Router 1X3 Design and verification project.

1. module router_fifo#(parameter DEPTH=16,WIDTH=9,ADD=5) (input clock,resetn,soft_reset,write_enb,read_enb,lfd_state, input [7:0]data_in,output full,empty,output reg[7:0]data_out);

```
        integer i;
        reg[(WIDTH-1):0]mem[(DEPTH-1):0];
        reg[ADD-1:0]write_add,read_add;
        reg [5:0]count;
        reg header;

        always@(posedge clock)
                begin
                        if(~resetn)
                                header=1'b0;
                        else
                                header=lfd_state;
                end
//write
//reset
        always@(posedge clock)
                begin
                        if(~resetn)
                                begin
                                        for(i=0;i<DEPTH;i=i+1)
                                                begin
                                                        mem[i]=0;
                                                end
                                        end
           else if(soft_reset)
                                                                                        begin

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

        begin

        mem[i]=0;
                                                                                                        end
                                                                                        end
                                                        else if(write_enb&&~full)
                                                begin

        {mem[write_add[3:0]][8],mem[write_add[3:0]][7:0]}={header,data_in};
                                                        end
                    end
//read
                always@(posedge clock)
                        begin
                                if(~resetn)
                                        begin
                                                data_out<=2;
                                        end
                                else if(soft_reset)
                                                                begin
                                                                        data_out<=8'dz;
                                                                end
                                                        else if(read_enb&&~empty)
                                                                                        begin

        data_out<=mem[read_add[3:0]][7:0];
                                                                                        end
                                                                                        else if(count==0 &&
empty)

        begin

                data_out<=8'dz;

        end

                        end


//counter
                always@(posedge clock)
                        begin
                                if(~resetn)
                                        begin
                                                count<=0;
                                        end
                                else if(soft_reset)
                                        begin
                                                count<=0;
                                        end
                                        else if(read_enb&&~empty)
                                        begin
                                if(mem[read_add[3:0]][8]==1)
                                                        begin
                                                                count<=mem[read_add[3:0]][7:2]+1'b1;
                                                        end
                                                else if(count!=0)
                                                        begin
                                                                count<=count-1'b1;
                                                        end
                                        else
                                                        begin
                                                        count<=0;
                                                        end
                        end
                        end
//write pointer
                always@(posedge clock)
                        begin
                                if(~resetn)
                                        begin
                                                write_add<=0;
                                        end
                                else if(soft_reset)
                                          begin
                                                  write_add<=0;
                                          end
                        else if(write_enb&&~full)
                                                                begin
                                                                write_add<=write_add+1'b1;
                                                                end
                        end
//read pointer
                always@(posedge clock)
                        begin
                                if(~resetn)
                                        begin
                                                read_add<=0;
                                        end
                                else if(soft_reset)
                                          begin
                                                  read_add<=0;
                                          end
                        else if(read_enb&&~empty)
                                                                begin
                                                                read_add<=read_add+1'b1;
                                                                end
                        end
//full&&empty
          assign empty=(write_add==read_add);
                        assign full=(write_add=={~read_add[4],read_add[3:0]})?1'b1:1'b0;
endmodule
```

2. module router_synchronizer(input clock,resetn,detect_add,empty_0,empty_1,empty_2,write_enb_reg,read_enb_0,read_enb_1,read_enb_2,full_0,full_1,full_2,

```
                                                        input [1:0]data_in,
                                                        output reg[0:2]write_enb,
                                                        output reg
fifo_full,soft_reset_0,soft_reset_1,soft_reset_2,
                                                        output
vld_out_0,vld_out_1,vld_out_2);
        reg [1:0]temp_add;
        reg [4:0]count_0,count_1,count_2;
//temp add
        always@(posedge clock)
        begin
            if(~resetn)
                begin
                    temp_add<=0;
                end
            else if(detect_add)
                            begin
                                    temp_add<=data_in;
                            end
        end
//write_enb
    always@ *
        begin
            if(write_enb_reg)
                begin
                        case(temp_add)
                            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
//fifo_full
    always@ *
            begin
                case(temp_add)
                    2'b00 :fifo_full=full_0;
                    2'b01 :fifo_full=full_1;
                    2'b10 :fifo_full=full_2;
                    default :fifo_full=0;
                endcase
            end
//vld_out
    assign vld_out_0 = ~empty_0;
    assign vld_out_1 = ~empty_1;
    assign vld_out_2 = ~empty_2;
//soft_reset
    always@(posedge clock)
        begin
            if(~resetn)
                begin
                    count_0<=0;
                    soft_reset_0<=0;
                end
            else if(vld_out_0)
                begin
                    if(read_enb_0)
                        begin
                            count_0<=0;
                        soft_reset_0<=0;
                    end
                        else
                            begin
                                if (count_0==5'd30)
                                    begin
                                        soft_reset_0<=1;
                                        count_0<=0;
                                    end
                   else
                                    begin

count_0<=count_0+1'b1;
                                        soft_reset_0<=0;
                                    end
                            end
            end
    end
always@(posedge clock)
    begin
        if(~resetn)
            begin
                count_1<=0;
                soft_reset_1<=0;
            end
        else if(vld_out_1)
            begin
                if(read_enb_1)
                    begin
                        count_1<=0;
                        soft_reset_1<=0;
                    end
                        else
                            begin
                                if (count_1==5'd30)
                                    begin
                                        soft_reset_1<=1;
                                        count_1<=0;
                                    end
                   else
                                    begin

count_1<=count_1+1'b1;
                                        soft_reset_1<=0;
                                    end
                            end
            end
    end
always@(posedge clock)
    begin
        if(~resetn)
            begin
```

```
                                count_2<=0;
                                soft_reset_2<=0;
                           end
                      else if(vld_out_2)
                           begin
                                if(read_enb_2)
                                     begin
                                          count_2<=0;
                                          soft_reset_2<=0;
                                     end
                                          else
                                               begin
                                                    if (count_2==5'd30)
                                                         begin
                                                              soft_reset_2<=1;
                                                              count_2<=0;
                                                         end
                                   else
                                                         begin

      count_2<=count_2+1'b1;
                                                              soft_reset_2<=0;
                                                         end
                                               end
                           end
                 end
endmodule
```

---

3. module router_reg(input [7:0]data_in,

```
                                                    input
clock,resetn,pkt_valid,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,
                                                    output reg[7:0] dout,
                                                    output reg
err,parity_done,low_packet_valid);

      reg [7:0]header;
      reg [7:0]fifo_full_state;
      reg [7:0]internal_parity;
      reg [7:0]packet_parity_byte;

      always@(posedge clock)
           begin
                if(~resetn)
                     begin
                          header<=0;
                          dout<=0;
                     end
                else if(detect_add&&data_in[1:0]!=2'b11)
                                          begin
```

```
                                                header<=data_in;
                                           end
                                  else if(lfd_state)
                                                     begin
                                                          dout<=header;
                                                     end
                                                else if(ld_state && ~fifo_full)
                                                                          begin

     dout<=data_in;

                                                                          end
                                                          else if(ld_state &&
fifo_full)

     begin

               fifo_full_state<=data_in;

     end
                                                                               else
if(laf_state)

                    begin

                         dout<=fifo_full_state;

                    end
          end

     /*always@(posedge clock)
          begin
               if(~resetn)
                    fifo_full_state<=0;
               else if(ld_state &&fifo_full)
                                           begin
                                                fifo_full_state<=data_in;
                                           end
                                  else if(laf_state)
                                                     begin
                                                          dout<=fifo_full_state;
                                                     end
          end*/

//parity done
     always@(posedge clock)
          begin
               if(~resetn)
                    begin
                         parity_done<=1'b0;
                    end
               else if(detect_add)
                                           begin
```

```
                                                            parity_done<=1'b0;
                                                      end
                        else if(ld_state && ~pkt_valid && ~fifo_full)
                                                      begin
                                                            parity_done<=1'b1;
                                                      end
                        else if(laf_state && low_packet_valid && ~parity_done)
                                                      begin
                                                            parity_done<=1'b1;
                                                      end
                  end

//low_packet_valid
            always@(posedge clock)
                  begin
                        if(~resetn)
                              begin
                                    low_packet_valid<=1'b0;
                              end
                        else if(rst_int_reg)
                                                      low_packet_valid<=1'b0;
                                    else if(~pkt_valid && ld_state)
                                                      low_packet_valid<=1'b1;
                  end
//parity cheak(internal parity)
            always@(posedge clock)
                  begin
                        if(~resetn)
                              begin
                                    internal_parity<=1'b0;
                              end
                        else if(detect_add)
                                    internal_parity<=1'b0;
                        else if(lfd_state)
                                                      begin
                                                            internal_parity<=internal_parity^header;
                                                      end
                                    else if(ld_state && pkt_valid && ~full_state)
                                                      begin

            internal_parity<=internal_parity^data_in;
                                                      end
                  end

//packet_parity_byte(external parity)
            always@(posedge clock)
                  begin
                        if(~pkt_valid && ld_state)
                                    begin
                                          packet_parity_byte<=data_in[7:0];
                                    end
             end
```

```
//err
        always@(posedge clock)
                begin
                        if(~resetn)
                                begin
                                        err<=0;
                                end
                        else if(parity_done==1)
                                                if(internal_parity==packet_parity_byte)
                                                                        begin
                                                                                err<=1'b0;
                                                                        end
                                                                        else
                                                                                begin
                                                                                        err<=1'b1;
                                                                                end
                end

endmodule
```

4. module router_fsm(input clock,resetn,pkt_valid,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_reset_1,soft_reset_2,parity_done,

```
                                                                        low_packet_valid,
                                                                        input [1:0]data_in,
                                                                        output
write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy);

                                                                        parameter DECODE_ADDRESS
                =8'b10000000,

        LOAD_FIRST_DATA                 =8'b01000000,

        LOAD_DATA                               =8'b00100000,

        LOAD_PARITY                             =8'b00010000,

        FIFO_FULL_STATE         =8'b00001000,

        LOAD_AFTER_FULL                 =8'b00000100,

        WAIT_TILL_EMPTY                 =8'b00000010,

        CHECK_PARITY_ERROR=8'b00000001;

        reg[7:0]state,next_state;
        reg [1:0]add;

        always@(posedge clock)
                begin
                        if(detect_add)
                                add<=data_in[1:0];
                end

        always@(posedge clock)
                begin
                        if(~resetn)
                                state<=DECODE_ADDRESS;
                        else
if((soft_reset_0&&add==2'b00)||(soft_reset_1&&add==2'b01)||(soft_reset_2&&add==2'b10))
                                state<=DECODE_ADDRESS;
                        else
                                state<=next_state;
                end

        always@*
                        begin
                                next_state=DECODE_ADDRESS;
                                case(state)
                                        DECODE_ADDRESS          :if((pkt_valid &&
(data_in[1:0]==0)&&fifo_empty_0)||

                                (pkt_valid && (data_in[1:0]==1)&&fifo_empty_1)||

                                (pkt_valid && (data_in[2:0]==2)&&fifo_empty_2))

                                begin

                                        next_state=LOAD_FIRST_DATA;

                                end

                        else if((pkt_valid && (data_in[1:0]==0)&&~fifo_empty_0)|

                                (pkt_valid && (data_in[1:0]==1)&&~fifo_empty_1)|

                                (pkt_valid && (data_in[1:0]==2)&&~fifo_empty_2))

                                begin

                                        next_state=WAIT_TILL_EMPTY;

                                end

                                        LOAD_FIRST_DATA
        :next_state=LOAD_DATA;

                                        LOAD_DATA
        :if(fifo_full==1)

                        begin

                                next_state=FIFO_FULL_STATE;
```

```
                    end

                else if(fifo_full==0&&pkt_valid==0)

                                    begin

                                next_state=LOAD_PARITY;

                                    end

                                else

                                    begin

                                        next_state=LOAD_DATA;

                                    end

                                LOAD_PARITY
            :next_state=CHECK_PARITY_ERROR;

                                FIFO_FULL_STATE                 :if(fifo_full==1)

                            next_state=FIFO_FULL_STATE;

                else if(fifo_full==0)

                                        next_state=LOAD_AFTER_FULL;

                                LOAD_AFTER_FULL                     :if(parity_done==0
    && low_packet_valid==0)

                                next_state=LOAD_DATA;

                else if(parity_done==0 && low_packet_valid==1)

                                        next_state=LOAD_PARITY;

                    else if(parity_done==1)


            next_state=DECODE_ADDRESS;

                                WAIT_TILL_EMPTY
            :if(~fifo_empty_0||~fifo_empty_1||~fifo_empty_2)

                                next_state=WAIT_TILL_EMPTY;

                else if(fifo_empty_0&&data_in==2'b00||

                                    fifo_empty_1&&data_in==2'b01||
                                    fifo_empty_2&&data_in==2'b10)

                                next_state=LOAD_FIRST_DATA;

                                CHECK_PARITY_ERROR  :if(fifo_full==1)

                            next_state=FIFO_FULL_STATE;

                else if(fifo_full==0)

                            next_state=DECODE_ADDRESS;
                            endcase
                end


            assign write_enb_reg  =
    ((state==LOAD_DATA)||(state==LOAD_AFTER_FULL)||(state==LOAD_PARITY))?1'b1:1'b0;
            assign detect_add                 = (state==DECODE_ADDRESS);
            assign ld_state                   = (state==LOAD_DATA);
            assign laf_state     = (state==LOAD_AFTER_FULL);
            assign lfd_state                  = (state==LOAD_FIRST_DATA);
            assign full_state    = (state==FIFO_FULL_STATE);
            assign rst_int_reg   = (state==CHECK_PARITY_ERROR);
            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));
    endmodule
```

---

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

```
        wire [7:0]dout;
        wire [2:0]write_enb;

//router1
  router_fifo FIFO1(clock,resetn,soft_reset_0,write_enb[0],read_enb_0,lfd_state,//ip
                                                                dout,

        fifo_full_0,fifo_empty_0,data_out_0);//op

//rouetr2
        router_fifo FIFO2(clock,resetn,soft_reset_1,write_enb[1],read_enb_1,lfd_state,//ip
                                                                dout,

        fifo_full_1,fifo_empty_1,data_out_1);//op

//router_3
        router_fifo FIFO3(clock,resetn,soft_reset_2,write_enb[2],read_enb_2,lfd_state,//ip
                                                                                         dout,

        fifo_full_2,fifo_empty_2,data_out_2);//op

//synchronizer
        router_synchronizer SYN(clock,resetn,detect_add,//ip

fifo_empty_0,fifo_empty_1,fifo_empty_2,
                                                                                          write_enb_reg,

read_enb_0,read_enb_1,read_enb_2,

fifo_full_0,fifo_full_1,fifo_full_2,
                                                                                          data_in[1:0],
                                                                                          write_enb,//op

fifo_full,soft_reset_0,soft_reset_1,soft_reset_2,
                                                                                          vld_out_0,vld_out_1,vld_out_2);

//fsm
        router_fsm
FSM(clock,resetn,pkt_valid,fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_re
set_1,soft_reset_2,parity_done,//ip
                                                                                  low_packet_valid,
                                                                                  data_in[1:0],

write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy);//op

//register
        router_reg REG(data_in,//ip

clock,resetn,pkt_valid,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,
                                                                                  dout,//op
                                                                                  err,parity_done,low_packet_valid);
endmodule
```

---

6. module router_top_tb();

```
        reg [7:0]data_in;
        reg clock,resetn,read_enb_0,read_enb_1,read_enb_2,pkt_valid;
        wire [7:0]data_out_0,data_out_1,data_out_2;
        wire vld_out_0,vld_out_1,vld_out_2,err,busy;
        parameter CYCLE=20;

        router_top DUT(data_in,

        clock,resetn,read_enb_0,read_enb_1,read_enb_2,pkt_valid,
                                                                       data_out_0,data_out_1,data_out_2,
                                                                       vld_out_0,vld_out_1,vld_out_2,err,busy);


        always
```

```
          begin
                    #(CYCLE/2);
                              clock=1'b0;
                    #(CYCLE/2);
                              clock=1'b1;
          end

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

task pkt_gen();
          reg [7:0] payload_data,parity,header;
          reg [5:0]payload_leng;
          reg [1:0]addr;
          integer i;

begin
          wait(~busy)
          @(negedge clock);
          payload_leng=14;
          addr=2'b01;
          header={payload_leng,addr};
          parity=0;
          data_in=header;
          pkt_valid=1;
          parity=parity^header;

          @(negedge clock);
          wait(~busy)
          for(i=0;i<payload_leng;i=i+1)

          begin
                    @(negedge clock)
                    wait(~busy)
                    payload_data={$random}%256;
                    data_in=payload_data;
                    parity=parity^payload_data;
          end

                    @(negedge clock)
                    wait(~busy)
                    pkt_valid=0;
                    data_in=parity;
          end
endtask
```

```
task pkt_gen_16();
        reg [7:0] payload_data,parity,header;
        reg [5:0]payload_leng;
        reg [1:0]addr;
        integer i;

begin
        wait(~busy)
        @(negedge clock);
        payload_leng=16;
        addr=2'b00;
        header={payload_leng,addr};
        parity=0;
        data_in=header;
        pkt_valid=1;
        parity=parity^header;

        @(negedge clock);
        wait(~busy)
        for(i=0;i<payload_leng;i=i+1)

        begin
                @(negedge clock)
                wait(~busy)
                payload_data={$random}%256;
                data_in=payload_data;
                parity=parity^payload_data;
        end

                @(negedge clock)
                wait(~busy)
                pkt_valid=0;
                data_in=parity;
        end
endtask


initial
        begin
                reset();
                pkt_gen();
                repeat(2)
                @(negedge clock);
                read_enb_1=1;
                wait(~vld_out_1)
                #20;
                read_enb_1=0;
                @(negedge clock);
                pkt_gen_16();
                repeat(2)
                @(negedge clock);
                read_enb_0=1;
                wait(~vld_out_0)
                #20;
                read_enb_0=0;
                @(negedge clock);
                #1000;
        end
endmodule
```