Router 1x3 Project Report

By,

T Raja Aadhithan

BRN10

INDEX

FIFO :

RTL:

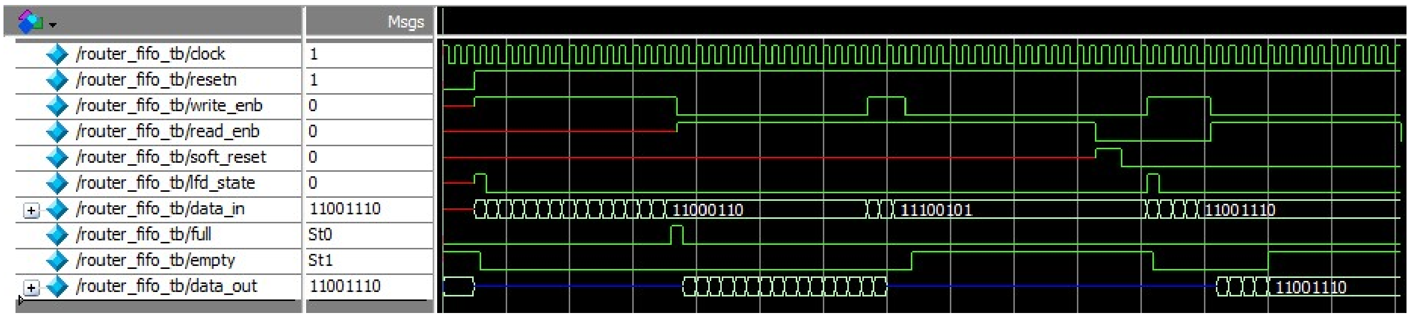
```
module router_fifo(input clock, resetn, write_enb, read_enb, soft_reset, lf
d_state,
                   input [7:0] data_in, output full, empty, output [7:0] da
ta_out);

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 :

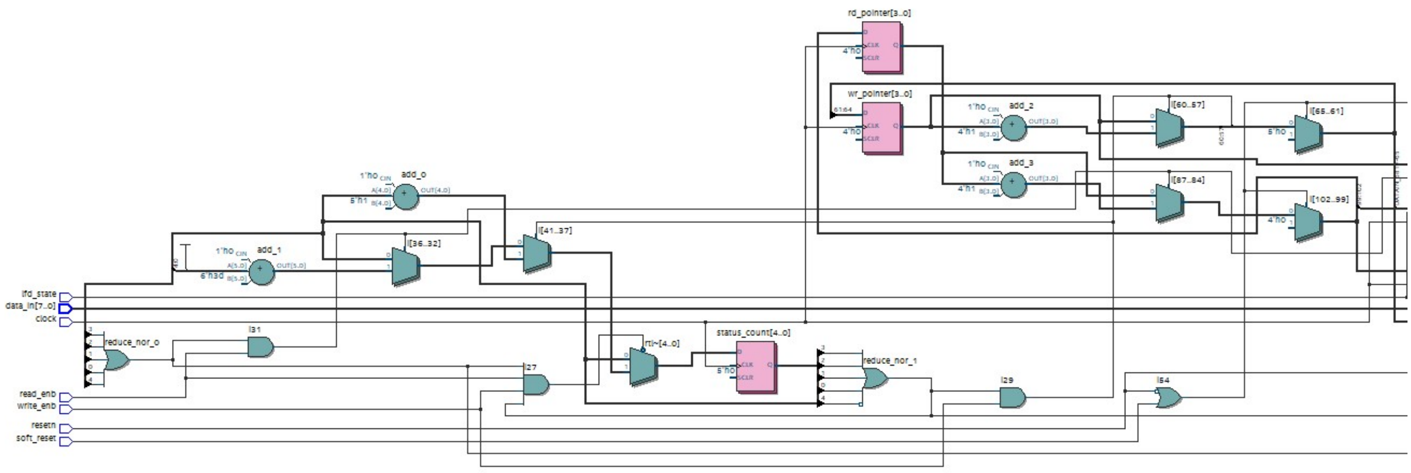
```
module router_fifo_tb();

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
    $monitor("@time: %t, data_in: %h, data_out: %h, lfd: %b, full: %b, empt
y: %b",$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
```

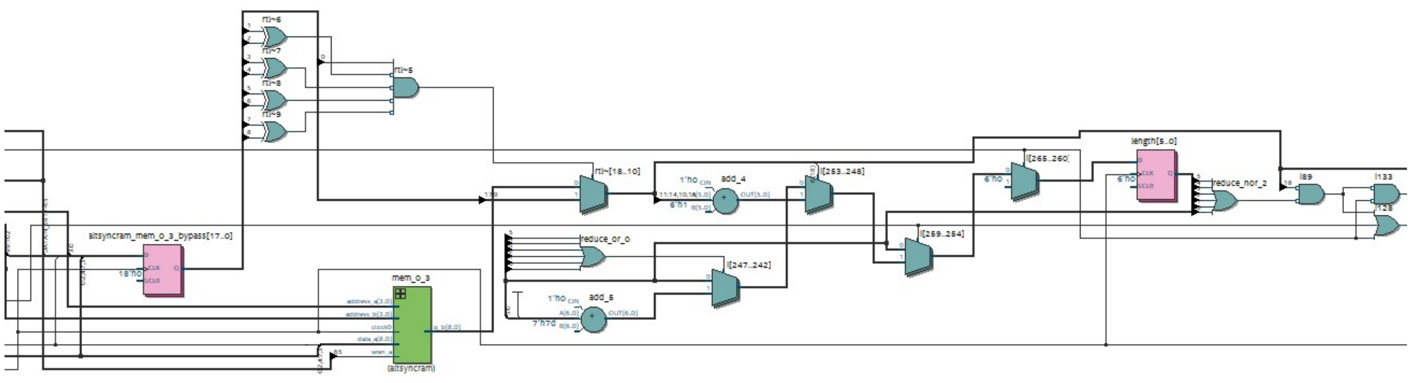
Waveform :

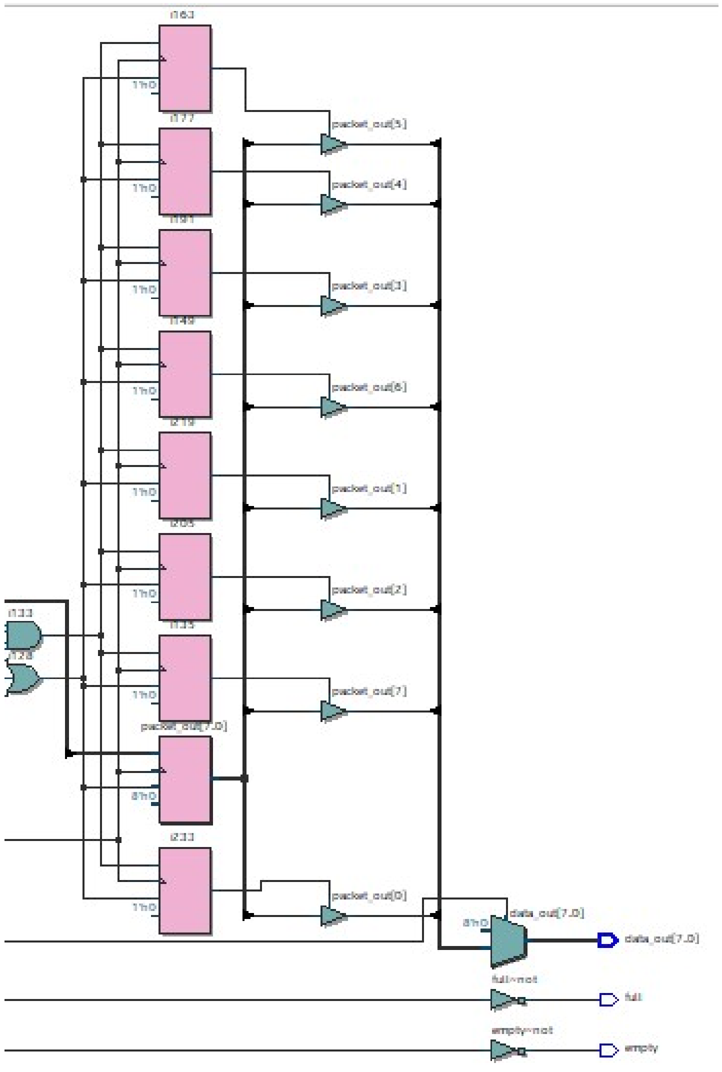


```
                            Msgs
/router_fifo_tb/clock       1
/router_fifo_tb/resetn      1
/router_fifo_tb/write_enb   0
/router_fifo_tb/read_enb    0
/router_fifo_tb/soft_reset  0
/router_fifo_tb/lfd_state   0
/router_fifo_tb/data_in     11001110
/router_fifo_tb/full        St0
/router_fifo_tb/empty       St1
/router_fifo_tb/data_out    11001110
```

RTL Netlist:




rti~6
rti~7
rti~8
rti~9
rti~5
altsyncram_mem_o_3_bypass[17..0]
18'h0
mem_o_3
(altsyncram)
rti~[18..10]
1'h0
add_4
6'h1
reduce_or_o
1'h0
add_5
7'h7d
l[253..248]
l[247..242]
l[259..254]
l[265..260]
6'h0
length[5..0]
6'h0
reduce_nor_2
i89
i133
i128


i163
1'h0
i177
1'h0
i191
1'h0
i149
1'h0
i219
1'h0
i205
1'h0
i135
1'h0
packet_out[7..0]
8'h0
i233
1'h0
i133
i128
packet_out[5]
packet_out[4]
packet_out[3]
packet_out[6]
packet_out[1]
packet_out[2]
packet_out[7]
packet_out[0]
8'h0
data_out[7..0]
data_out[7..0]
full~not
full
empty~not
empty

FSM :

RTL:

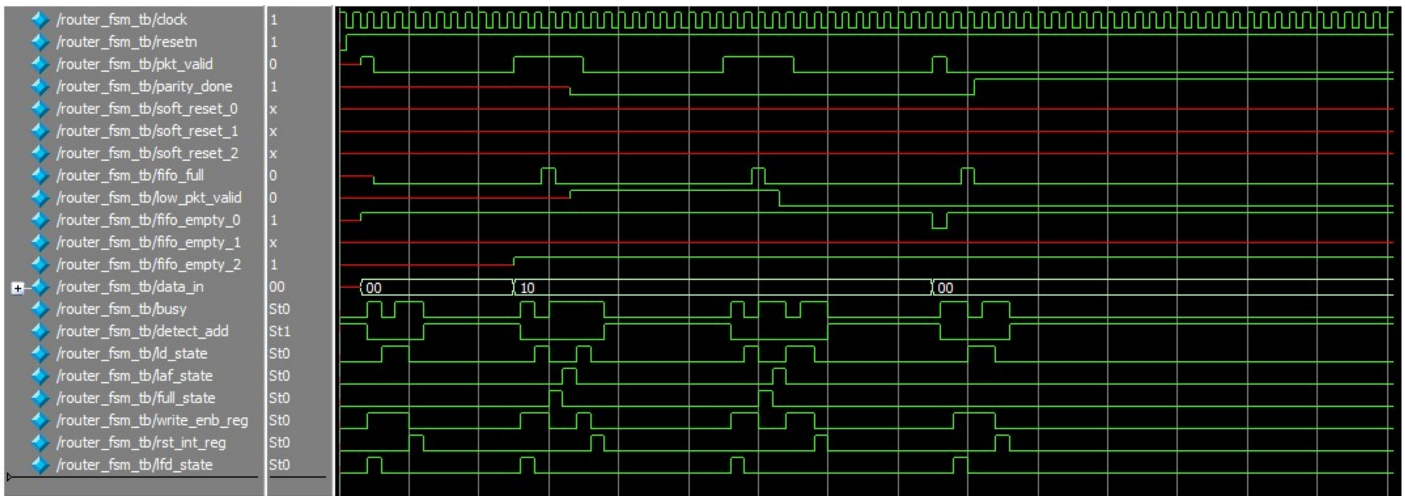
```
module router_fsm(input clock, resetn, pkt_valid, parity_done,
                        soft_reset_0, soft_reset_1, soft_reset_2,
                        fifo_full, low_pkt_valid,
                        fifo_empty_0, fifo_empty_1, fifo_empty_2,
                        input [1:0] data_in,
                        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_Add
ress;
    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
```

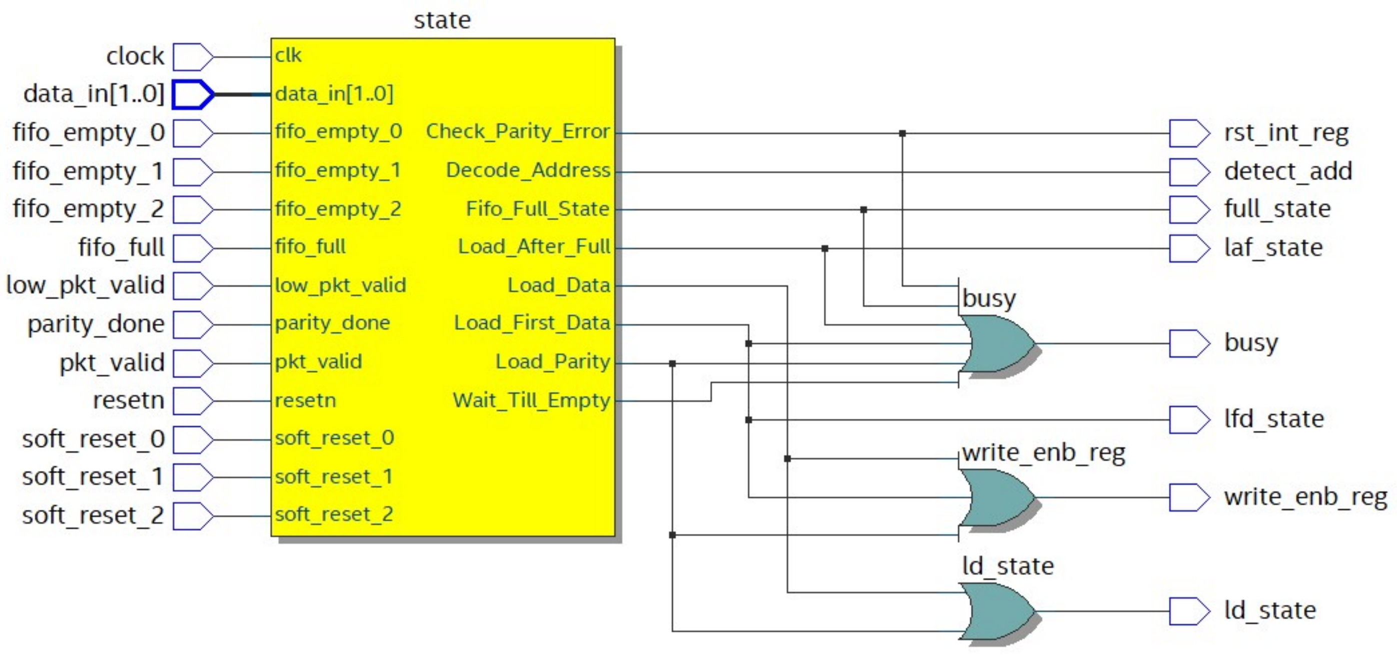
```
task ts1();
    begin
        resetn = 0;
        @(negedge clock);
        resetn = 1;
        @(negedge clock);
        pkt_valid = 1'b1;
        data_in = 2'b0;
        fifo_empty_0 = 1'b1;
        @(negedge clock);
        fifo_full = 1'b0;
        pkt_valid = 1'b0;
    end
endtask

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 :



/router_fsm_tb/clock 1
/router_fsm_tb/resetn 1
/router_fsm_tb/pkt_valid 0
/router_fsm_tb/parity_done 1
/router_fsm_tb/soft_reset_0 x
/router_fsm_tb/soft_reset_1 x
/router_fsm_tb/soft_reset_2 x
/router_fsm_tb/fifo_full 0
/router_fsm_tb/low_pkt_valid 0
/router_fsm_tb/fifo_empty_0 1
/router_fsm_tb/fifo_empty_1 x
/router_fsm_tb/fifo_empty_2 1
/router_fsm_tb/data_in 00
/router_fsm_tb/busy St0
/router_fsm_tb/detect_add St1
/router_fsm_tb/ld_state St0
/router_fsm_tb/laf_state St0
/router_fsm_tb/full_state St0
/router_fsm_tb/write_enb_reg St0
/router_fsm_tb/rst_int_reg St0
/router_fsm_tb/lfd_state St0
00
10
00

RTL Netlist :



Synchronizer:

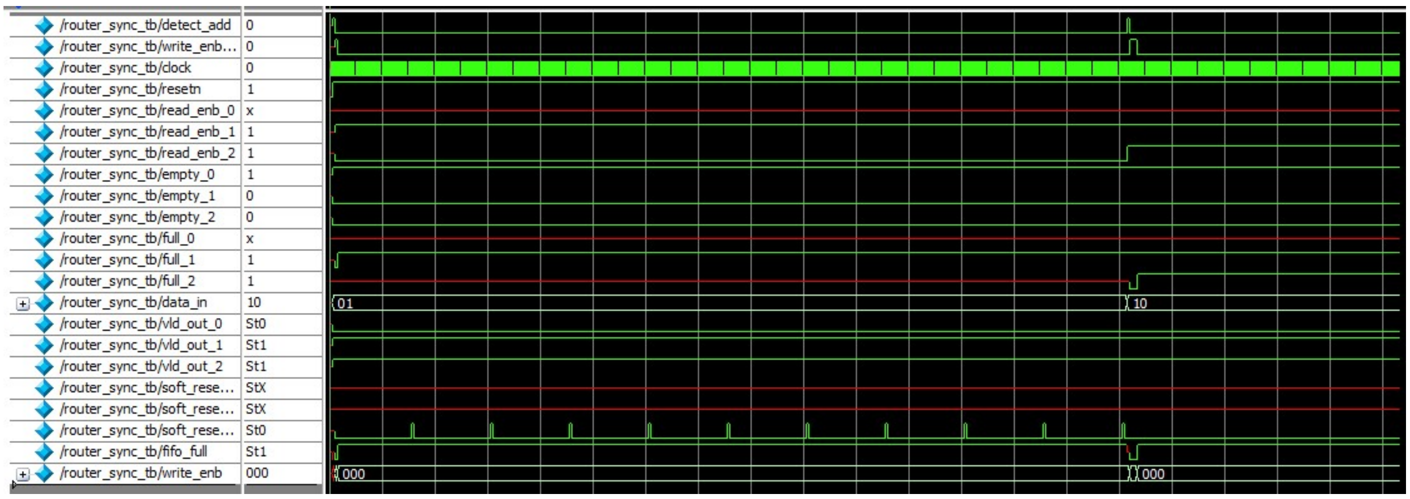
RTL:

```
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:

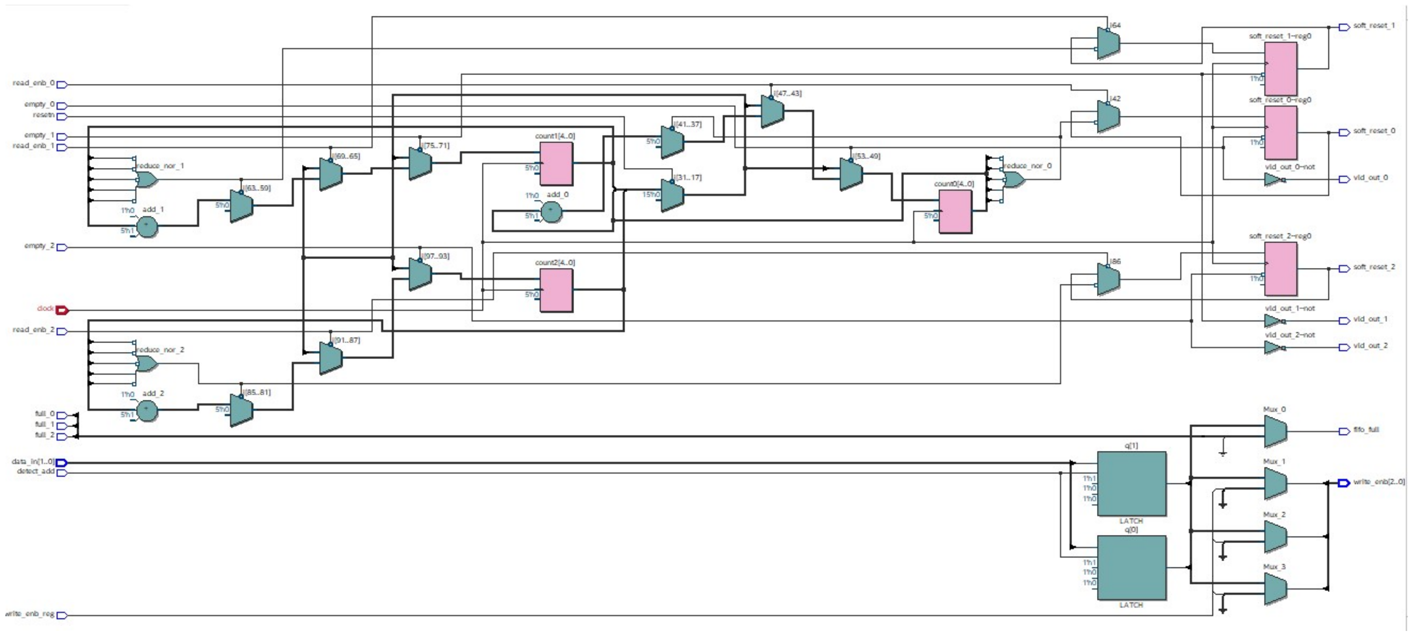
```
module router_sync_tb();
  reg detect_add, write_enb_reg, clock, resetn,
      read_enb_0, read_enb_1, read_enb_2,
      empty_0, empty_1, empty_2,
      full_0, full_1, full_2;
  reg [1:0] data_in;
  wire vld_out_0, vld_out_1, vld_out_2,
       soft_reset_0, soft_reset_1, soft_reset_2,
       fifo_full;
  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,               v
ld_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_ou
t_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:



```
/router_sync_tb/detect_add      0
/router_sync_tb/write_enb...    0
/router_sync_tb/clock           0
/router_sync_tb/resetn          1
/router_sync_tb/read_enb_0      x
/router_sync_tb/read_enb_1      1
/router_sync_tb/read_enb_2      1
/router_sync_tb/empty_0         1
/router_sync_tb/empty_1         0
/router_sync_tb/empty_2         0
/router_sync_tb/full_0          x
/router_sync_tb/full_1          1
/router_sync_tb/full_2          1
/router_sync_tb/data_in         10
/router_sync_tb/vld_out_0       St0
/router_sync_tb/vld_out_1       St1
/router_sync_tb/vld_out_2       St1
/router_sync_tb/soft_rese...    StX
/router_sync_tb/soft_rese...    StX
/router_sync_tb/soft_rese...    St0
/router_sync_tb/fifo_full       St1
/router_sync_tb/write_enb       000
```

RTL Netlist:



Register:

RTL:

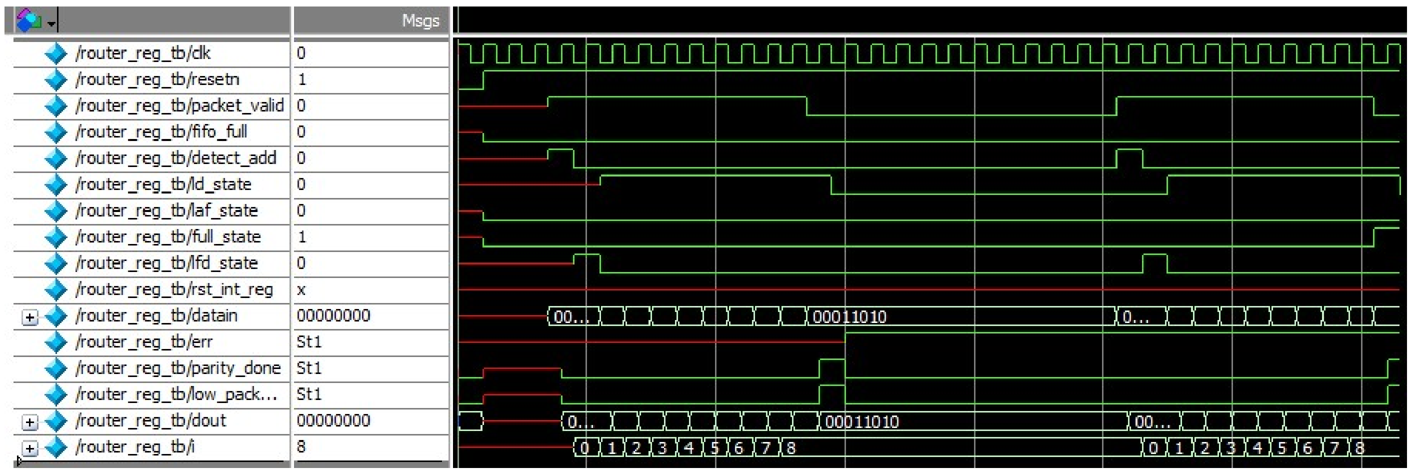
```
module router_reg(input clock, resetn, pkt_valid, fifo_full, rst_int_reg, d
etect_add,
                        ld_state, laf_state, full_state, lfd_state,
                  output reg parity_done, low_pkt_valid, err,
                  input [7:0] data_in, output reg [7:0] dout);
reg [7:0] a,c,d,e;
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_val
id e : dout) ) ) : 8'd0;
end
endmodule
```

TB :

```

module router_reg_tb();

reg clk, resetn, packet_valid,fifo_full, detect_add, ld_state, laf_state, f
ull_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;
```

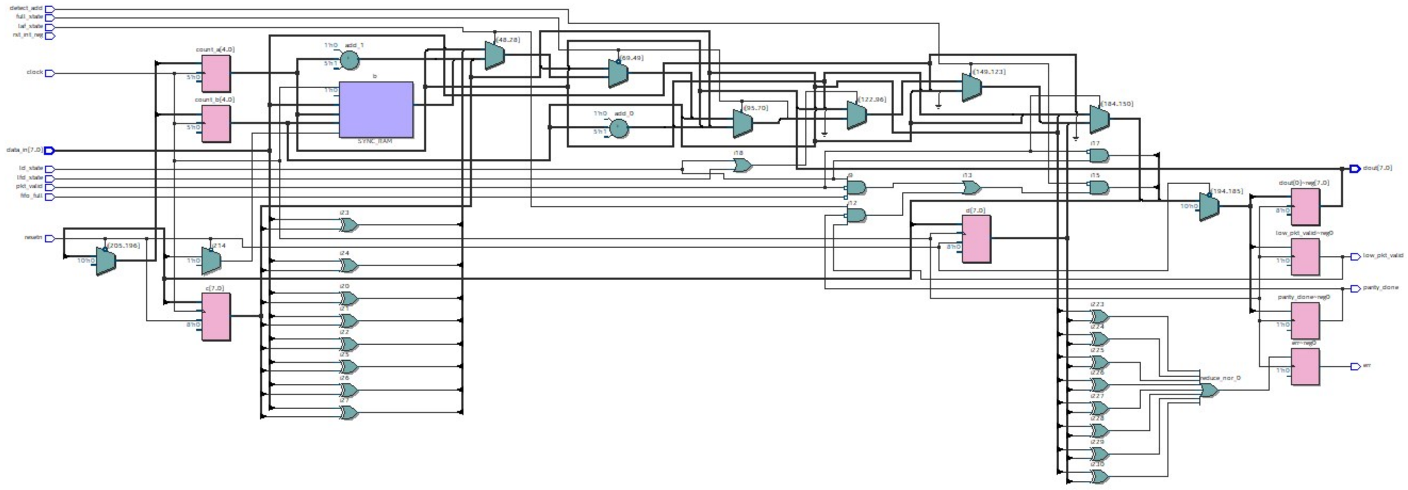
```verilog
                @(negedge clk);
                detect_add=1'b0;
                lfd_state=1'b1;

                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 :



```
                              Msgs
/router_reg_tb/clk            0
/router_reg_tb/resetn         1
/router_reg_tb/packet_valid   0
/router_reg_tb/fifo_full      0
/router_reg_tb/detect_add     0
/router_reg_tb/ld_state       0
/router_reg_tb/laf_state      0
/router_reg_tb/full_state     1
/router_reg_tb/lfd_state      0
/router_reg_tb/rst_int_reg    x
/router_reg_tb/datain         00000000
/router_reg_tb/err            St1
/router_reg_tb/parity_done    St1
/router_reg_tb/low_pack...    St1
/router_reg_tb/dout           00000000
/router_reg_tb/i              8
```

RTL Netlist:



Router top module:

RTL:

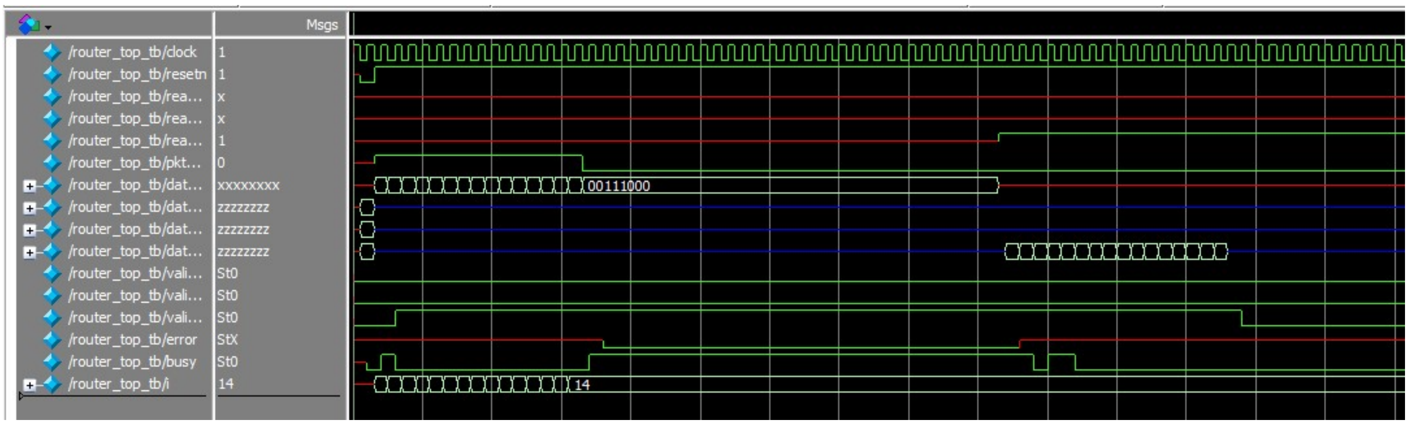
```
module router_top(input clock, resetn, read_enb_0, read_enb_1, read_enb_2, pkt_valid,
                  input [7:0] data_in, output [7:0] data_out_0, data_out_1, data_out_2,
                  output valid_out_0, valid_out_1, valid_out_2, error, busy);

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:

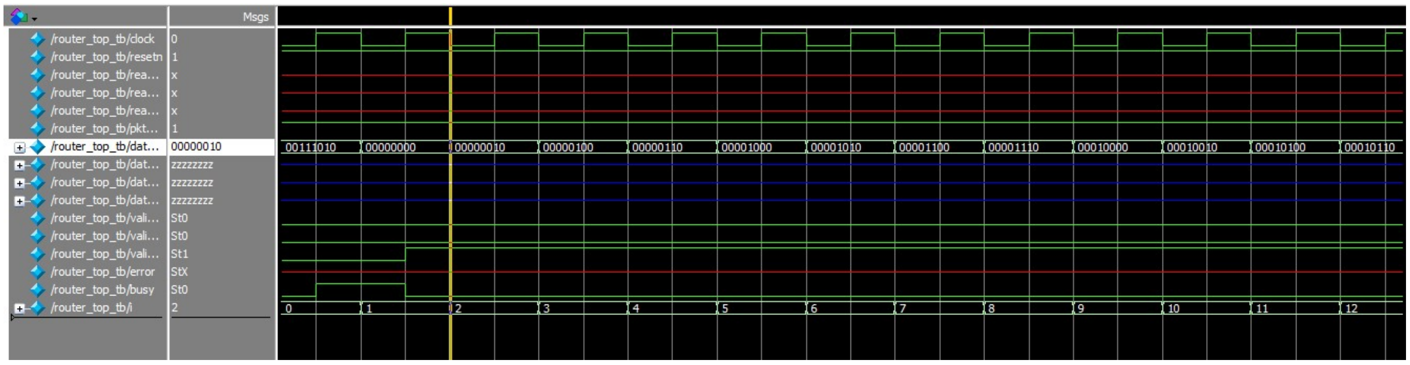
```
module router_top_tb();
reg clock, resetn, read_enb_0, read_enb_1, read_enb_2, pkt_valid;
reg [7:0] data_in;
wire [7:0] data_out_0, data_out_1, data_out_2;
wire valid_out_0, valid_out_1, valid_out_2, error, busy;
integer i;

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, err
or, 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
```

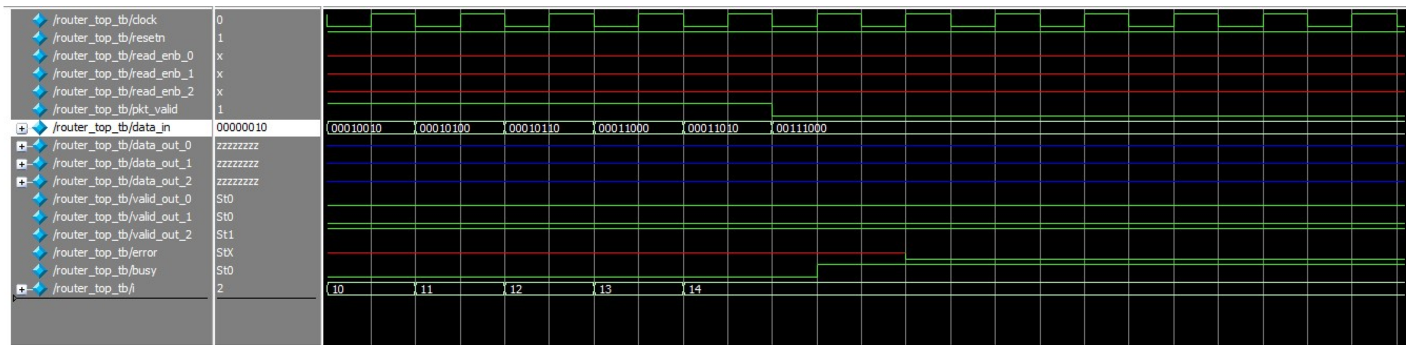
```
        wait(!busy)
        @(negedge clock)
        data_in = i*2;
        parity = parity^data_in;
    end
    wait(!busy);
    @(negedge clock)
    pkt_valid = 1'b0;
    data_in = parity;
    repeat(30)
    @(negedge clock);
    data_in = 8'bx;
    read_enb_2 = 1'b1;

end
endtask

initial begin
    reset;
    pkt16;
    #1000;
    $finish;
end
endmodule
```

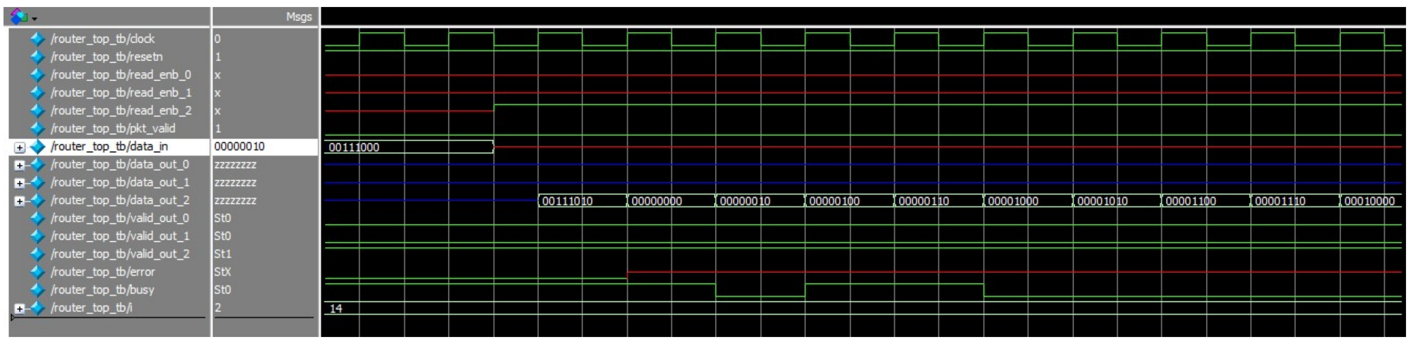
WAVE (packet 14):

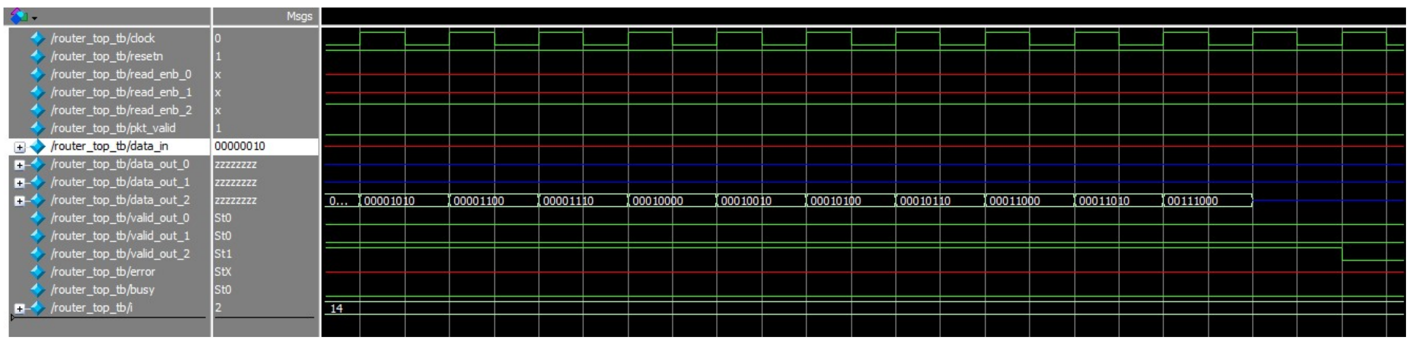


```
                         Msgs
/router_top_tb/clock     1
/router_top_tb/resetn    1
/router_top_tb/rea...    x
/router_top_tb/rea...    x
/router_top_tb/rea...    1
/router_top_tb/pkt...    0
/router_top_tb/dat...    xxxxxxxx
/router_top_tb/dat...    zzzzzzzz
/router_top_tb/dat...    zzzzzzzz
/router_top_tb/dat...    zzzzzzzz
/router_top_tb/vali...   St0
/router_top_tb/vali...   St0
/router_top_tb/vali...   St0
/router_top_tb/error     StX
/router_top_tb/busy      St0
/router_top_tb/i         14
```

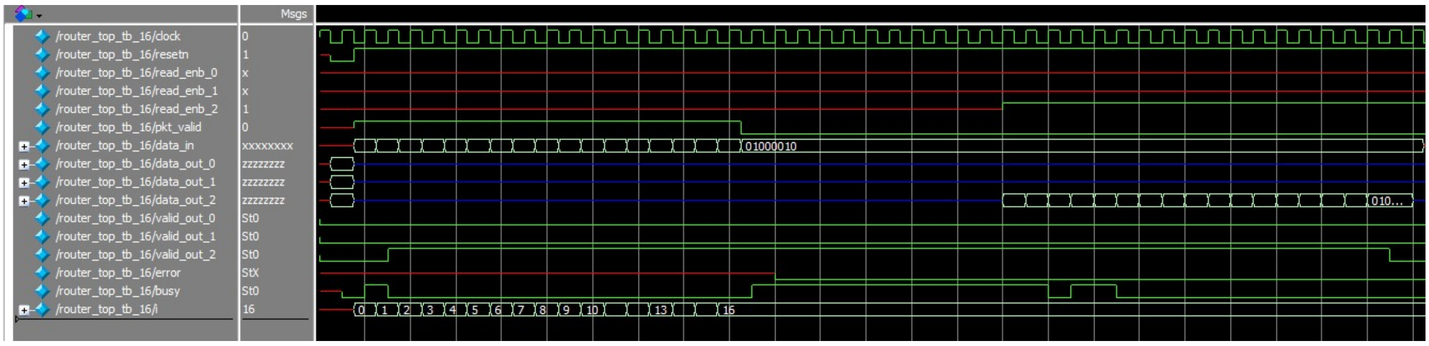

Msgs
/router_top_tb/clock 0
/router_top_tb/resetn 1
/router_top_tb/rea... x
/router_top_tb/rea... x
/router_top_tb/rea... x
/router_top_tb/pkt... 1
/router_top_tb/dat... 00000010
/router_top_tb/dat... zzzzzzzz
/router_top_tb/dat... zzzzzzzz
/router_top_tb/dat... zzzzzzzz
/router_top_tb/vali... St0
/router_top_tb/vali... St0
/router_top_tb/vali... St1
/router_top_tb/error StX
/router_top_tb/busy St0
/router_top_tb/i 2
00111010 00000000 00000010 00000100 00000110 00001000 00001010 00001100 00001110 00010000 00010010 00010100 00010110
0 1 2 3 4 5 6 7 8 9 10 11 12

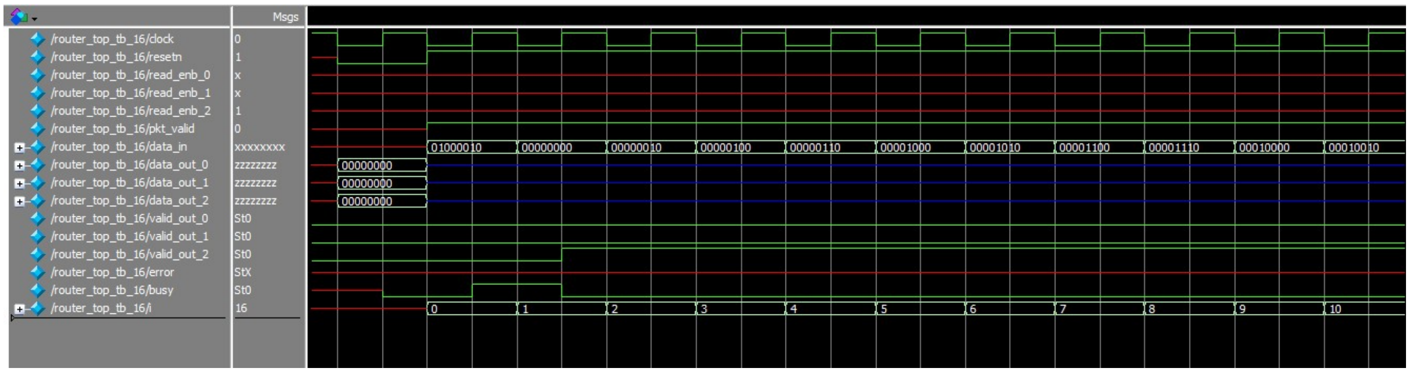

/router_top_tb/clock 0
/router_top_tb/resetn 1
/router_top_tb/read_enb_0 x
/router_top_tb/read_enb_1 x
/router_top_tb/read_enb_2 x
/router_top_tb/pkt_valid 1
/router_top_tb/data_in 00000010
/router_top_tb/data_out_0 zzzzzzzz
/router_top_tb/data_out_1 zzzzzzzz
/router_top_tb/data_out_2 zzzzzzzz
/router_top_tb/valid_out_0 St0
/router_top_tb/valid_out_1 St0
/router_top_tb/valid_out_2 St1
/router_top_tb/error StX
/router_top_tb/busy St0
/router_top_tb/i 2
00010010 00010100 00010110 00011000 00011010 00111000
10 11 12 13 14

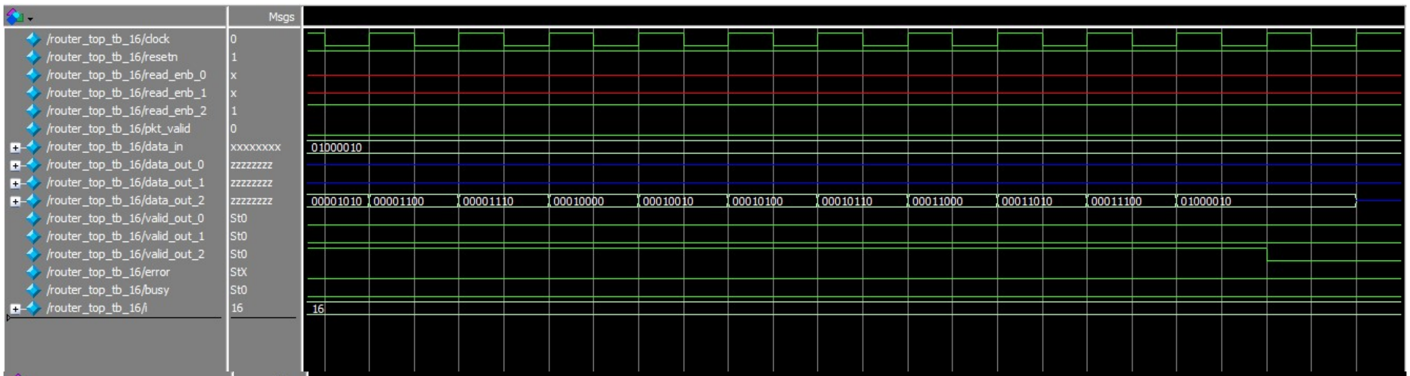

Msgs
/router_top_tb/clock 0
/router_top_tb/resetn 1
/router_top_tb/read_enb_0 x
/router_top_tb/read_enb_1 x
/router_top_tb/read_enb_2 x
/router_top_tb/pkt_valid 1
/router_top_tb/data_in 00000010
/router_top_tb/data_out_0 zzzzzzzz
/router_top_tb/data_out_1 zzzzzzzz
/router_top_tb/data_out_2 zzzzzzzz
/router_top_tb/valid_out_0 St0
/router_top_tb/valid_out_1 St0
/router_top_tb/valid_out_2 St1
/router_top_tb/error StX
/router_top_tb/busy St0
/router_top_tb/i 2
00111000
00111010 00000000 00000010 00000100 00000110 00001000 00001010 00001100 00001110 00010000
14

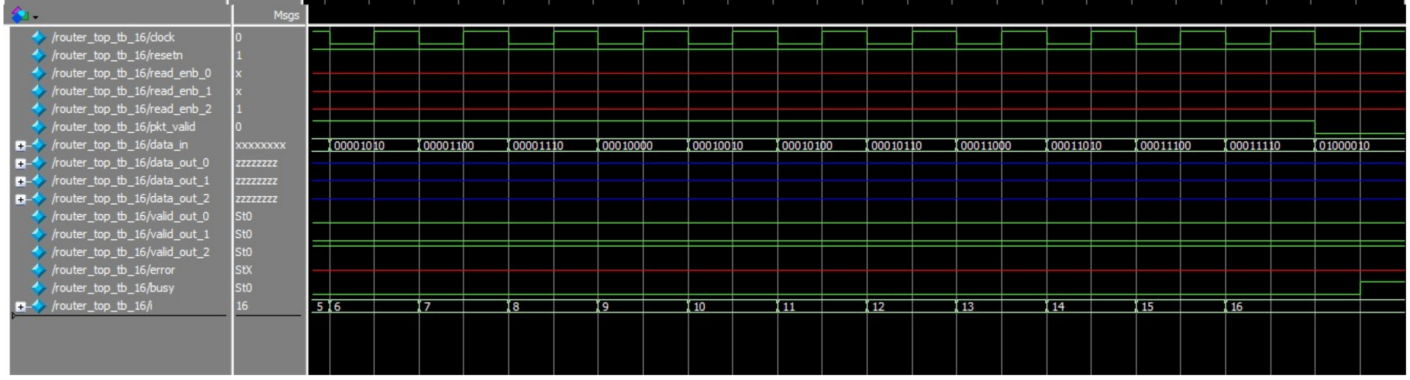

Msgs
/router_top_tb/clock 0
/router_top_tb/resetn 1
/router_top_tb/read_enb_0 x
/router_top_tb/read_enb_1 x
/router_top_tb/read_enb_2 x
/router_top_tb/pkt_valid 1
/router_top_tb/data_in 00000010
/router_top_tb/data_out_0 zzzzzzzz
/router_top_tb/data_out_1 zzzzzzzz
/router_top_tb/data_out_2 zzzzzzzz
/router_top_tb/valid_out_0 St0
/router_top_tb/valid_out_1 St0
/router_top_tb/valid_out_2 St1
/router_top_tb/error StX
/router_top_tb/busy St0
/router_top_tb/i 2
0... 00001010 00001100 00001110 00010000 00010010 00010100 00010110 00011000 00011010 00111000
14

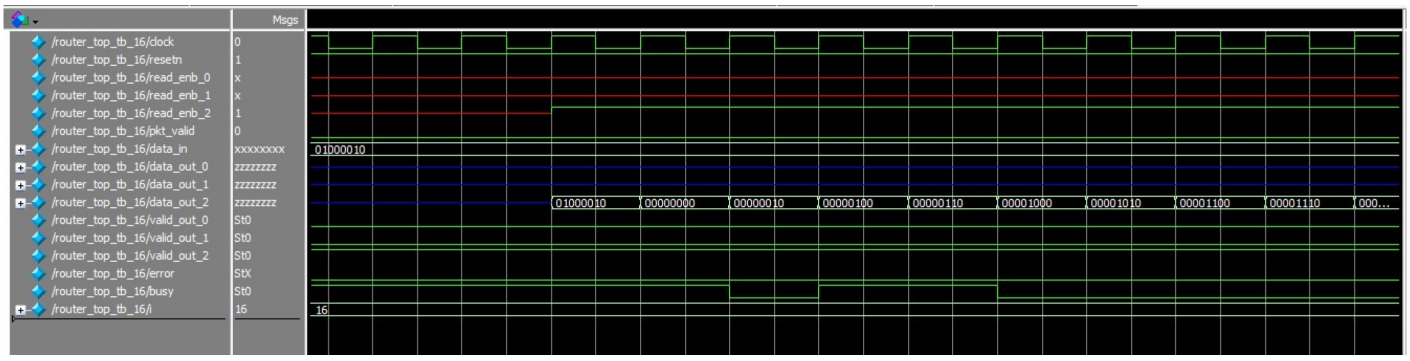
WAVE (packet 16):


Msgs
/router_top_tb_16/clock 0
/router_top_tb_16/resetn 1
/router_top_tb_16/read_enb_0 x
/router_top_tb_16/read_enb_1 x
/router_top_tb_16/read_enb_2 1
/router_top_tb_16/pkt_valid 0
/router_top_tb_16/data_in xxxxxxxx
/router_top_tb_16/data_out_0 zzzzzzzz
/router_top_tb_16/data_out_1 zzzzzzzz
/router_top_tb_16/data_out_2 zzzzzzzz
/router_top_tb_16/valid_out_0 St0
/router_top_tb_16/valid_out_1 St0
/router_top_tb_16/valid_out_2 St0
/router_top_tb_16/error StX
/router_top_tb_16/busy St0
/router_top_tb_16/i 16
01000010
010...
0 1 2 3 4 5 6 7 8 9 10 13 16


Msgs
/router_top_tb_16/clock 0
/router_top_tb_16/resetn 1
/router_top_tb_16/read_enb_0 x
/router_top_tb_16/read_enb_1 x
/router_top_tb_16/read_enb_2 1
/router_top_tb_16/pkt_valid 0
/router_top_tb_16/data_in xxxxxxxx
/router_top_tb_16/data_out_0 zzzzzzzz
/router_top_tb_16/data_out_1 zzzzzzzz
/router_top_tb_16/data_out_2 zzzzzzzz
/router_top_tb_16/valid_out_0 St0
/router_top_tb_16/valid_out_1 St0
/router_top_tb_16/valid_out_2 St0
/router_top_tb_16/error StX
/router_top_tb_16/busy St0
/router_top_tb_16/i 16
01000010 00000000 00000010 00000100 00000110 00001000 00001010 00001100 00001110 00010000 00010010
00000000
00000000
00000000
0 1 2 3 4 5 6 7 8 9 10

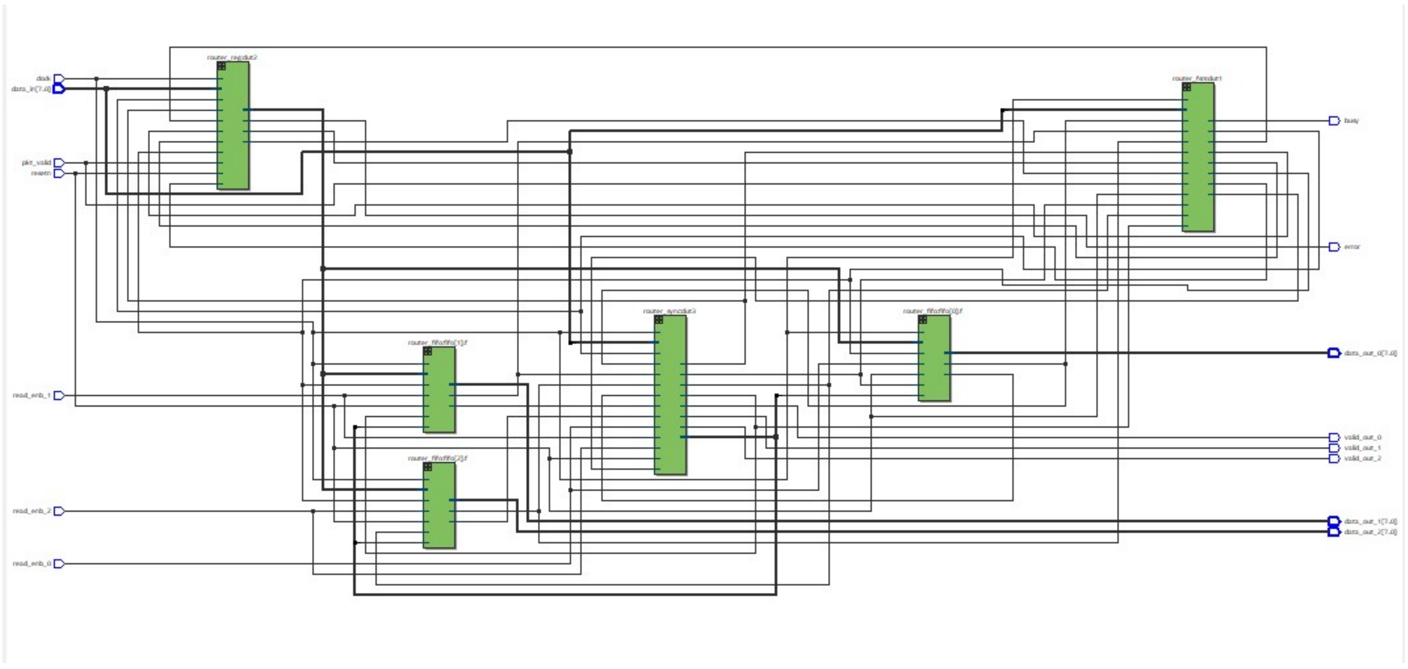

Msgs
/router_top_tb_16/clock 0
/router_top_tb_16/resetn 1
/router_top_tb_16/read_enb_0 x
/router_top_tb_16/read_enb_1 x
/router_top_tb_16/read_enb_2 1
/router_top_tb_16/pkt_valid 0
/router_top_tb_16/data_in xxxxxxxx
/router_top_tb_16/data_out_0 zzzzzzzz
/router_top_tb_16/data_out_1 zzzzzzzz
/router_top_tb_16/data_out_2 zzzzzzzz
/router_top_tb_16/valid_out_0 St0
/router_top_tb_16/valid_out_1 St0
/router_top_tb_16/valid_out_2 St0
/router_top_tb_16/error StX
/router_top_tb_16/busy St0
/router_top_tb_16/i 16
01000010
00001010 00001100 00001110 00010000 00010010 00010100 00010110 00011000 00011010 00011100 01000010
16


Msgs
/router_top_tb_16/clock 0
/router_top_tb_16/resetn 1
/router_top_tb_16/read_enb_0 x
/router_top_tb_16/read_enb_1 x
/router_top_tb_16/read_enb_2 1
/router_top_tb_16/pkt_valid 0
/router_top_tb_16/data_in xxxxxxxx
/router_top_tb_16/data_out_0 zzzzzzzz
/router_top_tb_16/data_out_1 zzzzzzzz
/router_top_tb_16/data_out_2 zzzzzzzz
/router_top_tb_16/valid_out_0 St0
/router_top_tb_16/valid_out_1 St0
/router_top_tb_16/valid_out_2 St0
/router_top_tb_16/error StX
/router_top_tb_16/busy St0
/router_top_tb_16/i 16
00001010 00001100 00001110 00010000 00010010 00010100 00010110 00011000 00011010 00011100 00011110 01000010
5 6 7 8 9 10 11 12 13 14 15 16



```
                               Msgs
/router_top_tb_16/clock        0
/router_top_tb_16/resetn       1
/router_top_tb_16/read_enb_0   x
/router_top_tb_16/read_enb_1   x
/router_top_tb_16/read_enb_2   1
/router_top_tb_16/pkt_valid    0
/router_top_tb_16/data_in      xxxxxxxx   01000010
/router_top_tb_16/data_out_0   zzzzzzzz
/router_top_tb_16/data_out_1   zzzzzzzz
/router_top_tb_16/data_out_2   zzzzzzzz   01000010 00000000 00000010 00000100 00000110 00001000 00001010 00001100 00001110 000...
/router_top_tb_16/valid_out_0  St0
/router_top_tb_16/valid_out_1  St0
/router_top_tb_16/valid_out_2  St0
/router_top_tb_16/error        StX
/router_top_tb_16/busy         St0
/router_top_tb_16/i            16         16
```

RTL Netlist:



router_reg:dut2
router_fsm:dut1
router_sync:dut3
router_fifo:fifo[0].f
router_fifo:fifo[1].f
router_fifo:fifo[2].f
clock
data_in[7:0]
pkt_valid
resetn
read_enb_1
read_enb_2
read_enb_0
busy
error
data_out_0[7:0]
valid_out_0
valid_out_1
valid_out_2
data_out_1[7:0]
data_out_2[7:0]