**Router 1x3 Project Report**

****

By,

T Raja Aadhithan

BRN10

**INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MODULE** | FIFO | FSM | SYNC | REGISTER | TOP |
| **PAGE NO** | 2 | 8 | 14 | 19 | 25 |

**FIFO :**

RTL:

module router\_fifo(input clock, resetn, write\_enb, read\_enb, soft\_reset, lfd\_state,

                   input [7:0] data\_in, output full, empty, output [7:0] data\_out);

reg [8:0] mem [0:15];

reg [3:0] rd\_pointer, wr\_pointer;

reg [8:0] packet\_out;

wire [8:0]packet;

reg [5:0] length;

reg [4:0] status\_count = 0;

assign empty = (status\_count == 0 ) ? 1'b1 : 1'b0 ;

assign full = (status\_count == 5'd16) ? 1'b1 : 1'b0 ;

assign packet[7:0] = data\_in[7:0];

assign packet[8] = lfd\_state;

assign data\_out = resetn ?  packet\_out[7:0] : 8'd0;

always@(posedge clock) begin

    if(full == 1'b0 && write\_enb & empty == 1'b0 && read\_enb) status\_count <= status\_count;

    else if (full == 1'b0 && write\_enb) status\_count <= status\_count + 1'b1;

    else if(empty == 1'b0 && read\_enb ) status\_count <= status\_count - 1'b1;

    else status\_count <= status\_count;

end

//write block:

always@(posedge clock) begin

    if(!resetn || soft\_reset) begin //reset

        wr\_pointer <= 4'd0;

    end

    else if(full == 1'b0 && write\_enb) begin //only write

            mem[wr\_pointer[3:0]] <= packet;

            wr\_pointer <= wr\_pointer + 1'b1;

    end

end

//read block:

always@(posedge clock) begin

    if(!resetn || soft\_reset) begin //reset

        rd\_pointer <= 4'd0;

        packet\_out <= 8'dz;

    end

    else begin

        if(empty == 1'b0 && read\_enb ) begin //only write

                packet\_out <= mem[rd\_pointer[3:0]];

                rd\_pointer <= rd\_pointer + 1'b1;

        end

        if (length == 0 && !mem[rd\_pointer[3:0]][8]) packet\_out <= 8'bz;

    end

end

always@(posedge clock)begin

    if (!resetn || soft\_reset) begin

        length <= 0;

    end

    else if(empty == 1'b0 && read\_enb ) begin

        if(mem[rd\_pointer[3:0]][8])begin

                length <= mem[rd\_pointer[3:0]][7:2] + 1'b1;

        end

        else if (length != 0)  begin

                length <= length - 1'b1;

        end

    end

end

endmodule

TB :

module router\_fifo\_tb();

reg clock, resetn, write\_enb, read\_enb, soft\_reset, lfd\_state;

reg [7:0] data\_in;

wire full, empty;

wire [7:0] data\_out;

router\_fifo DUT(clock, resetn, write\_enb, read\_enb, soft\_reset, lfd\_state, data\_in, full, empty, data\_out);

initial begin

    clock = 1'b1;

    forever #5 clock = !clock;

end

task write(input [7:0]data);

    begin

        lfd\_state = 0;

        write\_enb = 1;

        data\_in = data;

        #10;

        write\_enb = 0;

    end

endtask

task header(input [5:0]data,input [1:0]set);

    begin

        write\_enb = 1;

        lfd\_state = 1;

        data\_in = {data,set};

        #10;

        write\_enb = 0;

    end

endtask

task read();

    begin

        read\_enb = 1;

        #10;

        read\_enb = 0;

    end

endtask

task readandwrite(input [7:0]data);

begin

        lfd\_state = 0;

        write\_enb = 1;

        data\_in = data;

        read\_enb = 1;

        #10;

        read\_enb = 0;

        write\_enb = 0;

    end

endtask

initial begin

    $monitor("@time: %t, data\_in: %h, data\_out: %h, lfd: %b, full: %b, empty: %b",$time, data\_in, data\_out, lfd\_state, full, empty);

    resetn = 1'b0;

    repeat(3)

    @(negedge clock);

    resetn = 1'b1;

    header(6'd14,{$random}%3);// 6'dx denotes the paylength, %3 selects the fifo from 0 to 2

    repeat(15) // payload length is denoted here

    write({$random}%256);

    repeat(15)

    read();

    repeat(3)

    readandwrite({$random}%256);

    repeat(15)

    read();

    soft\_reset = 1'b1;

    #20;

    soft\_reset = 1'b0;

    #20;

    header(6'd4,{$random}%3);

    repeat(4) // payload length is denoted here

    write({$random}%256);

    repeat(15)

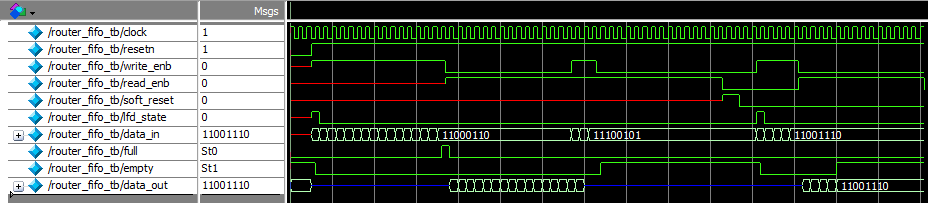
    read();

    $finish;

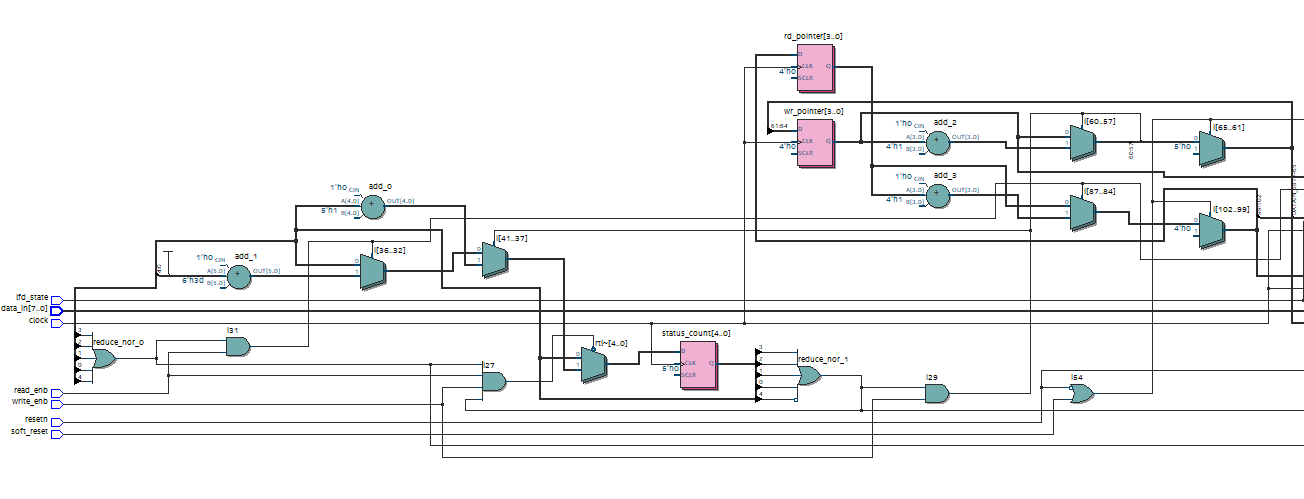
end

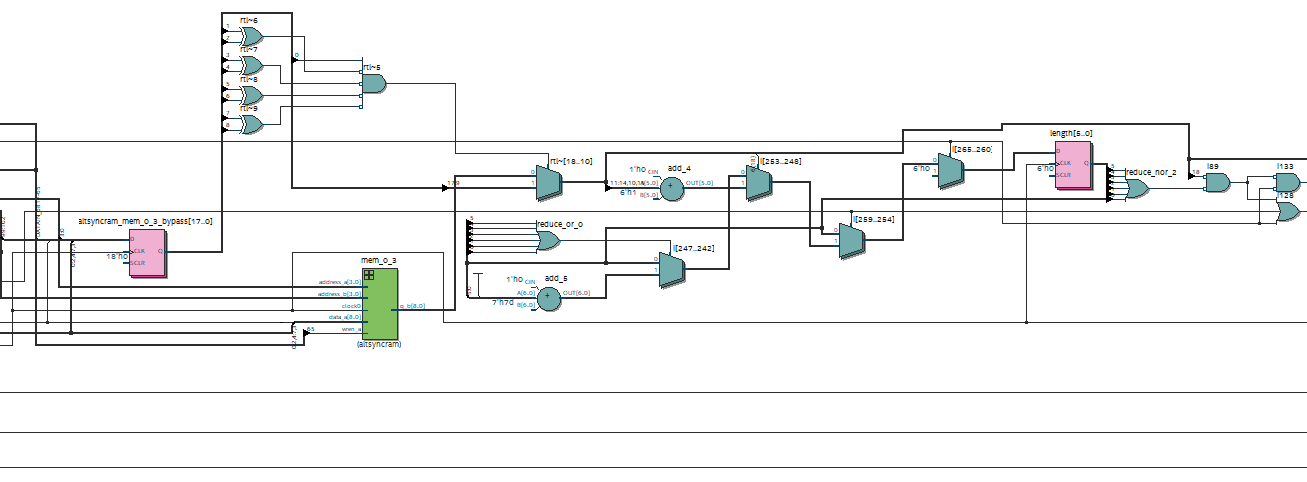
endmodule

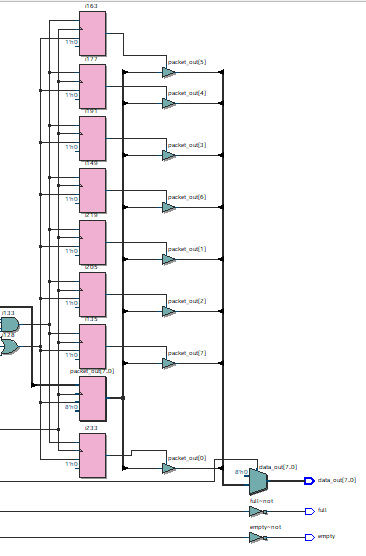
Waveform :



RTL Netlist:







**FSM :**

RTL:

module router\_fsm(input clock, resetn, pkt\_valid, parity\_done,

                        soft\_reset\_0, soft\_reset\_1, soft\_reset\_2,

                        fifo\_full, low\_pkt\_valid,

                        fifo\_empty\_0, fifo\_empty\_1, fifo\_empty\_2,

                        input [1:0] data\_in,

                        output busy, detect\_add, ld\_state, laf\_state,

                        full\_state, write\_enb\_reg, rst\_int\_reg, lfd\_state);

parameter Decode\_Address = 3'b000,

          Load\_First\_Data = 3'b001,

          Wait\_Till\_Empty = 3'b010,

          Load\_Data = 3'b011,

          Check\_Parity\_Error = 3'b100,

          Load\_Parity = 3'b101,

          Fifo\_Full\_State = 3'b110,

          Load\_After\_Full = 3'b111;

reg [2:0] state, next\_state;

always @(posedge clock) begin

    if (!resetn) state <= Decode\_Address;

    else if((soft\_reset\_0 && (data\_in[1:0] == 0)) | (soft\_reset\_1 && (data\_in[1:0] == 1)) | (soft\_reset\_2 && (data\_in[1:0] == 2))) state <= Decode\_Address;

    else state <= next\_state;

end

always @(\*) begin

    case(state)

    Decode\_Address : begin //1st case

        if(pkt\_valid & (data\_in[1:0] == 0 & fifo\_empty\_0 | data\_in[1:0] == 1 & fifo\_empty\_1 | data\_in[1:0] == 2 & fifo\_empty\_2))

        begin

            next\_state = Load\_First\_Data;

        end

        else if(pkt\_valid & (data\_in[1:0] == 0 & !fifo\_empty\_0 | data\_in[1:0] == 1 & !fifo\_empty\_1 | data\_in[1:0] == 2 & !fifo\_empty\_2))

        begin

            next\_state = Wait\_Till\_Empty;

        end

        else next\_state = Decode\_Address;

    end

    Load\_First\_Data : next\_state = Load\_Data; //2nd case

    Wait\_Till\_Empty : begin //3rd case

        if( (fifo\_empty\_0 && (data\_in[1:0] == 0)) || (fifo\_empty\_1 && (data\_in[1:0] == 1)) || (fifo\_empty\_2 && (data\_in[1:0] == 2)) ) begin

            next\_state = Load\_First\_Data;

        end

        else next\_state = Wait\_Till\_Empty;

    end

    Load\_Data : begin //4th case

        if(fifo\_full) next\_state = Fifo\_Full\_State;

        else if (!fifo\_full & !pkt\_valid) next\_state = Load\_Parity;

        else next\_state = Load\_Data;

    end

    Fifo\_Full\_State : begin //5th case

        if(!fifo\_full) next\_state = Load\_After\_Full;

        else next\_state = Fifo\_Full\_State;

    end

    Load\_Parity : next\_state = Check\_Parity\_Error; //6th case

    Check\_Parity\_Error : begin //7th case

        if(fifo\_full) next\_state= Fifo\_Full\_State;

        else next\_state = Decode\_Address;

    end

    Load\_After\_Full : begin //8th case

        if (parity\_done) next\_state = Decode\_Address;

        else if(low\_pkt\_valid) next\_state = Load\_Parity;

        else next\_state = Load\_Data;

    end

    endcase

end

assign detect\_add = (state == Decode\_Address);

assign lfd\_state = (state == Load\_First\_Data);

assign busy = (state == Load\_First\_Data) || (state == Load\_Parity) || (state == Fifo\_Full\_State) || (state == Load\_After\_Full) || (state == Wait\_Till\_Empty) || (state == Check\_Parity\_Error);

assign ld\_state = (state == Load\_Data)||(state == Load\_Parity) ;

assign write\_enb\_reg = (state == Load\_First\_Data)|(state == Load\_Data)||(state == Load\_Parity);

assign full\_state = (state == Fifo\_Full\_State);

assign laf\_state = (state == Load\_After\_Full);

assign rst\_int\_reg = (state == Check\_Parity\_Error);

endmodule

TB :

module router\_fsm\_tb();

reg clock, resetn, pkt\_valid, parity\_done, soft\_reset\_0, soft\_reset\_1, soft\_reset\_2,

    fifo\_full, low\_pkt\_valid, fifo\_empty\_0, fifo\_empty\_1, fifo\_empty\_2;

reg [1:0] data\_in;

wire busy, detect\_add, ld\_state, laf\_state, full\_state, write\_enb\_reg, rst\_int\_reg, lfd\_state;

router\_fsm dut(clock, resetn, pkt\_valid, parity\_done, soft\_reset\_0, soft\_reset\_1, soft\_reset\_2,

    fifo\_full, low\_pkt\_valid, fifo\_empty\_0, fifo\_empty\_1, fifo\_empty\_2,data\_in,

    busy, detect\_add, ld\_state, laf\_state, full\_state, write\_enb\_reg, rst\_int\_reg, lfd\_state);

initial begin

    clock = 1'b1;

    forever #5 clock = !clock;

end

task ts1();

    begin

        resetn = 0;

        @(negedge clock);

        resetn = 1;

        @(negedge clock);

        pkt\_valid = 1'b1;

        data\_in = 2'b0;

        fifo\_empty\_0 = 1'b1;

        @(negedge clock);

        fifo\_full = 1'b0;

        pkt\_valid = 1'b0;

    end

endtask

task ts2();

begin

    pkt\_valid = 1'b1;

    data\_in = 2'b10;

    fifo\_empty\_2 = 1'b1;

    repeat(3)

    @(negedge clock);

    fifo\_full = 1'b1;

    @(negedge clock);

    fifo\_full = 1'b0;

    @(negedge clock);

    parity\_done = 1'b0;

    low\_pkt\_valid = 1'b1;

    @(negedge clock);

    fifo\_full = 1'b0;

    pkt\_valid = 1'b0;

end

endtask

task ts3();

begin

    pkt\_valid = 1'b1;

    data\_in = 2'b10;

    fifo\_empty\_2 = 1'b1;

    repeat(3)

    @(negedge clock);

    fifo\_full = 1'b1;

    @(negedge clock);

    fifo\_full = 1'b0;

    @(negedge clock);

    parity\_done = 1'b0;

    low\_pkt\_valid = 1'b0;

    @(negedge clock);

    fifo\_full = 1'b0;

    pkt\_valid = 1'b0;

end

endtask

task ts4();

begin

    pkt\_valid = 1'b1;

    data\_in = 2'b0;

    fifo\_empty\_0 = 1'b0;

    #10;

    fifo\_empty\_0 = 1'b1;

    @(negedge clock);

    fifo\_full = 1'b0;

    pkt\_valid = 1'b0;

    @(negedge clock);

    fifo\_full = 1'b1;

    @(negedge clock);

    fifo\_full = 1'b0;

    parity\_done = 1'b1;

end

endtask

initial begin

    ts1();

    #100;

    ts2();

    #100;

    ts3();

    #100;

    ts4();

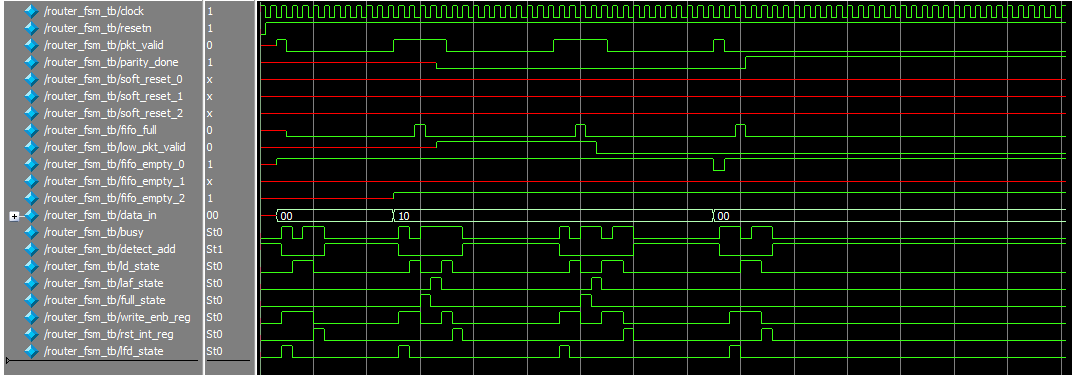
    #300;

    $finish;

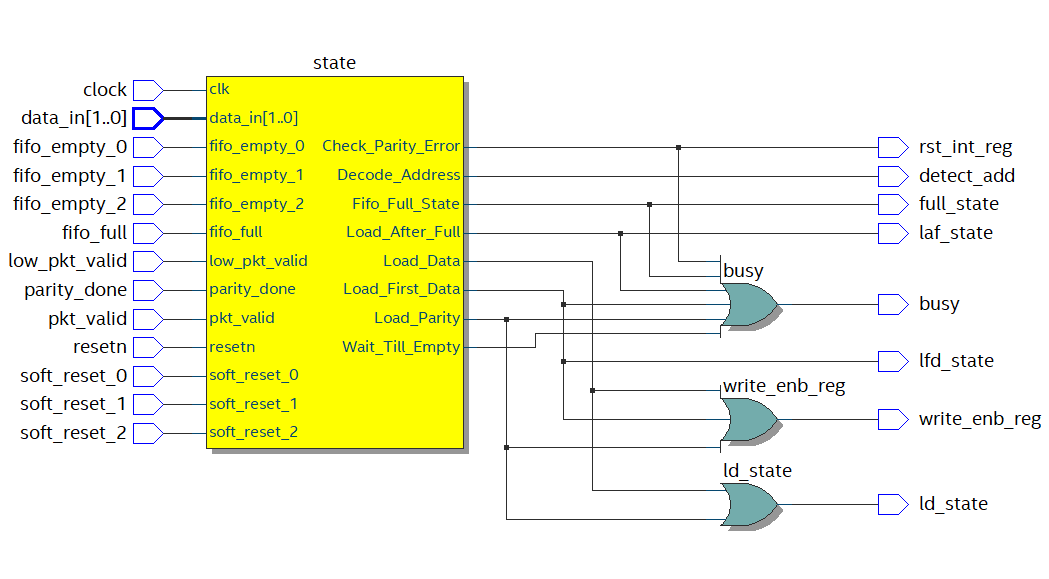
end

endmodule

Waveform :



RTL Netlist :



**Synchronizer:**

RTL:

module router\_sync(input detect\_add, write\_enb\_reg, clock, resetn,

                         read\_enb\_0, read\_enb\_1, read\_enb\_2,

                         empty\_0, empty\_1, empty\_2,

                         full\_0, full\_1, full\_2,

                   input [1:0] data\_in,

                   output reg vld\_out\_0, vld\_out\_1, vld\_out\_2,

                   output reg soft\_reset\_0, soft\_reset\_1, soft\_reset\_2,

                   output reg fifo\_full,

                   output reg [2:0] write\_enb);

reg [1:0] q;

reg [4:0] count0,count1,count2;

always@(\*)begin

    if(detect\_add) q<= data\_in;

    else q <= q;

    vld\_out\_0 <= !empty\_0;

    vld\_out\_1 <= !empty\_1;

    vld\_out\_2 <= !empty\_2;

    case(q)

    2'b00 : begin

            fifo\_full = full\_0;

            write\_enb = {2'b00, write\_enb\_reg};

            end

    2'b01 : begin

            fifo\_full = full\_1;

            write\_enb = {1'b0, write\_enb\_reg,1'b0};

            end

    2'b10 : begin

            fifo\_full = full\_2;

            write\_enb = {write\_enb\_reg, 2'b00};

            end

    default : begin

                fifo\_full = 0;

                write\_enb = 3'd0;

    end

    endcase

end

always@(posedge clock) begin

    if(!resetn)begin

        count0<=5'd0;

        count1<=5'd0;

        count2<=5'd0;

    end

    if (vld\_out\_0) begin

            if (!read\_enb\_0) begin

              if (count0 == 5'b11101)begin

                    soft\_reset\_0 <= 1'b1;

                    count0 <= 5'd0;

                end

                else begin

                    soft\_reset\_0 <= 1'b0;

                    count0 <= count0 + 1'b1;

                end

            end

    end

    if (vld\_out\_1) begin

            if (!read\_enb\_1) begin

              if (count1 == 5'b11101)begin

                    soft\_reset\_1 <= 1'b1;

                    count1 <= 5'd0;

                end

                else begin

                    soft\_reset\_1 <= 1'b0;

                    count1 <= count1 + 1'b1;

                end

            end

    end

    if (vld\_out\_2) begin

            if (!read\_enb\_2) begin

              if (count2 == 5'b11101)begin

                    soft\_reset\_2 <= 1'b1;

                    count2 <= 5'd0;

                end

                else begin

                    soft\_reset\_2 <= 1'b0;

                    count2 <= count2 + 1'b1;

                end

            end

    end

end

endmodule

TB:

module router\_sync\_tb();

  reg detect\_add, write\_enb\_reg, clock, resetn,

      read\_enb\_0, read\_enb\_1, read\_enb\_2,

      empty\_0, empty\_1, empty\_2,

      full\_0, full\_1, full\_2;

  reg [1:0] data\_in;

  wire vld\_out\_0, vld\_out\_1, vld\_out\_2,

       soft\_reset\_0, soft\_reset\_1, soft\_reset\_2,

       fifo\_full;

  wire [2:0] write\_enb;

  router\_sync dut(detect\_add, write\_enb\_reg, clock, resetn,

                  read\_enb\_0, read\_enb\_1, read\_enb\_2,

                  empty\_0, empty\_1, empty\_2,

                  full\_0, full\_1, full\_2,data\_in,vld\_out\_0,               vld\_out\_1, vld\_out\_2,

                  soft\_reset\_0, soft\_reset\_1, soft\_reset\_2,

                  fifo\_full,write\_enb);

  initial begin

    clock = 1'b1;

    forever #5 clock = !clock;

  end

  initial begin

    $monitor("@%3dns : q = %b, vldout = %b,%b,%b",$time,soft\_reset\_0,vld\_out\_0, vld\_out\_1, vld\_out\_2);

    resetn = 1'b0;

    #10;

    resetn = 1'b1;

    empty\_0 = 1'b1;

    empty\_1 = 1'b0;

    empty\_2 = 1'b0;

    data\_in = 2'b01;

    detect\_add = 1'b1;

    #10;

    detect\_add = 1'b0;

    full\_1 = 0;

    read\_enb\_1 = 1'b1;

    read\_enb\_2 = 1'b0;

    write\_enb\_reg = 1'b1;

    #10;

    full\_1 = 1;

    write\_enb\_reg = 1'b0;

    #3000;

    read\_enb\_2 = 1'b1;

    detect\_add = 1'b1;

    data\_in =2'b10;

    #10;

    detect\_add = 1'b0;

    full\_2 = 0;

    write\_enb\_reg = 1'b1;

    #30;

    full\_2 = 1;

    write\_enb\_reg = 1'b0;

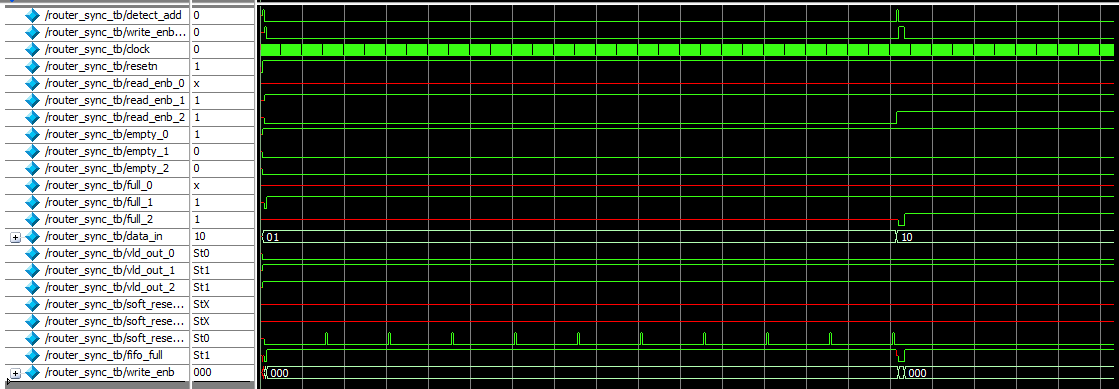
    #1000;

    $finish;

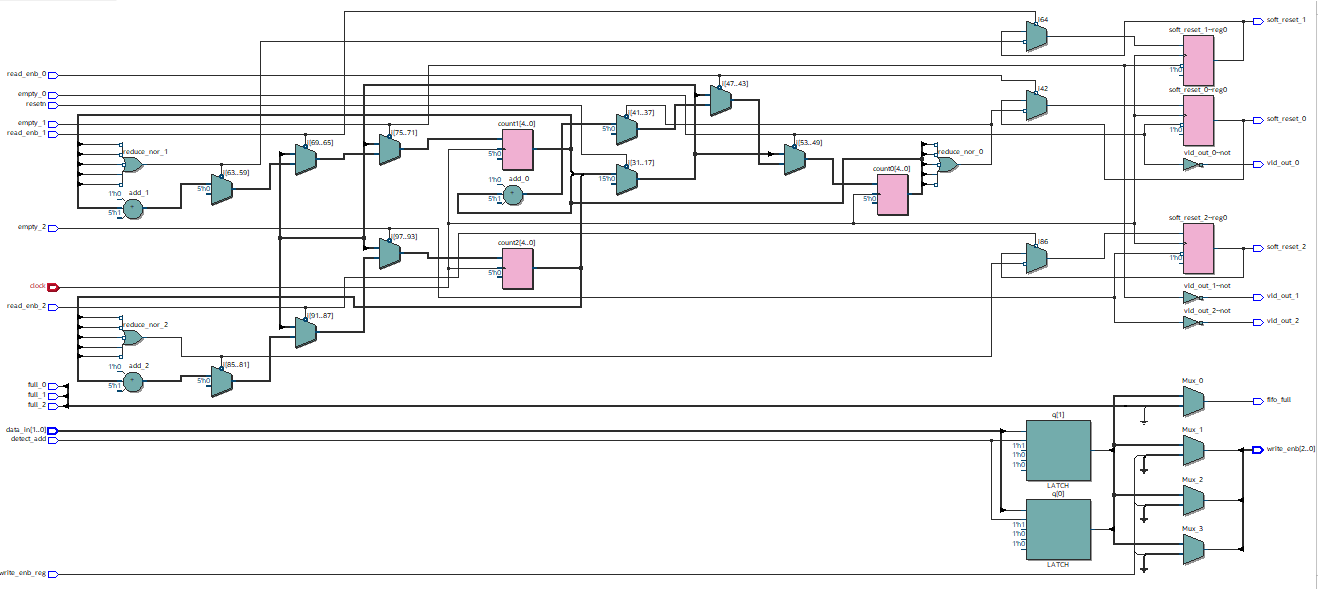
  end

endmodule

Waveform:



RTL Netlist:



**Register:**

RTL:

module router\_reg(input clock, resetn, pkt\_valid, fifo\_full, rst\_int\_reg, detect\_add,

                        ld\_state, laf\_state, full\_state, lfd\_state,

                  output reg parity\_done, low\_pkt\_valid, err,

                  input [7:0] data\_in, output reg [7:0] dout);

reg [7:0] a,c,d,e;

reg [8:0] b [31:0];

reg [4:0] count\_b,count\_a ;

always@(posedge clock) begin

    if(!resetn) begin

        dout <= 8'd0;

        parity\_done <= 1'b0;

        low\_pkt\_valid <= 1'b0;

        err <= 1'b0;

        count\_b <= 0;

        count\_a <= 0;

    end

    else begin

    parity\_done <= ((ld\_state & !fifo\_full & !pkt\_valid) | (laf\_state & low\_pkt\_valid & !parity\_done)) & !detect\_add;

    low\_pkt\_valid <= (ld\_state & !pkt\_valid);

        if(pkt\_valid) begin

            if(detect\_add) begin

                dout <= data\_in;

                c <= data\_in;

            end

            else if(ld\_state||lfd\_state) begin

                if(full\_state) begin

                    b[count\_b] <= data\_in;

                    count\_b <= count\_b + 1;

                end

                else if(!full\_state) begin

                    if(laf\_state) begin

                        dout <= b[count\_a];

                        count\_a <= count\_a + 1;

                    end

                    else begin

                        c <= c^data\_in;

                        dout <= data\_in;

                    end

                end

            end

        end

        else begin

            d <= data\_in;

            dout <= data\_in;

        end

    end

err <= (c==d) ? 0 : 1 ;

//dout <= resetn ? ( lfd\_state ? a : ( laf\_state ? b : (ld\_state ? (pkt\_valid e : dout) ) ) : 8'd0;

end

endmodule

TB :

module router\_reg\_tb();

reg clk, resetn, packet\_valid,fifo\_full, detect\_add, ld\_state, laf\_state, full\_state, lfd\_state, rst\_int\_reg;

reg [7:0] datain;

wire err, parity\_done, low\_packet\_valid;

wire [7:0]dout;

integer i;

router\_reg DUT( .clock(clk),

                .resetn(resetn),

                .pkt\_valid(packet\_valid),

                .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),

                .data\_in(datain),

                .err(err),

                .parity\_done(parity\_done),

                .low\_pkt\_valid(low\_packet\_valid),

                .dout(dout));

//clock generation

initial

    begin

    clk = 1;

    forever

    #5 clk=~clk;

    end

    task reset;

        begin

            resetn=1'b0;

            #10;

            resetn=1'b1;

        end

    endtask

    task packet1();

            reg [7:0]header, payload\_data, parity;

            reg [5:0]payloadlen;

            begin

                @(negedge clk);

                payloadlen=8;

                parity=0;

                detect\_add=1'b1;

                packet\_valid=1'b1;

                header={payloadlen,2'b10};

                datain=header;

                parity=parity^datain;

                @(negedge clk);

                detect\_add=1'b0;

                lfd\_state=1'b1;

                for(i=0;i<payloadlen;i=i+1)

                    begin

                    @(negedge clk);

                    lfd\_state=0;

                    ld\_state=1;

                    payload\_data={$random}%256;

                    datain=payload\_data;

                    parity=parity^datain;

                    end

                    @(negedge clk);

                    packet\_valid=0;

                    datain=parity;

                    @(negedge clk);

                    ld\_state=0;

                    end

endtask

task packet2();

            reg [7:0]header, payload\_data, parity;

            reg [5:0]payloadlen;

            begin

                @(negedge clk);

                payloadlen=8;

                parity=0;

                detect\_add=1'b1;

                packet\_valid=1'b1;

                header={payloadlen,2'b10};

                datain=header;

                parity=parity^datain;

                @(negedge clk);

                detect\_add=1'b0;

                lfd\_state=1'b1;

                for(i=0;i<payloadlen;i=i+1)

                    begin

                    @(negedge clk);

                    lfd\_state=0;

                    ld\_state=1;

                    payload\_data={$random}%256;

                    datain=payload\_data;

                    parity=parity^datain;

                    end

                    @(negedge clk);

                    packet\_valid=0;

                    full\_state = 1;

                    datain=!parity;

                    @(negedge clk);

                    ld\_state=0;

                    end

        endtask

initial

    begin

        reset;

        fifo\_full=1'b0;

        laf\_state=1'b0;

        full\_state=1'b0;

        #20;

        packet1();

        #105;

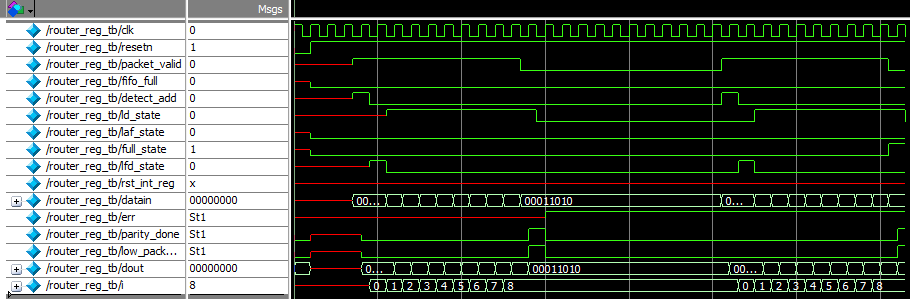
        packet2();

        $finish;

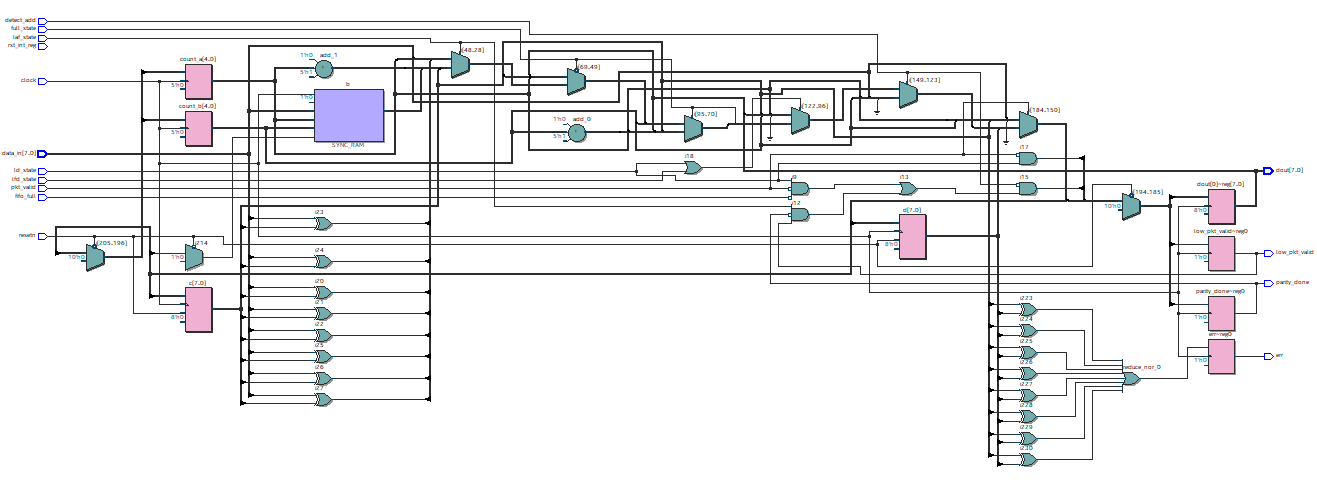
    end

endmodule

Waveform :



RTL Netlist:



**Router top module:**

RTL:

module router\_top(input clock, resetn, read\_enb\_0, read\_enb\_1, read\_enb\_2, pkt\_valid,

                  input [7:0] data\_in, output [7:0] data\_out\_0, data\_out\_1, data\_out\_2,

                  output valid\_out\_0, valid\_out\_1, valid\_out\_2, error, busy);

wire parity\_done,detect\_add;

wire ld\_state, laf\_state, full\_state, write\_enb\_reg, rst\_int\_reg, lfd\_state;

wire [2:0] fifo\_empty, full,w\_enb, write\_enb, soft\_reset\_temp;

wire fifo\_full,low\_pkt\_valid,soft\_reset\_0,soft\_reset\_1,soft\_reset\_2;

wire [7:0] dout;

wire read\_enb\_temp[2:0];

wire [7:0] data\_out\_temp[2:0];

assign soft\_reset\_temp[0] = soft\_reset\_0;

assign soft\_reset\_temp[1] = soft\_reset\_2;

assign soft\_reset\_temp[2] = soft\_reset\_1;

assign read\_enb\_temp[0] = read\_enb\_0;

assign read\_enb\_temp[1] = read\_enb\_1;

assign read\_enb\_temp[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];

router\_fsm dut1(.clock(clock),

                .resetn(resetn),

                .pkt\_valid(pkt\_valid),

                .parity\_done(parity\_done),

                .soft\_reset\_0(soft\_reset\_0),

                .soft\_reset\_1(soft\_reset\_1),

                .soft\_reset\_2(soft\_reset\_2),

                .fifo\_full(fifo\_full),

                .low\_pkt\_valid(low\_pkt\_valid),

                .fifo\_empty\_0(fifo\_empty[0]),

                .fifo\_empty\_1(fifo\_empty[1]),

                .fifo\_empty\_2(fifo\_empty[2]),

                .data\_in(data\_in[1:0]),

                .busy(busy),

                .detect\_add(detect\_add),

                .ld\_state(ld\_state),

                .laf\_state(laf\_state),

                .full\_state(full\_state),

                .write\_enb\_reg(write\_enb\_reg),

                .rst\_int\_reg(rst\_int\_reg),

                .lfd\_state(lfd\_state));

router\_reg dut2(.clock(clock),

                .resetn(resetn),

                .pkt\_valid(pkt\_valid),

                .fifo\_full(fifo\_full),

                .detect\_add(detect\_add),

                .ld\_state(ld\_state),

                .laf\_state(laf\_state),

                .full\_state(full\_state),

                .rst\_int\_reg(rst\_int\_reg),

                .lfd\_state(lfd\_state),

                .parity\_done(parity\_done),

                .low\_pkt\_valid(low\_pkt\_valid),

                .err(error),

                .data\_in(data\_in),

                .dout(dout));

router\_sync dut3(.clock(clock),

                .resetn(resetn),

                .detect\_add(detect\_add),

                .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(fifo\_empty[0]),

                .empty\_1(fifo\_empty[1]),

                .empty\_2(fifo\_empty[2]),

                .full\_0(full[0]),

                .full\_1(full[1]),

                .full\_2(full[2]),

                .data\_in(data\_in[1:0]),

                .vld\_out\_0(valid\_out\_0),

                .vld\_out\_1(valid\_out\_1),

                .vld\_out\_2(valid\_out\_2),

                .soft\_reset\_0(soft\_reset\_0),

                .soft\_reset\_1(soft\_reset\_1),

                .soft\_reset\_2(soft\_reset\_2),

                .fifo\_full(fifo\_full),

                .write\_enb(w\_enb));

genvar x;

generate for (x= 0 ; x<3 ; x = x+1)

begin:fifo

    router\_fifo f(.clock(clock),

                  .resetn(resetn),

                  .soft\_reset(soft\_reset\_temp[x]),

                  .lfd\_state(lfd\_state),

                  .write\_enb(w\_enb[x]),

                  .data\_in(dout),

                  .read\_enb(read\_enb\_temp[x]),

                  .full(full[x]),

                  .empty(fifo\_empty[x]),

                  .data\_out(data\_out\_temp[x]));

end

endgenerate

endmodule

Test bench:

module router\_top\_tb();

reg clock, resetn, read\_enb\_0, read\_enb\_1, read\_enb\_2, pkt\_valid;

reg [7:0] data\_in;

wire [7:0] data\_out\_0, data\_out\_1, data\_out\_2;

wire valid\_out\_0, valid\_out\_1, valid\_out\_2, error, busy;

integer i;

router\_top dut(clock, resetn, read\_enb\_0, read\_enb\_1, read\_enb\_2, pkt\_valid,data\_in,

               data\_out\_0, data\_out\_1, data\_out\_2, valid\_out\_0, valid\_out\_1, valid\_out\_2, error, busy);

initial begin

    clock = 1;

    forever #5 clock = !clock;

end

task reset();

begin

    @(negedge clock);

    resetn =1'b0;

    #10;

    resetn = 1'b1;

end

endtask

task pkt16();

reg [7:0] parity;

reg [5:0] payload\_len;

begin

    wait(!busy);

    begin

        @(negedge clock);

        payload\_len = 6'b001110;

        pkt\_valid = 1'b1;

        data\_in = {payload\_len,2'b10};

        parity = data\_in;

    end

    for(i=0 ; i < payload\_len; i=i+1 )

    begin

        wait(!busy)

        @(negedge clock)

        data\_in = i\*2;

        parity = parity^data\_in;

    end

    wait(!busy);

    @(negedge clock)

    pkt\_valid = 1'b0;

    data\_in = parity;

    repeat(30)

    @(negedge clock);

    data\_in = 8'bx;

    read\_enb\_2 = 1'b1;

end

endtask

initial begin

    reset;

    pkt16;

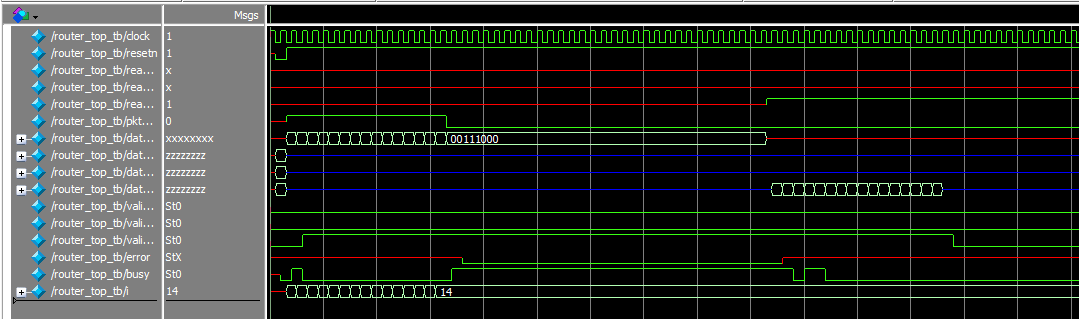
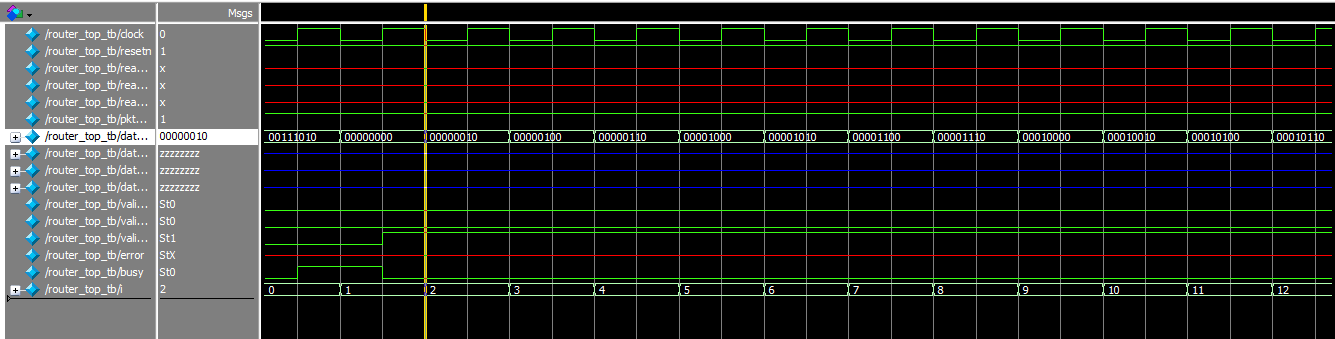
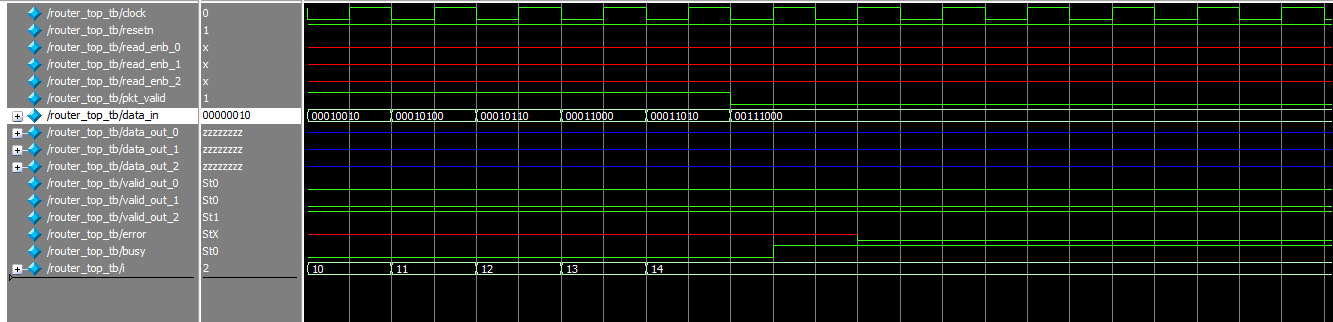
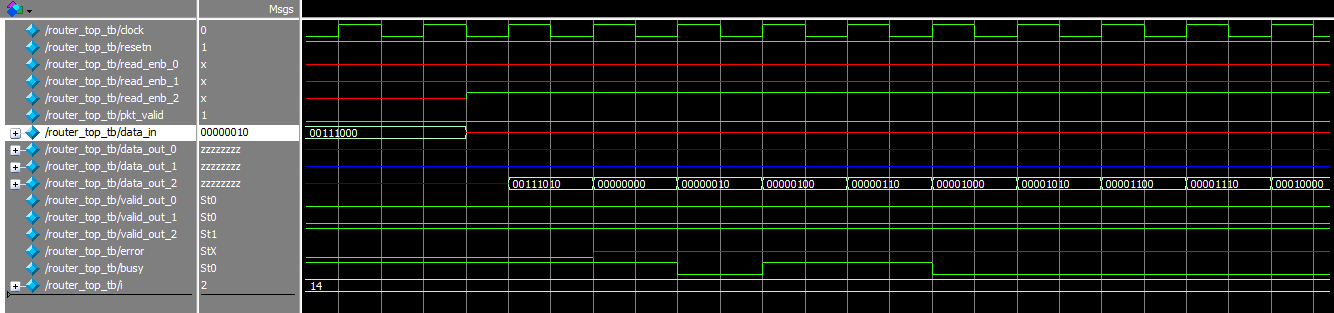
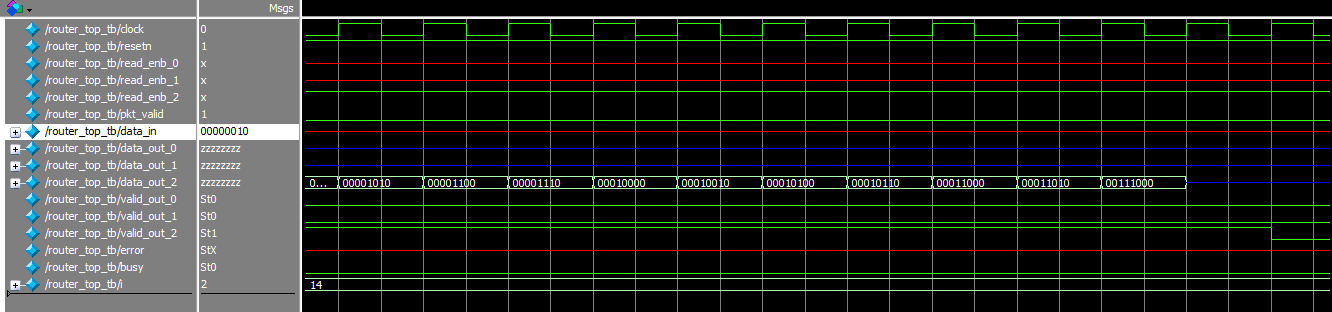
    #1000;

    $finish;

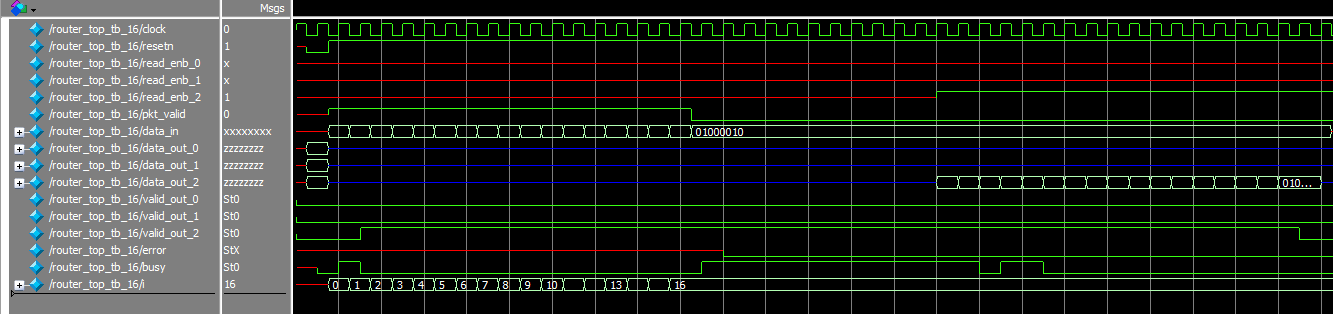
end

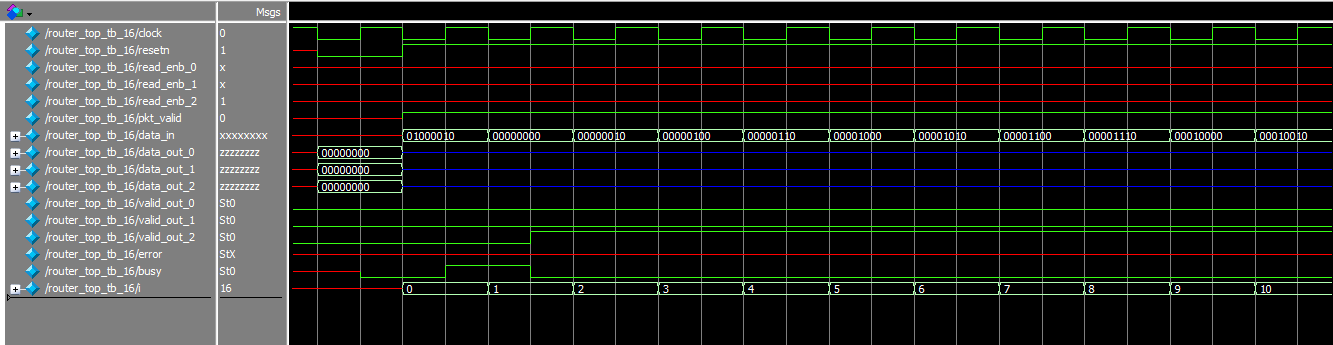
endmodule

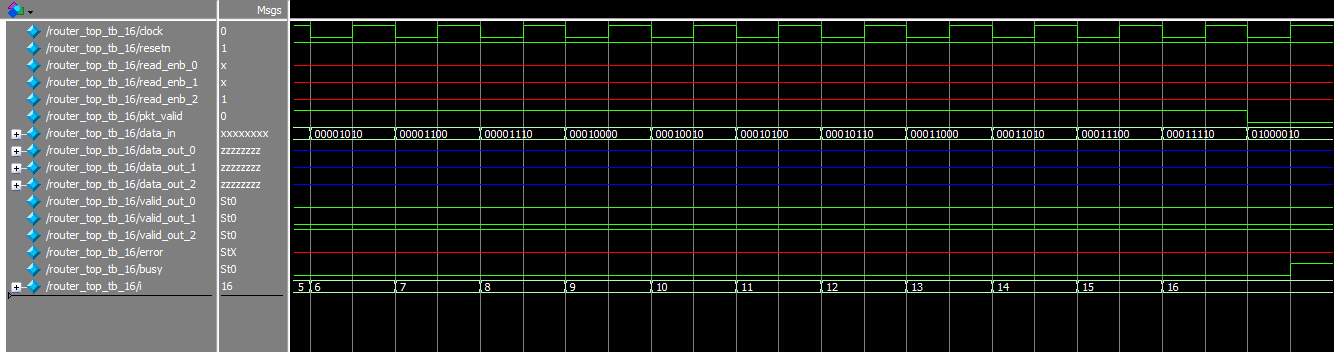
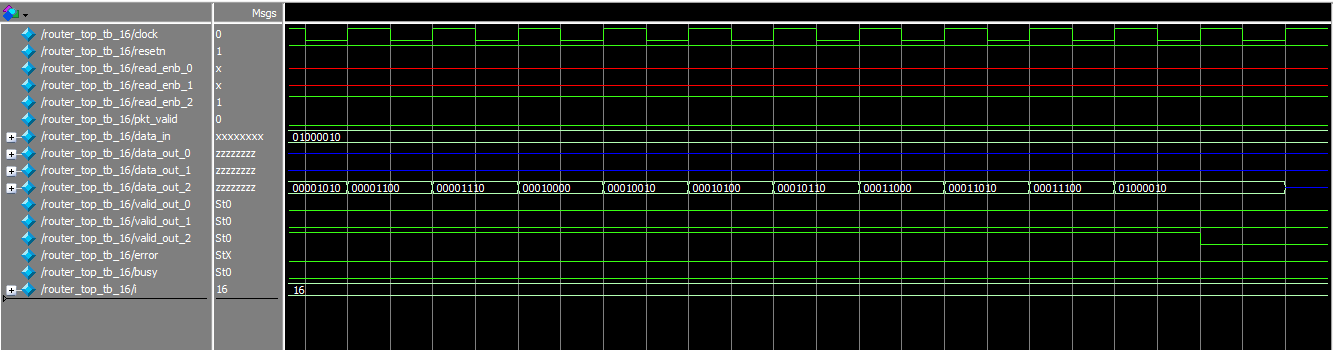
WAVE (packet 14):

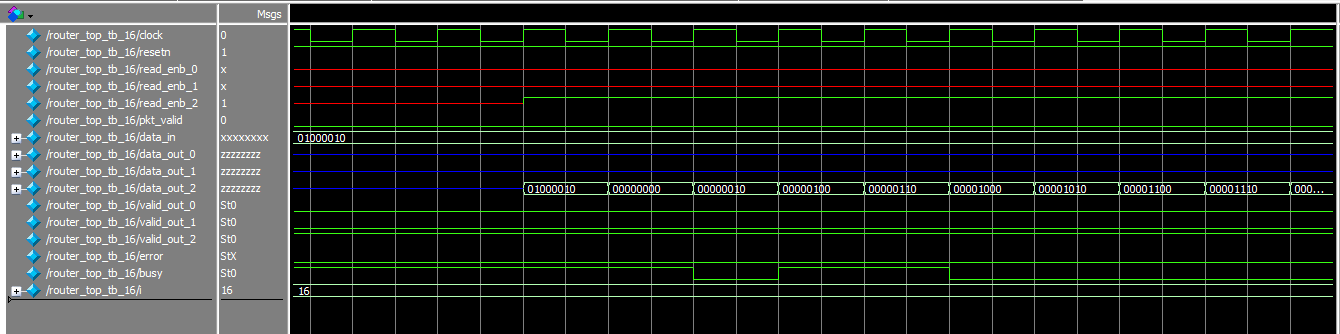
    

WAVE (packet 16):









RTL Netlist:

