```
module router_fifo(
   input clock,resetn,soft_reset,write_enb,read_enb,lfd_state,
   input [7:0]data_in,
   output empty,full,
   output reg [7:0] data_out
   );
reg [4:0]waddr,raddr;
reg [8:0]fifo[15:0];
integer i;
reg [6:0]count;
reg temp;
wire count_z = (count==7'b0000000)? 1'b1:1'b0;

always@(posedge clock)  // Delay lfd by 1 clock
begin
 if(!resetn)
   temp<= 1'b0;
 else
   temp<= lfd_state;
end

always@(posedge clock)  // Write operation
begin
 if(!resetn)
 begin  // Reset applied
   waddr<= 5'b00000;
   for (i=0; i<16; i=i+1) //FIFO initialization to 0
     fifo[i] <= 9'b000000000;
 end
 else if(soft_reset)
 begin  // Soft Reset applied
```

```
   waddr<= 5'b00000;
   for (i=0; i<16; i=i+1) //FIFO initialization to 0
    fifo[i] <= 9'b000000000;
  end
  else if((write_enb && !full)||(read_enb && !empty)) // Writing into fifo
  begin
   case({write_enb,read_enb})
          2'b10: begin
                fifo[waddr[3:0]]<= {temp,data_in};
                                waddr<= waddr+1'b1;
                        end
          2'b1x: begin
                fifo[waddr[3:0]]<= {temp,data_in};
                                waddr<= waddr+1'b1;
                        end
          2'b01: fifo[raddr[3:0]]<= 9'b000000000;
          2'bx1: fifo[raddr[3:0]]<= 9'b000000000;
          2'b11: begin
                fifo[waddr[3:0]]<= {temp,data_in};
                                waddr<= waddr+1'b1;
                fifo[raddr[3:0]]<= 9'b000000000;
               end
          default: fifo[waddr[3:0]]<= fifo[waddr[3:0]];
          endcase
  end
  else
   fifo[waddr[3:0]]<= fifo[waddr[3:0]];
end

always@(posedge clock)  // Read operation
begin
```

```
 if(!resetn)
 begin  // Reset applied
  raddr<= 5'b00000;
  data_out<= 8'h00;
 end
 else if(soft_reset)
 begin  // Soft Reset applied
  raddr<= 5'b00000;
  data_out<= 8'hzz;
 end
 /*else if(count_z)  // Packet transmission completed
 begin
  data_out<= 8'hzz;
 end*/
 else if(read_enb && !empty)
 begin
  data_out<= fifo[raddr[3:0]][7:0];
  raddr<= raddr+1'b1;
          end
 else if(count_z)  // Packet transmission completed
 begin
  data_out<= 8'hzz;
 end
 else
  data_out<= data_out;
end

always@(posedge clock)  // Internal counter logic
begin
 if(!resetn)
 begin
```

```
  count<= 7'b0000000;
 end
 else if(soft_reset)
 begin
  count<= 7'b0000000;
 end
 else if(fifo[raddr[3:0]][8])
 begin
  count<= (fifo[raddr[3:0]][7:2]) + 1'b1;
 end
 else if(count==0)
 begin
  count<= 7'b0000000;
 end
 else if(read_enb && !empty)
 begin
  count<= count-1'b1;
 end
 else
 begin
  count<= count;
 end
end

assign empty = (waddr==raddr);
assign full = ((waddr[4]!=raddr[4]) && (waddr[3:0]==raddr[3:0]));

endmodule
```