Router Project Report

Harsha Vardhan Duvvuru - BRN47
harshavardhanreddy527@gmail.com

Router- Top-Level Block

Description:

ROUTER is a device that forwards data packets between computer networks. It is an OSI layer 3 routing device. It drives an incoming packet to an output channel based on the address field contained in the packet header.

RTL:

```
module router_top(clock,resetn,pkt_valid,read_enb_0,read_enb_1,read_enb_2,data_in,

busy,err,vld_out_0,vld_out_1,vld_out_2,data_out_0,data_out_1,data_out_2);

  input [7:0]data_in;
  input pkt_valid,clock,resetn,read_enb_0,read_enb_1,read_enb_2;
  output [7:0]data_out_0,data_out_1,data_out_2;
  output vld_out_0,vld_out_1,vld_out_2,err,busy;

          wire
soft_reset_0,full_0,empty_0,soft_reset_1,full_1,empty_1,soft_reset_2,full_2,empty_2,
        fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,
        parity_done,low_packet_valid,write_enb_reg;
          wire [2:0]write_enb;
          wire [7:0]d_in;


   //-------FiFO Instantiation-----

          router_fifo FIFO_0(.clk(clock),
                        .rstn(resetn),
                           .soft_rst(soft_reset_0),
                              .write_enb(write_enb[0]),
```

```
                .read_enb(read_enb_0),
            .lfd_state(lfd_state),
                .data_in(d_in),
                .full(full_0),
                .empty(empty_0),
                .data_out(data_out_0));


    router_fifo FIFO_1(.clk(clock),
                .rstn(resetn),
                .soft_rst(soft_reset_1),
                .write_enb(write_enb[1]),
                .read_enb(read_enb_1),
            .lfd_state(lfd_state),
                .data_in(d_in),
                .full(full_1),
                .empty(empty_1),
                .data_out(data_out_1));


    router_fifo FIFO_2(.clk(clock),
                .rstn(resetn),
                .soft_rst(soft_reset_2),
                .write_enb(write_enb[2]),
                .read_enb(read_enb_2),
            .lfd_state(lfd_state),
                .data_in(d_in),
                .full(full_2),
                .empty(empty_2),
                .data_out(data_out_2));

//-------Register Instantiation-----

    router_reg REGISTER(.clock(clock),
                .resetn(resetn),
                .pkt_valid(pkt_valid),
            .data_in(data_in),
                .fifo_full(fifo_full),
            .detect_add(detect_add),
     .ld_state(ld_state),
                .laf_state(laf_state),
                .full_state(full_state),
          .lfd_state(lfd_state),
                .rst_int_reg(rst_int_reg),
```

```
                        .err(err),
        .parity_done(parity_done),
                        .low_packet_valid(low_packet_valid),
                        .dout(d_in));

//-------Synchronizer Instantiation-----

        router_sync SYNCHRONIZER(.clock(clock),
                        .resetn(resetn),
                                .data_in(data_in[1:0]),
                                .detect_add(detect_add),
                                .full_0(full_0),
                                .full_1(full_1),
                                .full_2(full_2),
                                .empty_0(empty_0),
                                .empty_1(empty_1),
                                .empty_2(empty_2),
                                .write_enb_reg(write_enb_reg),
                                .read_enb_0(read_enb_0),
                                .read_enb_1(read_enb_1),
                                .read_enb_2(read_enb_2),
                                .write_enb(write_enb),
                                .fifo_full(fifo_full),
                                .vld_out_0(vld_out_0),
                                .vld_out_1(vld_out_1),
                                .vld_out_2(vld_out_2),
                                .soft_reset_0(soft_reset_0),
                                .soft_reset_1(soft_reset_1),
                                .soft_reset_2(soft_reset_2));

//-------FSM Instantiation-----

        router_fsm FSM(.clock(clock),
                    .resetn(resetn),
                       .pkt_valid(pkt_valid),
                       .data_in(data_in[1:0]),
                       .fifo_full(fifo_full),
                       .fifo_empty_0(empty_0),
                       .fifo_empty_1(empty_1),
                       .fifo_empty_2(empty_2),
         .soft_reset_0(soft_reset_0),
                       .soft_reset_1(soft_reset_1),
                       .soft_reset_2(soft_reset_2),
                       .parity_done(parity_done),
```

```
                .low_packet_valid(low_packet_valid),
        .write_enb_reg(write_enb_reg),
                .detect_add(detect_add),
                        .ld_state(ld_state),
                        .laf_state(laf_state),
                        .lfd_state(lfd_state),
                        .full_state(full_state),
                .rst_int_reg(rst_int_reg),
                .busy(busy));

endmodule
```

TestBench:

```
module router_top_tb();

reg clk, resetn, read_enb_0, read_enb_1, read_enb_2, packet_valid;
reg [7:0]datain;
wire [7:0]data_out_0, data_out_1, data_out_2;
wire vld_out_0, vld_out_1, vld_out_2, err, busy;
integer i;

  router_top DUT(.clock(clk),
                        .resetn(resetn),
                        .read_enb_0(read_enb_0),
                        .read_enb_1(read_enb_1),
                        .read_enb_2(read_enb_2),
                        .pkt_valid(packet_valid),
                        .data_in(datain),
                        .data_out_0(data_out_0),
                        .data_out_1(data_out_1),
                        .data_out_2(data_out_2),
                        .vld_out_0(vld_out_0),
                        .vld_out_1(vld_out_1),
                        .vld_out_2(vld_out_2),
                        .err(err),
                        .busy(busy) );

//clock generation

initial
        begin
        clk = 1;
```

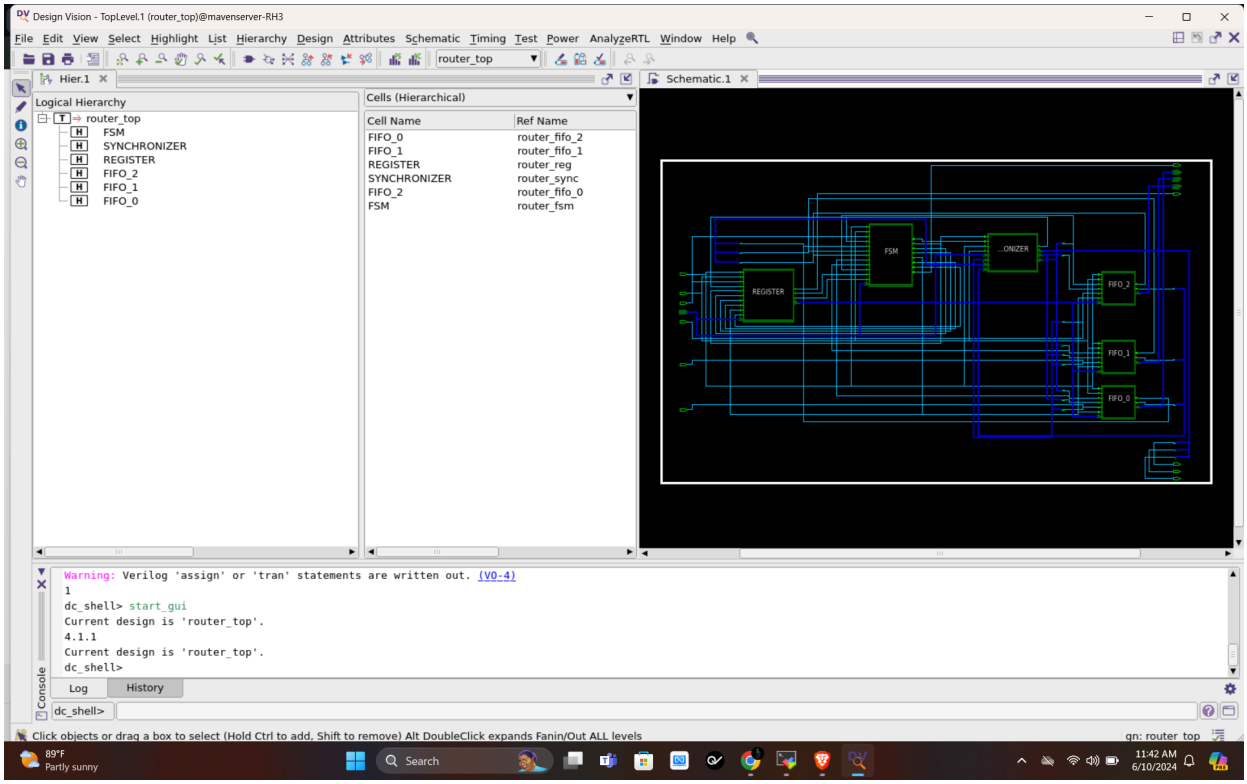
```
	forever
	#5 clk=~clk;
	end


	task reset;
		begin
		   @(negedge clk)
			resetn=1'b0;
			@(negedge clk)
			resetn=1'b1;
		end
	endtask

	task initialize;
	   begin
			resetn = 1'b1;
			{read_enb_0, read_enb_1, read_enb_2, packet_valid}=0;
		end
endtask


	task pktm_gen_5;	// packet generation payload 5
			reg [7:0]header, payload_data, parity;
			reg [8:0]payloadlen;

			begin
				parity=0;
				wait(!busy)
				begin
				@(negedge clk);
				payloadlen=5;
				packet_valid=1'b1;
				header={payloadlen,2'b10};
				datain=header;
				parity=parity^datain;
				end
				@(negedge clk);

				for(i=0;i<payloadlen;i=i+1)
					begin
					wait(!busy)
	begin
					@(negedge clk);
```

```
                                        payload_data={$random}%256;
                                        datain=payload_data;
                                        parity=parity^datain;
            end
                                        end

        wait(!busy)
            begin
                                        @(negedge clk);
                                        packet_valid=0;
                                        datain=parity;
            end
        repeat(2)
                        @(negedge clk);
                        read_enb_2=1'b1;

        wait(DUT.FIFO_2.empty)
      @(negedge clk)
      read_enb_2=0;
                        end

endtask

        task pktm_gen_14;     // packet generation payload 14
                        reg [7:0]header, payload_data, parity;
                        reg [8:0]payloadlen;

                        begin
                                parity=0;
                                wait(!busy)
                                begin
                                @(negedge clk);
                                payloadlen=14;
                                packet_valid=1'b1;
                                header={payloadlen,2'b01};
                                datain=header;
                                parity=parity^datain;
                                end
                                @(negedge clk);

                                for(i=0;i<payloadlen;i=i+1)
                                        begin
                                        wait(!busy)
            begin
```

```
                                        @(negedge clk);
                                        payload_data={$random}%256;
                                        datain=payload_data;
                                        parity=parity^datain;
            end
                                        end

          wait(!busy)
            begin
                                        @(negedge clk);
                                        packet_valid=0;
                                        datain=parity;
            end
          repeat(2)
                        @(negedge clk);
                        read_enb_1=1'b1;

          wait(DUT.FIFO_1.empty)
        @(negedge clk)
        read_enb_1=0;
                        end
endtask

        task pktm_gen_17;     // packet generation payload 17
                        reg [7:0]header, payload_data, parity;
                        reg [8:0]payloadlen;

                        begin
                                parity=0;
                                wait(!busy)
                                begin
                                @(negedge clk);
                                payloadlen=17;
                                packet_valid=1'b1;
                                header={payloadlen,2'b00};
                                datain=header;
                                parity=parity^datain;
                                end
                                @(negedge clk);

                                for(i=0;i<payloadlen;i=i+1)
                                        begin
                                        wait(!busy)
            begin
```

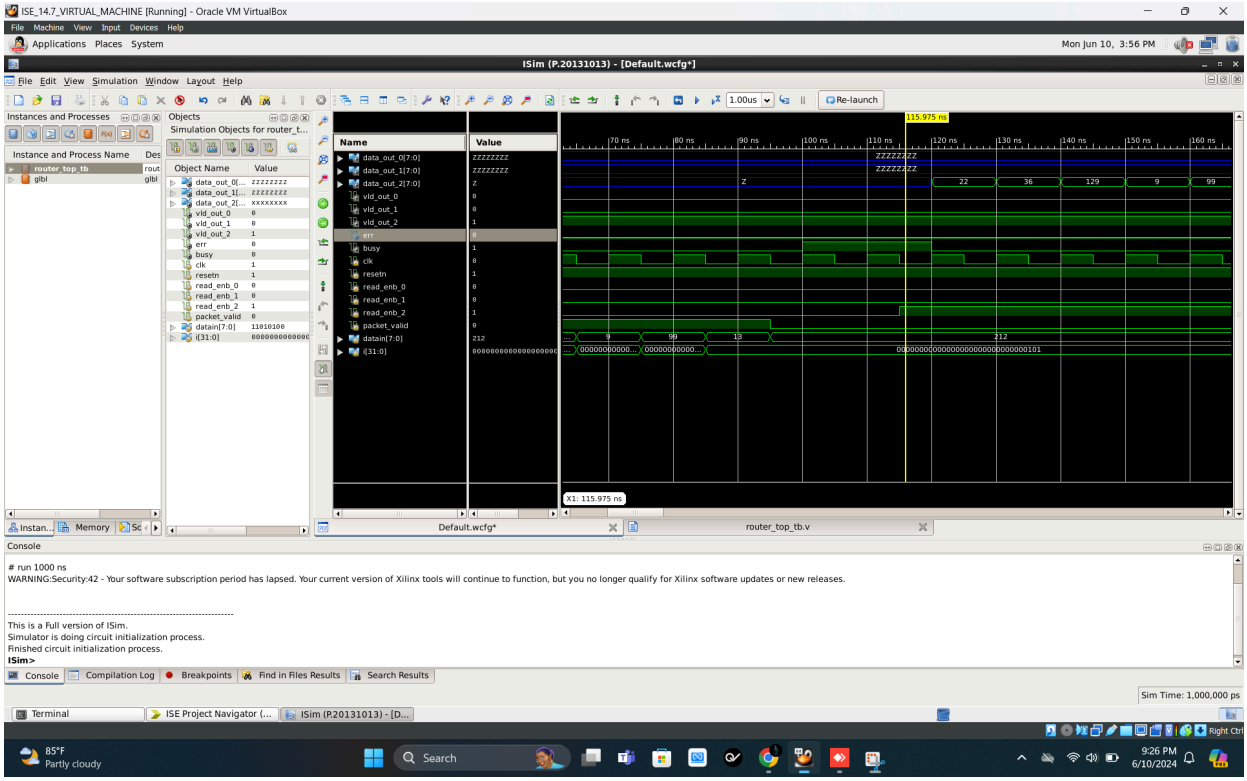
```
                                    @(negedge clk);
                                    payload_data={$random}%256;
                                    datain=payload_data;
                                    parity=parity^datain;
            end
                                    end

          wait(!busy)
            begin
                                    @(negedge clk);
                                    packet_valid=0;
                                    datain=parity;
            end
          repeat(2)
                        @(negedge clk);
                        read_enb_0=1'b1;

          wait(DUT.FIFO_0.empty)
        @(negedge clk)
        read_enb_0=0;
                        end
endtask



        initial
                begin
                  initialize;
                        reset;
                        #10;
                        pktm_gen_5;
        #100;
        //reset;
                        pktm_gen_14;
                        #100;
                        pktm_gen_17;
                        #700;
                        $finish;
                end

endmodule
```

Synthesis:

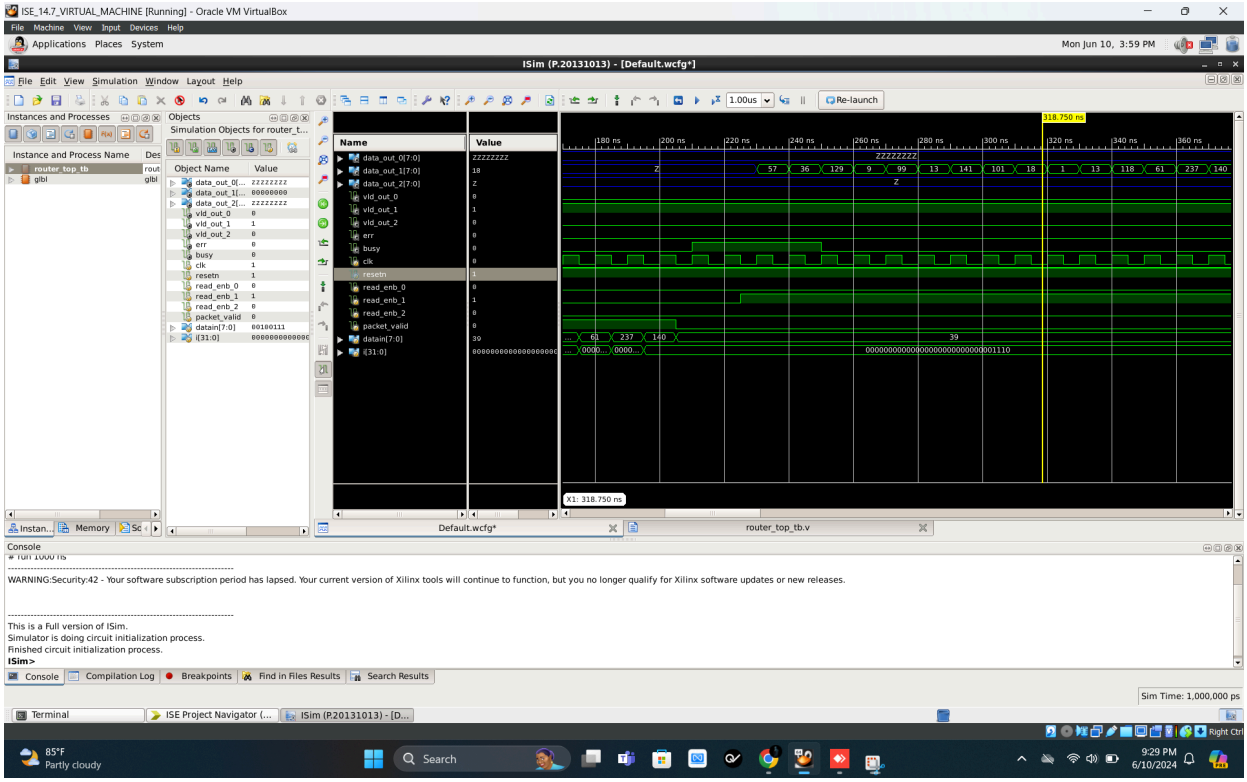


Simulation:

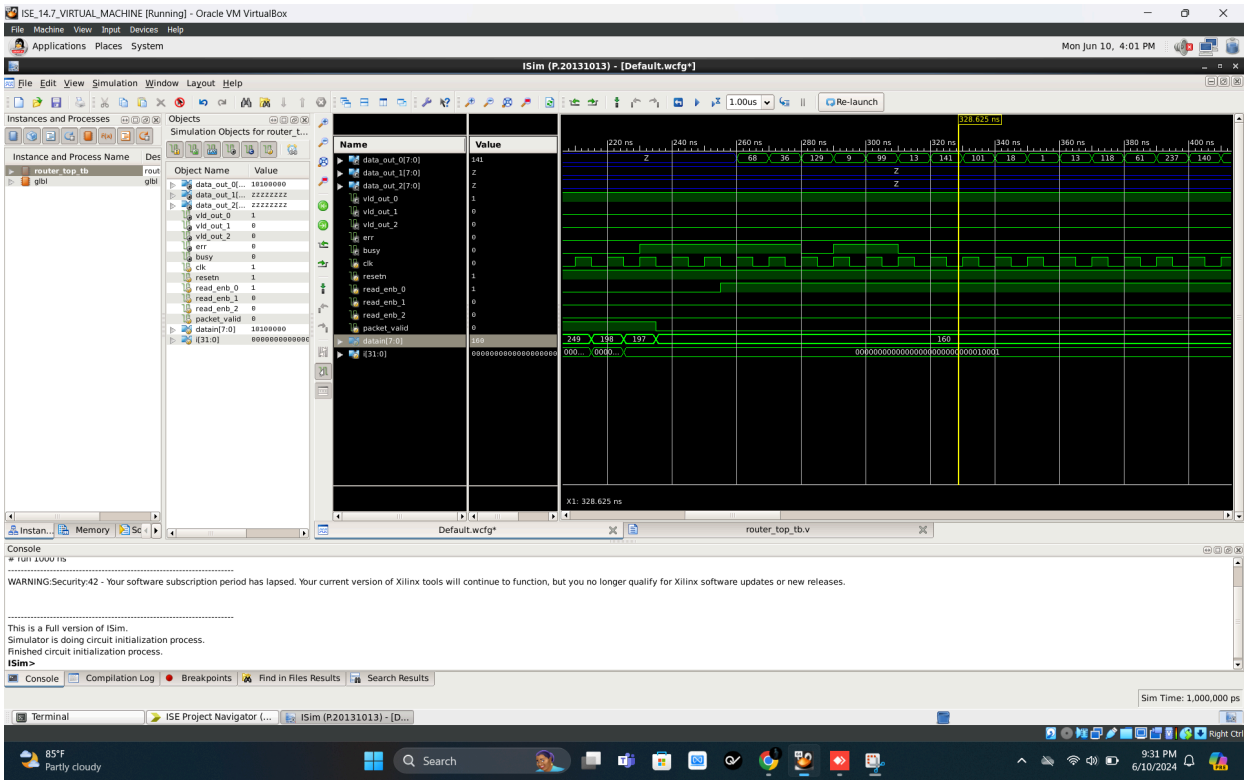
1) Payload Length = 5



2) Payload Length = 14



3) Payload Length = 17



Router- FIFO

Description: There are 3 FIFOs used in the router design. Each FIFO is of 9 bits width and of 16 locations as depth.The FIFO works on the system clock and is reset with a synchronous active low reset. The FIFO is also internally reset by an internal reset signal soft_reset .soft_reset is an active high signal which is generated by the SYNCHRONIZER block during the time out state of the ROUTER

RTL:

```
module router_fifo(clk,rstn,soft_rst,write_enb,read_enb,data_in,lfd_state,data_out,empty,full);
        //params declaration
        parameter FIFO_DEPTH = 16;
        parameter FIFO_WIDTH = 9;
```

```
// ports declaration

input clk,rstn,soft_rst,write_enb,read_enb,lfd_state;
input [7:0] data_in;
output reg [7:0] data_out;
output empty,full;
integer i;
reg [5:0]counter;
// memory declaration

reg [FIFO_WIDTH-1:0]mem[FIFO_DEPTH-1:0];

reg [4:0] rp,wp;


// write logic

always@(posedge clk)
	begin
		if(!rstn)
			begin
				for(i=0;i<FIFO_DEPTH;i=i+1)
					begin
						mem[i]<=0;
					end
				wp<=0;
			end
		else if(soft_rst)
			begin
				for(i=0;i<FIFO_DEPTH;i=i+1)
					begin
						mem[i]<=0;
					end
				wp<=0;
			end
		else if(write_enb && !full)
			begin
				mem[wp]<={lfd_state,data_in};
				wp<=wp+1;
			end
	end

// data reading logic
```

```
always@(posedge clk)
	begin
		if(!rstn)
			begin
				data_out<=8'b0;
				rp<=0;
			end
		else if(soft_rst)
			begin
				data_out<=8'bz;
				rp<=0;
			end

		else if(read_enb && !empty)
			begin
				data_out<=mem[rp];
				rp<=rp+1;
			end
		else if(counter==1'b0)
			begin
				data_out<=8'bz;
			end
	end

// counter logic
always@(posedge clk)
	begin
		if(!rstn || soft_rst)
		begin
			counter<=0;
		end
		else if(read_enb && !empty)
			if(mem[0][8]==1)
				begin
					counter<=data_in[7:2]+1;
				end
			else
				begin
					counter<=counter-1;
				end
	end


// empty and fill status assignment
```

```
        assign full = (wp==5'b10000 && rp==5'b0000);
        assign empty = (wp==rp);

endmodule
```

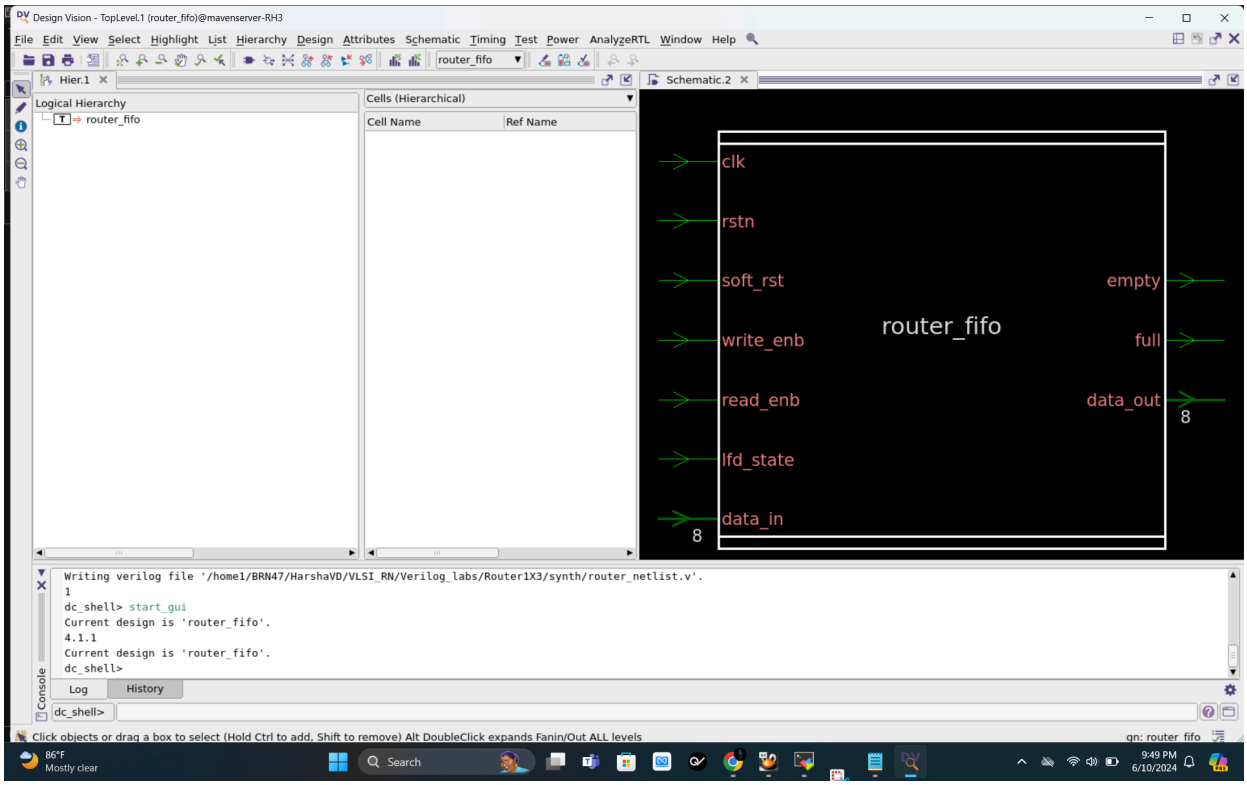
Test Bench:

```
module router_fifo_tb();

 // Inputs
 reg clock;
 reg resetn;
 reg write_enb;
 reg soft_reset;
 reg read_enb;
 reg [7:0] data_in;
 reg lfd_state;

 integer i,j;

 // Outputs
 wire empty;
 wire [7:0] data_out;
 wire full;

 // Instantiate the Unit Under Test (UUT)
 router_fifo uut (
  .clk(clock),
  .rstn(resetn),
  .write_enb(write_enb),
  .soft_rst(soft_reset),
  .read_enb(read_enb),
  .data_in(data_in),
  .lfd_state(lfd_state),
  .empty(empty),
  .data_out(data_out),
  .full(full)
 );

// Clock generation
    always #5 clock = ~clock;
```

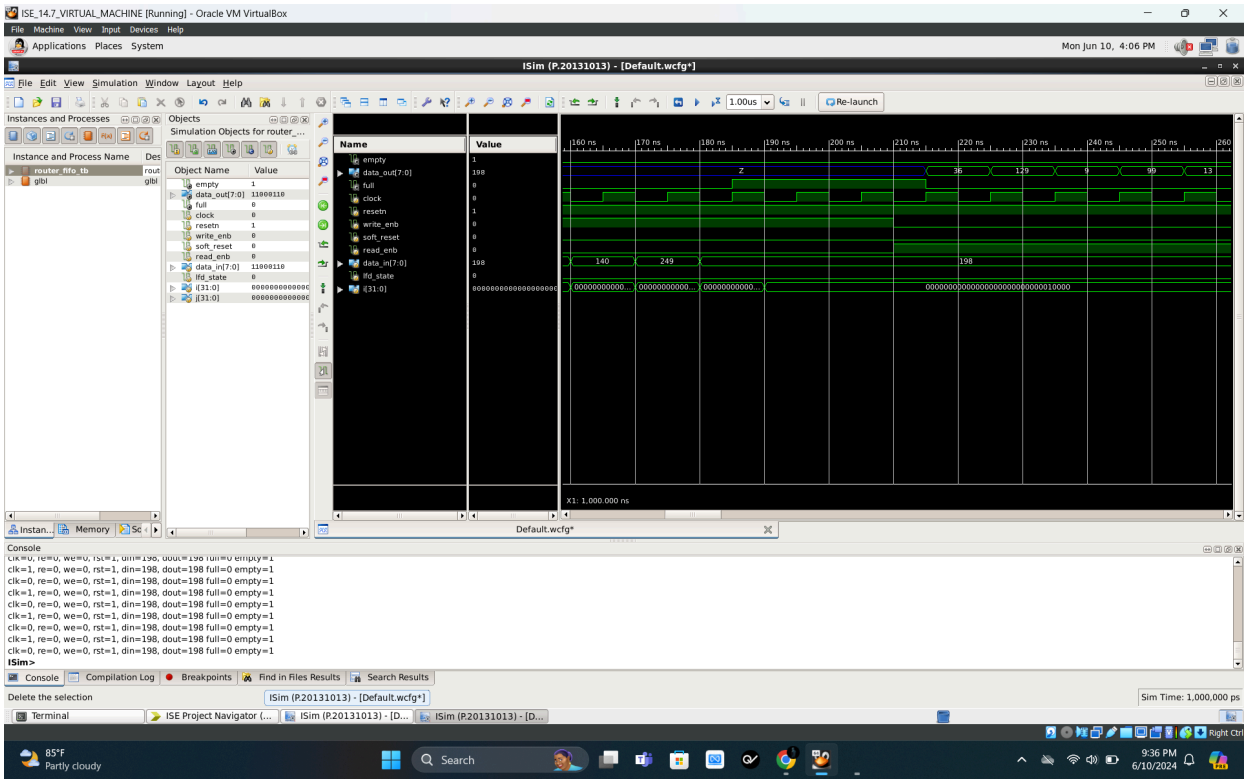
```
    task write(input [7:0] in, input e);
            begin
                    @(negedge clock);
                    write_enb=e;
                    data_in=in;
            end
    endtask

    task read(input ee);
            begin
                    @(negedge clock);
                    read_enb=ee;
            end
    endtask

    task reset;
            begin
            @(negedge clock);resetn=1'b0;
            @(negedge clock);resetn=1'b1;
            end
    endtask

    initial
            begin
              clock = 0;
resetn = 1;
soft_reset = 0;
write_enb = 0;
read_enb = 0;
data_in = 9'b0;
// Apply reset
reset();
                    for(i=0;i<16;i=i+1)
                            begin
                                    if(i==0)
                                            begin
                                                    lfd_state = 1;
                                                    write({$random},1);
                                            end
                                    else
                                            begin
                                                    lfd_state = 0;
                                                    write({$random},1);
                                            end
```

```
                #10;
                end
            #20;write_enb=0;
            for(j=0;j<16;j=j+1)
                begin
                    read(1);
                    #10;
                end
            #20;read_enb=0;
        end
    initial $monitor("clk=%b, re=%b, we=%b, rst=%b, din=%d, dout=%d full=%d empty=%d"
    ,clock, read_enb, write_enb, resetn, data_in, data_out,uut.full,uut.empty);
    initial #10000 $finish();

endmodule
```

Synthesis:



Simulation:



Router - Synchronizer:

Description: This module provides synchronization between router FSM and router FIFO modules. It provides faithful communication between the single input port and three output ports.

RTL:

module router_sync(clock,resetn,data_in,detect_add,full_0,full_1,full_2,empty_0,empty_1,empty_2,write_enb_reg,read_enb_0,read_enb_1,read_enb_2,write_enb,fifo_full,vld_out_0,vld_out_1,vld_out_2,soft_reset_0,soft_reset_1,soft_reset_2);

input clock,resetn,detect_add,full_0,full_1,full_2,empty_0,empty_1,empty_2,write_enb_reg,read_enb_0,read_enb_1,read_enb_2;
input [1:0]data_in;
output reg[2:0]write_enb;
output reg fifo_full,soft_reset_0,soft_reset_1,soft_reset_2;

```
output vld_out_0,vld_out_1,vld_out_2;

  reg [1:0] data_in_tmp;
  reg[4:0]count0,count1,count2;

  always@(posedge clock)
  begin
    if(~resetn)
    data_in_tmp<=0;
    else if(detect_add)
    data_in_tmp<=data_in;
  end



//-----------Address decoding & fifo empty ----------------
always@(*)
  begin
    case(data_in_tmp)
    2'b00:begin
              fifo_full<=full_0;
              if(write_enb_reg)
              write_enb<=3'b001;
              else
              write_enb<=0;
              end
    2'b01:begin
              fifo_full<=full_1;
              if(write_enb_reg)
              write_enb<=3'b010;
              else
              write_enb<=0;
              end
    2'b10:begin
              fifo_full<=full_2;
              if(write_enb_reg)
              write_enb<=3'b100;
              else
              write_enb<=0;
              end
    default:begin
              fifo_full<=0;
              write_enb<=0;
              end
```

```
   endcase
  end



//----------------------------------Valid Byte block--------------------------------

assign vld_out_0 = (~empty_0);
assign vld_out_1 = (~empty_1);
assign vld_out_2 = (~empty_2);


//----------------------------------Soft Reset block--------------------------------

always@(posedge clock)
  begin

  if(~resetn)
  begin
  count0<=0;
  soft_reset_0<=0;
  end

  else if(vld_out_0)
  begin
  if(~read_enb_0)

    begin
    if(count0==29)
      begin
      soft_reset_0<=1'b1;
      count0<=0;
      end
    else
      begin
      soft_reset_0<=1'b0;
      count0<=count0+1'b1;
      end
    end
  else
  count0<=0;
  end
  end
```

```
always@(posedge clock)
  begin

  if(~resetn)
  begin
  count1<=0;
  soft_reset_1<=0;
  end

  else if(vld_out_1)
  begin
  if(~read_enb_1)

    begin
    if(count1==29)
      begin
      soft_reset_1<=1'b1;
      count1<=0;
      end
    else
      begin
      soft_reset_1<=1'b0;
      count1<=count1+1'b1;
      end
    end
  else
  count1<=0;
  end
  end

always@(posedge clock)
  begin

  if(~resetn)
  begin
  count2<=0;
  soft_reset_2<=0;
  end

  else if(vld_out_2)
  begin
  if(~read_enb_2)

    begin
```

```
    if(count2==29)
      begin
      soft_reset_2<=1'b1;
      count2<=0;
      end
    else
      begin
      soft_reset_2<=1'b0;
      count2<=count2+1'b1;
      end
    end
  else
  count2<=0;
  end
  end

endmodule
```

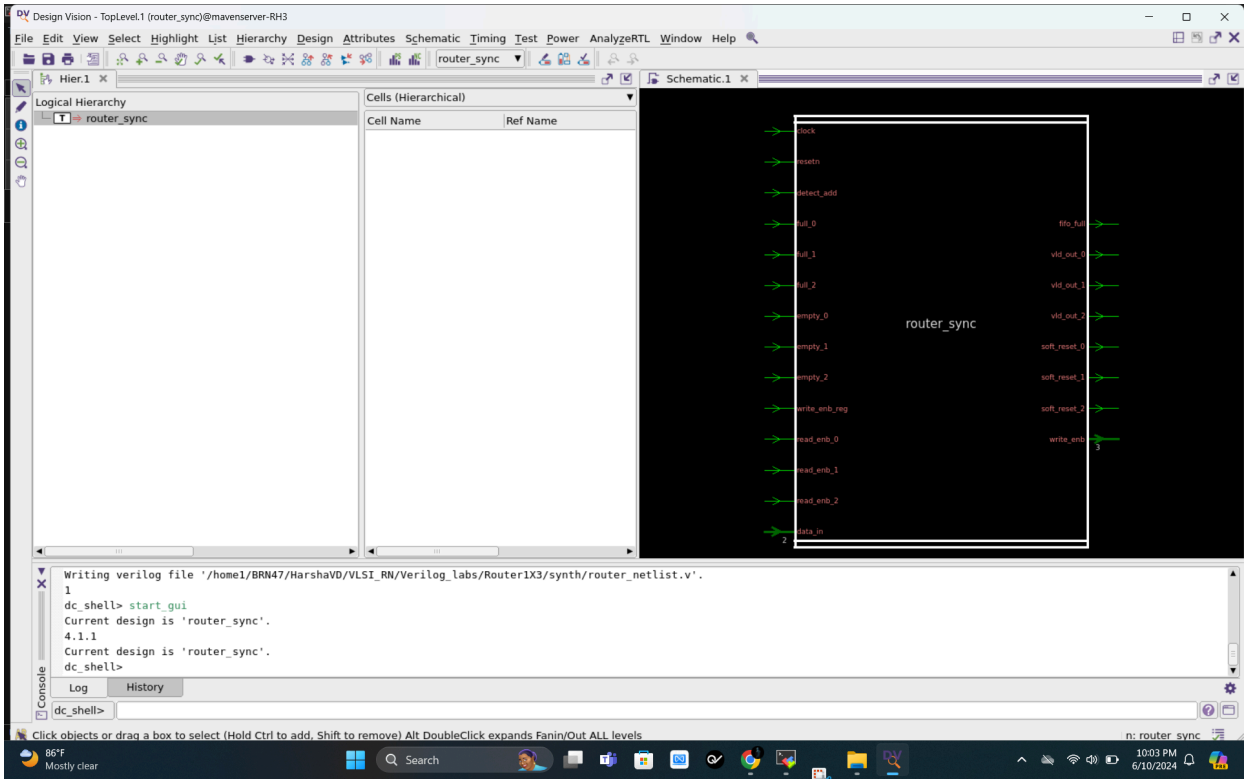
Test Bench:

```
module router_sync_tb();
wire [2:0]write_enb;
wire fifo_full;
wire vld_out_0,vld_out_1,vld_out_2;
wire soft_reset_0,soft_reset_1,soft_reset_2;
reg  clock,resetn,detect_add;
reg  [1:0]data_in;
reg  full_0,full_1,full_2;
reg  empty_0,empty_1,empty_2;
reg write_enb_reg;
reg read_enb_0,read_enb_1,read_enb_2;
parameter CYCLE=10;

router_sync
DUT(clock,resetn,data_in,detect_add,full_0,full_1,full_2,empty_0,empty_1,empty_2,write_enb_r
eg,read_enb_0,read_enb_1,read_enb_2,write_enb,fifo_full,vld_out_0,vld_out_1,vld_out_2,soft_
reset_0,soft_reset_1,soft_reset_2);

          initial
            begin
              clock=1'b0;
```

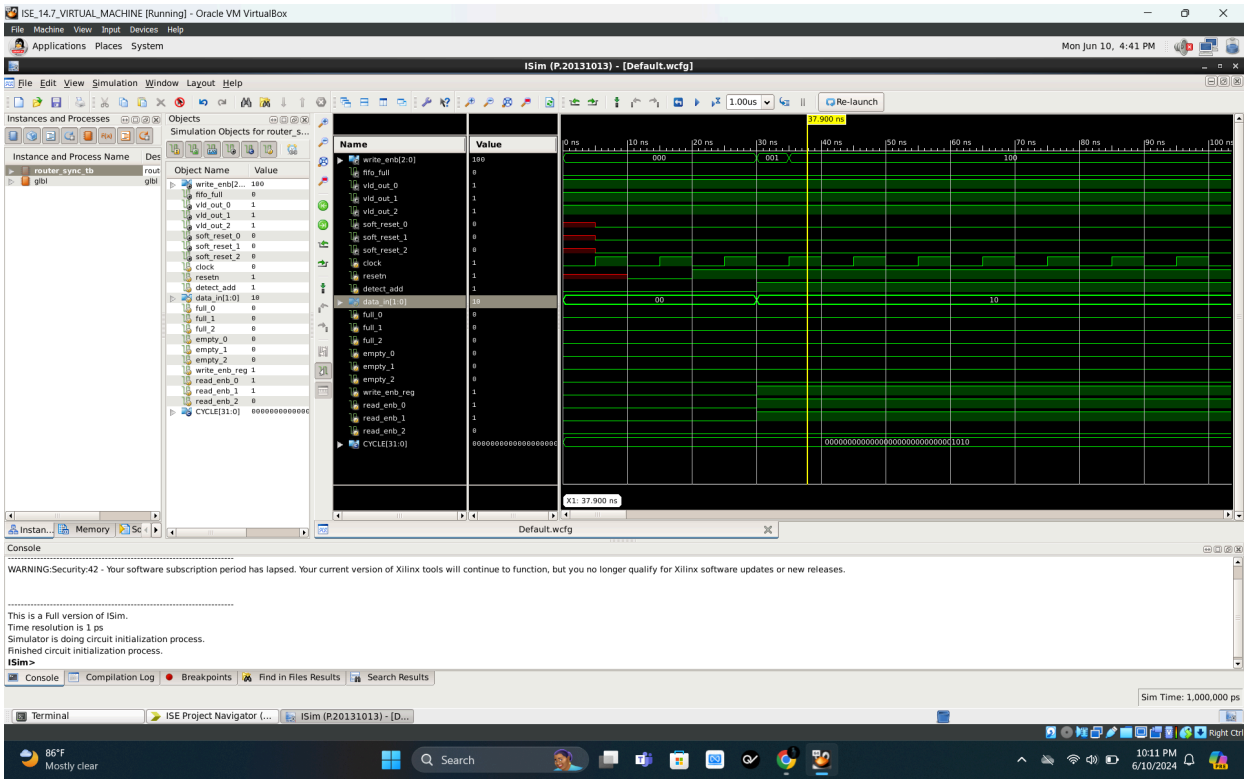
```
            forever #(CYCLE/2) clock=~clock;
          end
//task initialize
        task initialize;
              begin
                    {detect_add,data_in,full_0,full_1,full_2}=0;

{write_enb_reg,read_enb_0,read_enb_1,read_enb_2,empty_0,empty_1,empty_2}=0;

               end
              endtask
//task reset
             task reset_dut();
             begin
               @(negedge clock)
                 resetn=1'b0;
                          @(negedge clock)
                 resetn=1'b1;
             end
           endtask
//task readenb
            task readenb(input r1,r2,r3);
              begin
                  {read_enb_0,read_enb_1,read_enb_2}={r1,r2,r3};
                          end
            endtask
//task input and detect addresee
             task detect_ad(input [1:0]d1,input detect_ad1);
               begin
                 data_in=d1;
                            detect_add=detect_ad1;
               end
                  endtask
//task fifo_full
                  task fifo_ful(input f1,f2,f3);
               begin
                 full_0=f1;
                 full_1=f2;
                 full_2=f3;
                end
            endtask
//task empty
            task empty_dut(input e1,e2,e3);
              begin
```

```
                    empty_0=e1;
                    empty_1=e2;
            empty_2=e3;
                                end
                  endtask

 //task write enb reg
          task write_reg(input l1);
             begin
             write_enb_reg=l1;

             end
          endtask
     initial
      begin
             initialize;
             reset_dut();
          @(negedge clock)
             readenb(1,1,0);
     detect_ad(2'b10,1);
       fifo_ful(0,0,0);
     write_reg(1);
       empty_dut(0,0,0);
          #1000;
       $finish;
      end

endmodule
```

Synthesis:


Design Vision - TopLevel.1 (router_sync)@mavenserver-RH3
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
router_sync
Hier.1
Logical Hierarchy
router_sync
Cells (Hierarchical)
Cell Name
Ref Name
Schematic.1
router_sync
clock
resetn
detect_add
full_0
full_1
full_2
empty_0
empty_1
empty_2
write_enb_reg
read_enb_0
read_enb_1
read_enb_2
data_in
fifo_full
vld_out_0
vld_out_1
vld_out_2
soft_reset_0
soft_reset_1
soft_reset_2
write_enb
Writing verilog file '/home1/BRN47/HarshaVD/VLSI_RN/Verilog_labs/Router1X3/synth/router_netlist.v'.
1
dc_shell> start_gui
Current design is 'router_sync'.
4.1.1
Current design is 'router_sync'.
dc_shell>
Log
History
Console
dc_shell>
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove) Alt DoubleClick expands Fanin/Out ALL levels
n: router_sync
86°F
Mostly clear
Search
10:03 PM
6/10/2024

Simulation:



Router - FSM:

Description: The FSM module is the controller circuit for the ROUTER. This module generates all the control signals when a new packet is received by the ROUTER. These control signals are used by other design components in order to transfer the packet to the output port

RTL:

```
module
router_fsm(clock,resetn,pkt_valid,data_in,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,
                soft_reset_0,soft_reset_1,soft_reset_2,parity_done,low_packet_valid,
                write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy);

                    input clock,resetn,pkt_valid,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2;
                    input soft_reset_0,soft_reset_1,soft_reset_2,parity_done,low_packet_valid;
                    input [1:0] data_in;

                    output
write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy;
```

```
        parameter DECODE_ADDRESS     = 3'b000,
                                  LOAD_FIRST_DATA  = 3'b001,
                                  LOAD_DATA        = 3'b010,
                                  WAIT_TILL_EMPTY  = 3'b011,
                                  CHECK_PARITY_ERROR = 3'b100,
                                  LOAD_PARITY      = 3'b101,
                                  FIFO_FULL_STATE  = 3'b110,
                                  LOAD_AFTER_FULL  = 3'b111;

        reg [2:0] PS,NS;

    always@(posedge clock)
        begin
            if(!resetn)
              PS <= DECODE_ADDRESS;
            else if(soft_reset_0 || soft_reset_1 || soft_reset_2)
              PS <= DECODE_ADDRESS;
            else
              PS <= NS;
        end

        always@(*)
  begin
    NS = DECODE_ADDRESS;
    case(PS)

     DECODE_ADDRESS :
          begin
            if((pkt_valid && (data_in[1:0]==0) && fifo_empty_0)||
              (pkt_valid && (data_in[1:0]==1) && fifo_empty_1)||
              (pkt_valid && (data_in[1:0]==2) && fifo_empty_2))

              NS = LOAD_FIRST_DATA;
                                        else if((pkt_valid && (data_in[1:0]==0) &&
(~fifo_empty_0))||
                 (pkt_valid && (data_in[1:0]==1) && (~fifo_empty_1))||
                 (pkt_valid && (data_in[1:0]==2) && (~fifo_empty_2)))

              NS = WAIT_TILL_EMPTY;

             else
              NS = DECODE_ADDRESS;
          end
```

```
                LOAD_FIRST_DATA :  NS = LOAD_DATA;

        LOAD_DATA      :
                begin
                                        if(fifo_full)
                                                NS=FIFO_FULL_STATE;
                 else if(!fifo_full && !pkt_valid)
                                                NS=LOAD_PARITY;
                                        else
                                                NS=LOAD_DATA;
                                        end
                 WAIT_TILL_EMPTY  :
                                    begin
                                      if((!fifo_empty_0) || (!fifo_empty_1) || (!fifo_empty_2))
                                NS=WAIT_TILL_EMPTY;
                                  else if(fifo_empty_0||fifo_empty_1||fifo_empty_2)
                                NS=LOAD_FIRST_DATA;
                                  else
                                NS=WAIT_TILL_EMPTY;
                                        end
                CHECK_PARITY_ERROR:
                                begin
                                        if(fifo_full)
                                                NS=FIFO_FULL_STATE;
                                        else
                                        NS=DECODE_ADDRESS;

                                         end
                LOAD_PARITY          :       NS=CHECK_PARITY_ERROR;


                                 FIFO_FULL_STATE    :
                                                                begin
                                                                        if(!fifo_full)

NS=LOAD_AFTER_FULL;
                                                                                else if(fifo_full)

NS=FIFO_FULL_STATE;
                                                                end

                LOAD_AFTER_FULL:
                                begin
                                        if((!parity_done) && (!low_packet_valid))
```

```
                                        NS=LOAD_DATA;
                                else if((!parity_done) && (low_packet_valid))
                                        NS=LOAD_PARITY;
                                else if(parity_done)
                                        NS=DECODE_ADDRESS;
                                        end

    endcase

  end

                assign detect_add = ((PS==DECODE_ADDRESS)?1:0);
                assign
write_enb_reg=((PS==LOAD_DATA||PS==LOAD_PARITY||PS==LOAD_AFTER_FULL)?1:0);
                assign full_state=((PS==FIFO_FULL_STATE)?1:0);
                assign lfd_state=((PS==LOAD_FIRST_DATA)?1:0);
                assign
busy=((PS==FIFO_FULL_STATE||PS==LOAD_AFTER_FULL||PS==WAIT_TILL_EMPTY||

PS==LOAD_FIRST_DATA||PS==LOAD_PARITY||PS==CHECK_PARITY_ERROR)?1:0);
                assign ld_state=((PS==LOAD_DATA)?1:0);
                assign laf_state=((PS==LOAD_AFTER_FULL)?1:0);
                assign rst_int_reg=((PS==CHECK_PARITY_ERROR)?1:0);


endmodule
```

Test Bench:

```
module router_fsm_tb();

        reg
clock,resetn,pkt_valid,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_reset_
1,soft_reset_2,parity_done,low_packet_valid;
        reg [1:0] data_in;
        wire write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy;

        parameter cycle=10;

        router_fsm
DUT(clock,resetn,pkt_valid,data_in,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_
```

```
0,soft_reset_1,soft_reset_2,parity_done,low_packet_valid,write_enb_reg,detect_add,ld_state,laf
_state,lfd_state,full_state,rst_int_reg,busy);

  parameter DECODE_ADDRESS       = 3'b000,
                            LOAD_FIRST_DATA   = 3'b001,
                            LOAD_DATA                 = 3'b010,
                            WAIT_TILL_EMPTY    = 3'b011,
                            CHECK_PARITY_ERROR   = 3'b100,
                            LOAD_PARITY                   = 3'b101,
                            FIFO_FULL_STATE    = 3'b110,
                            LOAD_AFTER_FULL             = 3'b111;

  reg [3*8:0]string_cmd;

  always@(DUT.PS)
     begin
      case (DUT.PS)
            DECODE_ADDRESS     :  string_cmd = "DA";
            LOAD_FIRST_DATA    :  string_cmd = "LFD";
            LOAD_DATA             :  string_cmd = "LD";
            WAIT_TILL_EMPTY    :  string_cmd = "WTE";
            CHECK_PARITY_ERROR :  string_cmd = "CPE";
            LOAD_PARITY         :  string_cmd = "LP";
            FIFO_FULL_STATE    :  string_cmd = "FFS";
            LOAD_AFTER_FULL    :  string_cmd = "LAF";
            endcase
     end

  initial
  begin
   clock=1'b1;
   forever #cycle clock = ~clock;
  end

  task initialize;
  begin
  {pkt_valid,fifo_empty_0,fifo_empty_1,fifo_empty_2,fifo_full,parity_done,low_packet_valid}=0;
  end
  endtask

  task rst;
  begin
  @(negedge clock)
   resetn=1'b0;
```

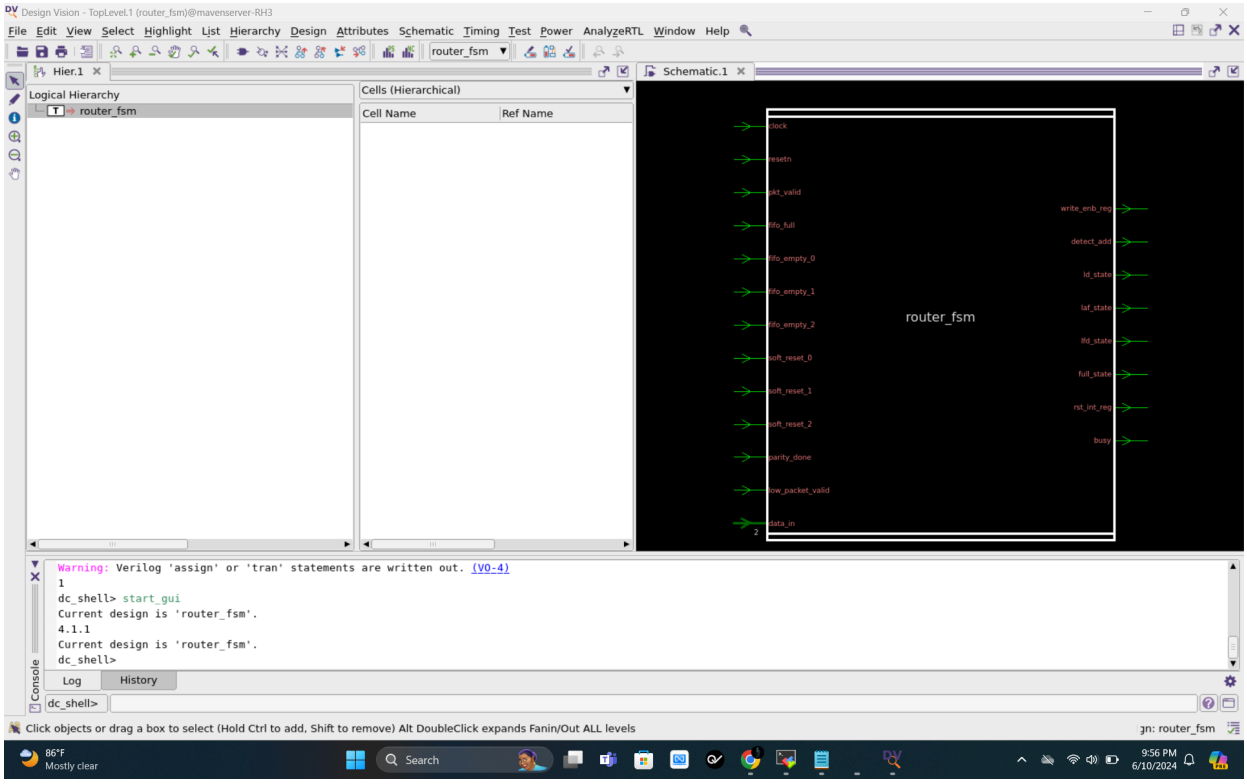
```
@(negedge clock)
 resetn=1'b1;
end
endtask

          task t1;
begin
@(negedge clock)  // LFD
begin
pkt_valid<=1;
data_in[1:0]<=0;
fifo_empty_0<=1;
end
@(negedge clock) //LD
@(negedge clock) //LP
begin
fifo_full<=0;
pkt_valid<=0;
end
@(negedge clock) // CPE
@(negedge clock) // DA
fifo_full<=0;
end
endtask

        task t2;
begin
@(negedge clock)//LFD
begin
pkt_valid<=1;
data_in[1:0]<=0;
fifo_empty_0<=1;
end
@(negedge clock)//LD
@(negedge clock)//FFS
fifo_full<=1;
@(negedge clock)//LAF
fifo_full<=0;
@(negedge clock)//LP
begin
parity_done<=0;
low_packet_valid<=1;
end
@(negedge clock)//CPE
```

```
  @(negedge clock)//DA
  fifo_full<=0;
  end
  endtask

task t3;
  begin
  @(negedge clock) //LFD
  begin
  pkt_valid<=1;
  data_in[1:0]<=0;
  fifo_empty_0<=1;
  end
  @(negedge clock) //LD
  @(negedge clock) // FFS
  fifo_full<=1;
  @(negedge clock) // LAF
  fifo_full<=0;
  @(negedge clock)  // LD
  begin
    low_packet_valid<=0;
        parity_done<=0;

  end  // LP
  @(negedge clock)
  begin
  fifo_full<=0;
  pkt_valid<=0;
  end
  @(negedge clock) // CPE
  @(negedge clock) // DA
  fifo_full<=0;
  end
  endtask

          task t4;
  begin
  @(negedge clock)  // LFD
  begin
  pkt_valid<=1;
  data_in[1:0]<=0;
  fifo_empty_0<=1;
  end
  @(negedge clock)   // LD
```

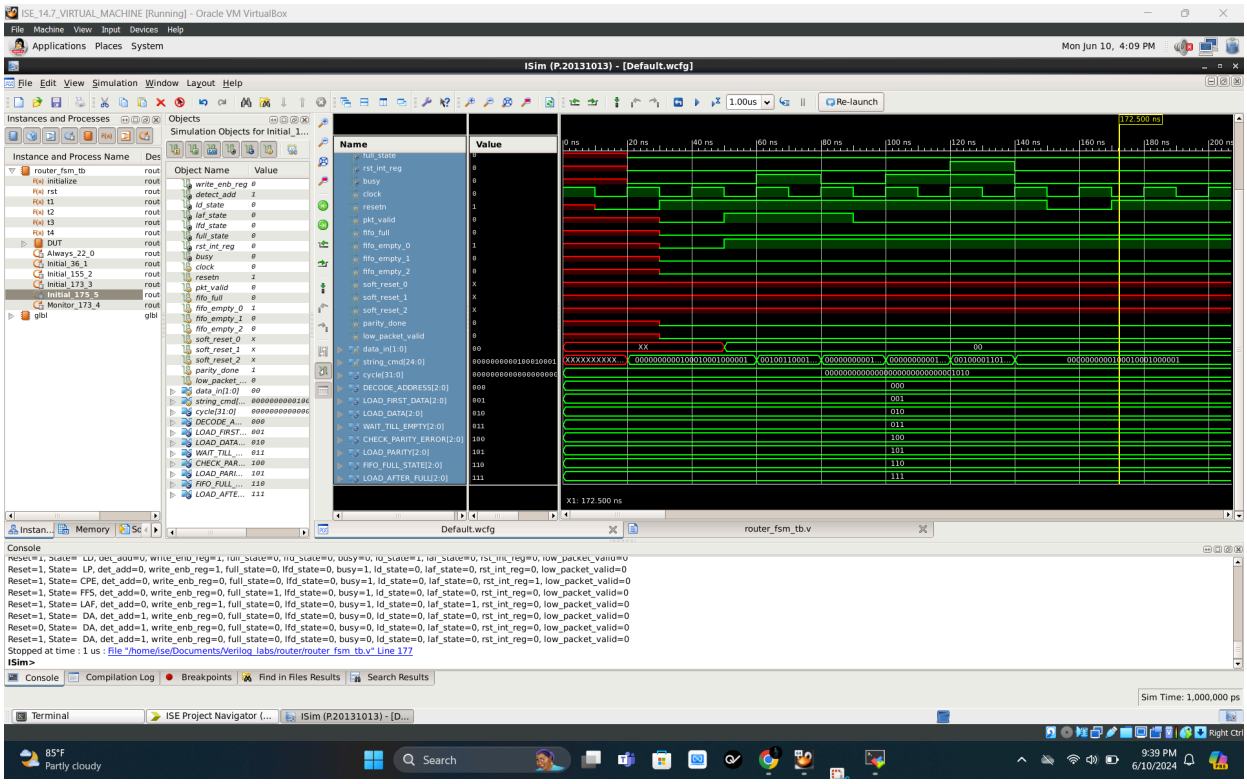
```
  @(negedge clock)   // LP
  begin
  fifo_full<=0;
  pkt_valid<=0;
  end
  @(negedge clock)   // CPE
  @(negedge clock)   // FFS
  fifo_full<=1;
  @(negedge clock)   // LAF
  fifo_full<=0;
 @(negedge clock)    // DA
  parity_done=1;
  end
  endtask

  initial
                  begin
                  rst;
                  initialize;

                          t1;
                          rst;
                          #30
                          t2;
                          rst;
                          #30
                          t3;
                          rst;
                          #30
                          t4;
                          rst;
                  end

          initial $monitor("Reset=%b, State=%s, det_add=%b, write_enb_reg=%b, full_state=%b,
lfd_state=%b, busy=%b, ld_state=%b, laf_state=%b, rst_int_reg=%b,
low_packet_valid=%b",resetn,string_cmd,detect_add,write_enb_reg,full_state,lfd_state,busy,ld_
state,laf_state,rst_int_reg,low_packet_valid);

  initial
  begin
  #1000 $finish;
  end
```

endmodule

Synthesis:



Simulation:



Router - Register:

Description: This module implements 4 internal registers in order to hold header byte, FIFO full state byte ,internal parity and packet parity byte

RTL:

```
module router_reg(clock,resetn,pkt_valid,data_in,fifo_full,detect_add,
                  ld_state,laf_state,full_state,lfd_state,rst_int_reg,err,
                                                    parity_done,low_packet_valid,dout);

        input
clock,resetn,pkt_valid,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg;
        input [7:0]data_in;
        output reg err,parity_done,low_packet_valid;
        output reg [7:0]dout;
        reg [7:0]header,int_reg,int_parity,ext_parity;

        //------------------------------DATA OUT LOGIC--------------------------------

        always@(posedge clock)
```

```
                begin
                if(!resetn)
                        begin
                  dout          <=0;
                  header        <=0;
                  int_reg       <=0;
                        end
                else if(detect_add && pkt_valid && data_in[1:0]!=2'b11)
                  header<=data_in;
                else if(lfd_state)
                  dout<=header;
                else if(ld_state && !fifo_full)
                  dout<=data_in;

                else if(ld_state && fifo_full)
            int_reg<=data_in;
     else if(laf_state)
            dout<=int_reg;
    end

 //--------------------------LOW PACKET VALID LOGIC---------------------------

          always@(posedge clock)
                begin
                  if(!resetn)
                        low_packet_valid<=0;
                        else if(rst_int_reg)
                        low_packet_valid<=0;

                  else if(ld_state && !pkt_valid)
                                low_packet_valid<=1;
          end

//--------------------------PARITY DONE LOGIC-------------------------------

          always@(posedge clock)
          begin
     if(!resetn)
            parity_done<=0;
    else if(detect_add)
            parity_done<=0;
     else if((ld_state && !fifo_full && !pkt_valid)
            ||(laf_state && low_packet_valid && !parity_done))
            parity_done<=1;
```

```
        end

//--------------------------PARITY CALCULATE LOGIC--------------------------

        always@(posedge clock)
        begin
    if(!resetn)
         int_parity<=0;
        else if(detect_add)
         int_parity<=0;
        else if(lfd_state && pkt_valid)
         int_parity<=int_parity^header;
        else if(ld_state && pkt_valid && !full_state)
         int_parity<=int_parity^data_in;
        else
         int_parity<=int_parity;
        end

//-------------------------------ERROR LOGIC---------------------------------

        always@(posedge clock)
                begin
        if(!resetn)
                                err<=0;
            else if(parity_done)
                        begin
                                        if (int_parity==ext_parity)
                                        err<=0;
                                        else
                                err<=1;
                                end
                    else
                            err<=0;
            end

//-------------------------------EXTERNAL PARITY LOGIC-------------------------

        always@(posedge clock)
        begin
    if(!resetn)
                        ext_parity<=0;
    else if(detect_add)
                        ext_parity<=0;
```

```
    else if((ld_state && !fifo_full && !pkt_valid) || (laf_state && !parity_done &&
low_packet_valid))
                        ext_parity<=data_in;
        end

endmodule
```

Test Bench:

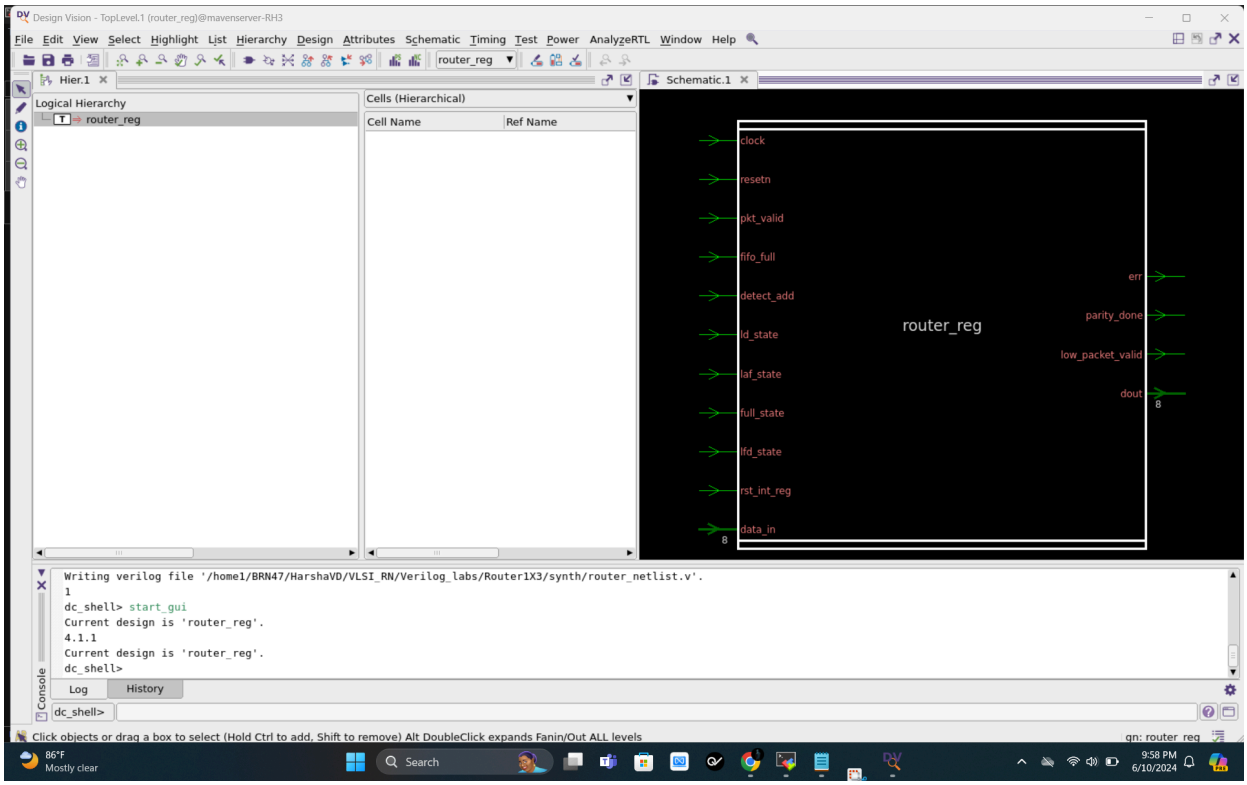
```
module router_reg_tb();

reg clock,resetn,pkt_valid,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg;
reg [7:0]data_in;
wire err,parity_done,low_packet_valid;
wire [7:0]dout;
integer i;
parameter cycle=10;


router_reg
DUT(clock,resetn,pkt_valid,data_in,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_i
nt_reg,err,parity_done,low_packet_valid,dout);

initial
begin
  clock=1'b0;
  forever #(cycle/2) clock=~clock;
end

task rst();
  begin
    @(negedge clock)
    resetn=1'b0;
    @(negedge clock)
    resetn=1'b1;
  end
endtask

task initialize();
  begin
   pkt_valid<=1'b0;
```

```
   fifo_full<=1'b0;
   detect_add<=1'b0;
   ld_state<=1'b0;
   laf_state<=1'b0;
   full_state<=1'b0;
   lfd_state<=1'b0;
   rst_int_reg<=1'b0;
  end
endtask


task good_pkt_gen_reg;

        reg[7:0]payload_data,parity1,header1;
        reg[5:0]payload_len;
        reg[1:0]addr;

        begin
                @(negedge clock)
                payload_len=6'd5;
                addr=2'b10;
                pkt_valid=1;
                detect_add=1;
                header1={payload_len,addr};
                parity1=0^header1;
                data_in=header1;
                @(negedge clock);
                detect_add=0;
                lfd_state=1;
                full_state=0;
                fifo_full=0;
                laf_state=0;

                for(i=0;i<payload_len;i=i+1)
                begin
                        @(negedge clock);
                         lfd_state=0;
                         ld_state=1;
                         payload_data={$random}%256;
                         data_in=payload_data;
                         parity1=parity1^data_in;
                        end
                        @(negedge clock);
                        pkt_valid=0;
```

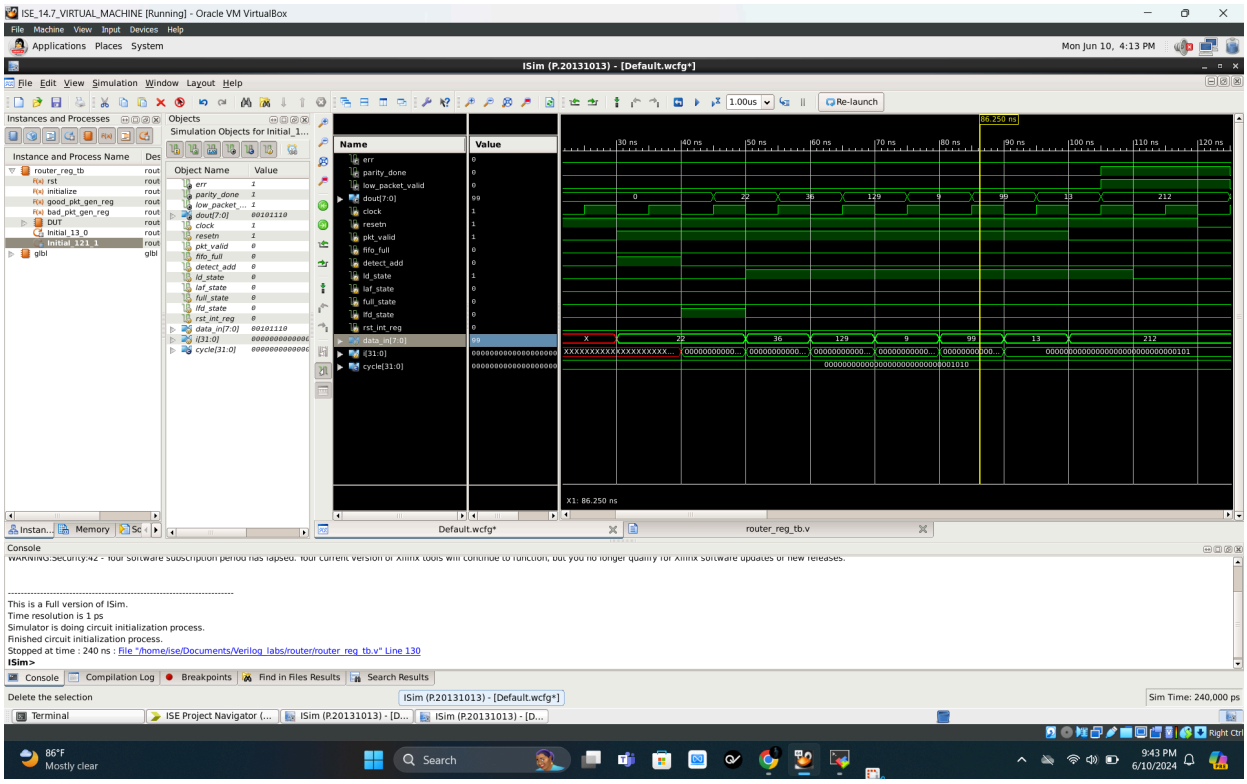
```
                        data_in=parity1;
                        @(negedge clock);
                        ld_state=0;
                end
 endtask

task bad_pkt_gen_reg;

        reg[7:0]payload_data,parity1,header1;
        reg[5:0]payload_len;
        reg[1:0]addr;

        begin
         @(negedge clock)
         payload_len=6'd5;
         addr=2'b10;
         pkt_valid=1;
         detect_add=1;
         header1={payload_len,addr};
         parity1=0^header1;
         data_in=header1;
         @(negedge clock);
         detect_add=0;
         lfd_state=1;
         full_state=0;
         fifo_full=0;
         laf_state=0;

        for(i=0;i<payload_len;i=i+1)
                begin
                        @(negedge clock);
                         lfd_state=0;
                         ld_state=1;
                         payload_data={$random}%256;
                         data_in=payload_data;
                         parity1=parity1^data_in;
                end
                @(negedge clock);
                pkt_valid=0;
                data_in=46;
                @(negedge clock);
                ld_state=0;
        end
endtask
```

```
initial
 begin
 rst();
 initialize();
 good_pkt_gen_reg;
 rst();
 bad_pkt_gen_reg;
 #20

 $finish;
 end

 endmodule
```

Synthesis:



Simulation:


ISE_14.7_VIRTUAL_MACHINE [Running] - Oracle VM VirtualBox
ISim (P.20131013) - [Default.wcfg*]
Re-launch
Default.wcfg*
router_reg_tb.v
Console
This is a full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 240 ns : File "/home/ise/Documents/Verilog_labs/router/router_reg_tb.v" Line 130
ISim>
Delete the selection
Sim Time: 240,000 ps
86°F
Mostly clear
Search
9:43 PM
6/10/2024