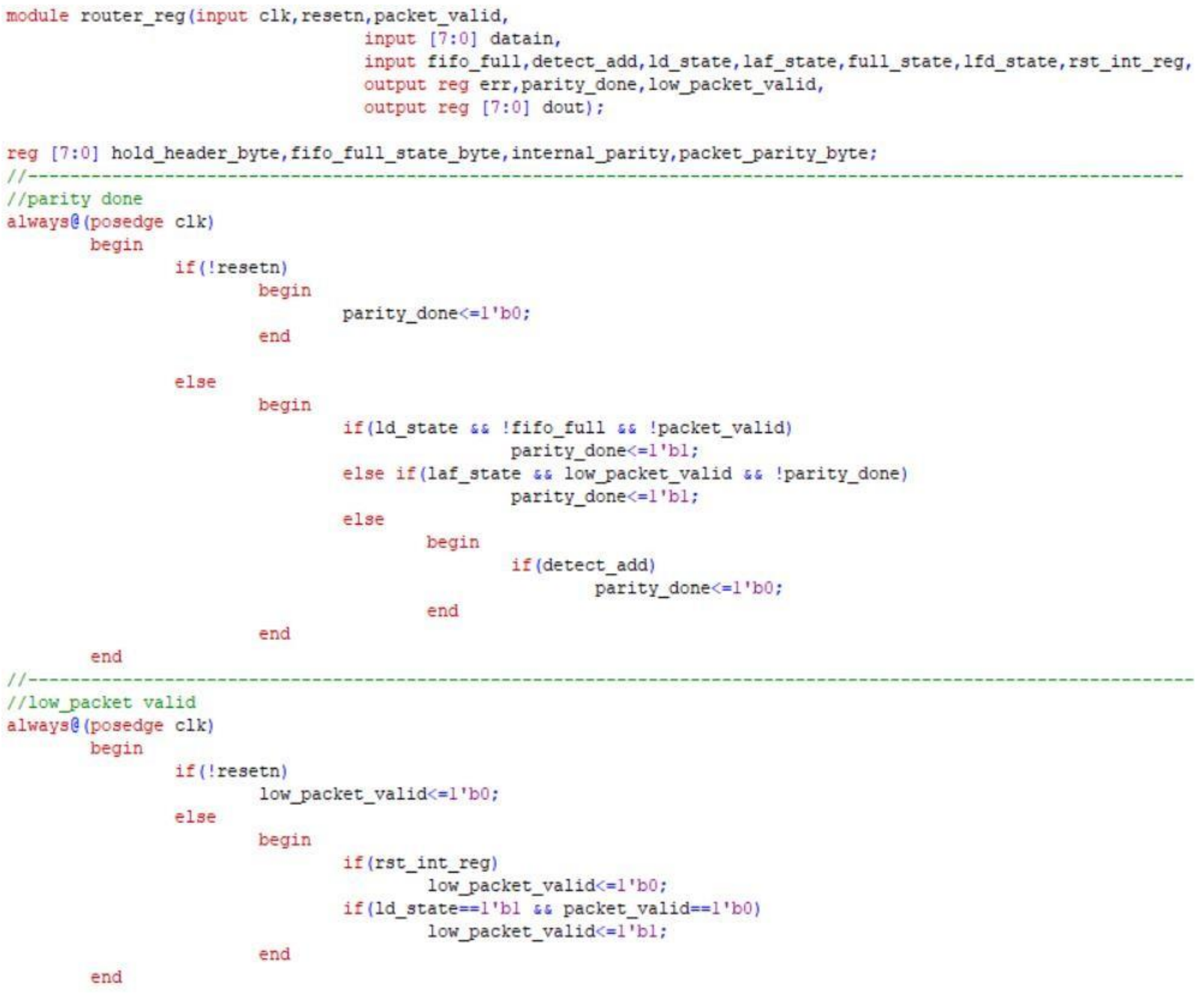
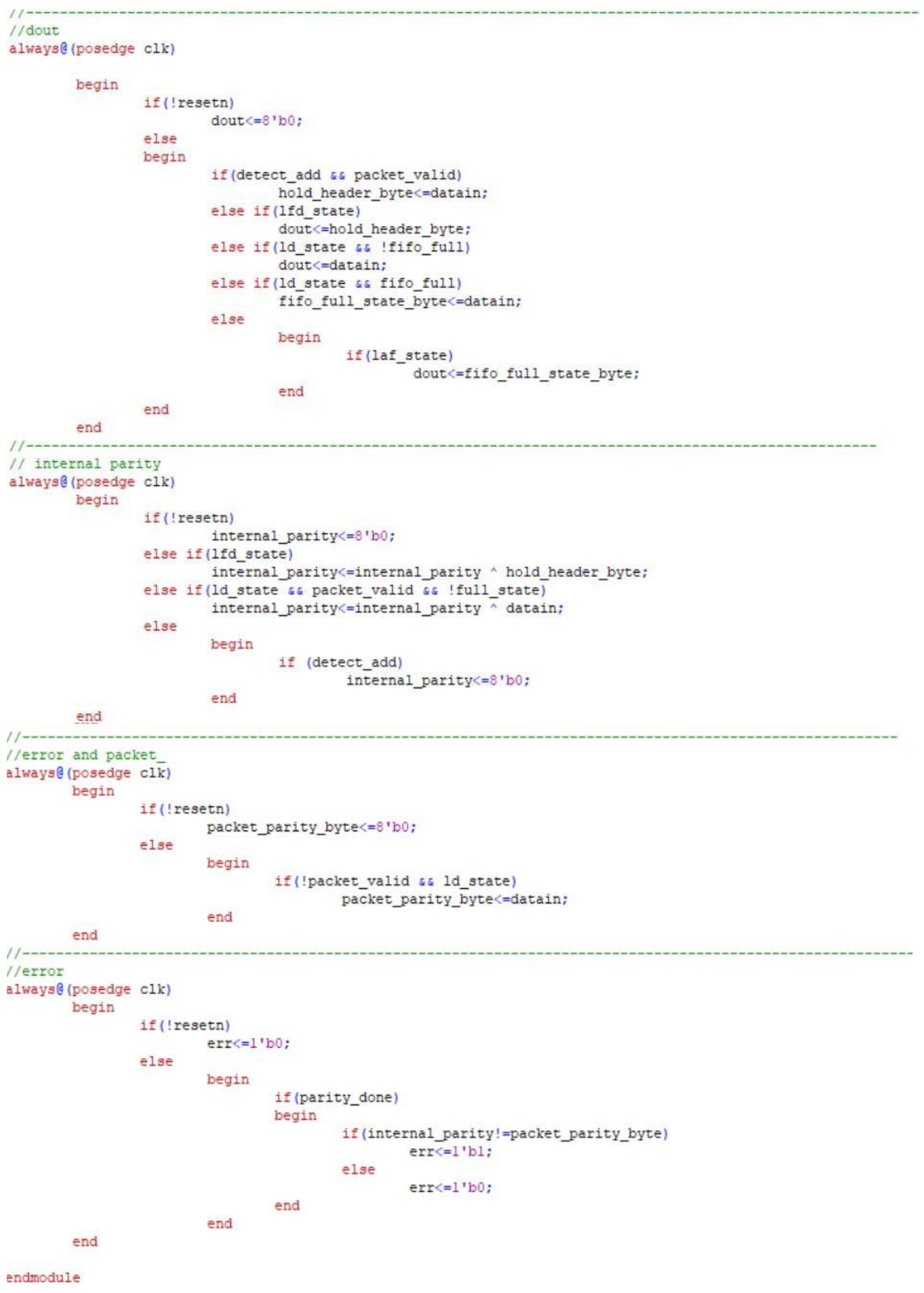
**Name:** CHANDRA NIKHITA M
**Batch:** BRN39  MS 21-22/5978

# ROUTER PROJECT REPORT

## Code: ROUTER Register RTL

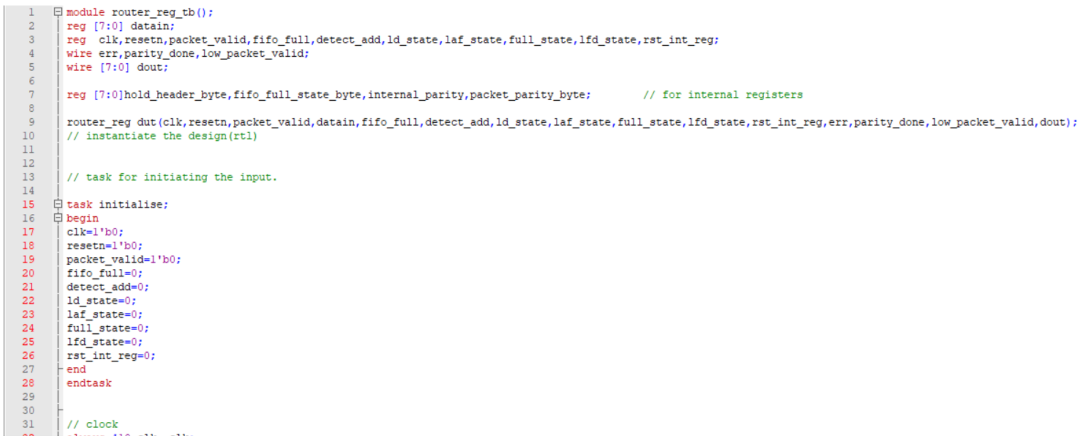


```
module router_reg(input clk,resetn,packet_valid,
                                  input [7:0] datain,
                                  input fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,
                                  output reg err,parity_done,low_packet_valid,
                                  output reg [7:0] dout);

reg [7:0] hold_header_byte,fifo_full_state_byte,internal_parity,packet_parity_byte;
//---------------------------------------------------------------------------------------------------------------
//parity done
always@(posedge clk)
        begin
                if(!resetn)
                        begin
                                parity_done<=1'b0;
                        end

                else
                        begin
                                if(ld_state && !fifo_full && !packet_valid)
                                                parity_done<=1'b1;
                                else if(laf_state && low_packet_valid && !parity_done)
                                                parity_done<=1'b1;
                                else
                                        begin
                                                if(detect_add)
                                                        parity_done<=1'b0;
                                        end
                        end
        end
//----------------------------------------------------------------------------------------------------------------
//low_packet valid
always@(posedge clk)
        begin
                if(!resetn)
                        low_packet_valid<=1'b0;
                else
                        begin
                                if(rst_int_reg)
                                        low_packet_valid<=1'b0;
                                if(ld_state==1'b1 && packet_valid==1'b0)
                                        low_packet_valid<=1'b1;
                        end
        end
```

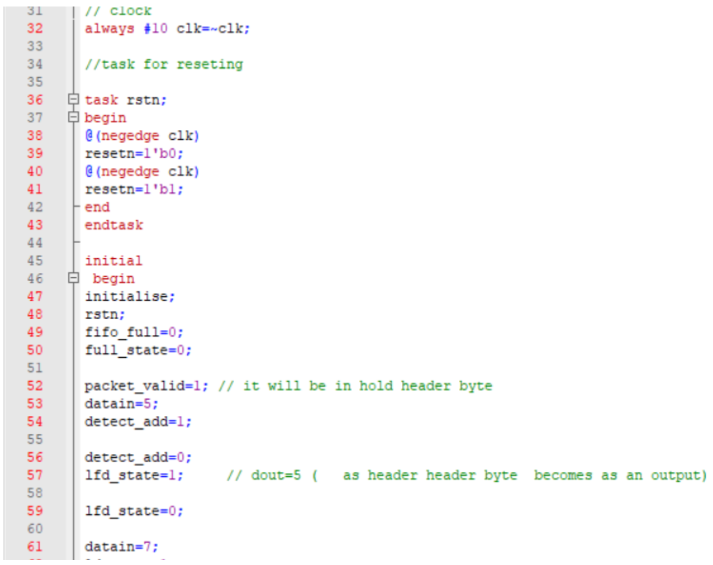


```
//--------------------------------------------------------------------------------------------------------------
//dout
always@(posedge clk)

        begin
                if(!resetn)
                        dout<=8'b0;
                else
                begin
                        if(detect_add && packet_valid)
                                hold_header_byte<=datain;
                        else if(lfd_state)
                                dout<=hold_header_byte;
                        else if(ld_state && !fifo_full)
                                dout<=datain;
                        else if(ld_state && fifo_full)
                                fifo_full_state_byte<=datain;
                        else
                                begin
                                        if(laf_state)
                                                dout<=fifo_full_state_byte;
                                end
                end
        end
//--------------------------------------------------------------------------------------------------------
// internal parity
always@(posedge clk)
        begin
                if(!resetn)
                        internal_parity<=8'b0;
                else if(lfd_state)
                        internal_parity<=internal_parity ^ hold_header_byte;
                else if(ld_state && packet_valid && !full_state)
                        internal_parity<=internal_parity ^ datain;
                else
                        begin
                                if (detect_add)
                                        internal_parity<=8'b0;
                        end
        end
//-----------------------------------------------------------------------------------------------------------
//error and packet_
always@(posedge clk)
        begin
                if(!resetn)
                        packet_parity_byte<=8'b0;
                else
                        begin
                                if(!packet_valid && ld_state)
                                        packet_parity_byte<=datain;
                        end
        end
//------------------------------------------------------------------------------------------------------------
//error
always@(posedge clk)
        begin
                if(!resetn)
                        err<=1'b0;
                else
                        begin
                                if(parity_done)
                                begin
                                        if(internal_parity!=packet_parity_byte)
                                                err<=1'b1;
                                        else
                                                err<=1'b0;
                                end
                        end
        end

endmodule
```

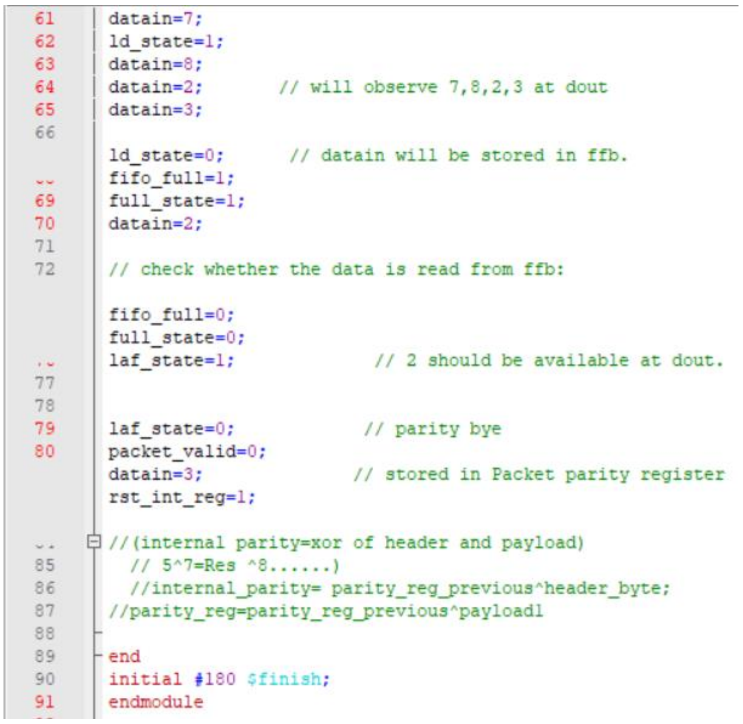
**ROUTER REGISTER TB:**



```
module router_reg_tb();
reg [7:0] datain;
reg  clk,resetn,packet_valid,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg;
wire err,parity_done,low_packet_valid;
wire [7:0] dout;

reg [7:0]hold_header_byte,fifo_full_state_byte,internal_parity,packet_parity_byte;       // for internal registers

router_reg dut(clk,resetn,packet_valid,datain,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,err,parity_done,low_packet_valid,dout);
// instantiate the design(rtl)


// task for initiating the input.

task initialise;
begin
clk=1'b0;
resetn=1'b0;
packet_valid=1'b0;
fifo_full=0;
detect_add=0;
ld_state=0;
laf_state=0;
full_state=0;
lfd_state=0;
rst_int_reg=0;
end
endtask


// clock
```

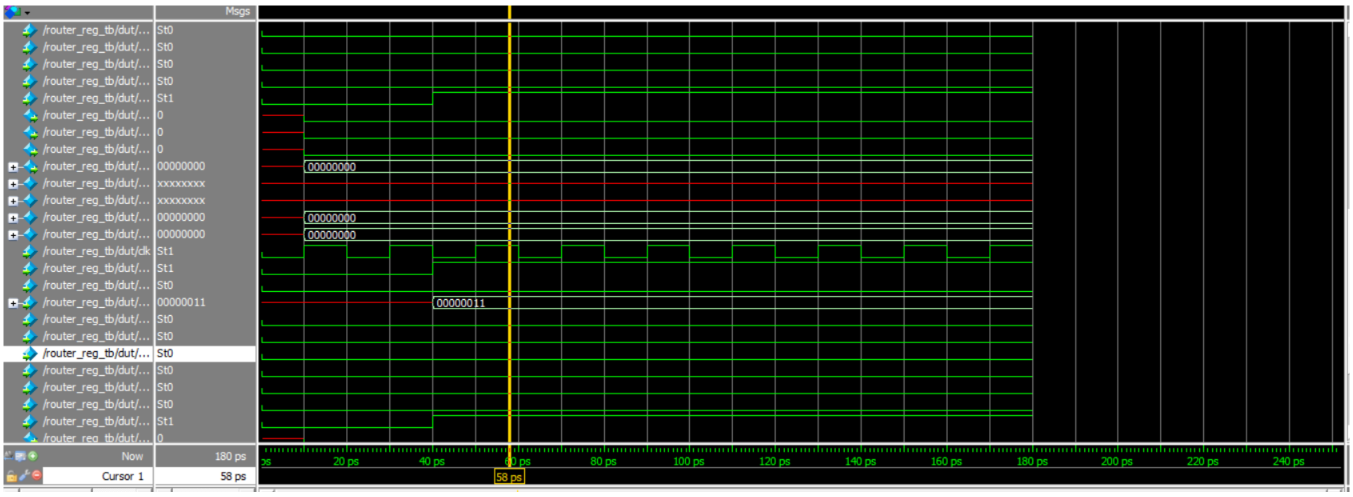


```
// clock
always #10 clk=~clk;

//task for reseting

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

initial
 begin
initialise;
rstn;
fifo_full=0;
full_state=0;

packet_valid=1; // it will be in hold header byte
datain=5;
detect_add=1;

detect_add=0;
lfd_state=1;      // dout=5 (   as header header byte  becomes as an output)

lfd_state=0;

datain=7;
```

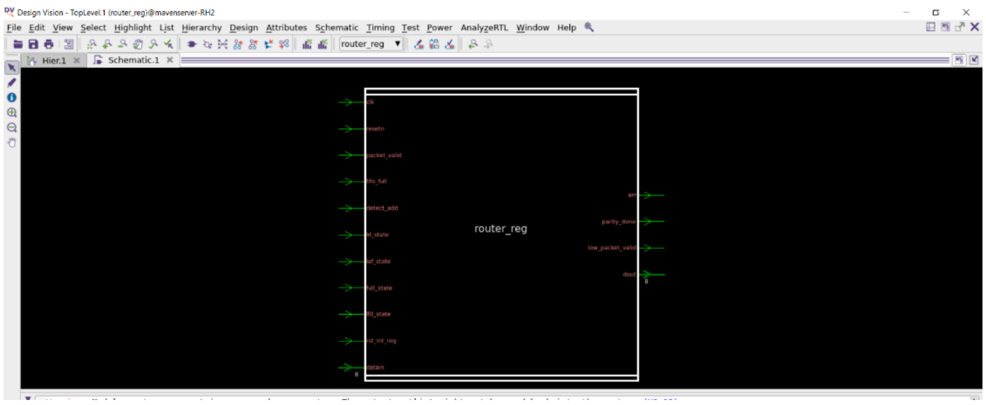


```
datain=7;
ld_state=1;
datain=8;
datain=2;        // will observe 7,8,2,3 at dout
datain=3;

ld_state=0;       // datain will be stored in ffb.
fifo_full=1;
full_state=1;
datain=2;

// check whether the data is read from ffb:

fifo_full=0;
full_state=0;
laf_state=1;               // 2 should be available at dout.


laf_state=0;              // parity bye
packet_valid=0;
datain=3;                // stored in Packet parity register
rst_int_reg=1;

//(internal parity=xor of header and payload)
  // 5^7=Res ^8......)
  //internal_parity= parity_reg_previous^header_byte;
//parity_reg=parity_reg_previous^payload1

end
initial #180 $finish;
endmodule
```

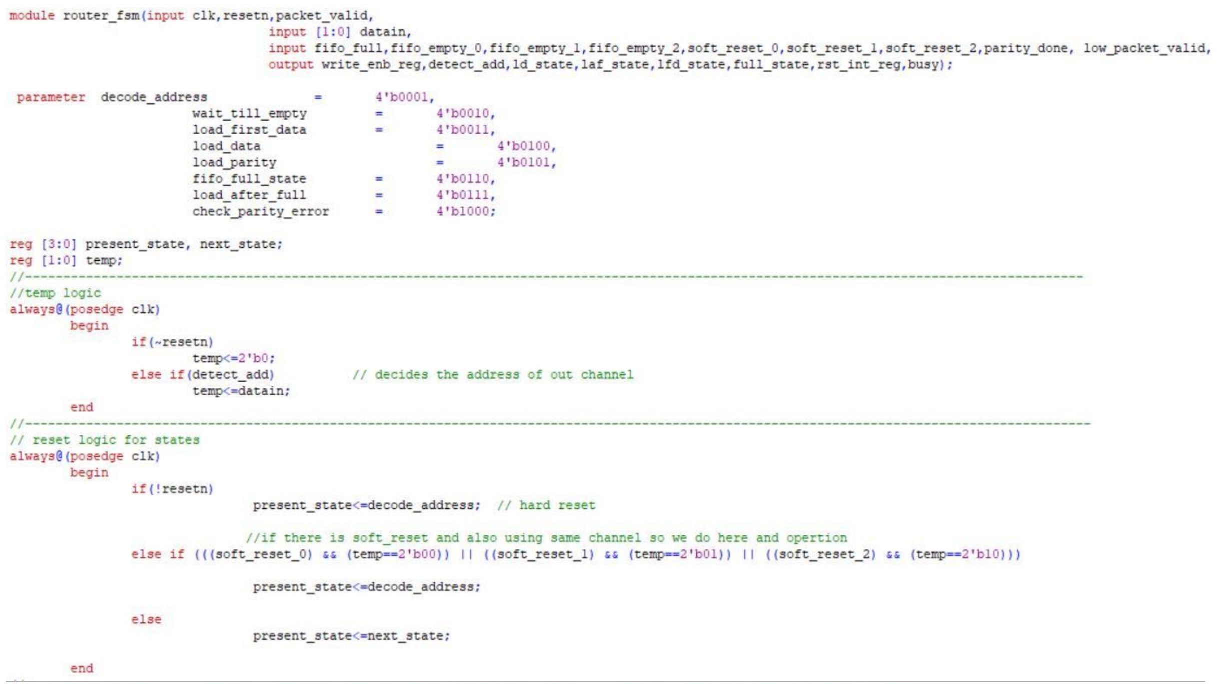
**SIMULATION:**


Msgs
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St1
/router_reg_tb/dut/... 0
/router_reg_tb/dut/... 0
/router_reg_tb/dut/... 0
/router_reg_tb/dut/... 00000000
/router_reg_tb/dut/... xxxxxxxx
/router_reg_tb/dut/... xxxxxxxx
/router_reg_tb/dut/... 00000000
/router_reg_tb/dut/... 00000000
/router_reg_tb/dut/clk St1
/router_reg_tb/dut/... St1
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... 00000011
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St0
/router_reg_tb/dut/... St1
/router_reg_tb/dut/... 0
00000000
00000000
00000000
00000011
Now 180 ps
Cursor 1 58 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps
240 ps
58 ps

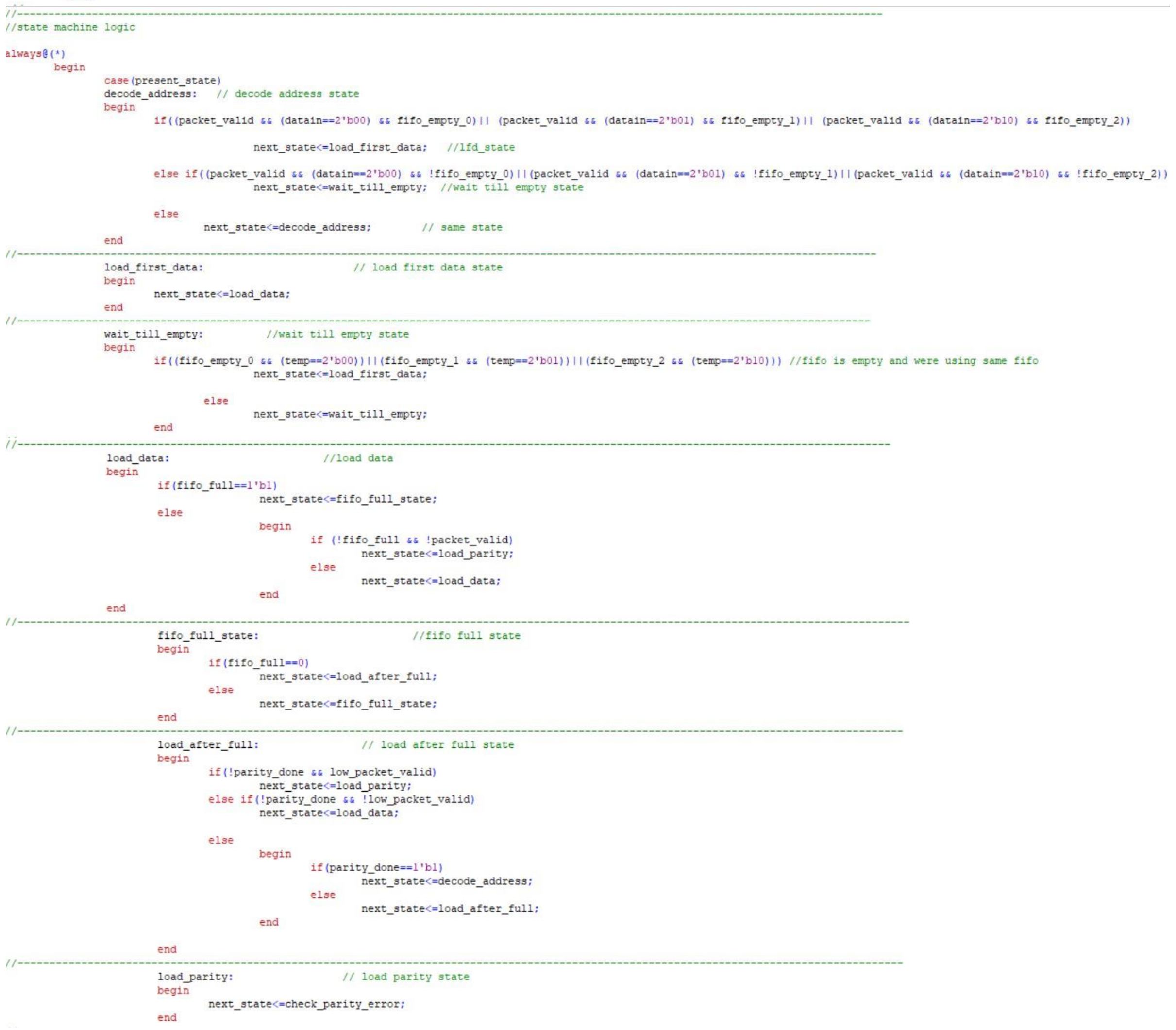
## SYNTHESIS:


Design Vision - TopLevel.1 (router_reg)@mavenserver-RH2
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
router_reg
Hier.1
Schematic.1
clk
resetn
packet_valid
fifo_full
detect_add
ld_state
laf_state
full_state
lfd_state
rst_int_reg
datain
8
router_reg
err
parity_done
low_packet_valid
dout
8

## FSM: RTL



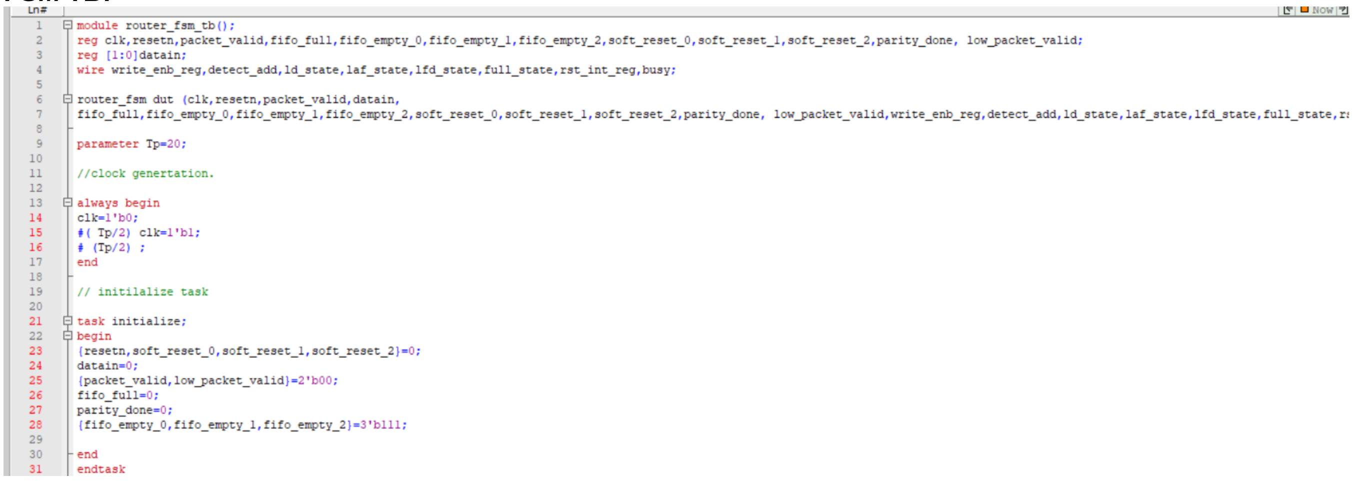
```verilog
module router_fsm(input clk,resetn,packet_valid,
                                input [1:0] datain,
                                input fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_reset_1,soft_reset_2,parity_done, low_packet_valid,
                                output write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy);

 parameter  decode_address              =       4'b0001,
                        wait_till_empty         =       4'b0010,
                        load_first_data         =       4'b0011,
                        load_data                       =       4'b0100,
                        load_parity                     =       4'b0101,
                        fifo_full_state         =       4'b0110,
                        load_after_full         =       4'b0111,
                        check_parity_error      =       4'b1000;

reg [3:0] present_state, next_state;
reg [1:0] temp;
//--------------------------------------------------------------------------------------------------------------------
//temp logic
always@(posedge clk)
        begin
                if(~resetn)
                        temp<=2'b0;
                else if(detect_add)             // decides the address of out channel
                        temp<=datain;
        end
//--------------------------------------------------------------------------------------------------------------------
// reset logic for states
always@(posedge clk)
        begin
                if(!resetn)
                                present_state<=decode_address;  // hard reset

                               //if there is soft_reset and also using same channel so we do here and opertion
                else if (((soft_reset_0) && (temp==2'b00)) || ((soft_reset_1) && (temp==2'b01)) || ((soft_reset_2) && (temp==2'b10)))

                                present_state<=decode_address;

                else
                                present_state<=next_state;

        end
```



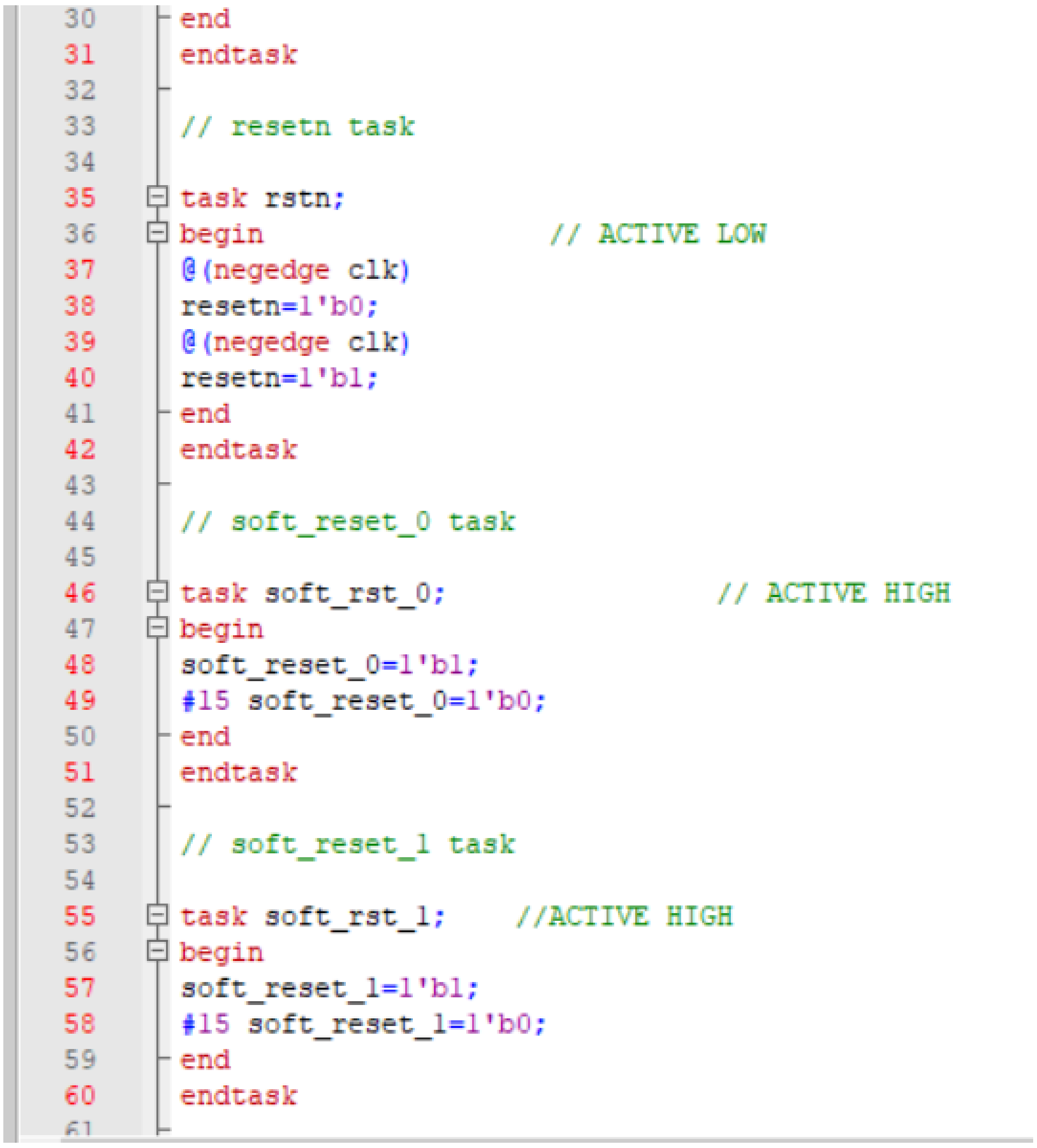
```verilog
//--------------------------------------------------------------------------------------------------------------------
//state machine logic

always@(*)
        begin
                case(present_state)
                decode_address:   // decode address state
                begin
                        if((packet_valid && (datain==2'b00) && fifo_empty_0)|| (packet_valid && (datain==2'b01) && fifo_empty_1)|| (packet_valid && (datain==2'b10) && fifo_empty_2))

                                        next_state<=load_first_data;   //lfd_state

                        else if((packet_valid && (datain==2'b00) && !fifo_empty_0)||(packet_valid && (datain==2'b01) && !fifo_empty_1)||(packet_valid && (datain==2'b10) && !fifo_empty_2))
                                        next_state<=wait_till_empty;  //wait till empty state

                        else
                                next_state<=decode_address;        // same state
                end
//--------------------------------------------------------------------------------------------------------------------
                load_first_data:                        // load first data state
                begin
                        next_state<=load_data;
                end
//--------------------------------------------------------------------------------------------------------------------
                wait_till_empty:          //wait till empty state
                begin
                        if((fifo_empty_0 && (temp==2'b00))||(fifo_empty_1 && (temp==2'b01))||(fifo_empty_2 && (temp==2'b10))) //fifo is empty and were using same fifo
                                        next_state<=load_first_data;

                                else
                                        next_state<=wait_till_empty;
                        end
//--------------------------------------------------------------------------------------------------------------------
                load_data:                          //load data
                begin
                        if(fifo_full==1'b1)
                                        next_state<=fifo_full_state;
                        else
                                        begin
                                                if (!fifo_full && !packet_valid)
                                                        next_state<=load_parity;
                                                else
                                                        next_state<=load_data;
                                        end
                end
//--------------------------------------------------------------------------------------------------------------------
                        fifo_full_state:                        //fifo full state
                        begin
                                if(fifo_full==0)
                                        next_state<=load_after_full;
                                else
                                        next_state<=fifo_full_state;
                        end
//--------------------------------------------------------------------------------------------------------------------
                        load_after_full:                // load after full state
                        begin
                                if(!parity_done && low_packet_valid)
                                        next_state<=load_parity;
                                else if(!parity_done && !low_packet_valid)
                                        next_state<=load_data;

                                else
                                        begin
                                                if(parity_done==1'b1)
                                                        next_state<=decode_address;
                                                else
                                                        next_state<=load_after_full;
                                        end

                        end
//--------------------------------------------------------------------------------------------------------------------
                        load_parity:                // load parity state
                        begin
                                next_state<=check_parity_error;
                        end
```

```
//---------------------------------------------------------------------------------------------------------------------------------------
                        check_parity_error:                     // check parity error
                        begin
                                if(!fifo_full)
                                        next_state<=decode_address;
                                else
                                        next_state<=fifo_full_state;
                        end
//----------------------------------------------------------------------------------------------------------------------------------------
                        default:                                        //default state
                                next_state<=decode_address;

                endcase         // state machine completed
        end
//------------------------------------------------------------------------------------------------------------------------------------
// output logic


assign busy=((present_state==load_first_data)||(present_state==load_parity)||(present_state==fifo_full_state)||(present_state==load_after_full)||(present_state==wait_till_empty)
              ||(present_state==check_parity_error))?1:0;
assign detect_add=((present_state==decode_address))?1:0;
assign lfd_state=((present_state==load_first_data))?1:0;
assign ld_state=((present_state==load_data))?1:0;
assign write_enb_reg=((present_state==load_data)||(present_state==load_after_full)||(present_state==load_parity))?1:0;
assign full_state=((present_state==fifo_full_state))?1:0;
assign laf_state=((present_state==load_after_full))?1:0;
assign rst_int_reg=((present_state==check_parity_error))?1:0;

endmodule
```

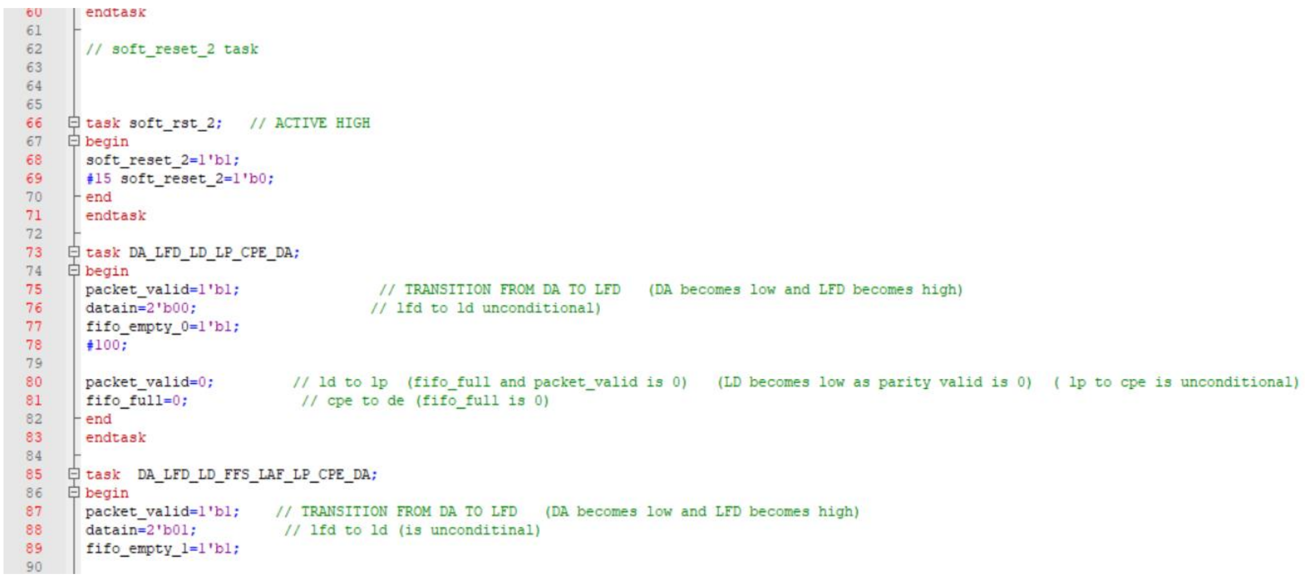
## FSM TB:



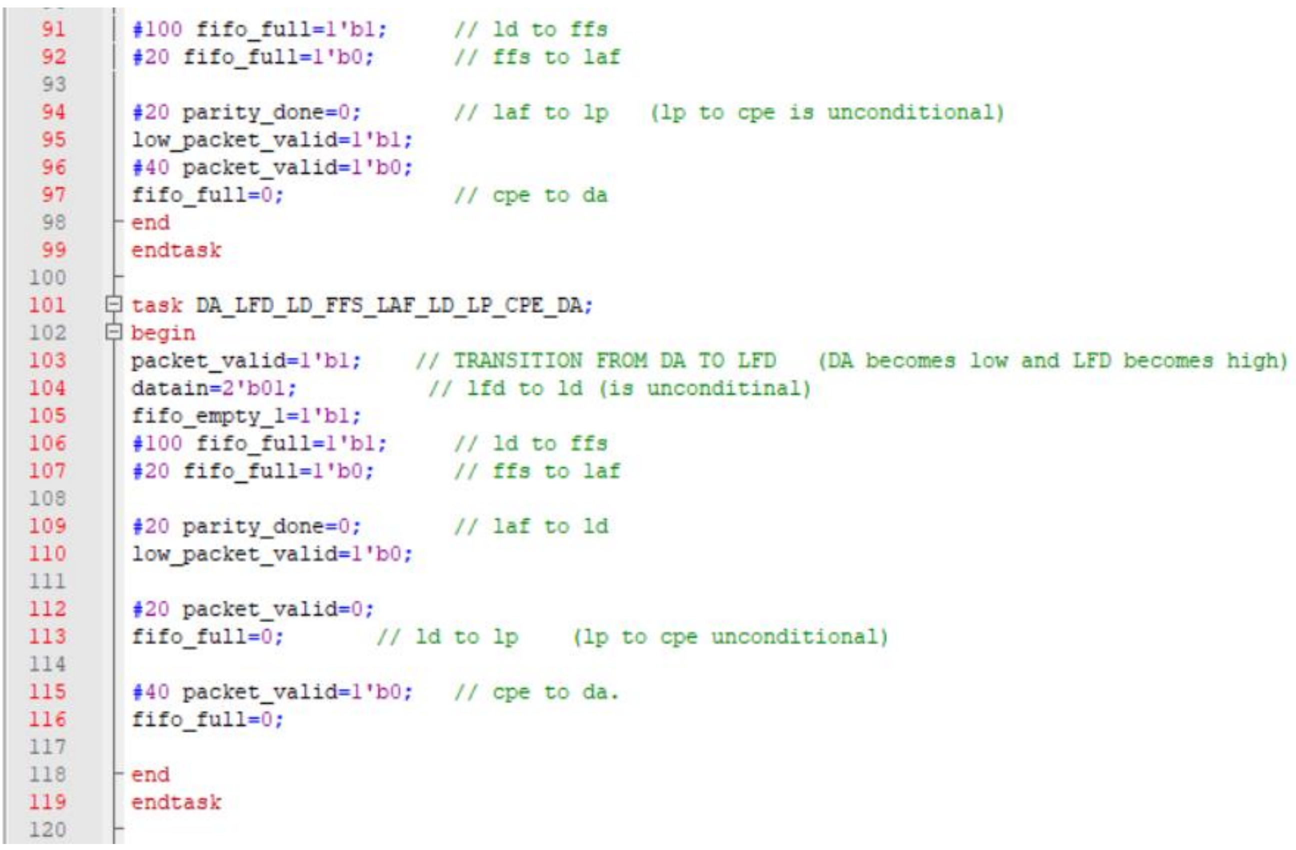
```
module router_fsm_tb();
reg clk,resetn,packet_valid,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_reset_1,soft_reset_2,parity_done, low_packet_valid;
reg [1:0]datain;
wire write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy;

router_fsm dut (clk,resetn,packet_valid,datain,
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,r

parameter Tp=20;

//clock genertation.

always begin
clk=1'b0;
#( Tp/2) clk=1'b1;
# (Tp/2) ;
end

// initilalize task

task initialize;
begin
{resetn,soft_reset_0,soft_reset_1,soft_reset_2}=0;
datain=0;
{packet_valid,low_packet_valid}=2'b00;
fifo_full=0;
parity_done=0;
{fifo_empty_0,fifo_empty_1,fifo_empty_2}=3'b111;

end
endtask
```



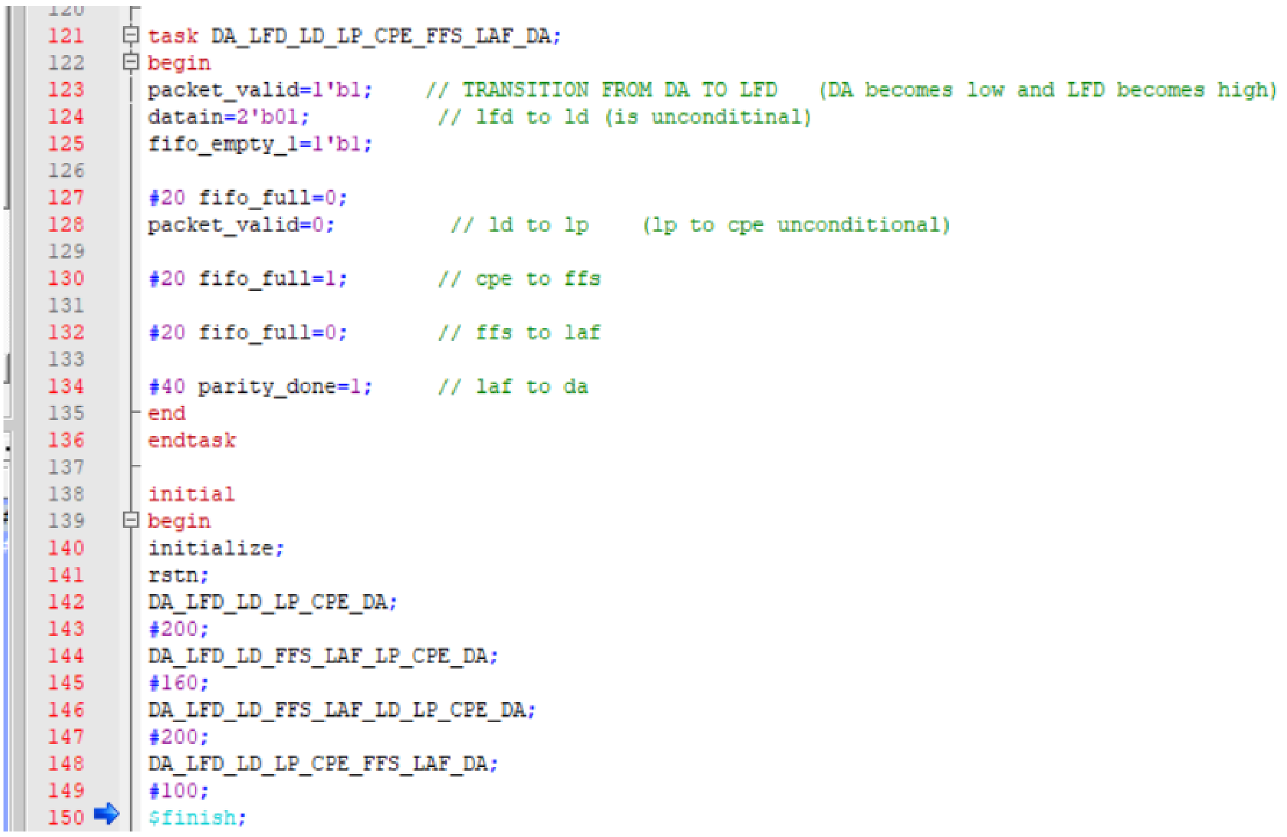
```verilog
end
endtask

// resetn task

task rstn;
begin                          // ACTIVE LOW
@(negedge clk)
resetn=1'b0;
@(negedge clk)
resetn=1'b1;
end
endtask

// soft_reset_0 task

task soft_rst_0;                          // ACTIVE HIGH
begin
soft_reset_0=1'b1;
#15 soft_reset_0=1'b0;
end
endtask

// soft_reset_1 task

task soft_rst_1;    //ACTIVE HIGH
begin
soft_reset_1=1'b1;
#15 soft_reset_1=1'b0;
end
endtask
```



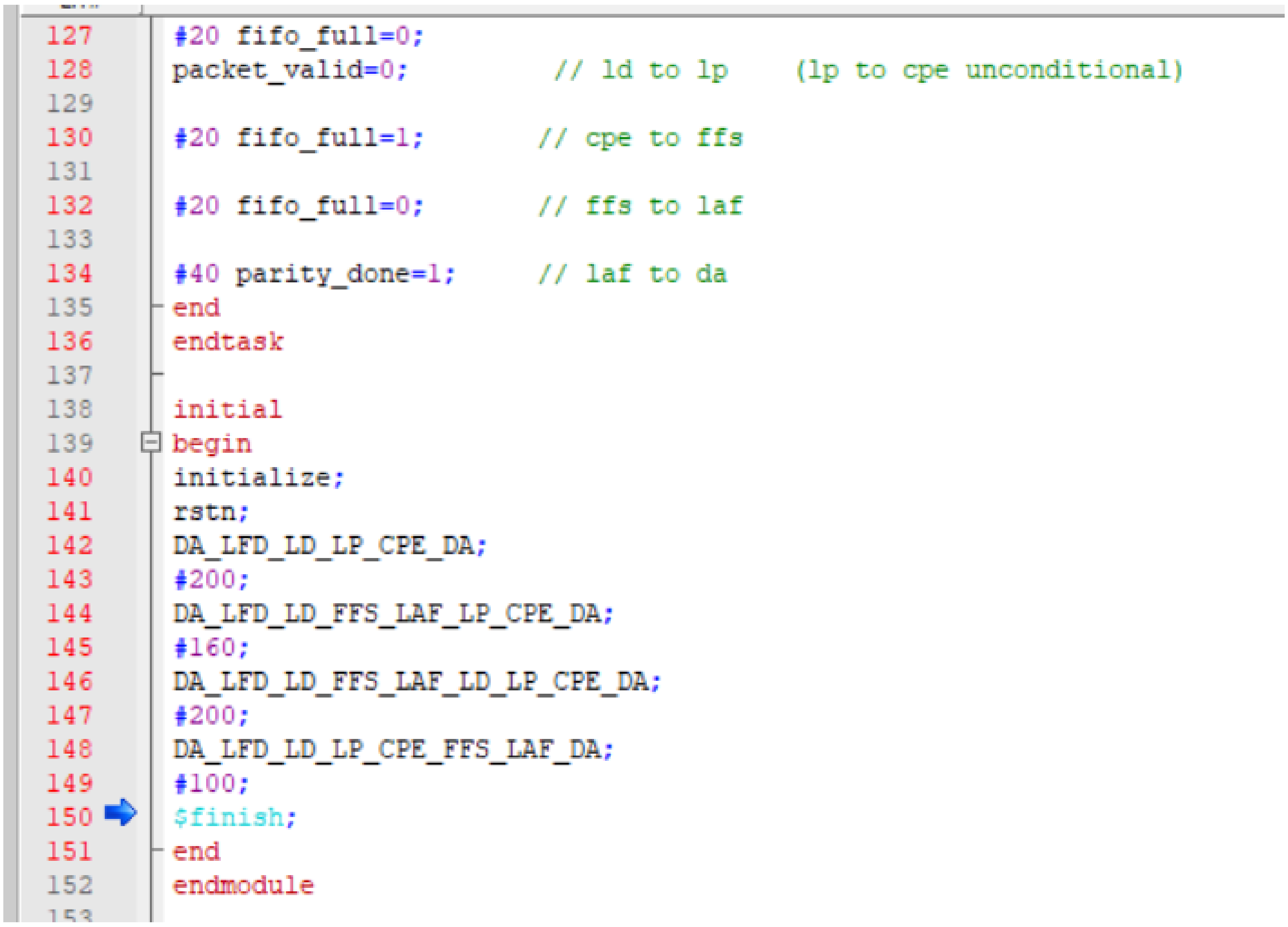
```verilog
endtask

// soft_reset_2 task



task soft_rst_2;   // ACTIVE HIGH
begin
soft_reset_2=1'b1;
#15 soft_reset_2=1'b0;
end
endtask

task DA_LFD_LD_LP_CPE_DA;
begin
packet_valid=1'b1;                // TRANSITION FROM DA TO LFD   (DA becomes low and LFD becomes high)
datain=2'b00;                    // lfd to ld unconditional)
fifo_empty_0=1'b1;
#100;

packet_valid=0;         // ld to lp  (fifo_full and packet_valid is 0)   (LD becomes low as parity valid is 0)  ( lp to cpe is unconditional)
fifo_full=0;             // cpe to de (fifo_full is 0)
end
endtask

task  DA_LFD_LD_FFS_LAF_LP_CPE_DA;
begin
packet_valid=1'b1;    // TRANSITION FROM DA TO LFD   (DA becomes low and LFD becomes high)
datain=2'b01;          // lfd to ld (is unconditinal)
fifo_empty_1=1'b1;
```



```verilog
#100 fifo_full=1'b1;      // ld to ffs
#20 fifo_full=1'b0;       // ffs to laf

#20 parity_done=0;        // laf to lp   (lp to cpe is unconditional)
low_packet_valid=1'b1;
#40 packet_valid=1'b0;
fifo_full=0;              // cpe to da
end
endtask

task DA_LFD_LD_FFS_LAF_LD_LP_CPE_DA;
begin
packet_valid=1'b1;     // TRANSITION FROM DA TO LFD   (DA becomes low and LFD becomes high)
datain=2'b01;           // lfd to ld (is unconditinal)
fifo_empty_1=1'b1;
#100 fifo_full=1'b1;      // ld to ffs
#20 fifo_full=1'b0;       // ffs to laf

#20 parity_done=0;        // laf to ld
low_packet_valid=1'b0;

#20 packet_valid=0;
fifo_full=0;         // ld to lp    (lp to cpe unconditional)

#40 packet_valid=1'b0;    // cpe to da.
fifo_full=0;

end
endtask
```

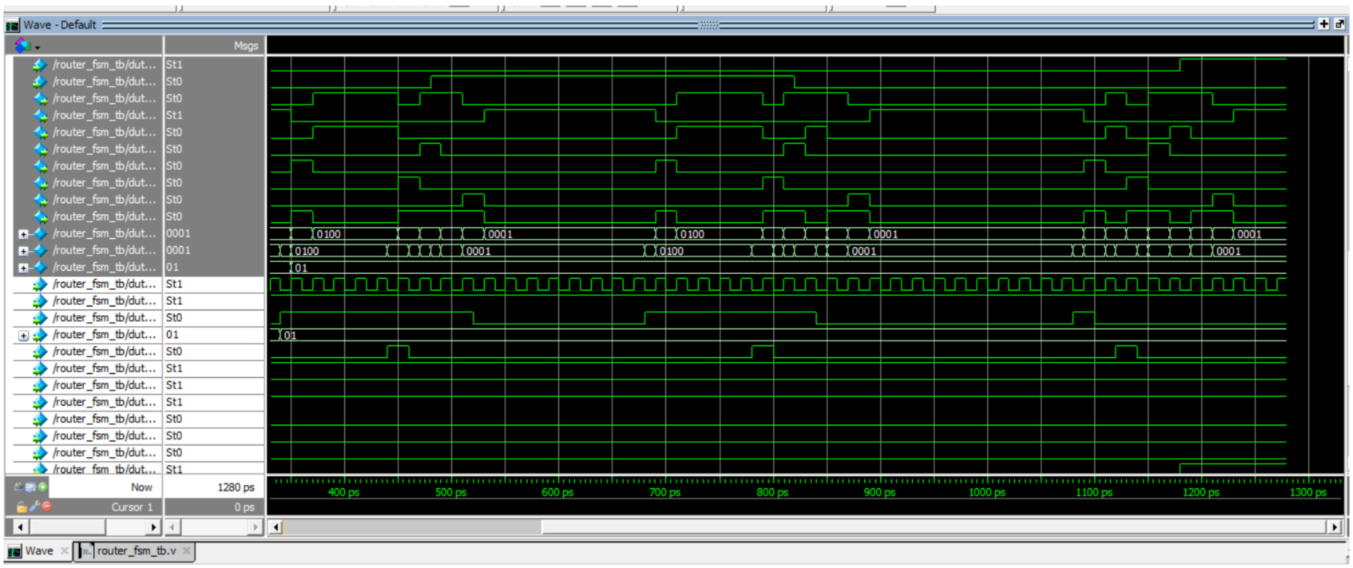


```verilog
task DA_LFD_LD_LP_CPE_FFS_LAF_DA;
begin
packet_valid=1'b1;     // TRANSITION FROM DA TO LFD    (DA becomes low and LFD becomes high)
datain=2'b01;           // lfd to ld (is unconditinal)
fifo_empty_1=1'b1;

#20 fifo_full=0;
packet_valid=0;           // ld to lp    (lp to cpe unconditional)

#20 fifo_full=1;        // cpe to ffs

#20 fifo_full=0;        // ffs to laf

#40 parity_done=1;      // laf to da
end
endtask

initial
begin
initialize;
rstn;
DA_LFD_LD_LP_CPE_DA;
#200;
DA_LFD_LD_FFS_LAF_LP_CPE_DA;
#160;
DA_LFD_LD_FFS_LAF_LD_LP_CPE_DA;
#200;
DA_LFD_LD_LP_CPE_FFS_LAF_DA;
#100;
$finish;
```

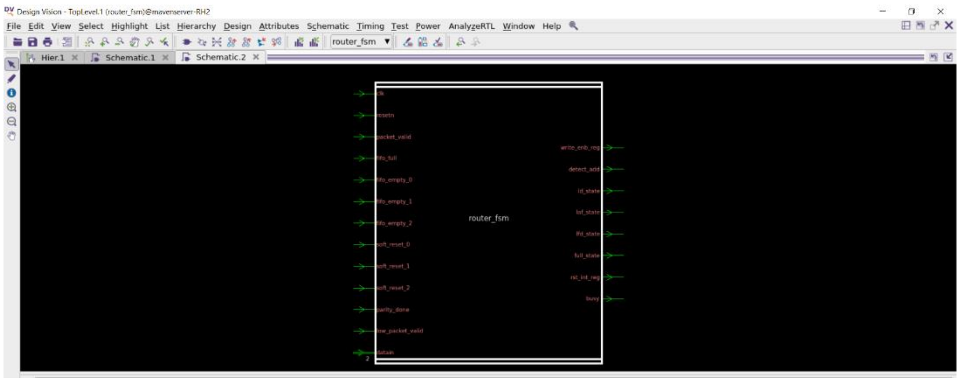


```
#20 fifo_full=0;
packet_valid=0;          // ld to lp     (lp to cpe unconditional)

#20 fifo_full=1;        // cpe to ffs

#20 fifo_full=0;        // ffs to laf

#40 parity_done=1;      // laf to da
end
endtask

initial
begin
initialize;
rstn;
DA_LFD_LD_LP_CPE_DA;
#200;
DA_LFD_LD_FFS_LAF_LP_CPE_DA;
#160;
DA_LFD_LD_FFS_LAF_LD_LP_CPE_DA;
#200;
DA_LFD_LD_LP_CPE_FFS_LAF_DA;
#100;
$finish;
end
endmodule
```

**SIMULATION:**

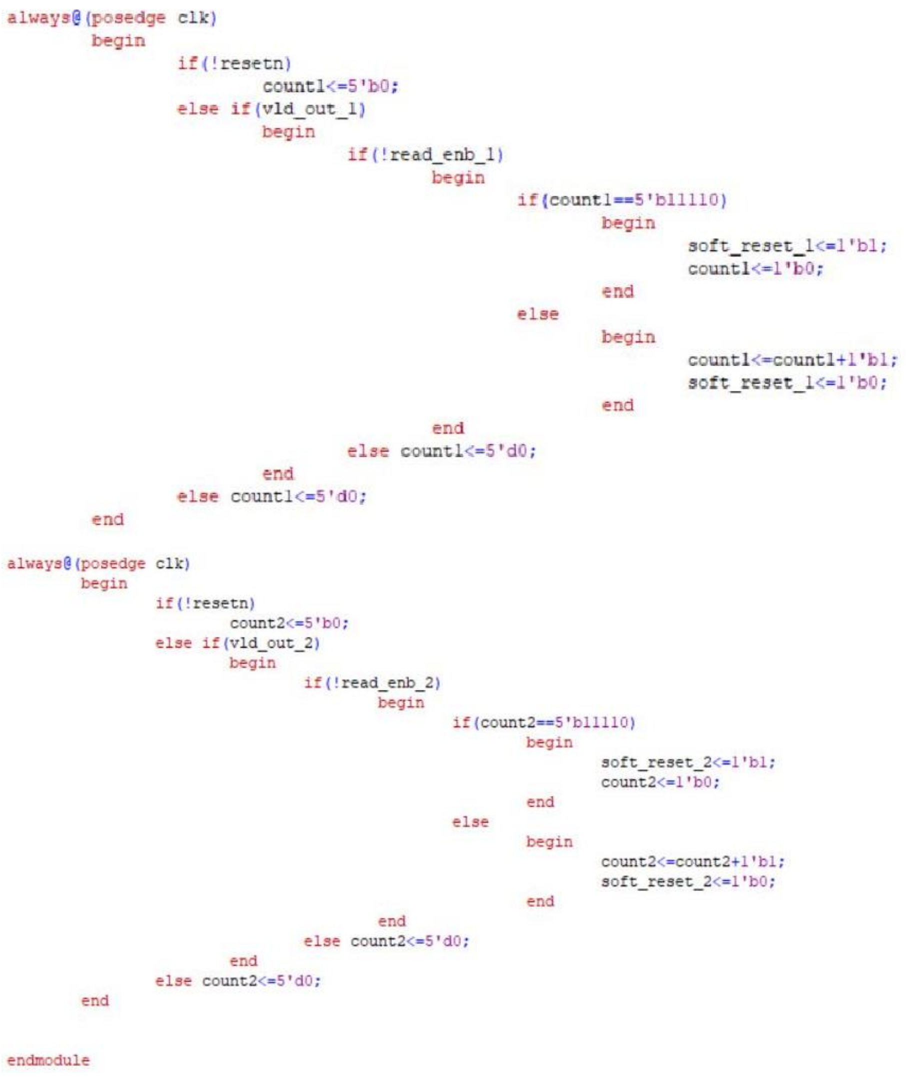

Wave - Default
Msgs
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... 0001
/router_fsm_tb/dut... 0001
/router_fsm_tb/dut... 01
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... 01
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St1
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St0
/router_fsm_tb/dut... St1
0100
0001
01
Now 1280 ps
Cursor 1 0 ps
400 ps
500 ps
600 ps
700 ps
800 ps
900 ps
1000 ps
1100 ps
1200 ps
1300 ps
Wave
router_fsm_tb.v

**Synthesis:**



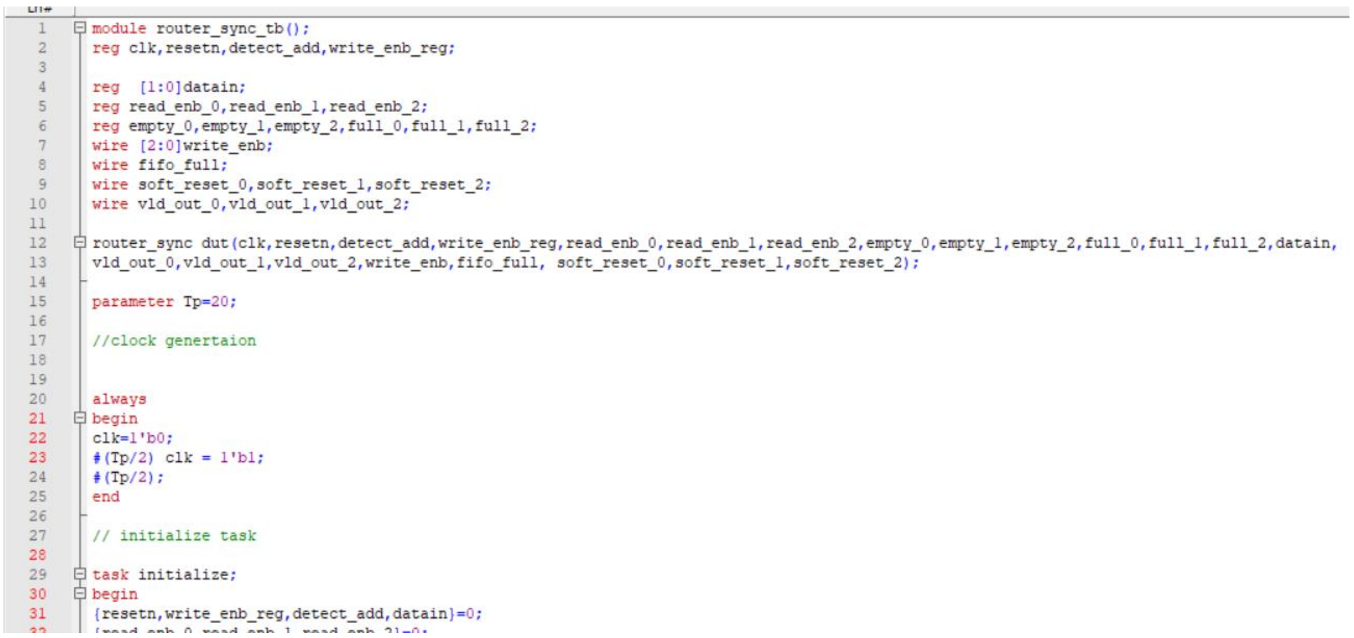
## SYNCHRONIZER RTL:

```
module router_sync( input clk,resetn,detect_add,write_enb_reg,read_enb_0,read_enb_1,read_enb_2,empty_0,empty_1,empty_2,full_0,full_1,full_2,
                                        input [1:0]datain,
                                        output wire vld_out_0,vld_out_1,vld_out_2,
                                        output reg [2:0]write_enb,
                                        output reg fifo_full, soft_reset_0,soft_reset_1,soft_reset_2);

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

//-----------------------------------------------------------------------------------------------------------------------------
always@(posedge clk)
        begin
                if(!resetn)
                        temp <= 2'd0;
                else if(detect_add)
                        temp<=datain;
        end

//----------------------------------------------------------------------------------------------------------------------------
//for fifo full
always@(*)
        begin
                case(temp)
                        2'b00: fifo_full=full_0;                // fifo fifo_full takes the value of full of fifo_0
                        2'b01: fifo_full=full_1;                // fifo fifo_full takes the value of full of fifo_1
                        2'b10: fifo_full=full_2;                                // fifo fifo_full takes the value of full of fifo_2
                        default fifo_full=0;
                endcase
        end
//----------------------------------------------------------------------------------------------------------------------------
//write enable
always@(*)
        begin
                                if(write_enb_reg)
                                begin
                                        case(temp)
                                                2'b00: write_enb=3'b001;
                                                2'b01: write_enb=3'b010;
                                                2'b10: write_enb=3'b100;
                                                default: write_enb=3'b000;
                                        endcase
                                end
                                else
                                        write_enb = 3'b000;
        end
//----------------------------------------------------------------------------------------------------------------------------

//valid out
assign vld_out_0 = !empty_0;
assign vld_out_1 = !empty_1;
assign vld_out_2 = !empty_2;
//----------------------------------------------------------------------------------------------------------------------------
//soft reset counter
always@(posedge clk)
        begin
                if(!resetn)
                        count0<=5'b0;
                else if(vld_out_0)
                        begin
                                if(!read_enb_0)
                                        begin
                                                if(count0==5'b11110)
                                                        begin
                                                                soft_reset_0<=1'b1;
                                                                count0<=1'b0;
                                                        end
                                                else
                                                        begin
                                                                count0<=count0+1'b1;
                                                                soft_reset_0<=1'b0;
                                                        end
                                        end
                                else count0<=5'd0;
                        end
                else count0<=5'd0;
        end
```

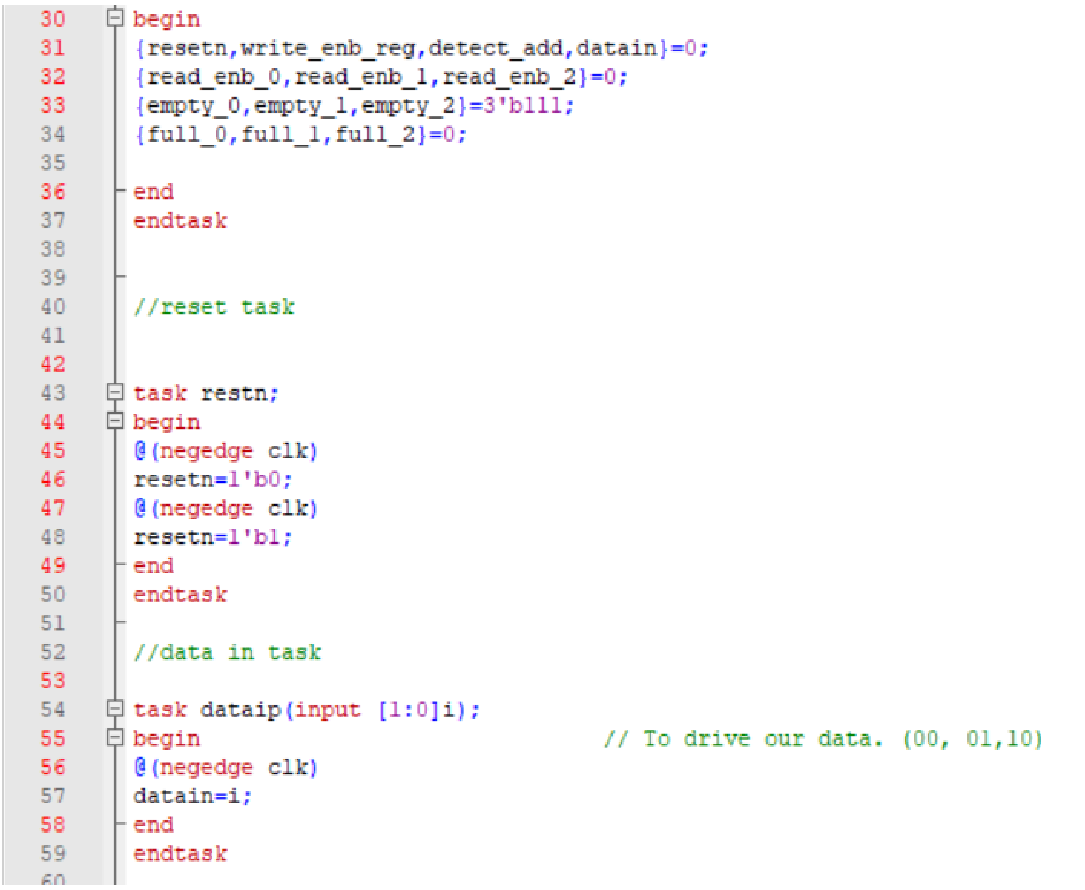


```
always@(posedge clk)
        begin
                if(!resetn)
                        count1<=5'b0;
                else if(vld_out_1)
                        begin
                                if(!read_enb_1)
                                        begin
                                                if(count1==5'b11110)
                                                        begin
                                                                soft_reset_1<=1'b1;
                                                                count1<=1'b0;
                                                        end
                                                else
                                                        begin
                                                                count1<=count1+1'b1;
                                                                soft_reset_1<=1'b0;
                                                        end
                                        end
                                else count1<=5'd0;
                        end
                else count1<=5'd0;
        end

always@(posedge clk)
        begin
                if(!resetn)
                        count2<=5'b0;
                else if(vld_out_2)
                        begin
                                if(!read_enb_2)
                                        begin
                                                if(count2==5'b11110)
                                                        begin
                                                                soft_reset_2<=1'b1;
                                                                count2<=1'b0;
                                                        end
                                                else
                                                        begin
                                                                count2<=count2+1'b1;
                                                                soft_reset_2<=1'b0;
                                                        end
                                        end
                                else count2<=5'd0;
                        end
                else count2<=5'd0;
        end


endmodule
```

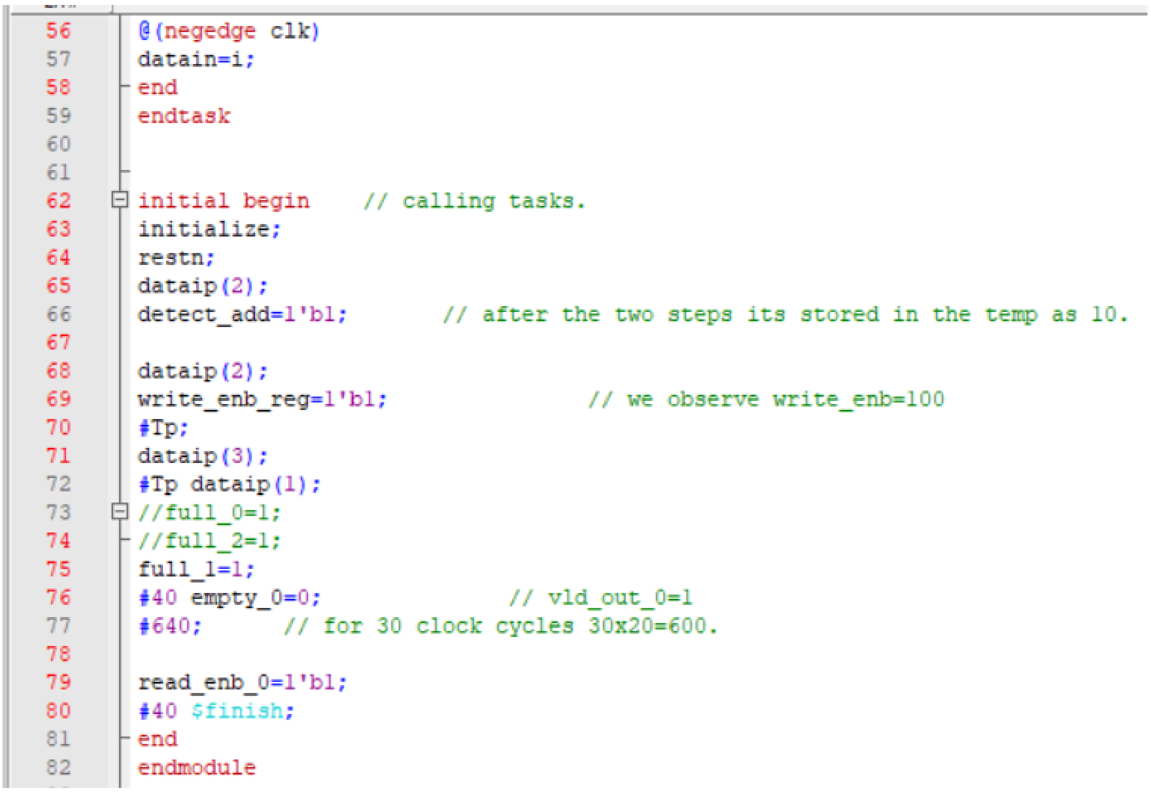
**TB:**



```
module router_sync_tb();
reg clk,resetn,detect_add,write_enb_reg;

reg  [1:0]datain;
reg read_enb_0,read_enb_1,read_enb_2;
reg empty_0,empty_1,empty_2,full_0,full_1,full_2;
wire [2:0]write_enb;
wire fifo_full;
wire soft_reset_0,soft_reset_1,soft_reset_2;
wire vld_out_0,vld_out_1,vld_out_2;

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

parameter Tp=20;

//clock genertaion


always
begin
clk=1'b0;
#(Tp/2) clk = 1'b1;
#(Tp/2);
end

// initialize task

task initialize;
begin
{resetn,write_enb_reg,detect_add,datain}=0;
{read_enb_0,read_enb_1,read_enb_2}=0;
```

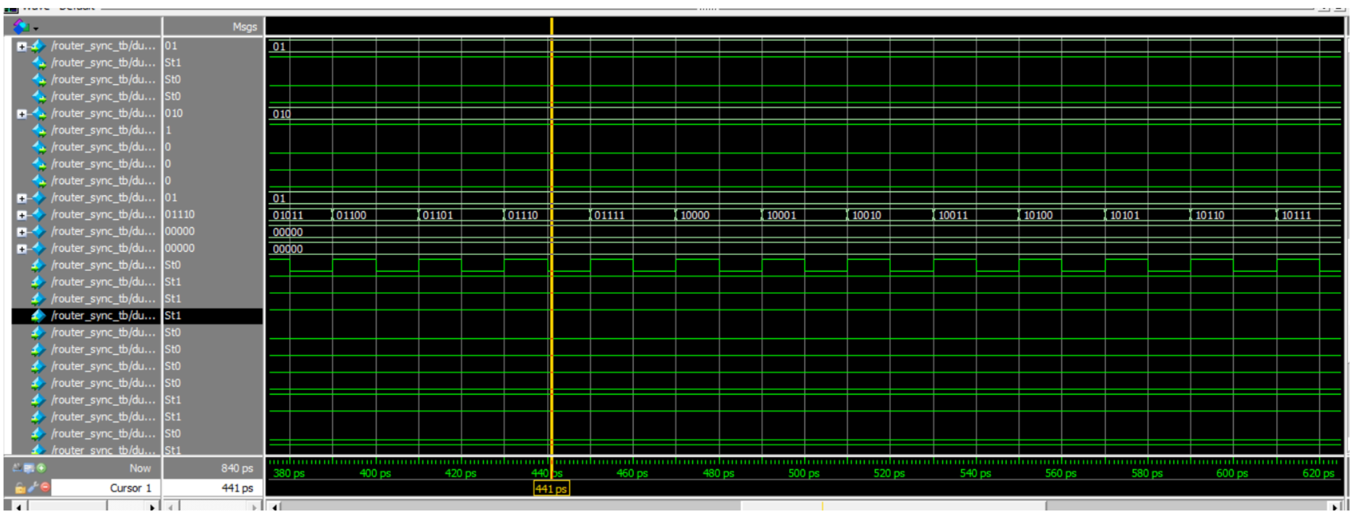


```verilog
begin
{resetn,write_enb_reg,detect_add,datain}=0;
{read_enb_0,read_enb_1,read_enb_2}=0;
{empty_0,empty_1,empty_2}=3'b111;
{full_0,full_1,full_2}=0;

end
endtask


//reset task


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

//data in task

task dataip(input [1:0]i);
begin                               // To drive our data. (00, 01,10)
@(negedge clk)
datain=i;
end
endtask
```

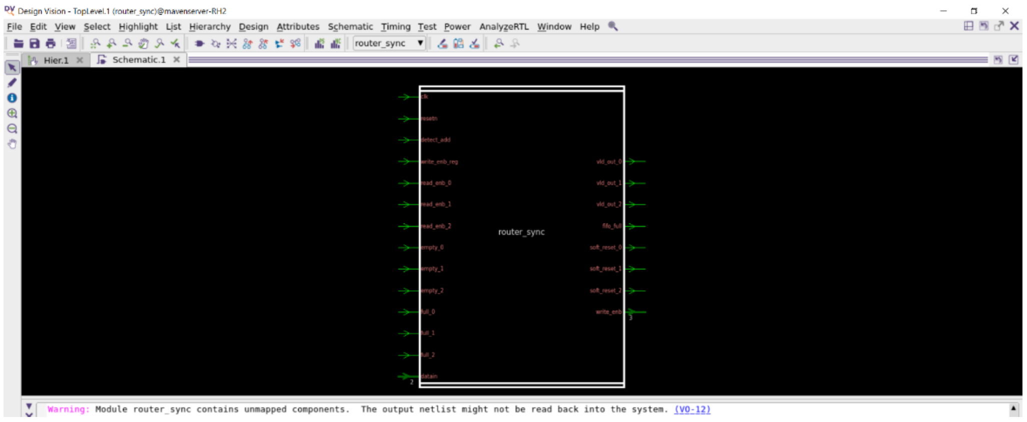


```verilog
@(negedge clk)
datain=i;
end
endtask


initial begin    // calling tasks.
initialize;
restn;
dataip(2);
detect_add=1'b1;       // after the two steps its stored in the temp as 10.

dataip(2);
write_enb_reg=1'b1;               // we observe write_enb=100
#Tp;
dataip(3);
#Tp dataip(1);
//full_0=1;
//full_2=1;
full_1=1;
#40 empty_0=0;               // vld_out_0=1
#640;      // for 30 clock cycles 30x20=600.

read_enb_0=1'b1;
#40 $finish;
end
endmodule
```

## SIMULATION:

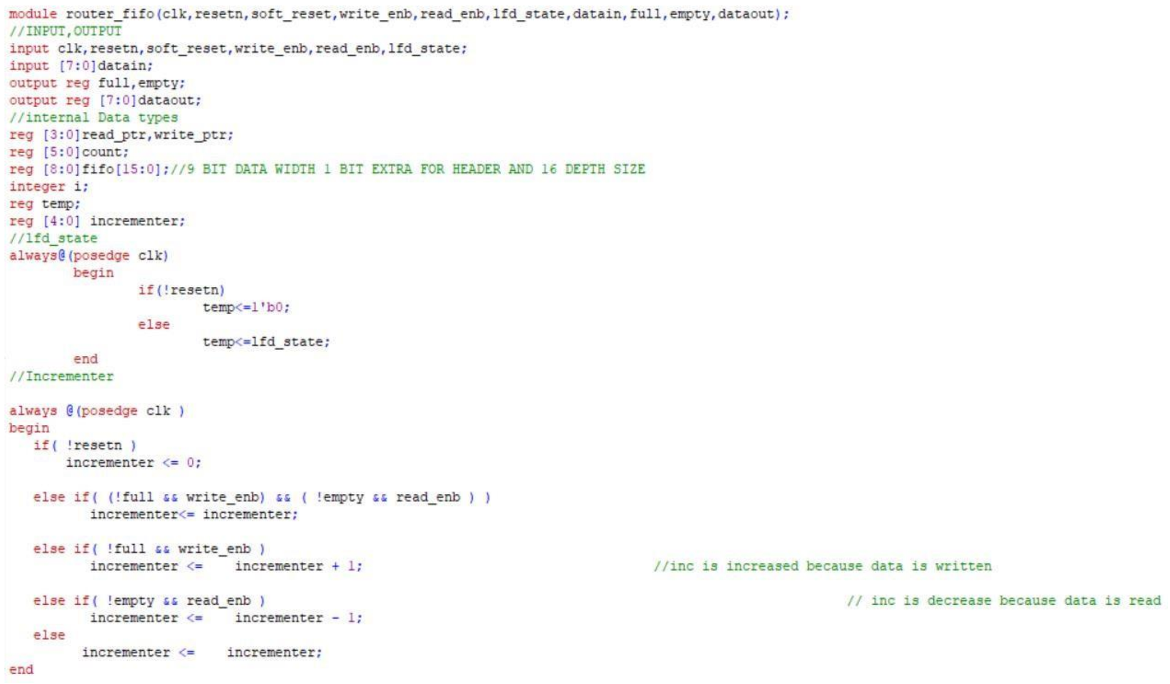


## SYNTHESIS:

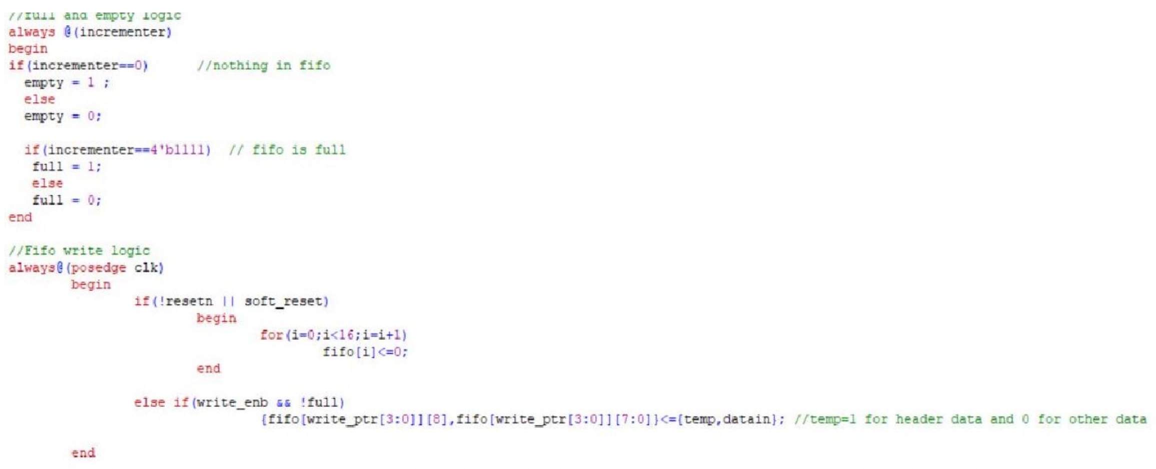


```
Warning: Module router_sync contains unmapped components.  The output netlist might not be read back into the system. (VO-12)
```

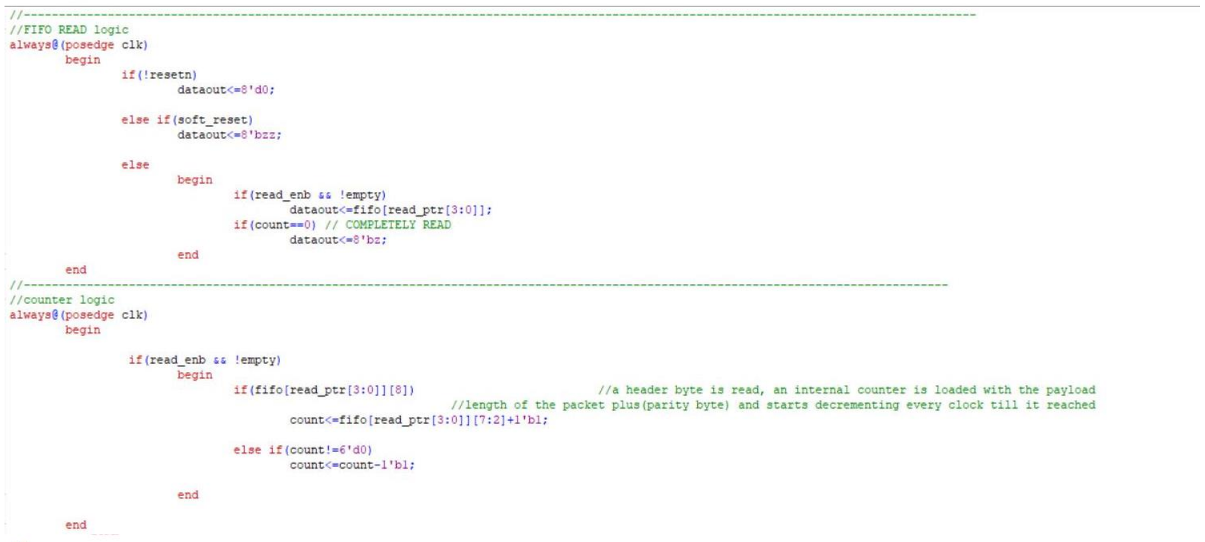
## ROUTER FIFO RTL



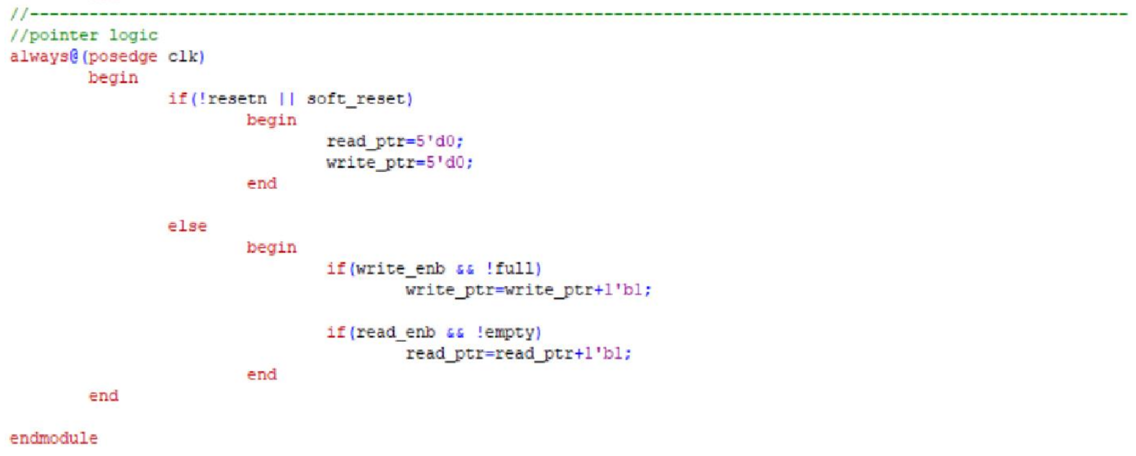
```
module router_fifo(clk,resetn,soft_reset,write_enb,read_enb,lfd_state,datain,full,empty,dataout);
//INPUT,OUTPUT
input clk,resetn,soft_reset,write_enb,read_enb,lfd_state;
input [7:0]datain;
output reg full,empty;
output reg [7:0]dataout;
//internal Data types
reg [3:0]read_ptr,write_ptr;
reg [5:0]count;
reg [8:0]fifo[15:0];//9 BIT DATA WIDTH 1 BIT EXTRA FOR HEADER AND 16 DEPTH SIZE
integer i;
reg temp;
reg [4:0] incrementer;
//lfd_state
always@(posedge clk)
        begin
                if(!resetn)
                        temp<=1'b0;
                else
                        temp<=lfd_state;
        end
//Incrementer

always @(posedge clk )
begin
   if( !resetn )
       incrementer <= 0;

   else if( (!full && write_enb) && ( !empty && read_enb ) )
          incrementer<= incrementer;

   else if( !full && write_enb )
          incrementer <=    incrementer + 1;                                    //inc is increased because data is written

   else if( !empty && read_enb )                                                                    // inc is decrease because data is read
          incrementer <=    incrementer - 1;
   else
         incrementer <=    incrementer;
end
```



```
//full and empty logic
always @(incrementer)
begin
if(incrementer==0)       //nothing in fifo
  empty = 1 ;
  else
  empty = 0;

  if(incrementer==4'b1111)  // fifo is full
   full = 1;
   else
   full = 0;
end

//Fifo write logic
always@(posedge clk)
        begin
                if(!resetn || soft_reset)
                        begin
                                for(i=0;i<16;i=i+1)
                                        fifo[i]<=0;
                        end

                else if(write_enb && !full)
                                {fifo[write_ptr[3:0]][8],fifo[write_ptr[3:0]][7:0]}<={temp,datain}; //temp=1 for header data and 0 for other data

        end
```

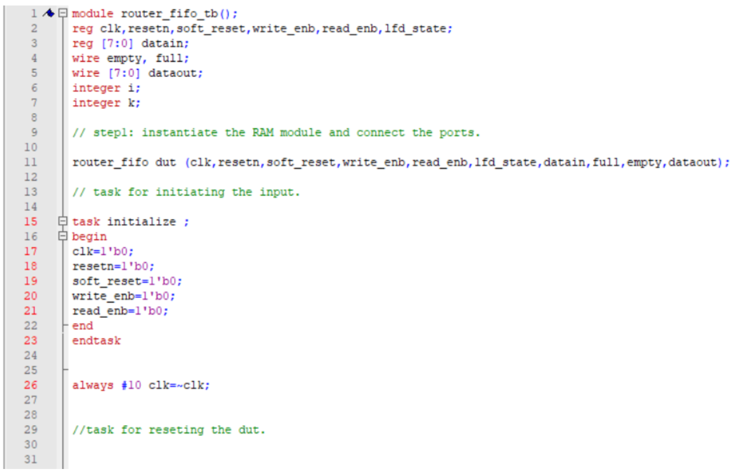


```
//-----------------------------------------------------------------------------------------------------------------------------
//FIFO READ logic
always@(posedge clk)
        begin
                if(!resetn)
                        dataout<=8'd0;

                else if(soft_reset)
                        dataout<=8'bzz;

                else
                        begin
                                if(read_enb && !empty)
                                        dataout<=fifo[read_ptr[3:0]];
                                if(count==0) // COMPLETELY READ
                                        dataout<=8'bz;
                        end
        end
//--------------------------------------------------------------------------------------------------------------------------
//counter logic
always@(posedge clk)
        begin

                 if(read_enb && !empty)
                        begin
                                if(fifo[read_ptr[3:0]][8])                          //a header byte is read, an internal counter is loaded with the payload
                                                                        //length of the packet plus(parity byte) and starts decrementing every clock till it reached
                                        count<=fifo[read_ptr[3:0]][7:2]+1'b1;

                                else if(count!=6'd0)
                                        count<=count-1'b1;

                        end

        end
```

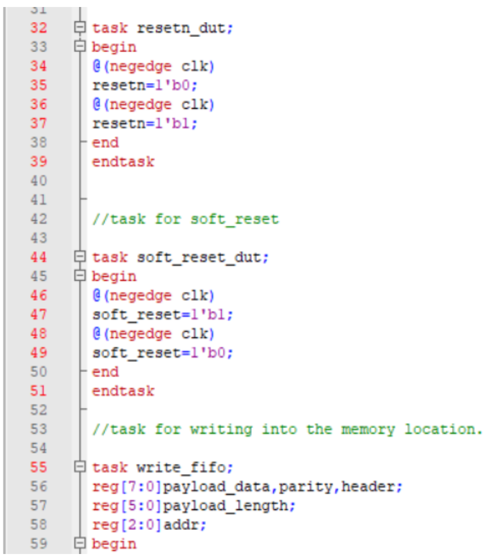


```
//-----------------------------------------------------------------------------------------------------------
//pointer logic
always@(posedge clk)
        begin
                if(!resetn || soft_reset)
                        begin
                                read_ptr=5'd0;
                                write_ptr=5'd0;
                        end

                else
                        begin
                                if(write_enb && !full)
                                        write_ptr=write_ptr+1'b1;

                                if(read_enb && !empty)
                                        read_ptr=read_ptr+1'b1;
                        end
        end

endmodule
```

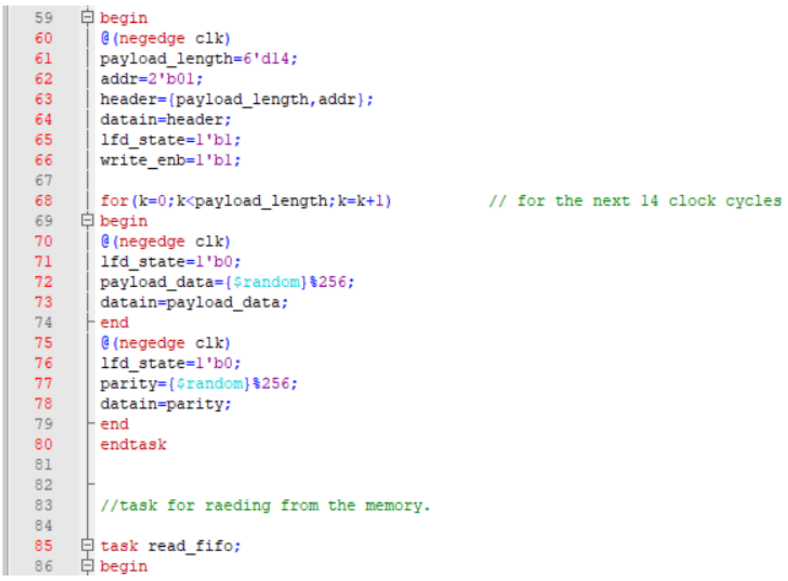
**FIFO TB:**



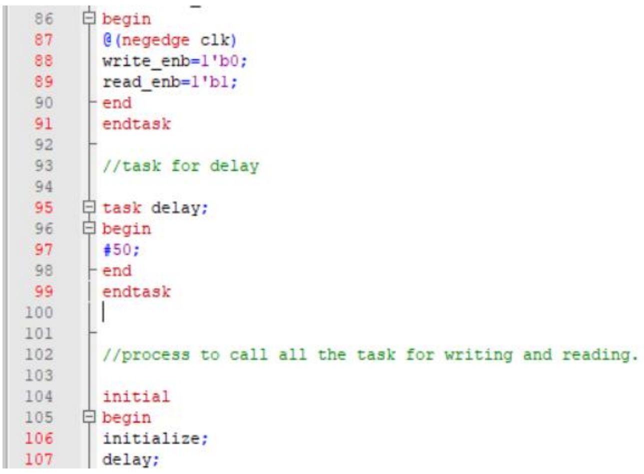
```
module router_fifo_tb();
reg clk,resetn,soft_reset,write_enb,read_enb,lfd_state;
reg [7:0] datain;
wire empty, full;
wire [7:0] dataout;
integer i;
integer k;

// step1: instantiate the RAM module and connect the ports.

router_fifo dut (clk,resetn,soft_reset,write_enb,read_enb,lfd_state,datain,full,empty,dataout);

// task for initiating the input.

task initialize ;
begin
clk=1'b0;
resetn=1'b0;
soft_reset=1'b0;
write_enb=1'b0;
read_enb=1'b0;
end
endtask


always #10 clk=~clk;


//task for reseting the dut.


```



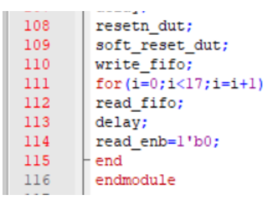
```

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


//task for soft_reset

task soft_reset_dut;
begin
@(negedge clk)
soft_reset=1'b1;
@(negedge clk)
soft_reset=1'b0;
end
endtask

//task for writing into the memory location.

task write_fifo;
reg[7:0]payload_data,parity,header;
reg[5:0]payload_length;
reg[2:0]addr;
begin
```



```
begin
@(negedge clk)
payload_length=6'd14;
addr=2'b01;
header={payload_length,addr};
datain=header;
lfd_state=1'b1;
write_enb=1'b1;

for(k=0;k<payload_length;k=k+1)          // for the next 14 clock cycles
begin
@(negedge clk)
lfd_state=1'b0;
payload_data={$random}%256;
datain=payload_data;
end
@(negedge clk)
lfd_state=1'b0;
parity={$random}%256;
datain=parity;
end
endtask


//task for raeding from the memory.

task read_fifo;
begin
```

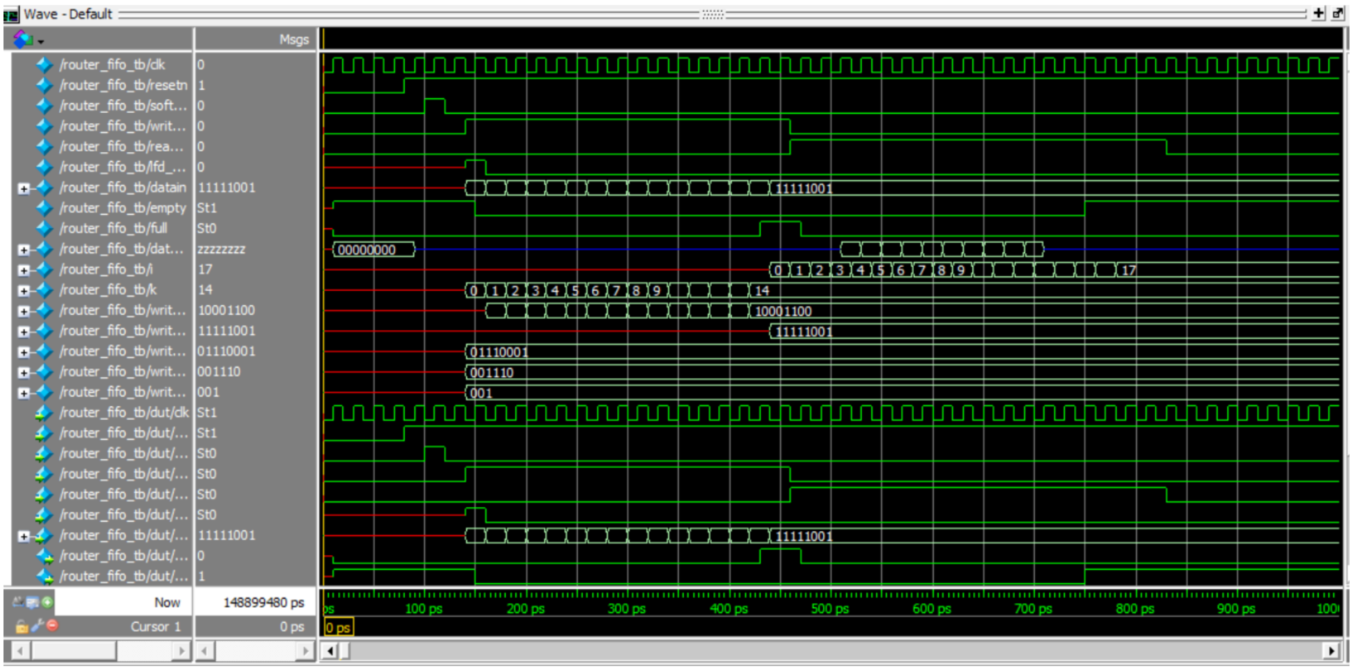


```
begin
@(negedge clk)
write_enb=1'b0;
read_enb=1'b1;
end
endtask

//task for delay

task delay;
begin
#50;
end
endtask


//process to call all the task for writing and reading.

initial
begin
initialize;
delay;
```



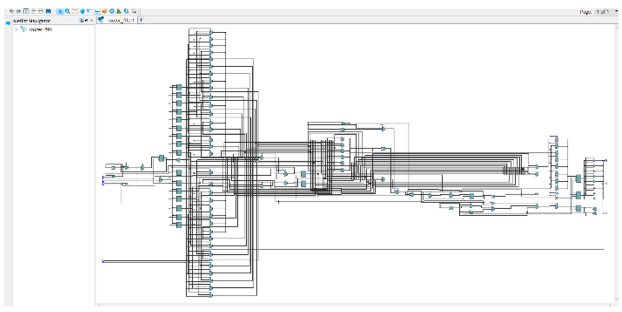
```
resetn_dut;
soft_reset_dut;
write_fifo;
for(i=0;i<17;i=i+1)
read_fifo;
delay;
read_enb=1'b0;
end
endmodule
```

## SIMULATION:

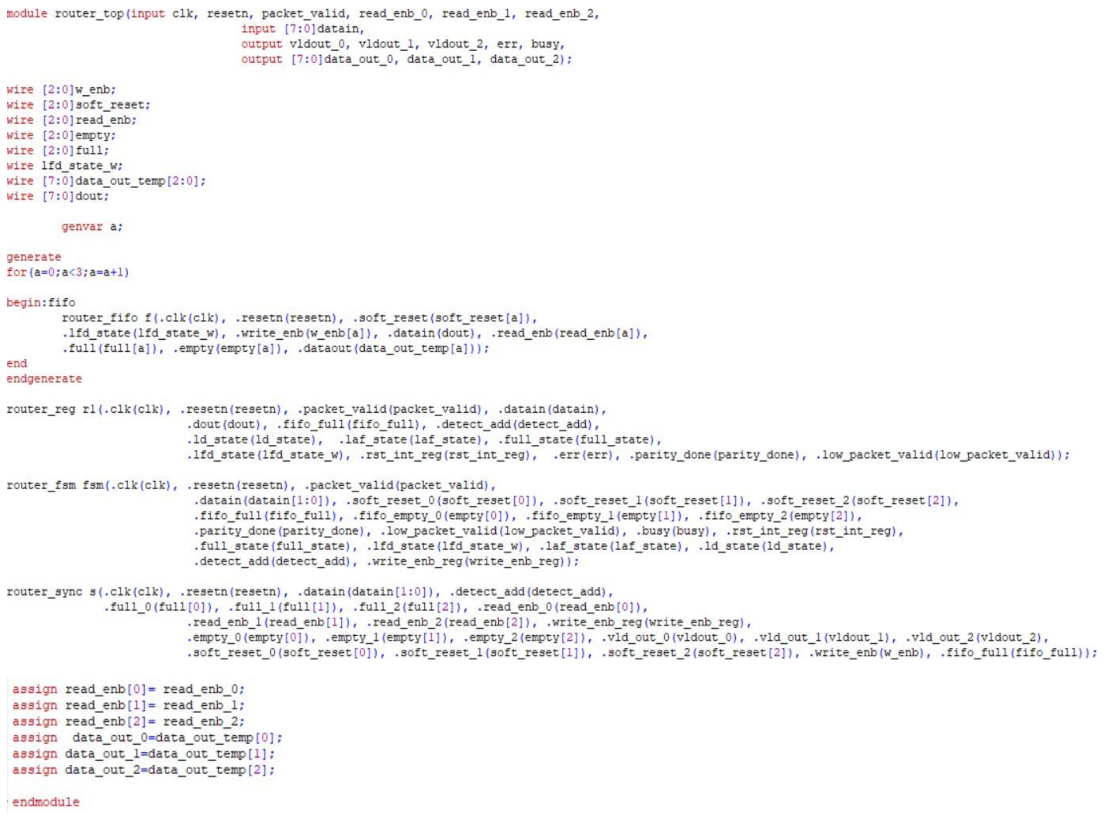


```
Wave - Default
                              Msgs
/router_fifo_tb/clk           0
/router_fifo_tb/resetn        1
/router_fifo_tb/soft...       0
/router_fifo_tb/writ...       0
/router_fifo_tb/rea...        0
/router_fifo_tb/lfd_...       0
/router_fifo_tb/datain        11111001
/router_fifo_tb/empty         St1
/router_fifo_tb/full          St0
/router_fifo_tb/dat...        zzzzzzzz
/router_fifo_tb/i             17
/router_fifo_tb/k             14
/router_fifo_tb/writ...       10001100
/router_fifo_tb/writ...       11111001
/router_fifo_tb/writ...       01110001
/router_fifo_tb/writ...       001110
/router_fifo_tb/writ...       001
/router_fifo_tb/dut/clk       St1
/router_fifo_tb/dut/...       St1
/router_fifo_tb/dut/...       St0
/router_fifo_tb/dut/...       St0
/router_fifo_tb/dut/...       St0
/router_fifo_tb/dut/...       St0
/router_fifo_tb/dut/...       11111001
/router_fifo_tb/dut/...       0
/router_fifo_tb/dut/...       1
Now                           148899480 ps
Cursor 1                      0 ps
```

## SYNTHESIS:

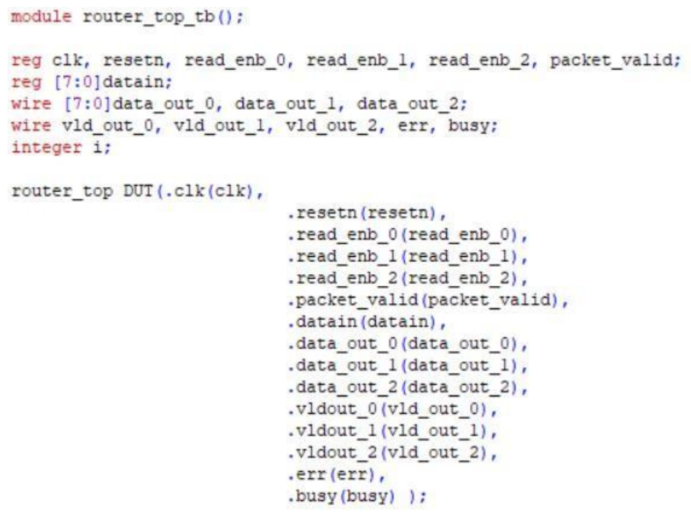


**ROUTER TOP : RTL**

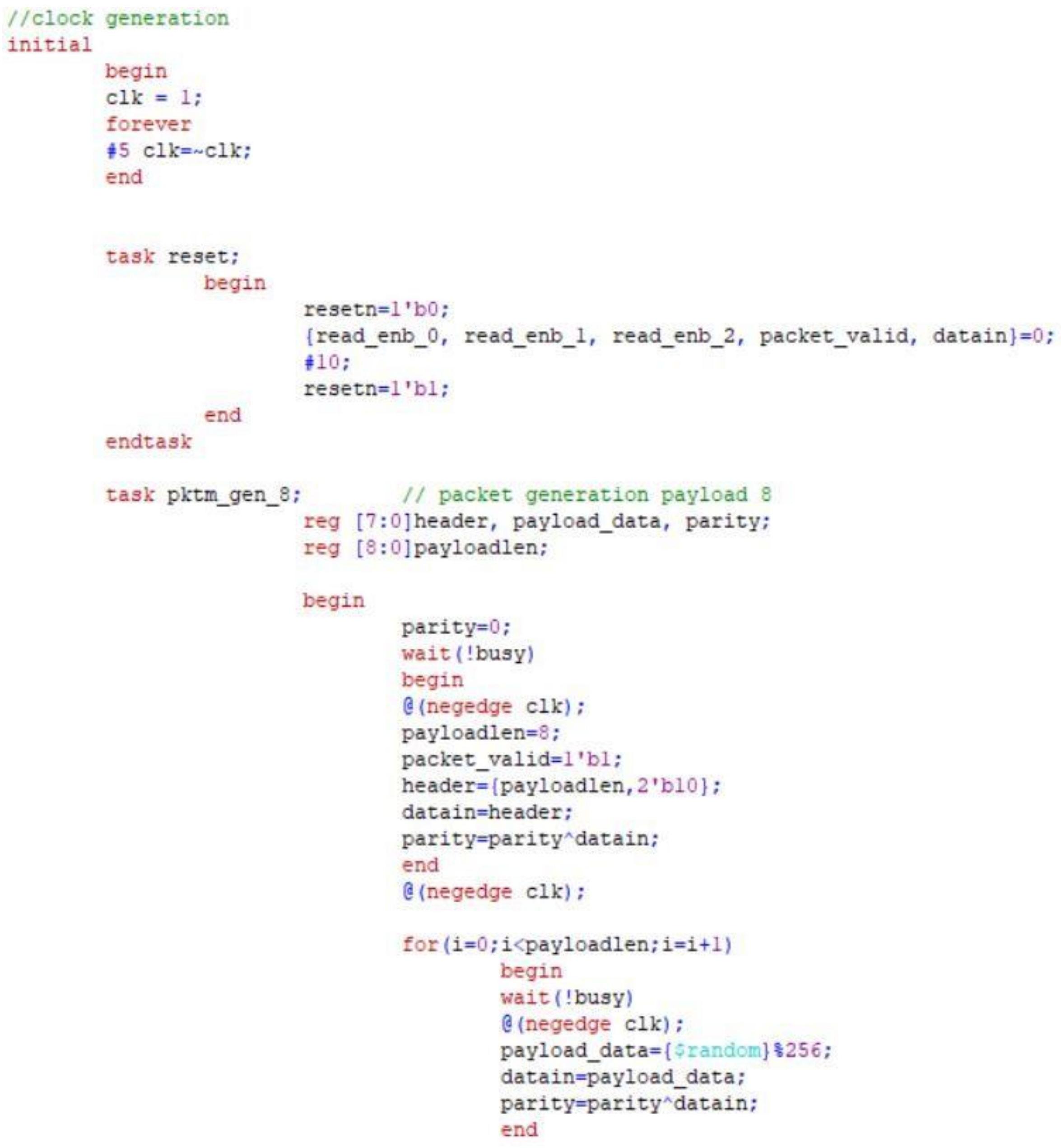


```
module router_top(input clk, resetn, packet_valid, read_enb_0, read_enb_1, read_enb_2,
                                input [7:0]datain,
                                output vldout_0, vldout_1, vldout_2, err, busy,
                                output [7:0]data_out_0, data_out_1, data_out_2);

wire [2:0]w_enb;
wire [2:0]soft_reset;
wire [2:0]read_enb;
wire [2:0]empty;
wire [2:0]full;
wire lfd_state_w;
wire [7:0]data_out_temp[2:0];
wire [7:0]dout;

        genvar a;

generate
for(a=0;a<3;a=a+1)

begin:fifo
        router_fifo f(.clk(clk), .resetn(resetn), .soft_reset(soft_reset[a]),
        .lfd_state(lfd_state_w), .write_enb(w_enb[a]), .datain(dout), .read_enb(read_enb[a]),
        .full(full[a]), .empty(empty[a]), .dataout(data_out_temp[a]));
end
endgenerate

router_reg r1(.clk(clk), .resetn(resetn), .packet_valid(packet_valid), .datain(datain),
                          .dout(dout), .fifo_full(fifo_full), .detect_add(detect_add),
                          .ld_state(ld_state),  .laf_state(laf_state), .full_state(full_state),
                          .lfd_state(lfd_state_w), .rst_int_reg(rst_int_reg),  .err(err), .parity_done(parity_done), .low_packet_valid(low_packet_valid));

router_fsm fsm(.clk(clk), .resetn(resetn), .packet_valid(packet_valid),
                           .datain(datain[1:0]), .soft_reset_0(soft_reset[0]), .soft_reset_1(soft_reset[1]), .soft_reset_2(soft_reset[2]),
                           .fifo_full(fifo_full), .fifo_empty_0(empty[0]), .fifo_empty_1(empty[1]), .fifo_empty_2(empty[2]),
                           .parity_done(parity_done), .low_packet_valid(low_packet_valid), .busy(busy), .rst_int_reg(rst_int_reg),
                           .full_state(full_state), .lfd_state(lfd_state_w), .laf_state(laf_state), .ld_state(ld_state),
                           .detect_add(detect_add), .write_enb_reg(write_enb_reg));

router_sync s(.clk(clk), .resetn(resetn), .datain(datain[1:0]), .detect_add(detect_add),
             .full_0(full[0]), .full_1(full[1]), .full_2(full[2]), .read_enb_0(read_enb[0]),
                          .read_enb_1(read_enb[1]), .read_enb_2(read_enb[2]), .write_enb_reg(write_enb_reg),
                          .empty_0(empty[0]), .empty_1(empty[1]), .empty_2(empty[2]), .vld_out_0(vldout_0), .vld_out_1(vldout_1), .vld_out_2(vldout_2),
                          .soft_reset_0(soft_reset[0]), .soft_reset_1(soft_reset[1]), .soft_reset_2(soft_reset[2]), .write_enb(w_enb), .fifo_full(fifo_full));

assign read_enb[0]= read_enb_0;
assign read_enb[1]= read_enb_1;
assign read_enb[2]= read_enb_2;
assign  data_out_0=data_out_temp[0];
assign data_out_1=data_out_temp[1];
assign data_out_2=data_out_temp[2];

endmodule
```

**TOP TB:**

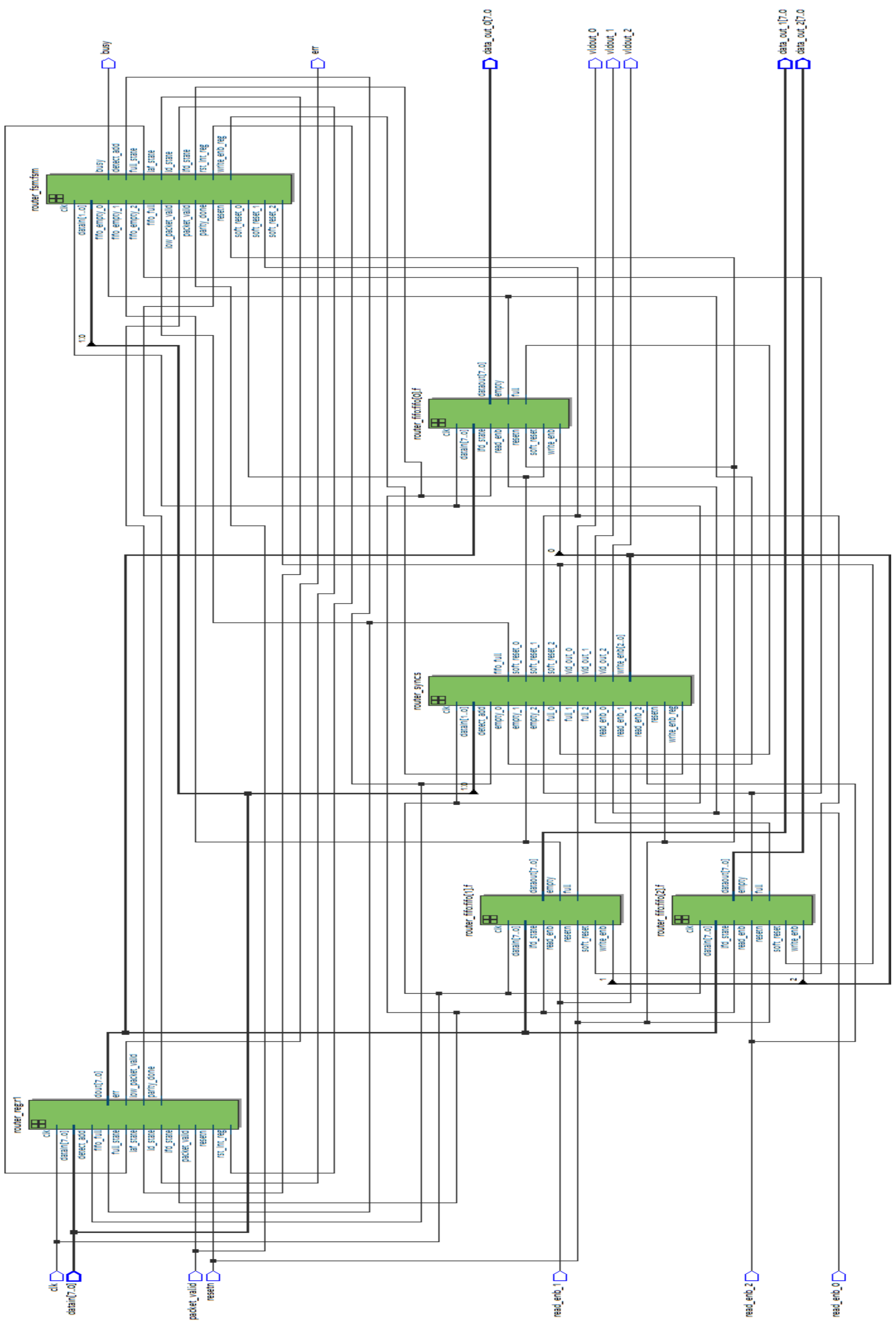


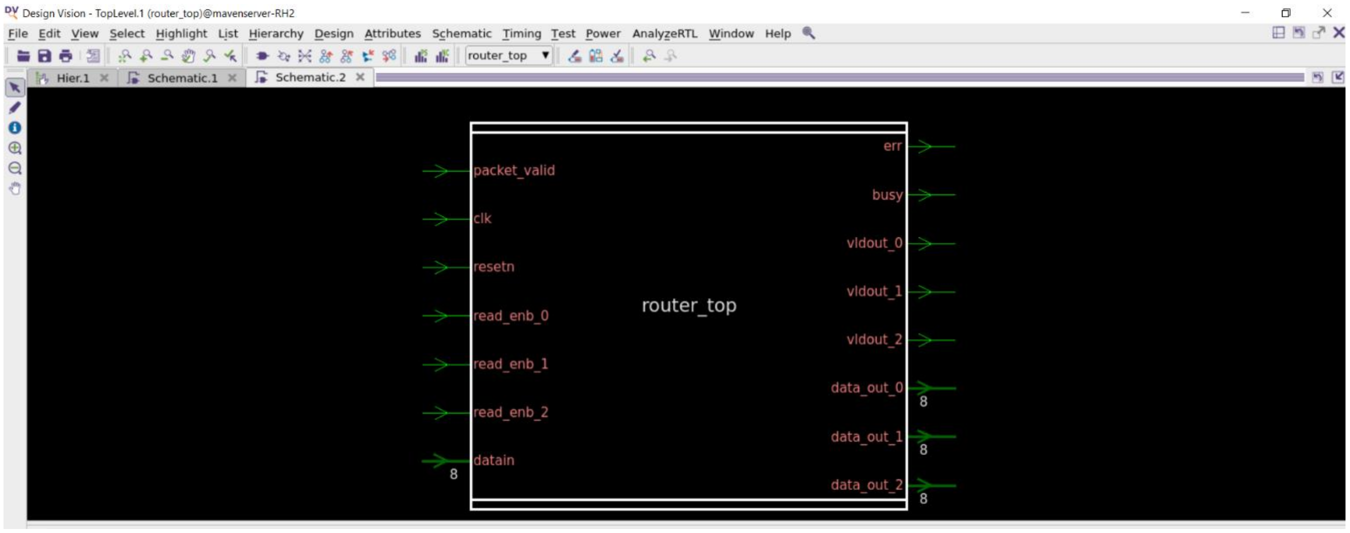
```
module router_top_tb();

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(.clk(clk),
                        .resetn(resetn),
                        .read_enb_0(read_enb_0),
                        .read_enb_1(read_enb_1),
                        .read_enb_2(read_enb_2),
                        .packet_valid(packet_valid),
                        .datain(datain),
                        .data_out_0(data_out_0),
                        .data_out_1(data_out_1),
                        .data_out_2(data_out_2),
                        .vldout_0(vld_out_0),
                        .vldout_1(vld_out_1),
                        .vldout_2(vld_out_2),
                        .err(err),
                        .busy(busy) );
```



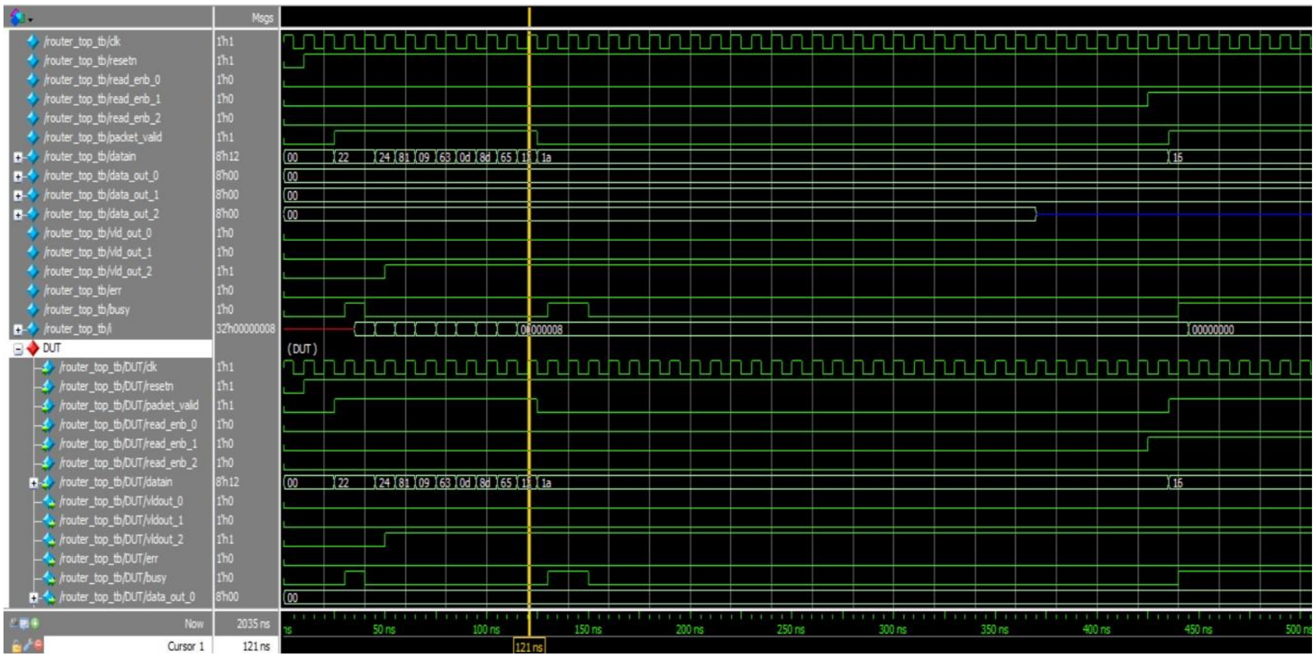
```
//clock generation
initial
        begin
        clk = 1;
        forever
        #5 clk=~clk;
        end


        task reset;
                begin
                        resetn=1'b0;
                        {read_enb_0, read_enb_1, read_enb_2, packet_valid, datain}=0;
                        #10;
                        resetn=1'b1;
                end
        endtask

        task pktm_gen_8;        // packet generation payload 8
                        reg [7:0]header, payload_data, parity;
                        reg [8:0]payloadlen;

                        begin
                                parity=0;
                                wait(!busy)
                                begin
                                @(negedge clk);
                                payloadlen=8;
                                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)
                                        @(negedge clk);
                                        payload_data={$random}%256;
                                        datain=payload_data;
                                        parity=parity^datain;
                                        end
                wait(!busy)
                    @(negedge clk);
                    packet_valid=0;
                    datain=parity;
                    repeat(30)
            @(negedge clk);
            read_enb_1=1'b1;
            end
endtask

    task pktm_gen_5;        // packet generation payload 8
            reg [7:0]header, payload_data, parity;
            reg [4:0]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)
                    @(negedge clk);
                    payload_data={$random}%256;
                    datain=payload_data;
                    parity=parity^datain;
                    end

                wait(!busy)
                    @(negedge clk);
                    packet_valid=0;
                    datain=parity;
                    repeat(30)
            @(negedge clk);
            read_enb_2=1'b1;
            end
endtask
    initial
        begin
            reset;
            #10;
            pktm_gen_8;
            pktm_gen_5;
            #1000;
            $finish;
        end


endmodule
```

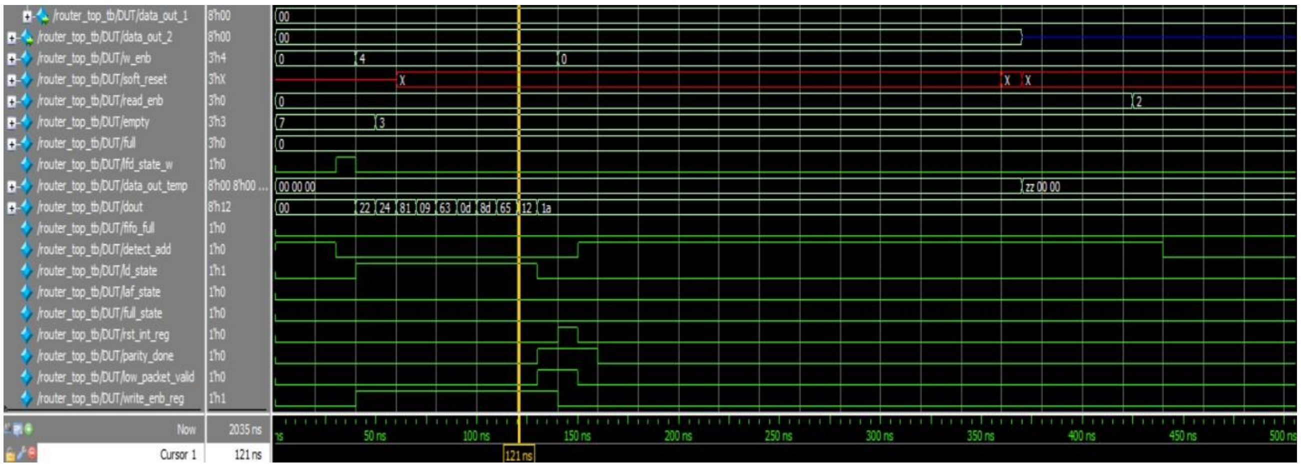
**Synthesis Circuit:**




Design Vision - TopLevel.1 (router_top)@mavenserver-RH2
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
router_top
Hier.1
Schematic.1
Schematic.2
router_top
packet_valid
clk
resetn
read_enb_0
read_enb_1
read_enb_2
datain
8
err
busy
vldout_0
vldout_1
vldout_2
data_out_0
8
data_out_1
8
data_out_2
8

**Simulation Waveform:**


Msgs
/router_top_tb/clk 1'h1
/router_top_tb/resetn 1'h1
/router_top_tb/read_enb_0 1'h0
/router_top_tb/read_enb_1 1'h0
/router_top_tb/read_enb_2 1'h0
/router_top_tb/packet_valid 1'h1
/router_top_tb/datain 8'h12
/router_top_tb/data_out_0 8'h00
/router_top_tb/data_out_1 8'h00
/router_top_tb/data_out_2 8'h00
/router_top_tb/vld_out_0 1'h0
/router_top_tb/vld_out_1 1'h0
/router_top_tb/vld_out_2 1'h1
/router_top_tb/err 1'h0
/router_top_tb/busy 1'h0
/router_top_tb/i 32'h00000008
DUT
/router_top_tb/DUT/clk 1'h1
/router_top_tb/DUT/resetn 1'h1
/router_top_tb/DUT/packet_valid 1'h1
/router_top_tb/DUT/read_enb_0 1'h0
/router_top_tb/DUT/read_enb_1 1'h0
/router_top_tb/DUT/read_enb_2 1'h0
/router_top_tb/DUT/datain 8'h12
/router_top_tb/DUT/vldout_0 1'h0
/router_top_tb/DUT/vldout_1 1'h0
/router_top_tb/DUT/vldout_2 1'h1
/router_top_tb/DUT/err 1'h0
/router_top_tb/DUT/busy 1'h0
/router_top_tb/DUT/data_out_0 8'h00
Now 2035 ns
Cursor 1 121 ns
121 ns


/router_top_tb/DUT/data_out_1 8'h00
/router_top_tb/DUT/data_out_2 8'h00
/router_top_tb/DUT/w_enb 3'h4
/router_top_tb/DUT/soft_reset 3'hX
/router_top_tb/DUT/read_enb 3'h0
/router_top_tb/DUT/empty 3'h3
/router_top_tb/DUT/full 3'h0
/router_top_tb/DUT/lfd_state_w 1'h0
/router_top_tb/DUT/data_out_temp 8'h00 8'h00 ...
/router_top_tb/DUT/dout 8'h12
/router_top_tb/DUT/fifo_full 1'h0
/router_top_tb/DUT/detect_add 1'h0
/router_top_tb/DUT/ld_state 1'h1
/router_top_tb/DUT/laf_state 1'h0
/router_top_tb/DUT/full_state 1'h0
/router_top_tb/DUT/rst_int_reg 1'h0
/router_top_tb/DUT/parity_done 1'h0
/router_top_tb/DUT/low_packet_valid 1'h0
/router_top_tb/DUT/write_enb_reg 1'h1
Now 2035 ns
Cursor 1 121 ns
121 ns