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

interface xge_mac_interface (   input   clk_156m25,
                                input   clk_xgmii_rx,
                                input   clk_xgmii_tx,
                                input   wb_clk_i,
                                input   reset_156m25_n,
                                input   reset_xgmii_rx_n,
                                input   reset_xgmii_tx_n,
                                input   wb_rst_i            );

  logic         pkt_rx_ren, pkt_tx_eop, pkt_tx_sop, pkt_tx_val;
  logic         wb_cyc_i, wb_stb_i, wb_we_i, wb_ack_o, wb_int_o;
  logic         pkt_rx_avail, pkt_rx_eop, pkt_rx_err, pkt_rx_sop, pkt_rx_val,
pkt_tx_full;
  logic [63:0]  pkt_tx_data, xgmii_rxd, pkt_rx_data, xgmii_txd;
  logic [31:0]  wb_dat_i, wb_dat_o;
  logic [7:0]   wb_adr_i, xgmii_rxc, xgmii_txc;
  logic [2:0]   pkt_tx_mod, pkt_rx_mod;


  parameter INPUT_SKEW  = 1;
  parameter OUTPUT_SKEW = 1;

  default clocking cb @(posedge clk_156m25);
    default input   #INPUT_SKEW;
    default output  #OUTPUT_SKEW;
    input   #1  pkt_rx_avail;
    input   #1  pkt_rx_data;
    input   #1  pkt_rx_eop;
    input   #1  pkt_rx_err;
    input   #1  pkt_rx_mod;
    input   #1  pkt_rx_sop;
    input   #1  pkt_rx_val;
    input   #1  pkt_tx_full;
    input   #1  wb_ack_o;
    input   #1  wb_dat_o;
    input   #1  wb_int_o;
    input   #1  xgmii_txc;
    input   #1  xgmii_txd;
    output  #1  pkt_rx_ren;
    output  #1  pkt_tx_data;
    output  #1  pkt_tx_eop;
    output  #1  pkt_tx_mod;
    output  #1  pkt_tx_sop;
    output  #1  pkt_tx_val;
    output  #1  wb_adr_i;
    output  #1  wb_cyc_i;
    output  #1  wb_dat_i;
    output  #1  wb_stb_i;
    output  #1  wb_we_i;
    output  #1  xgmii_rxc;
```

```
60      output  #1  xgmii_rxd;
61    endclocking
62
63    // modport to connect to the testcase
64    modport testcase_port ( clocking cb );
65
66
67    // task to wait a given number of nanoseconds
68    task wait_ns(input [31:0] delay);
69      #(1000*delay);
70    endtask : wait_ns
71
72    // task to drive all the DUT input signals to some
73    // appropriate value after the DUT comes out of reset
74    task init_tb_signals();
75      pkt_rx_ren      <= 1'b0;
76      pkt_tx_data     <= { $urandom, $urandom_range(0,65535) };
77      pkt_tx_val      <= 1'b0;
78      pkt_tx_sop      <= 1'b0;
79      pkt_tx_eop      <= 1'b0;
80      pkt_tx_mod      <= $urandom_range(0,7);
81      wb_adr_i        <= $urandom_range(0,255);
82      wb_cyc_i        <= 1'b0;
83      wb_dat_i        <= $urandom;
84      wb_stb_i        <= 1'b0;
85      wb_we_i         <= $urandom_range(0,1);
86    endtask : init_tb_signals
87
88    // task to configure the DUT in loopback mode
89    // input 'xgmii_rxc' connected to output 'xgmii_txc'
90    // input 'xgmii_rxd' connected to output 'xgmii_txd'
91    task make_loopback_connection();
92      assign  xgmii_rxc = xgmii_txc;
93      assign  xgmii_rxd = xgmii_txd;
94    endtask : make_loopback_connection
95
96    // task to write into the DUT registers.  The address and
97    // the data to be written are passed as arguments while
98    // calling this task.
99    // Configuration register 0   : Address 0x00
100   // Interrupt Pending Register : Address 0x08
101   // Interrupt Status Register  : Address 0x0C
102   // Interrupt Mask Register    : Address 0x010
103   task wishbone_write_task(bit[7:0] wr_addr, bit[31:0] wr_data);
104     assert ( wr_addr==8'h00 || wr_addr==8'h08 ||
105              wr_addr==8'h0C || wr_addr==8'h10   );
106     ##10;
107     wb_adr_i    <= wr_addr;
108     wb_cyc_i    <= 1'b1;
109     wb_stb_i    <= 1'b1;
110     wb_we_i     <= 1'b1;
111     wb_dat_i    <= wr_data;
112     ##2;
113     wb_adr_i    <= $urandom_range(0,255);
114     wb_cyc_i    <= 1'b0;
115     wb_stb_i    <= 1'b0;
116     wb_we_i     <= 1'b0;
117     wb_dat_i    <= $urandom;
118     ##10;
119   endtask : wishbone_write_task
```

```
120
121   // task to read from the DUT registers.  The address to read
122   // from is passed as an argument while invoking this task.
123   // Configuration register 0   : Address 0x00
124   // Interrupt Pending Register : Address 0x08
125   // Interrupt Status Register  : Address 0x0C
126   // Interrupt Mask Register    : Address 0x010
127   task wishbone_read_task(bit[7:0] rd_addr);
128     assert ( rd_addr==8'h00 || rd_addr==8'h08 ||
129              rd_addr==8'h0C || rd_addr==8'h10   );
130     wb_adr_i    <= rd_addr;
131     wb_cyc_i    <= 1'b1;
132     wb_stb_i    <= 1'b1;
133     wb_dat_i    <= $urandom;
134     ##2;
135     wb_adr_i    <= $urandom_range(0,255);
136     wb_cyc_i    <= 1'b0;
137     wb_stb_i    <= 1'b0;
138     wb_dat_i    <= $urandom;
139     ##10;
140   endtask : wishbone_read_task
141
142
143   // ================================ Assertions
144   property no_two_consecutive_sop_without_eop;
145     @(cb) pkt_rx_sop |=> !pkt_rx_sop throughout pkt_rx_eop [->1];
146   endproperty
147   assert property ( no_two_consecutive_sop_without_eop )
148     else $error ("ASSERTION FAILED : Got 2 consecutive SOPs without EOP in
between");
149
150   property no_two_consecutive_eop_without_sop;
151     @(cb) pkt_rx_eop |=> !pkt_rx_eop throughout pkt_rx_sop [->1];
152   endproperty
153   assert property ( no_two_consecutive_eop_without_sop )
154     else $error ("ASSERTION FAILED : Got 2 consecutive EOPs without SOP in
between");
155
156   property data_never_not_unknown_on_valid;
157     @(cb) pkt_rx_val |-> not($isunknown(pkt_rx_data));
158   endproperty
159   assert property ( data_never_not_unknown_on_valid )
160     else $error ("ASSERTION FAILED : Packet RX Data is unknown when valid is
asserted");
161   // ========================================
162
163
164 endinterface : xge_mac_interface
165
```