Router top module code and waveform – Raja Aadhithan.

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;

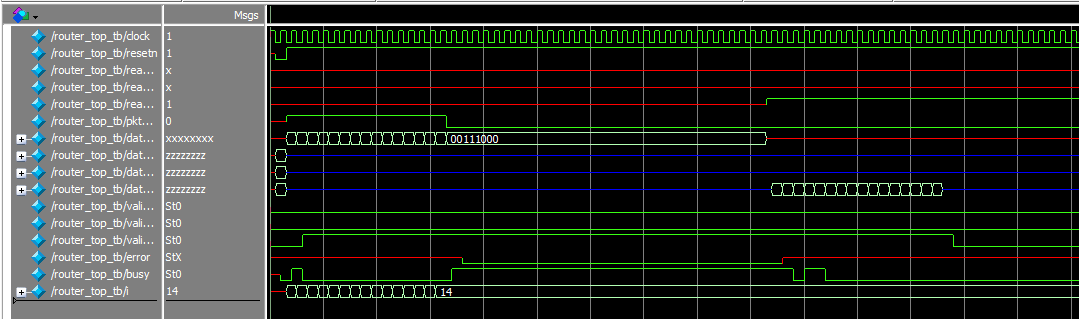
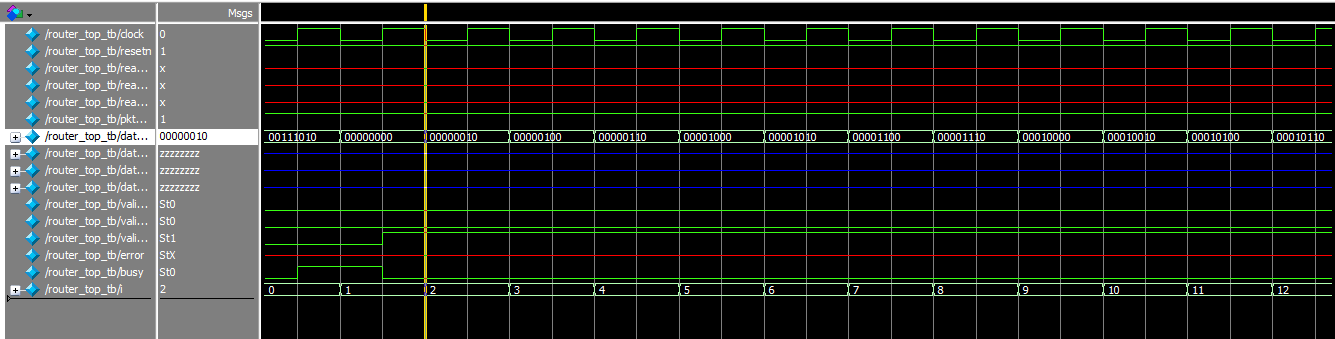
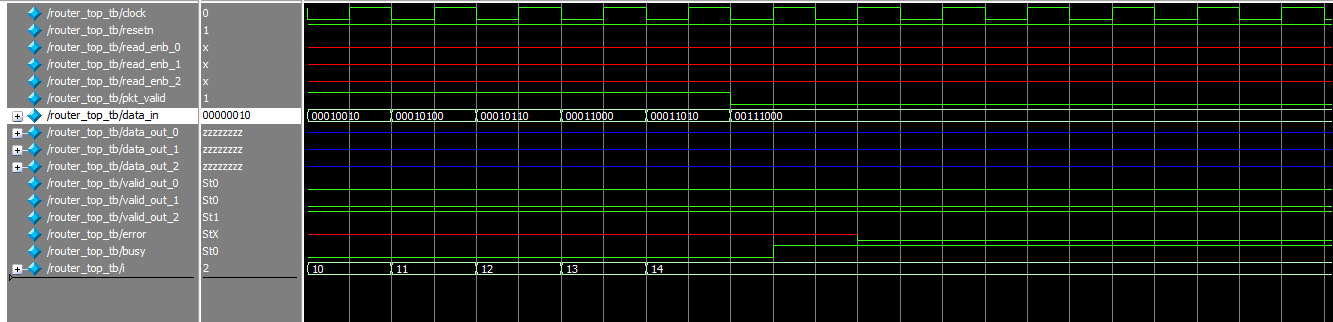
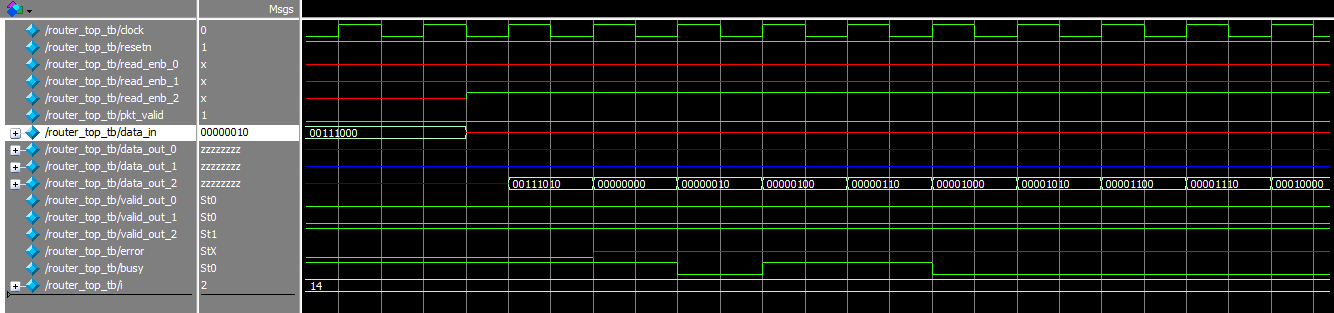
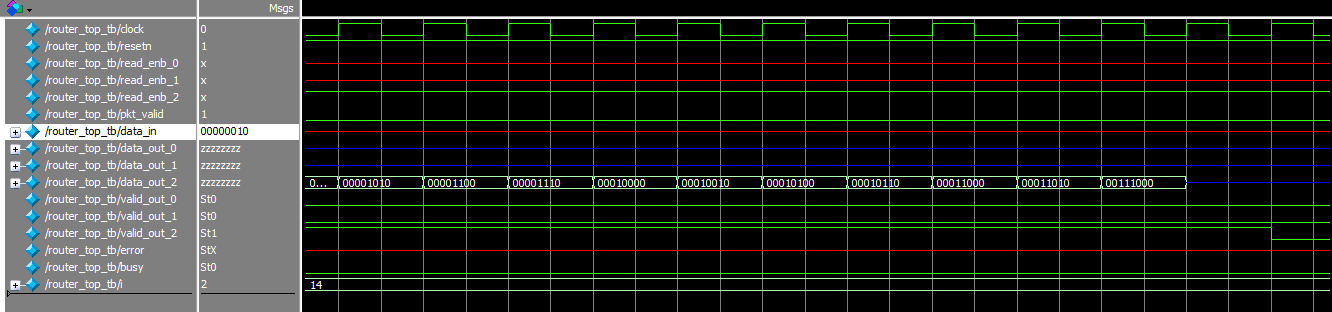
    #1000;

    $finish;

end

endmodule

WAVE (packet 14):

WAVE (packet 16):

