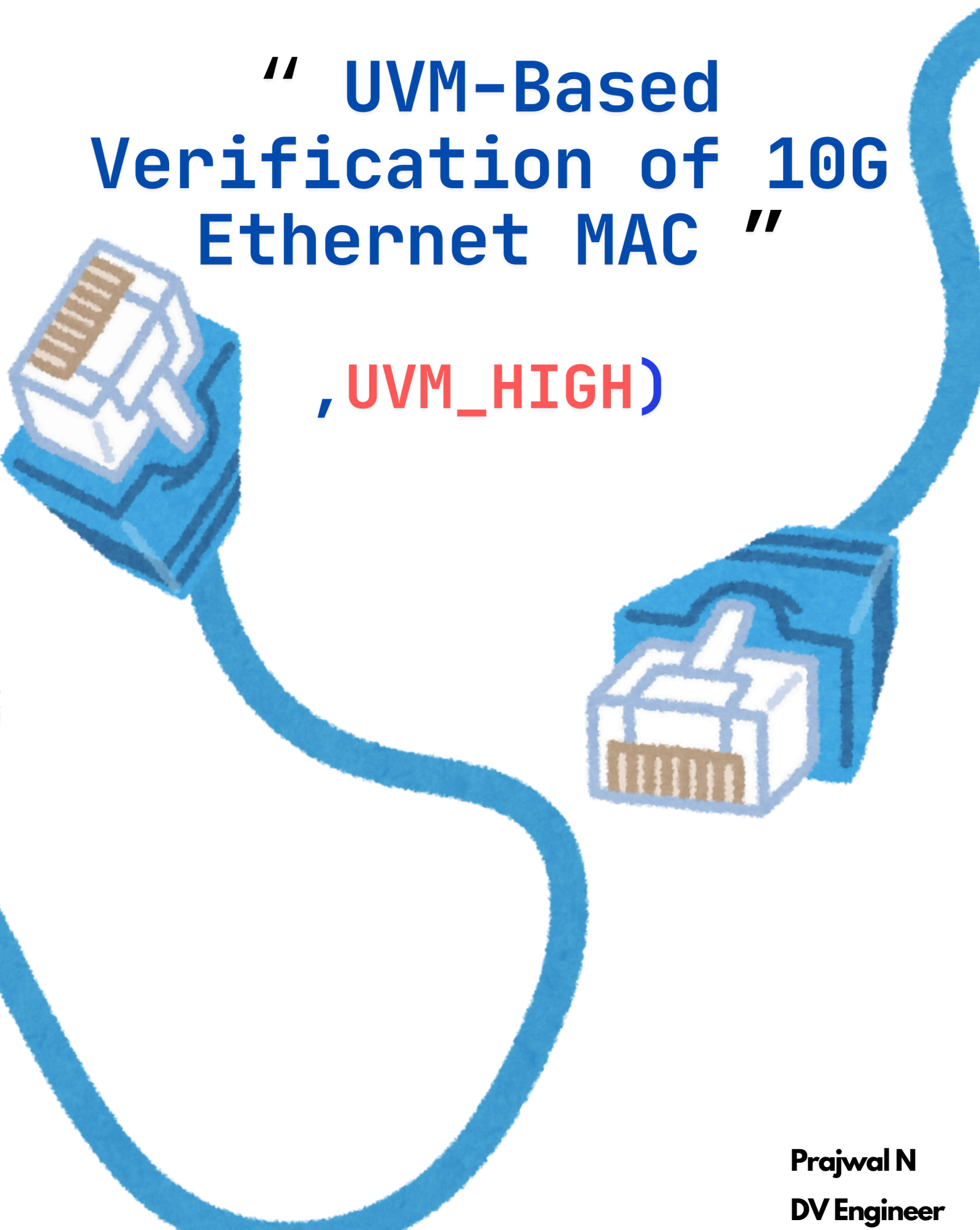
`uvm_info(get_type_name(),

" UVM-Based Verification of 10G Ethernet MAC "

,UVM_HIGH)



Prajwal N

DV Engineer

Introduction to Ethernet MAC

The Ethernet Media Access Control (MAC) layer is a critical component of the Ethernet networking protocol, responsible for managing how data packets are transmitted over the network. The MAC layer operates at the Data Link Layer (Layer 2) of the OSI model and is essential for ensuring that devices can communicate effectively across a shared medium. The primary functions of the MAC layer include frame delimiting, addressing, error checking, and controlling access to the physical transmission medium.

This guide is useful for anyone involved in top-level verification of the Ethernet MAC 10G. It provides an overview of the theory behind Ethernet MAC, the testbench architecture, and the test cases employed, and packet construction logic in verification. The frame format depicted in this guide is tailored for top-level verification; please note that the actual frame format may differ. The design under verification is sourced from an open-source GitHub repository, and credit goes to the original authors for their contributions.

Ethernet Packet Format

An Ethernet frame consists of several fields, each serving a specific purpose in data transmission. Understanding these fields is crucial for verifying the functionality of an Ethernet MAC.

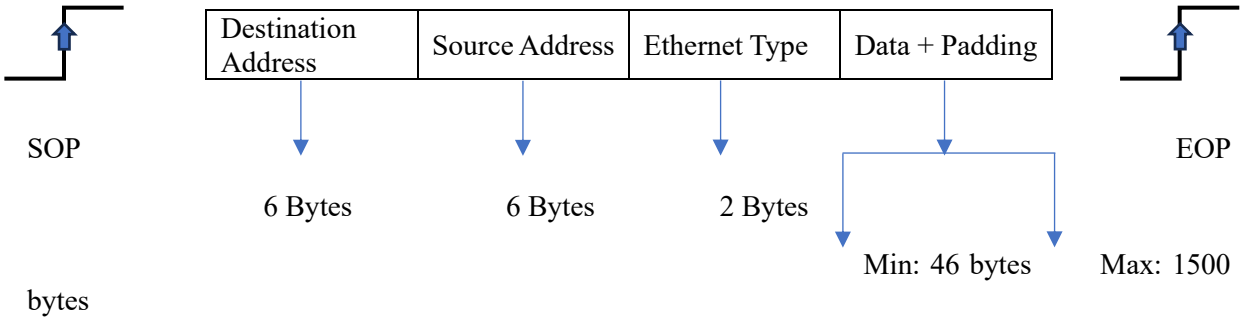


Fig:1

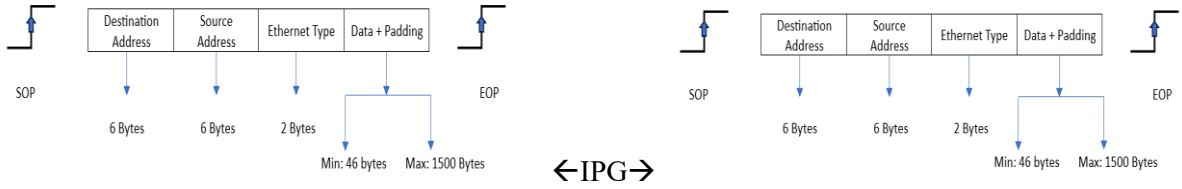


Fig:2

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

1. **Destination Address**:

The Destination Address is a 48-bit (6 Bytes) field that specifies the intended recipient of the frame. This address is unique to each device on the network and is typically assigned by the manufacturer. The MAC address allows devices to identify each other on a local area network (LAN).

2. **Source Address**:

The Source Address is also a 48-bit (6 Bytes) field that identifies the sender of the frame. Like the destination address, it is unique to each device and is used by the receiving device to know where the data originated.

3. **Ethernet Type**:

The Ethernet Type field indicates which protocol is encapsulated in the payload of the frame. Common values include:

- `0x0800` for IPv4
- `0x86DD` for IPv6
- `0x0806` for ARP

This field helps the receiving device determine how to process the incoming data.

4. **Payload**:

The Payload contains the actual data being transmitted. The size of this field can vary but must be between 46 bytes and 1500 bytes for standard Ethernet frames. If the payload is less than 46 bytes, padding of 0s or 1s is added to meet this minimum size requirement.

UVM-Based Verification Approach

The Universal Verification Methodology (UVM) provides a standardized framework for creating reusable verification environments. The test cases defined are as follows:

1. **Basic Bring-Up Loopback Case**: This test validates basic functionality by sending data back to itself, ensuring that the MAC can transmit and receive frames correctly.

2. **No Start of Packet (SOP) Case**: This scenario tests how well the MAC handles frames that do not signal their start correctly, which could lead to misinterpretation of incoming data.

3. **No End of Packet (EOP) Case**: This scenario tests how well the MAC handles frames received without proper end signals, which could result in incomplete data processing.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

4. **Undersized Packet Case**: This test checks how the MAC handles packets smaller than 45 bytes, ensuring they are appropriately rejected or padded as per standards.

5. **Oversized Packet Case**: This scenario evaluates processing of packets larger than 1500 bytes, which should be flagged as errors or dropped entirely.

6. **No Inter-Packet Gap (IPG)**: This test assesses how well the MAC manages packets sent back-to-back without required gaps, which could lead to collisions or data loss.

UVM Testbench Architecture:

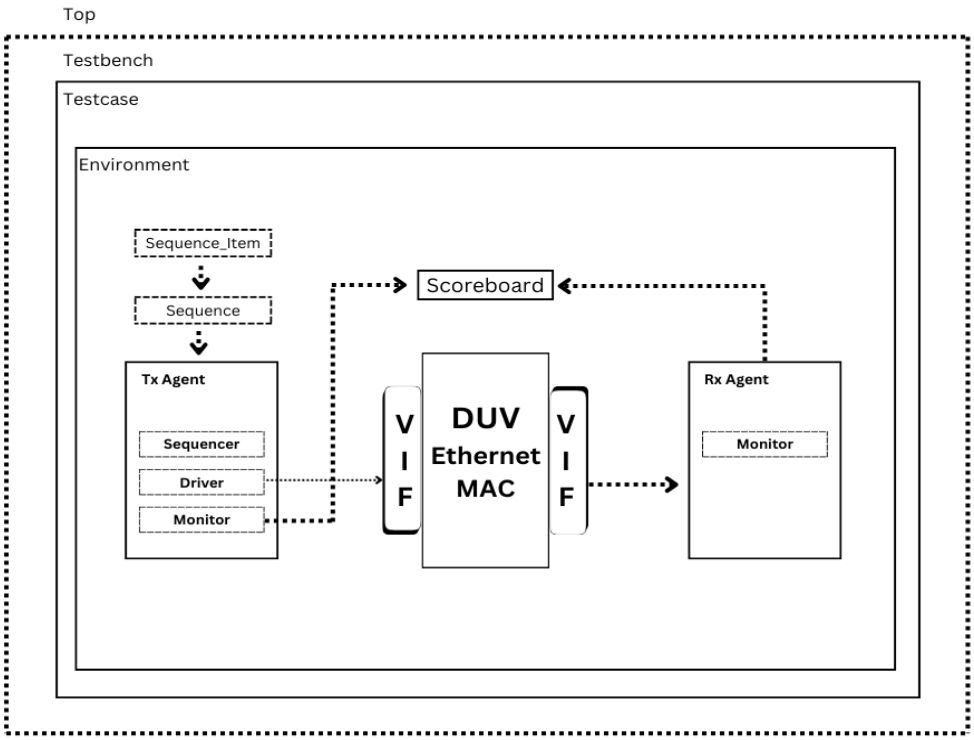


Fig:3

- **Sequence_Item**: Six `sequence_item` corresponding to each test case, utilizing `type_override` to customize behavior as needed for each test case scenario.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

- **Sequences**:

- **Idle Sequence**: Sends a continuous stream of zeros to simulate idle conditions on the bus. Utilizing idle_seq.start() to start Idle Sequence.

- **TX Data Sequence**: Sends actual data packets based on defined parameters and scenarios. Utilizing txdata_seq.start() to start TX Data Sequence.

TX Data Sequence:

Generates packets based on a payload size. Let's analyse the code step-by-step, particularly focusing on the scenario where the payload size is **46 bytes**. We will also detail each iteration of the loop that constructs the packets.

```
class txdata_sequence extends packet_sequence;
`uvm_object_utils ( txdat_sequence )
function new ( input string name = " txdata_sequence");
        super.new(name);
endfunction : new
task body();
        repeat(num_packets)
        begin
                `uvm_create ( req );
                assert ( req.randomize() );
                pkt_size = 6 + 6 + 2 + req.payload.size();
                pkt_size_mod8 = pkt_size % 8;
                pkt_size_div8 = pkt_size / 8;
                no_of_itrs = pkt_size_mod8 == 0 ? pkt_size_div8 : pkt_size_div8+1;
                last_full_row = pkt_size_mod8 == 0 ? no_of_itrs : no_of_itrs – 1;
                `uvm_info ( get_name() , $sformatf( " START PROCESS TXDATA_SEQUENCE
PRINT"),UVM_HIGH);
                $display("payload size = %d ",req.payload.size() );
                $display("ful packet size = %d ", pkt_size);
                $display("full packet size div 8 = %d " , pkt_size_div8 );
                $display("full packet size mod 8 = %d " , pkt_size_mod8 );
                $display("No of iteration = %d ",no of itrs );
                $display("last full row = %d" ,last_full_row);
                req.q_tx[0] = ( {ONE , ZERO , req.mac_dst_addr , req.mac_src_addr [47:32 ] } );
                req.q_tx[1] = ( { TWO , ZERO , req.mac_src_addr [ 31:0] , req.ether_type , req.payload
[ 0 ] , req.payload [1 ] } );
                if ( pkt_size_mod8 == 0)
                begin
                        k=last_full_row – 2;
                        for ( i=2 ; i<= k ; i=i+1 )
                        begin
                                j = -6+(i-1)*8;
```


Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

```
                                    req.q_tx[i] = ( {TWO , ZERO , req.payload [ j ] , req.payload[j+1] ,
req.payload [ j+2 ] , req.payload [ j+3 ] , req.payload [ j+4 ] , req.payload [ j+5 ] , req.payload [ j+6 ] ,
req.payload [ j+7 ] } );
                            end
                            i=last_full_row;
                            j = -6 + (i-1) * 8;
                            if ( pkt_size_mod8 == 1)
                            req.q_tx [ i ] = ( {THREE , ONE , req.payload [ j ] , {7{8'hFF} } } );
                            else  if ( pkt_size_mod8 == 2)
                            req.q_tx [ i ] = ( {THREE , TWO , req.payload [ j ] , req.payload [ j+1 ] ,
{6{8'hFF) } } );
                            else  if ( pkt_size_mod8 == 3)
                            req.q_tx [ i ] = ( {THREE , THREE , req.payload [ j ] , req.payload [ j+1 ] ,
req.payload [ j+2 ] , {5{8'hFF} } } );
                            else  if ( pkt_size_mod8 == 4)
                            req.q_tx [ i ] = ( {THREE , FOUR , req.payload [ j ] , req.payload [ j+1 ] ,
req.payload [ j+2 ] , req.payload [ j+3 ] , {4{8'hFF} } } );
                            else  if ( pkt_size_mod8 == 5)
                            req.q_tx [ i ] = ( {THREE , FIVE , req.payload [ j ] , req.payload [ j+1 ] ,
req.payload [ j+2 ] ,  req.payload [ j+3 ] , req.payload [ j+4 ] , {3{8'hFF} } } );
                            else  if ( pkt_size_mod8 == 6)
                            req.q_tx [ i ] = ( {THREE , SIX , req.payload [ j ] , req.payload [ j+1 ] ,
req.payload [ j+2 ] ,  req.payload [ j+3 ] , req.payload [ j+4 ] , req.payload [ j+5 ] , {2{8'hFF} } } );
                            else  if ( pkt_size_mod8 == 7)
                            req.q_tx [ i ] = ( {THREE , SEVEN , req.payload [ j ] , req.payload [ j+1 ] ,
req.payload [ j+2 ] ,  req.payload [ j+3 ] , req.payload [ j+4 ] , req.payload [ j+5 ] , req.payload [ j+6 ]
{8'hFF}  } );
                    end
                    start_item ( req ) ;
                    req.print();
                    `uvm_info ( get_name() , $sformatf ( "END : PROCESS TXDATA SEQUENCE
PRINT"), UVM_HIGH );
                    finish_item(req);
                    req.q_tx_delete();
            end
endtask
endclass : txdata_sequence
```

Overview of Key Calculations:

- Packet Size Calculations: Given a payload size of 46 bytes, we calculate the total packet size as follows:

  pkt_size = 6 (MAC dst) + 6 (MAC src) + 2 (EtherType) + req.payload.size();

  pkt_size = 6 + 6 + 2 + 46 = 60 bytes.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

- Modulus and Division Calculations: We compute modulus and division for the packet as follows:

  pkt_size_mod8 = pkt_size % 8; = 60 % 8 = 4

  pkt_size_div8 = pkt_size / 8; = 60 / 8 = 7

  no_of_itrs = pkt_size_mod8 == 0 ? pkt_size_div8 : pkt_size_div8 + 1; = 7 + 1 = 8.

  last_full_row = pkt_size_mod8 == 0 ? no_of_itrs : no_of_itrs - 1; = 8 - 1 = 7.

Construction of First Two Packets:

Before entering the loop for additional packets, the first two packets are constructed as follows:

req.q_tx[0] = {ONE, ZERO, req.mac_dst_addr, req.mac_src_addr[47:32]};
req.q_tx[1] = {TWO, ZERO, req.mac_src_addr[31:0], req.ether_type, req.payload[0], req.payload[1]};

- req.q_tx[0]: Contains the MAC destination address (6 bytes) and part of the source address (2 bytes).
- req.q_tx: Contains the remaining part of the source address (4 bytes), EtherType (2 bytes), and the first two bytes of the payload.

Construction of Remaining Packets:

Since pkt_size_mod8 is **4**, we enter the conditional block for constructing additional packets. The loop starts at i=2 and continues until i <= k, where k is defined as last_full_row - 2. Given that last_full_row is **7**, we have:

k = last_full_row - 2; // k = 7 - 2 = 5

- Iteration i = 2:

  j = -6 + (2-1)*8; // j = -6 + 8 = 2;
  req.q_tx[2] = {TWO, ZERO, req.payload[2], req.payload[3], req.payload[4], req.payload[5], req.payload[6], req.payload[7], req.payload[8], req.payload[9]};

  Constructs packet with payload bytes from index **2** to **9**.

- Iteration i = 3:

  j = -6 + (3-1)*8; // j = -6 + 16 = 10;
  req.q_tx[3] = {TWO, ZERO, req.payload[10], req.payload[11], req.payload[12], req.payload[13], req.payload[14], req.payload[15], req.payload[16], req.payload[17]};

  Constructs packet with payload bytes from index **10** to 1**7**.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

- Iteration i = 4:

  j = -6 + (4-1)*8; // j = -6 + 24 = 18;
  req.q_tx[4] = {TWO, ZERO, req.payload[18], req.payload[19], req.payload[20], req.payload[21], req.payload[22], req.payload[23], req.payload[24], req.payload[25]};

  Constructs packet with payload bytes from index **18** to **25.**

- Iteration i = 5:

  j = -6 + (5-1)*8; // j = -6 + 32 = 26;
  req.q_tx[5] = {TWO, ZERO, req.payload[26], req.payload[27], req.payload[28], req.payload[29], req.payload[30], req.payload[31], req.payload[32], req.payload[33]};

  Constructs packet with payload bytes from index **26** to **33.**

- Iteration i = 6:

  j = -6 + (6-1) * 8; // j = -6 + 40 = 34;
  req.q_tx[6] = {TWO, ZERO, req.payload[34], req.payload[35], req.payload[36], req.payload[37], req.payload[38], req.payload[39], req.payload[40], req.payload[41]};

  Constructs packet with payload bytes from index **34** to **41.**

Construction of Last Full Row Packets:

After constructing packets for i=2 to i=6, we handle the last full row:

req.q_tx[i] = {THREE, FOUR, req.payload[42],req.payload[43],req.payload[44], req.payload[45],padding}; // Padding with remaining bits as needed.

Constructs packet with payload bytes from index **42** to **45.**

NOTE:

The first field of each packet is designated to indicate the type of packet being constructed.

ONE: Represents the Start of Packet (SOP). This indicates that the packet being transmitted is the beginning segment of a data transmission. It includes MAC address of Destination and Source.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

TWO: Denotes the Middle of Packet (MOP). This signifies that the packet is part of a larger sequence, continuing from a previous segment. It includes additional Source MAC address, Ethernet type, and part of payload.

THREE: Signifies the End of Packet (EOP). This indicates that the current packet marks the conclusion of a data transmission sequence. It includes final data bytes and any necessary padding to ensure that the packet conforms to defined size constraints.

The second field, represented by constants ranging from **ZERO** to **SEVEN**, is used to manage padding within the packet structure. These values indicate how many bits are padded with either 0s or 1s to complete the remaining slots in an 8-byte frame format. This is crucial for adhering to protocol specifications that require packets to be aligned to specific byte boundaries.

Overview of the Packet Constructed:

	Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4]	Byte[5]	Byte[6]	Byte[7]
Clk 1	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]	S[5]	S[4]
Clk 2	S[3]	S[2]	S[1]	S[0]	ET[0]	ET[1]	P[0]	P[1]
Clk 3	P[2]	P[3]	P[4]	P[5]	P[6]	P[7]	P[8]	P[9]
Clk 4	P[10]	P[11]	P[12]	P[13]	P[14]	P[15]	P[16]	P[17]
Clk 5	P[18]	P[19]	P[20]	P[21]	P[22]	P[23]	P[24]	P[25]
Clk 6	P[26]	P[27]	P[28]	P[29]	P[30]	P[31]	P[32]	P[33]
Clk 7	P[34]	P[35]	P[36]	P[37]	P[38]	P[39]	P[40]	P[41]
Clk 8	Padding	Padding	Padding	Padding	P[45]	P[44]	P[43]	P[42]

Visit my GitHub repository for the complete UVM verification code: https://github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification/tree/main/Ethernet_MAC_10G