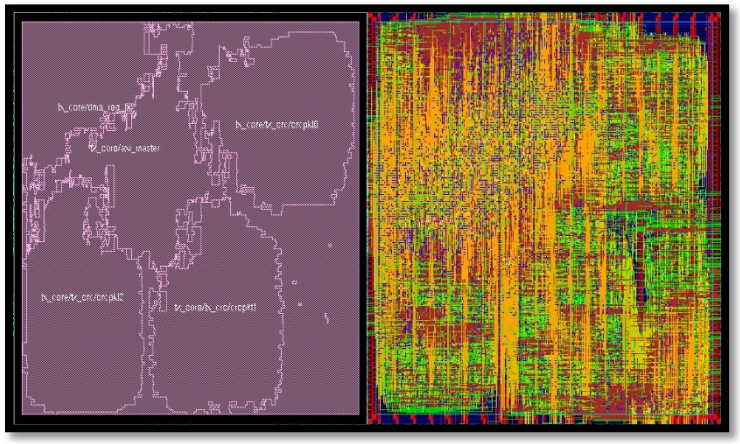


Figure 43. P&R 180nm with clock gate

Figure 51 shows the floorplan (left) and standard cell view (right) for 180 nm design with clock gate insertion. Some of statistics related to floor plan are given below. Routing congestion got reduced in clock inserted netlist. Clock tree synthesis was far better in clock gated netlist.

********** Analyze Floorplan **********
Die Area(um^2) : 2822300.00
Core Area(um^2) : 2559900.00
Chip Density (Counting Std Cells and MACROs and IOs): 46.351%
Core Density (Counting Std Cells and MACROs): 51.102%
Average utilization : 68.791%
Number of instance(s) : 21333
Number of Macro(s) : 0
Number of IO Pin(s) : 1264
Number of Power Domain(s) : 0
********** Estimation Results **********
Total wire length. : 1.8605e+06
Congestion (H). : 0.000%
Congestion (V). : 0.238%
****

P&R 45nm design without clock gate.

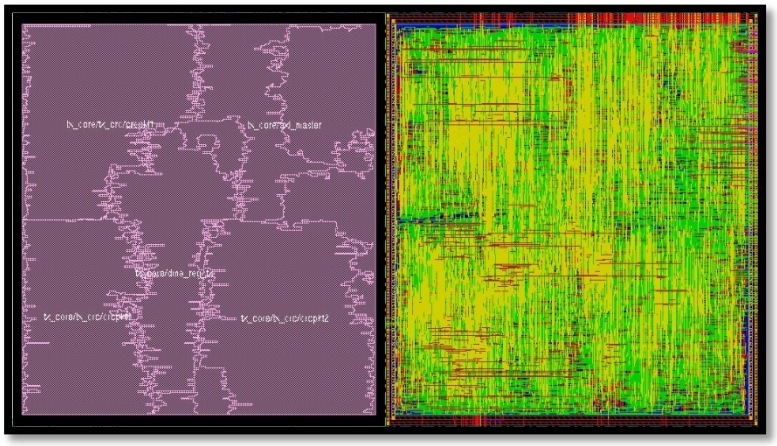


Figure 44.P&R 45nm without clock gate

Figure 52 shows the floorplan (left) and standard cell view (right) for 45 nm design without clock gate insertion. Some of statistics related to floor plan are given below.

********** Analyze Floorplan **********
Die Area(um^2) : 514614.14
Core Area(um^2) : 458635.19
Chip Density (Counting Std Cells and MACROs and IOs): 86.624%
Core Density (Counting Std Cells and MACROs): 97.196%
Average utilization : 97.224%
Number of instance(s) : 350397
Number of Macro(s) : 0
Number of IO Pin(s) : 1264
Number of Power Domain(s) : 0
********** Estimation Results **********
Total wire length. : 1.7708e+06
Congestion (H). : 0.000%
Congestion (V). : 0.003%
****

P&R 45nm design with clock gating

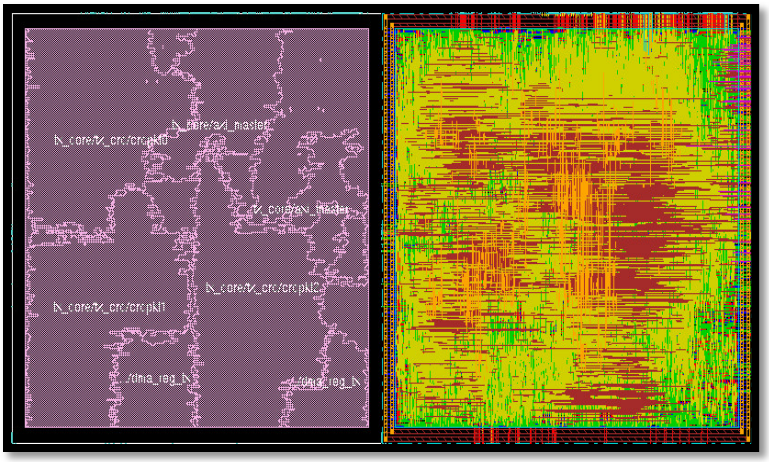


Figure 45. P&R 45nm with clock gate

Figure 53 shows the floorplan (left) and standard cell view (right) for 45 nm design with clock gate inserted netlist. Some of statistics related to floor plan are given below.

********** Analyze Floorplan **********
Die Area(um^2) : 309198.06
Core Area(um^2) : 266168.78
Chip Density (Counting Std Cells and MACROs and IOs): 80.573%
Core Density (Counting Std Cells and MACROs): 93.599%
Average utilization : 108.702%
Number of instance(s) : 192546
Number of Macro(s) : 0
Number of IO Pin(s) : 1264
Number of Power Domain(s) : 0
********** Estimation Results **********
Total wire length. : 7.2354e+05
Congestion (H). : 0.015%
Congestion (V). : 0.753%
****

Note: All the reg2reg hold violations in layout were fixed but there still some setup violations and DRC errors that are yet to be fixed.