```systemverilog
`timescale 1ns/10ps

module test;

bit iClk;
bit iRsn;
bit iBytePacketEn;
bit [7:0] iBytePacket;
wire oAddressEn;
wire [31:0] oAddress;

wire iBytePacketEN = iBytePacketEn;
wire iDataEn = iBytePacketEn;
wire [7:0] iData = iBytePacket;
wire oDataEn;
wire [31:0] oData;
assign oAddressEn = oDataEn;
assign oAddress = oData;


localparam TEST_NUM = 10;

// temporary
bit [31:0] addr;
bit [7:0] packets [0:6];
int packet_num;

int test_packet_num [0:TEST_NUM-1];
bit [7:0] test_packets [0:TEST_NUM-1][0:6];
bit [31:0] ref_addrs [0:TEST_NUM-1];

always #10ns iClk = ~iClk;

event new_test;

task reset ();
  iRsn = 0;
  #10ns;
  iRsn = 1;
  @ (posedge iClk);
  ->new_test;
endtask

initial begin
  // for (int i=0;i<8;i++) begin
  //   for (int j=0;j<3;j++) begin
  //     reset ();
  //     // testvector generation
  //     mk_tv (i); // 0~6: fixed mode, 7: random
  //     // driving
  //     drv (j); // 0: continuous, 1: 1-clock regular, 2: random
  //   end
  // end

  reset ();
  mk_tv (7); // 0~6: fixed mode, 7: random
  drv (2); // 0: continuous, 1: 1-clock regular, 2: random
  @(posedge iClk); //added//

  $finish(2);
end

function void mk_tv (int vec_mode = 7);
  for (int i=0;i<TEST_NUM;i++) begin
    mk_packet(vec_mode);
    test_packet_num[i] = packet_num;
    for (int j=0;j<packet_num;j++) begin
      test_packets[i][j] = packets[j];
    end
    ref_addrs[i] = addr;
    $write("[%03d] packet_num = %0d",i,packet_num);
    for (int j=0;j<packet_num;j++) begin
      $write("  0x%02h",packets[j]);
    end
    $display("  --> ADDR: 0x%08h",addr);
  end
endfunction

task drv (int drv_mode = 0);
  @ (posedge iClk);
  // 0: continuous, 1: one clock regular, 2: random
  for (int i=0;i<TEST_NUM;i++) begin
    for (int j=0;j<test_packet_num[i];j++) begin
      if (drv_mode == 0) begin
      end else if (drv_mode == 1) begin
        @ (posedge iClk);
      end else if (drv_mode == 2) begin
        repeat ($urandom_range(0,5)) @ (posedge iClk);
      end
      iBytePacketEn <= 1;
      iBytePacket <= test_packets[i][j];
      @ (posedge iClk);
      iBytePacketEn <= 0;
    end
  end
endtask

function void mk_packet (int mode = 7);

  bit [31:0] addr_to_update;
  bit [8:0] exception_to_update;
  bit ns_to_update;
  bit hyp_to_update;

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////// updated by sanggu


  bit has_exp, has_hyp;
  integer byte_pos = 1;
  integer addr_pos = 8;
  integer i = 0;

  if (mode == 7) begin
    mode = $urandom_range(0,4);                 // mode indicates the length of addr_bytes
    has_exp = ( mode != 0 ) && $urandom_range(0,1);
    has_hyp = has_exp && $urandom_range(0,1);
  end

  addr_to_update = $random();
  exception_to_update = $random();
  ns_to_update = $random();
  hyp_to_update = $random();
  packet_num = mode + has_exp + has_hyp + 1;
  i = 0;
  byte_pos = 1;
  addr_pos = 8;


  packets[0] = {(packet_num==1?1'b0:1'b1),addr_to_update[7:2],1'b1};

  while ( byte_pos <= mode) begin


      if( byte_pos == mode ) begin //write excp,hyp  packet

          if( byte_pos == 4 ) begin

                    packets[byte_pos][5:0] = {3'b001,addr_to_update[31:29]};
                        addr_pos += 3;

        end else begin

                addr_pos += 6;
                packets[byte_pos][5:0]  = addr_to_update[ addr_pos-1 -: 6 ];
        end


        case ( {has_hyp, has_exp} )

            0:  begin                 // exp = 0 , hyp = 0
                packets[byte_pos][7:6] = 2'b00;
            end
            1:  begin                 // exp = 1 , hyp = 0
                packets[byte_pos][7:6] = 2'b01;
                packets[byte_pos+1] = {3'b000,exception_to_update[3:0],ns_to_update} ;
            end
            3:  begin                 // exp = 1 , hyp = 1
                packets[byte_pos][7:6] = 2'b01;
                packets[byte_pos + 1] =
                {3'b100,exception_to_update[3:0],ns_to_update} ;
                packets[byte_pos + 2] = {2'h0,hyp_to_update,exception_to_update[8:4]};
            end

        endcase


          end else begin

        packets[byte_pos] = {1'b1,addr_to_update[  7 * (byte_pos + 1 ) -: 7 ] };
        addr_pos += 7;

          end

          byte_pos += 1;

  end // endwhile



while( addr_pos > 2) begin

    addr[ addr_pos-1] = addr_to_update[addr_pos-1];
    addr_pos -= 1 ;
end


/////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////   updated by sanggu




endfunction


// checker
int check_idx;
always @ (posedge iClk iff oAddressEn) begin
  if (oAddress === ref_addrs[check_idx]) begin
    $display ("(@ %0d ns) RESULT[%0d]: SUCCESS (0x%08h)",$time(),check_idx,oAddress);
  end else begin
    $display ("(@ %0d ns) RESULT[%0d]: FAIL (RTL:0x%08h vs
    REF:0x%08h)",$time(),check_idx,oAddress,ref_addrs[check_idx]);
  end
  check_idx++;
end
always @ (new_test) begin
  check_idx = 0;
end

decoder A_DUT (.*);

// general checker
bit [31:0] address_dump [0:1023];
int address_idx;
always @ (oAddress) begin
```

```
    address_dump[address_idx++] = oAddress;
  end

  final begin
    for (int i=0;i<TEST_NUM;i++) begin
      for (int j=0;j<address_idx;j++) begin
        if (address_dump[j] == ref_addrs[i]) begin
          $display ("[GEN checker] RESULT[%0d]: SUCCESS
          (RTL(%0d):0x%08h)",i,j,ref_addrs[i]);
          break;
        end
      end
    end

  end

  endmodule
```