```
////////////////////////////////////////////////////////////////////
//                                                                //
//  File name : driver.sv                                         //
//  Author    : G. Andres Mancera                                 //
//  License   : GNU Lesser General Public License                 //
//  Course    : Advanced Verification with SystemVerilog OOP      //
//              Testbench - UCSC Silicon Valley Extension         //
//                                                                //
////////////////////////////////////////////////////////////////////

class driver;

  virtual xge_mac_interface     drv_vi;
  packet                        xge_mac_pkt;
  mailbox                       drv2sb;

  // ======== Constructor ========
  function new( input virtual xge_mac_interface vif,
                input mailbox drv2sb                );
    $display("DRIVER :: inside new() function");
    this.drv_vi = vif;
    this.drv2sb = drv2sb;
    xge_mac_pkt = new();
  endfunction : new


  // ======== Class methods ========
  task send_packet(int num_packets);
    packet          drv_pkt;
    int unsigned    pkt_len_in_bytes;
    int unsigned    num_of_flits;
    bit [2:0]       last_flit_mod;
    bit [63:0]      tx_data;


    for (int j=0; j<num_packets; j++ ) begin
      drv_pkt     = new xge_mac_pkt;
      assert( drv_pkt.randomize() );
      pkt_len_in_bytes    = 6 + 6 + 2 + drv_pkt.payload.size();
      num_of_flits        = ( pkt_len_in_bytes%8 ) ? pkt_len_in_bytes/8 + 1 :
pkt_len_in_bytes/8;
      last_flit_mod       = pkt_len_in_bytes%8;
      //$display("DRIVER DEBUG :: pkt_len_in_bytes =%0d", pkt_len_in_bytes);
      //$display("DRIVER DEBUG :: num_of_flits     =%0d", num_of_flits);
      //$display("DRIVER DEBUG :: last_flit_mod    =%0d", last_flit_mod);

      for ( int i=0; i<num_of_flits; i++ ) begin
        tx_data = 64'h0;
        @(drv_vi.cb);
        if ( i==0 )  begin    // ------------------------------- SOP cycle --------
--------
          tx_data     = { drv_pkt.mac_dst_addr, drv_pkt.mac_src_addr[47:32] };
          drv_vi.cb.pkt_tx_val    <= 1'b1;
          drv_vi.cb.pkt_tx_sop    <= drv_pkt.sop_mark;
          drv_vi.cb.pkt_tx_eop    <= 1'b0;
          drv_vi.cb.pkt_tx_mod    <= $urandom_range(0,7);
          drv_vi.cb.pkt_tx_data   <= tx_data;
        end                   // ------------------------------- SOP cycle --------
--------
```

```
57        else if ( i==(num_of_flits-1) ) begin // ---------------- EOP cycle --------
--------
58          if ( num_of_flits==2 ) begin
59            tx_data[63:16]    = { drv_pkt.mac_src_addr[31:0], drv_pkt.ether_type };
60            tx_data[15:0]     = $urandom_range(0,16'hFFFF);
61            for ( int j=0; j<drv_pkt.payload.size(); j++ ) begin
62              if ( j==0 ) begin
63                tx_data[15:8] = drv_pkt.payload[0];
64              end
65              else begin
66                tx_data[7:0]  = drv_pkt.payload[1];
67              end
68            end
69          end
70          else begin
71            for ( int j=0; j<8; j++ ) begin
72              if (j<(((drv_pkt.payload.size()-3)%8)+1)) begin
73                tx_data       = tx_data | ( drv_pkt.payload[8*i+j-14] << (56-8*j) );
74              end
75              else begin
76                tx_data       = tx_data | ( $urandom_range(0,8'hFF) << (56-8*j) );
77              end
78            end
79          end
80          drv_vi.cb.pkt_tx_val    <= 1'b1;
81          drv_vi.cb.pkt_tx_sop    <= 1'b0;
82          drv_vi.cb.pkt_tx_eop    <= drv_pkt.eop_mark;
83          drv_vi.cb.pkt_tx_mod    <= last_flit_mod;
84          drv_vi.cb.pkt_tx_data   <= tx_data;
85        end                   // ------------------------------- EOP cycle --------
--------
86        else begin            // ------------------------------- MOP cycle --------
--------
87          if ( i==1 ) begin
88            tx_data           = { drv_pkt.mac_src_addr[31:0], drv_pkt.ether_type,
89                                  drv_pkt.payload[0], drv_pkt.payload[1] };
90          end
91          else begin
92            for ( int j=0; j<8; j++ ) begin
93              tx_data         = (tx_data<<8) | drv_pkt.payload[8*i+j-14];
94            end
95          end
96          drv_vi.cb.pkt_tx_val    <= 1'b1;
97          drv_vi.cb.pkt_tx_sop    <= 1'b0;
98          drv_vi.cb.pkt_tx_eop    <= 1'b0;
99          drv_vi.cb.pkt_tx_mod    <= $urandom_range(0,7);
100         drv_vi.cb.pkt_tx_data   <= tx_data;
101       end                   // ------------------------------- MOP cycle --------
--------
102     end
103     drv_pkt.increase_pktid();
104     drv_pkt.print("FROM DRIVER");
105     // Put the packet into the mailbox only when the SOP/EOP
106     // signals have been correctly driven.
107     if ( drv_pkt.sop_mark && drv_pkt.eop_mark ) begin
108       drv2sb.put(drv_pkt);
109     end
110     else begin
111       $display("DRIVER :: t=%2t, ERROR PACKET, WILL NOT SEND IT TO SCOREBOARD",
$time);
```

```
112       end
113       repeat ( drv_pkt.ipg ) begin
114         @(drv_vi.cb);
115         drv_vi.cb.pkt_tx_val    <= 1'b0;
116         drv_vi.cb.pkt_tx_sop    <= 1'b0;
117         drv_vi.cb.pkt_tx_eop    <= 1'b0;
118         drv_vi.cb.pkt_tx_mod    <= $urandom_range(0,7);
119         drv_vi.cb.pkt_tx_data   <= { $urandom, $urandom_range(0,65535) };
120       end
121       if ( (j==num_packets-1) && drv_pkt.ipg==0 ) begin
122         // Explicitly close the transfer of the last packet if its
123         // inter-packet gap is zero.  Otherwise, the repeat(drv_pkt.ipg)
124         // block does not get executed the the signals are kept at the
125         // same values that they had on the last EOP cycle
126         @(drv_vi.cb);
127         drv_vi.cb.pkt_tx_val    <= 1'b0;
128         drv_vi.cb.pkt_tx_sop    <= 1'b0;
129         drv_vi.cb.pkt_tx_eop    <= 1'b0;
130         drv_vi.cb.pkt_tx_mod    <= $urandom_range(0,7);
131         drv_vi.cb.pkt_tx_data   <= { $urandom, $urandom_range(0,65535) };
132       end
133       while ( drv_vi.cb.pkt_tx_full ) begin
134         // When the pkt_tx_full signal is asserted, transfers should
135         // be suspended at the end of the current packet.  Transfer of
136         // next packet can begin as soon as this signal is de-asserted.
137         @(drv_vi.cb);
138         drv_vi.cb.pkt_tx_val    <= 1'b0;
139         drv_vi.cb.pkt_tx_sop    <= 1'b0;
140         drv_vi.cb.pkt_tx_eop    <= 1'b0;
141         drv_vi.cb.pkt_tx_mod    <= $urandom_range(0,7);
142         drv_vi.cb.pkt_tx_data   <= { $urandom, $urandom_range(0,65535) };
143       end
144     end
145   endtask : send_packet
146
147 endclass
148
```