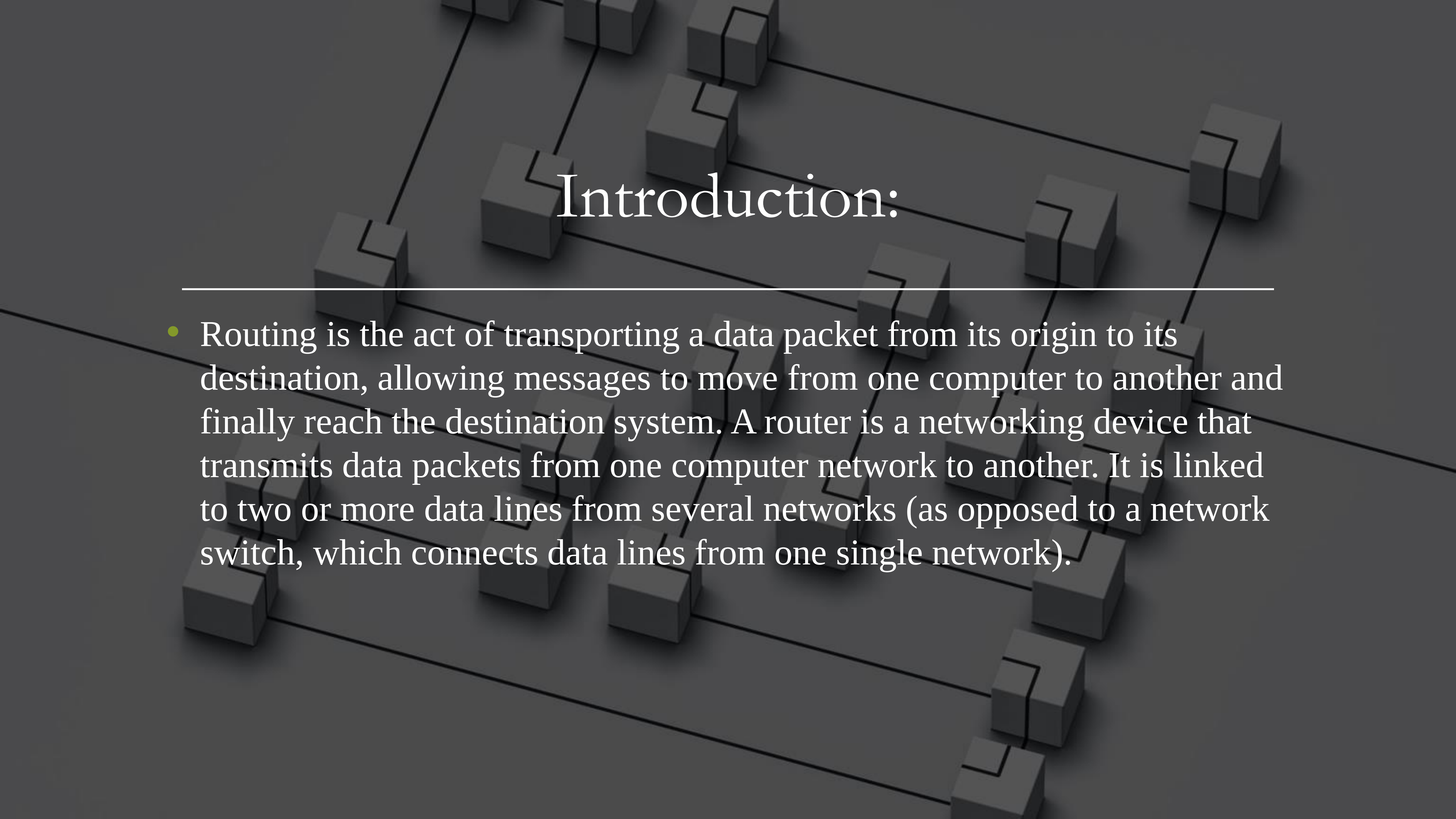
Router Design

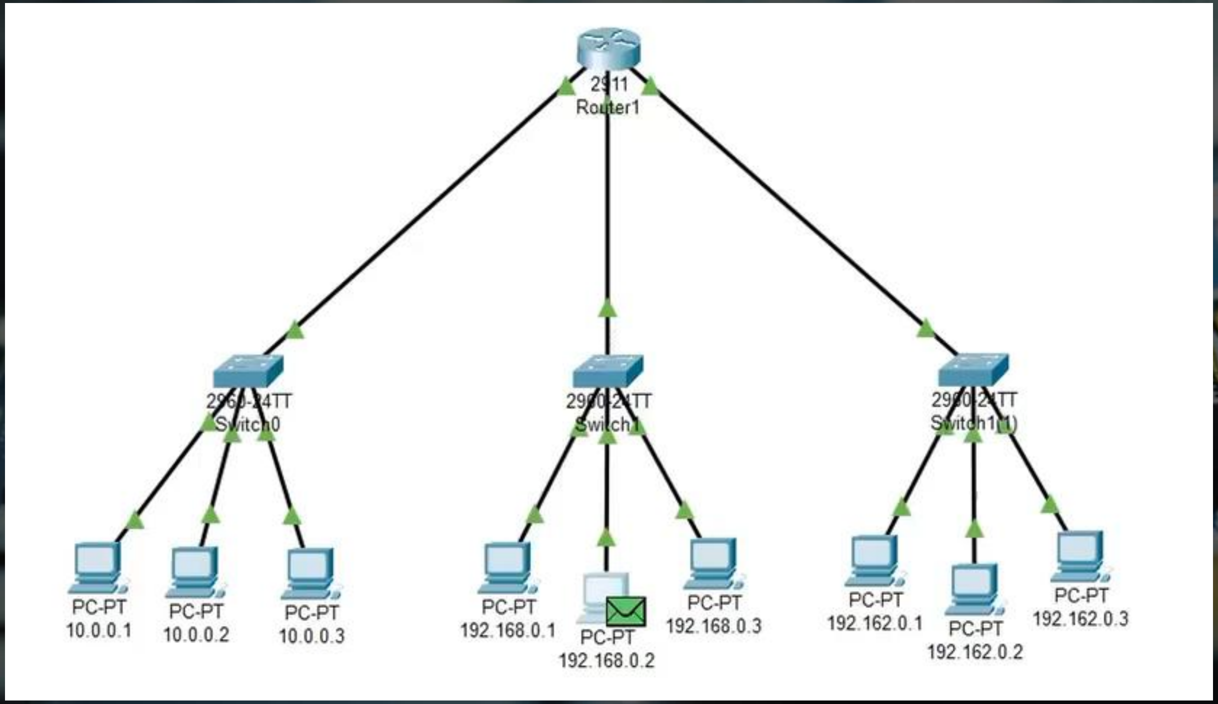
By: Kumar Sai Reddy Guntaka



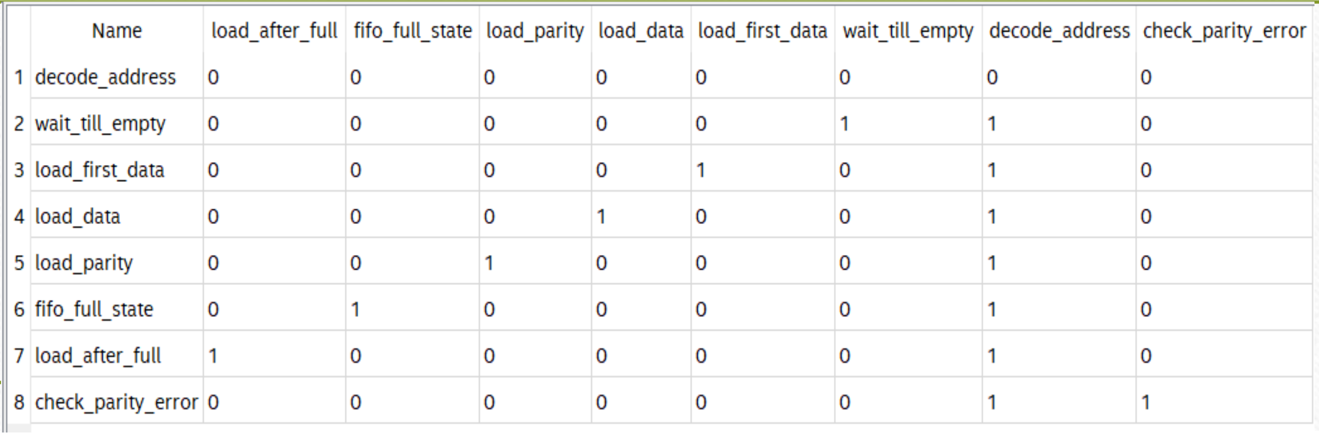
Introduction:

- Routing is the act of transporting a data packet from its origin to its destination, allowing messages to move from one computer to another and finally reach the destination system. A router is a networking device that transmits data packets from one computer network to another. It is linked to two or more data lines from several networks (as opposed to a network switch, which connects data lines from one single network).

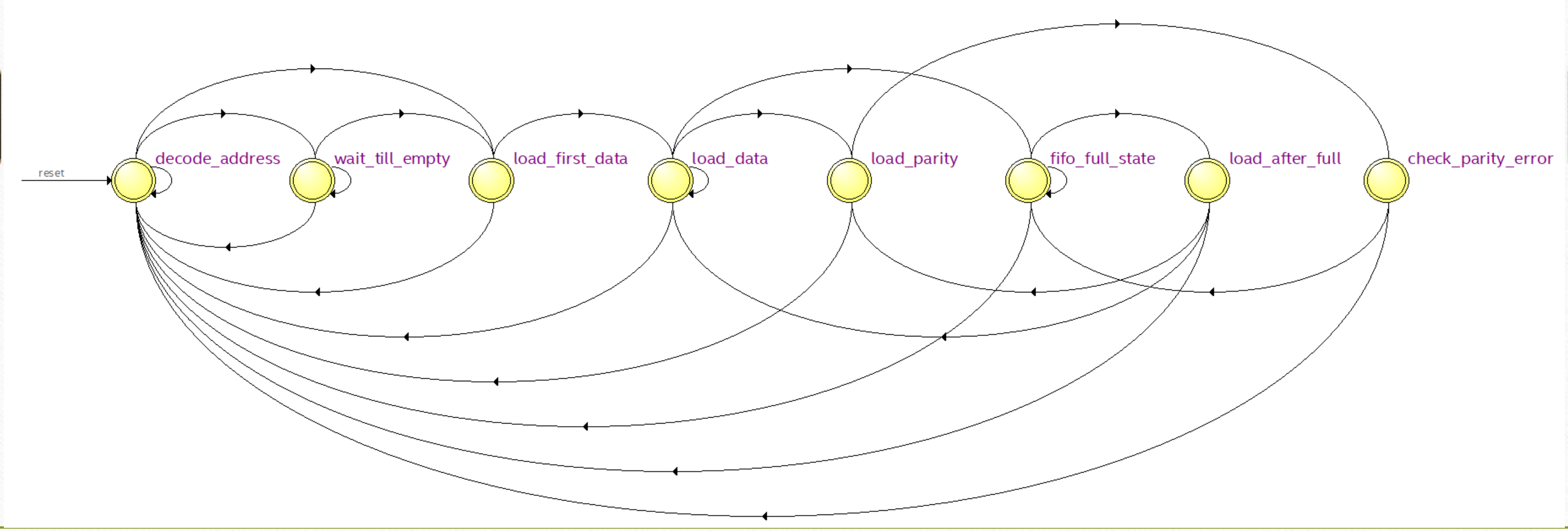
Cisco Packet Tracer



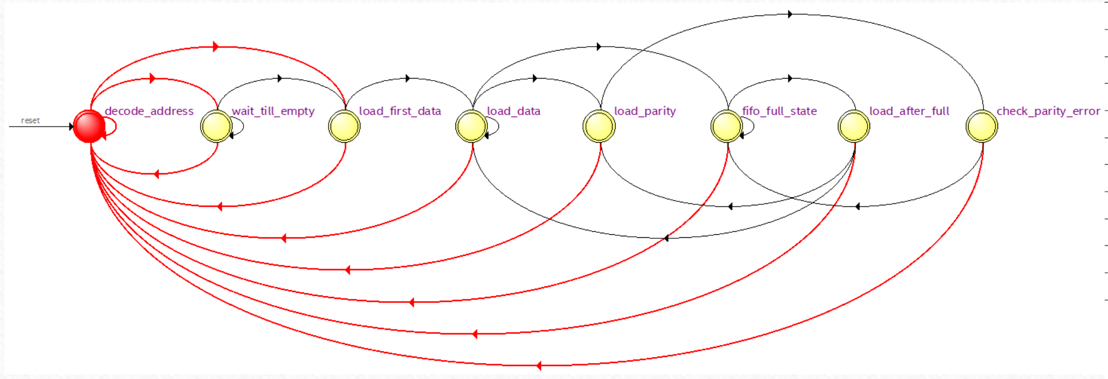
FSM:

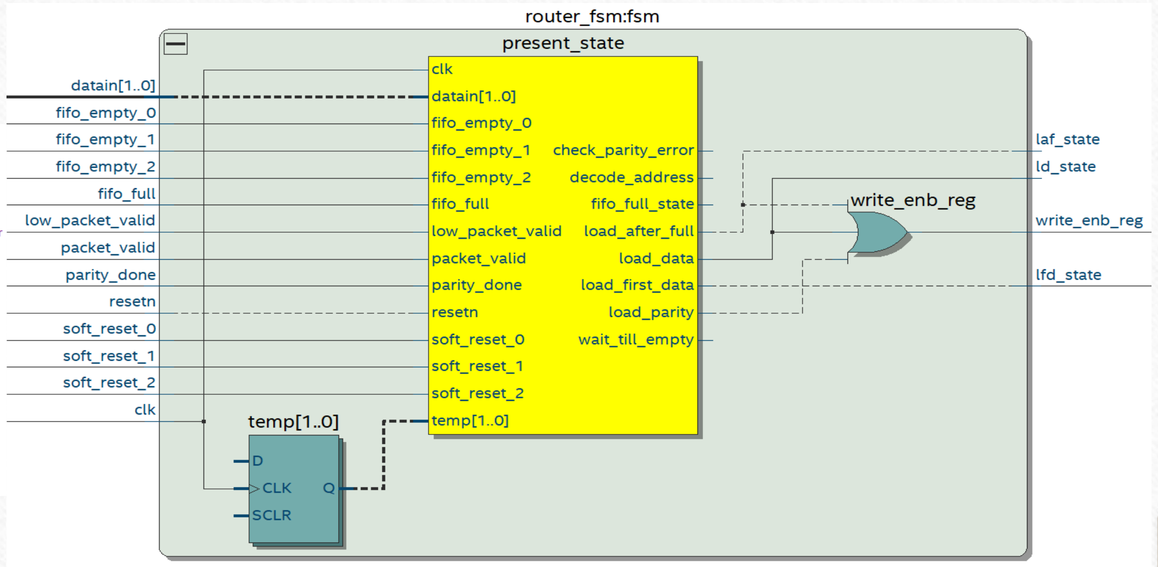


	Name	load_after_full	fifo_full_state	load_parity	load_data	load_first_data	wait_till_empty	decode_address	check_parity_error
1	decode_address	0	0	0	0	0	0	0	0
2	wait_till_empty	0	0	0	0	0	1	1	0
3	load_first_data	0	0	0	0	1	0	1	0
4	load_data	0	0	0	1	0	0	1	0
5	load_parity	0	0	1	0	0	0	1	0
6	fifo_full_state	0	1	0	0	0	0	1	0
7	load_after_full	1	0	0	0	0	0	1	0
8	check_parity_error	0	0	0	0	0	0	1	1

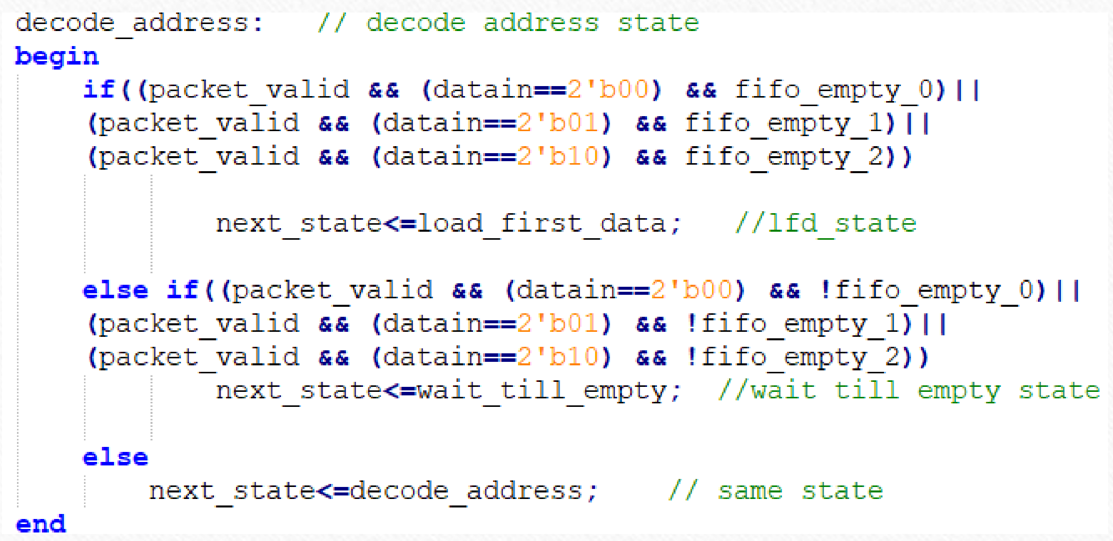


State1: Decode address



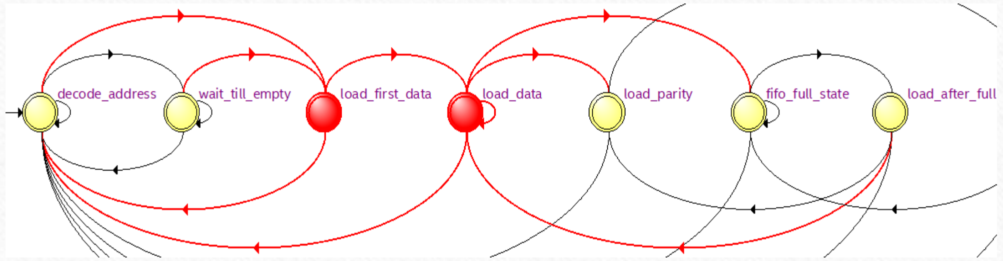


- FSM is the monitoring and controlling unit of the router architecture.
- State1: Decode_Address -> initial reset state
- Signal **detect_add** is asserted in this state which is used to detect an incoming packet. It is also used to latch the first byte as a header byte.



```
decode_address:   // decode address state
begin
    if((packet_valid && (datain==2'b00) && fifo_empty_0)||
    (packet_valid && (datain==2'b01) && fifo_empty_1)||
    (packet_valid && (datain==2'b10) && fifo_empty_2))

            next_state<=load_first_data;   //lfd_state

    else if((packet_valid && (datain==2'b00) && !fifo_empty_0)||
    (packet_valid && (datain==2'b01) && !fifo_empty_1)||
    (packet_valid && (datain==2'b10) && !fifo_empty_2))
            next_state<=wait_till_empty;  //wait till empty state

    else
        next_state<=decode_address;    // same state
end
```

State 2: Load_data and first data

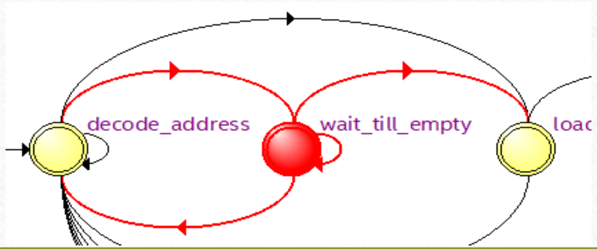


- The first data byte to the FIFO. Signal busy is also asserted in this state so that header byte that is already latched doesn't update to a new value for the current packet.
- This state is changed to LAOD_DATA state unconditionally in the next clock cycle.

```
load_first_data:              // load first data state
begin
    next_state<=load_data;
end
```

```
load_data:                              //load data
begin
    if(fifo_full==1'b1)
            next_state<=fifo_full_state;
    else
            begin
                if (!fifo_full && !packet_valid)
                    next_state<=load_parity;
                else
                    next_state<=load_data;
            end
end
```

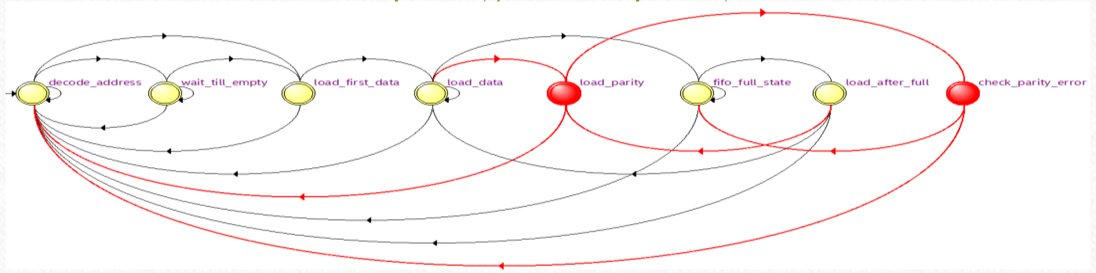
State3: Wait_till_empty



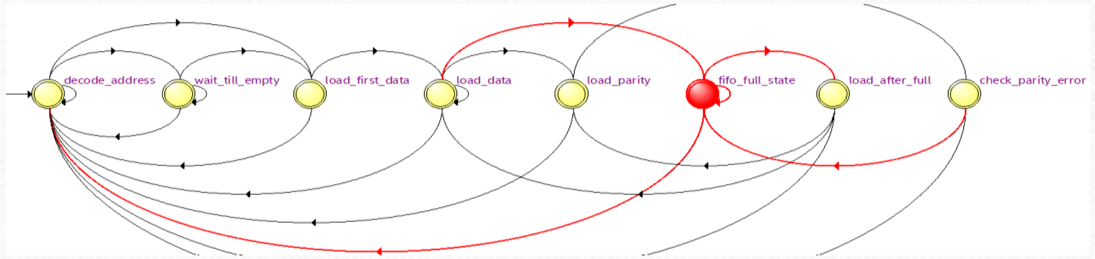
- Signal busy is de asserted in this state, so that ROUTER can receive the new data from input source every clock cycle,
- Signal write_enb_reg is asserted in this state in order to write the Packet information (Header+Payload+Parity) to the selected FIFO.
- This state transits to LAOD_PARITY state when pkt_valid goes low and to FIFO_FULL_STATE when FIFO is full.

```
wait_till_empty:             //wait till empty state
begin
    if((fifo_empty_0 && (temp==2'b00))||
    (fifo_empty_1 && (temp==2'b01))||
    (fifo_empty_2 && (temp==2'b10)))
    //fifo is empty and were using same fifo
            next_state<=load_first_data;

        else
            next_state<=wait_till_empty;
    end
```

State4&8: Load parity, check parity error



State5: FIFO full state

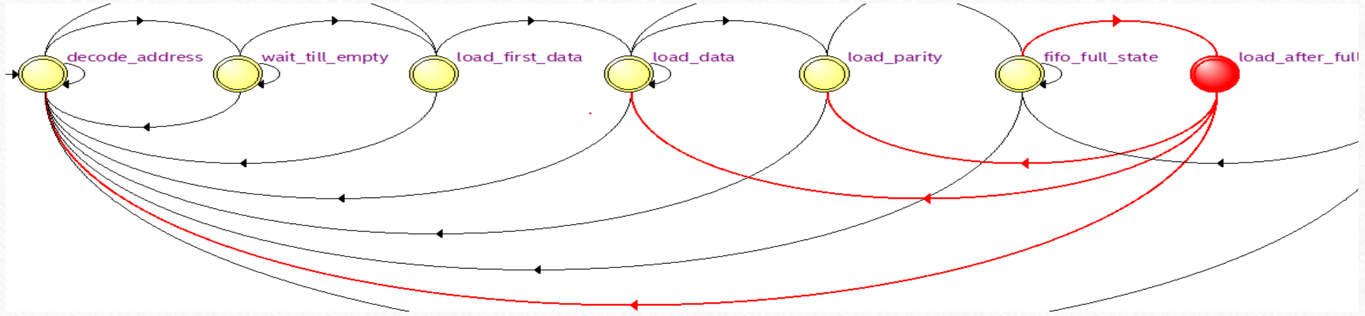


- In this state the last byte is latched which is the parity byte. It goes unconditionally to the state CHECK_PARITY_ERROR.
- Signal busy is asserted so that ROUTER doesn‟t accepts any new data. write_enb_reg is made high for latching the parity byte to FIFO.

- Busy signal is made high and write_enb_reg signal is made low.
- Signal full_state is asserted which detects the FIFO full state.

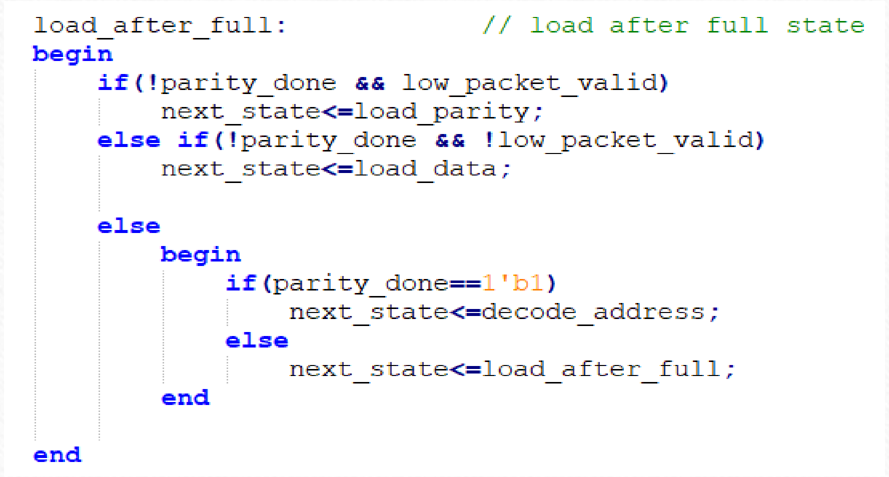
```
load_parity:                 // load parity state
begin
    next_state<=check_parity_error;
end
---------------------------------------------------
check_parity_error:         // check parity error
begin
    if(!fifo_full)
        next_state<=decode_address;
    else
        next_state<=fifo_full_state;
end
```

```
fifo_full_state:              //fifo full state
begin
    if(fifo_full==0)
        next_state<=load_after_full;
    else
        next_state<=fifo_full_state;
end
```

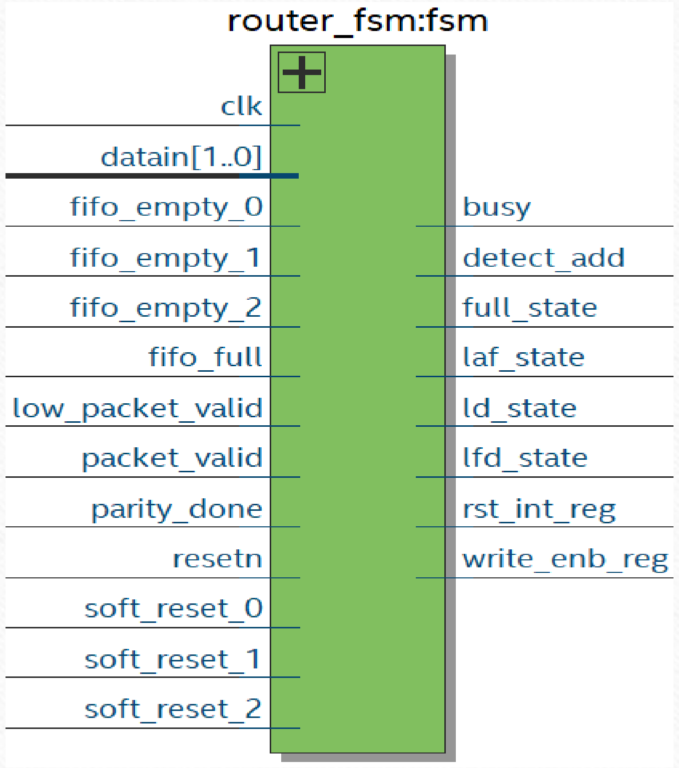


Load after full state:

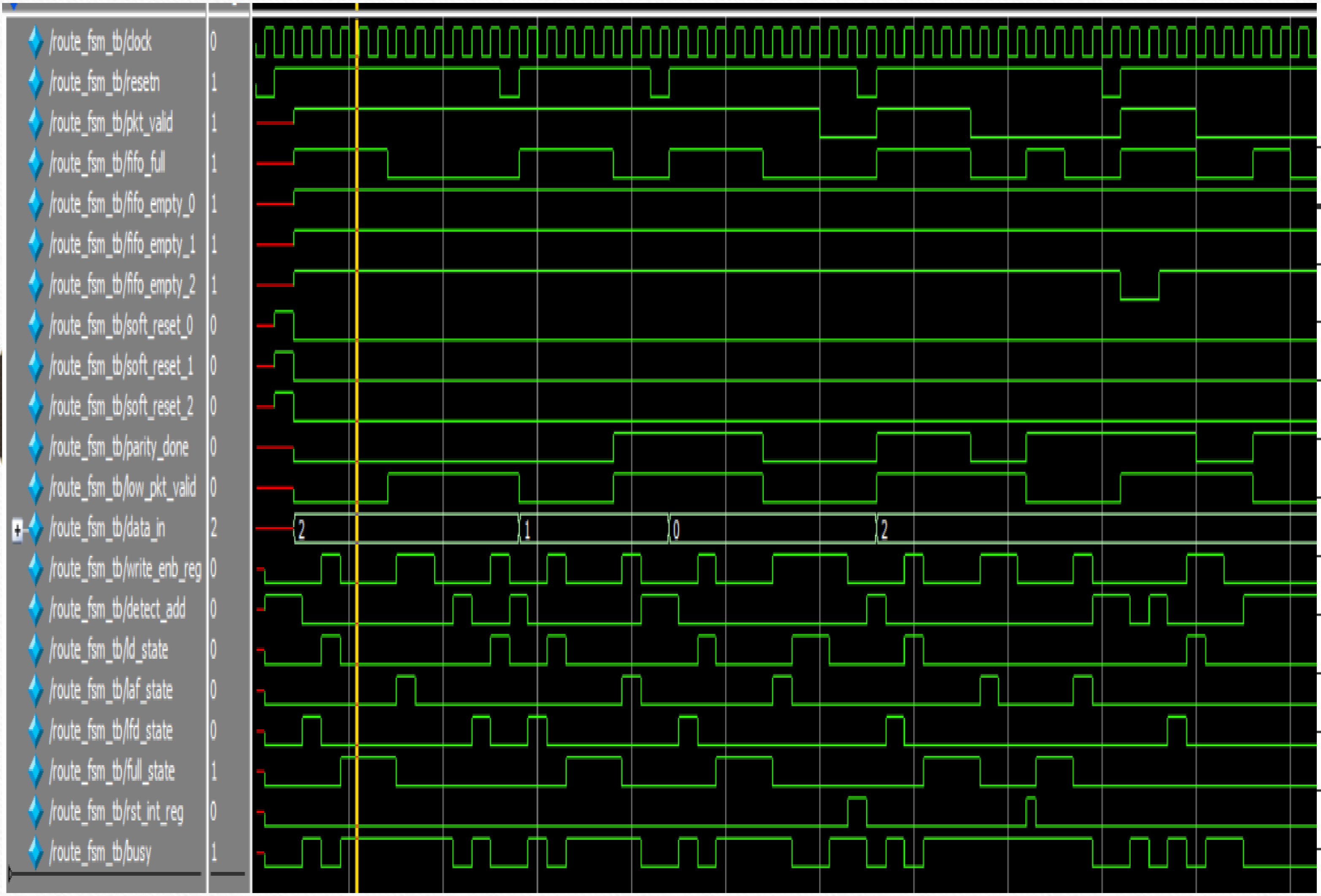
- The data after FIFO_FULL_STATE. Signal busy & write_enb_reg is asserted. It checks for parity_done signal and if it is high, shows that LOAD_PARITY state has been detected and it goes to the state DECODE_ADDRESS.
- If low_packet_valid is high it goes to LOAD_PARITY state otherwise it goes back to the LOAD_DATA state.



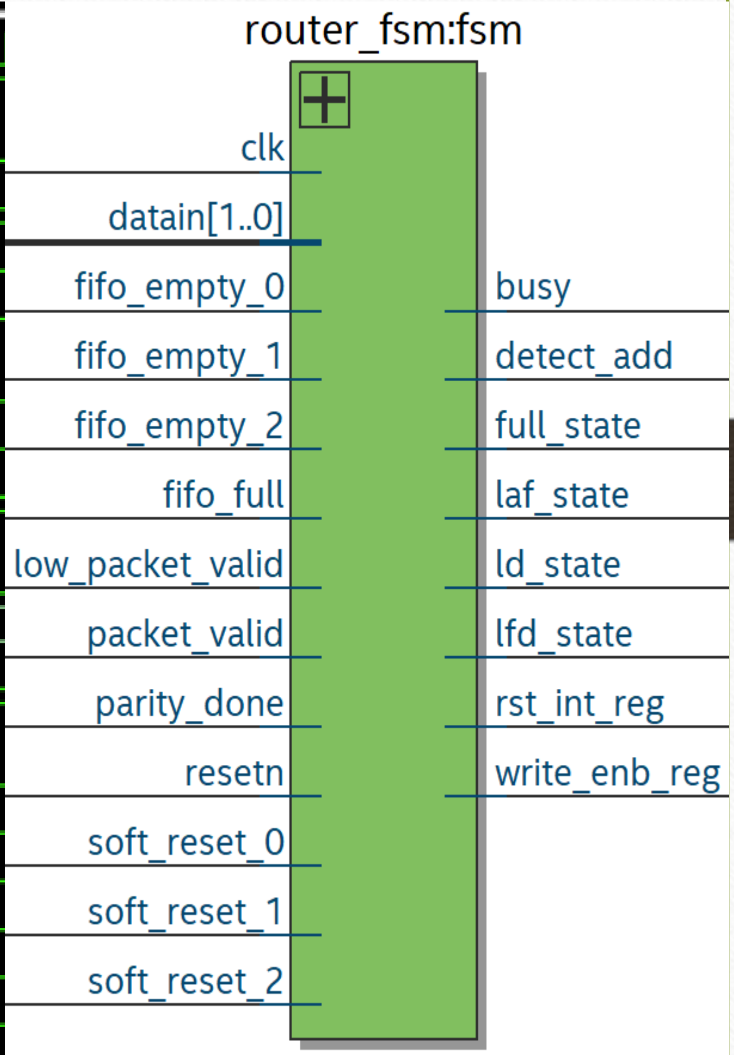
```
load_after_full:               // load after full state
begin
    if(!parity_done && low_packet_valid)
        next_state<=load_parity;
    else if(!parity_done && !low_packet_valid)
        next_state<=load_data;

    else
        begin
            if(parity_done==1'b1)
                next_state<=decode_address;
            else
                next_state<=load_after_full;
        end
end
```



FSM

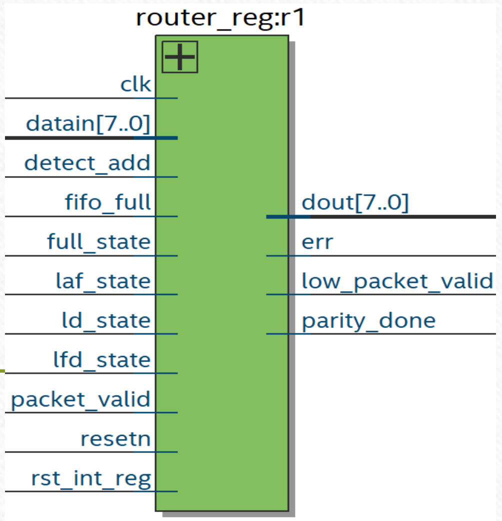


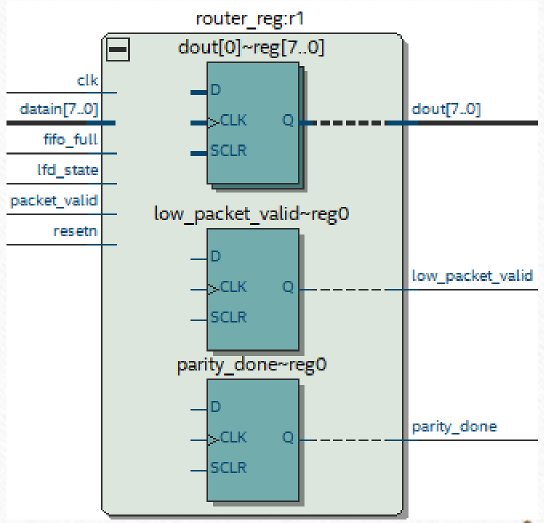
/route_fsm_tb/clock 0
/route_fsm_tb/resetn 1
/route_fsm_tb/pkt_valid 1
/route_fsm_tb/fifo_full 1
/route_fsm_tb/fifo_empty_0 1
/route_fsm_tb/fifo_empty_1 1
/route_fsm_tb/fifo_empty_2 1
/route_fsm_tb/soft_reset_0 0
/route_fsm_tb/soft_reset_1 0
/route_fsm_tb/soft_reset_2 0
/route_fsm_tb/parity_done 0
/route_fsm_tb/low_pkt_valid 0
/route_fsm_tb/data_in 2
/route_fsm_tb/write_enb_reg 0
/route_fsm_tb/detect_add 0
/route_fsm_tb/ld_state 0
/route_fsm_tb/laf_state 0
/route_fsm_tb/lfd_state 0
/route_fsm_tb/full_state 1
/route_fsm_tb/rst_int_reg 0
/route_fsm_tb/busy 1
2
1
0
2

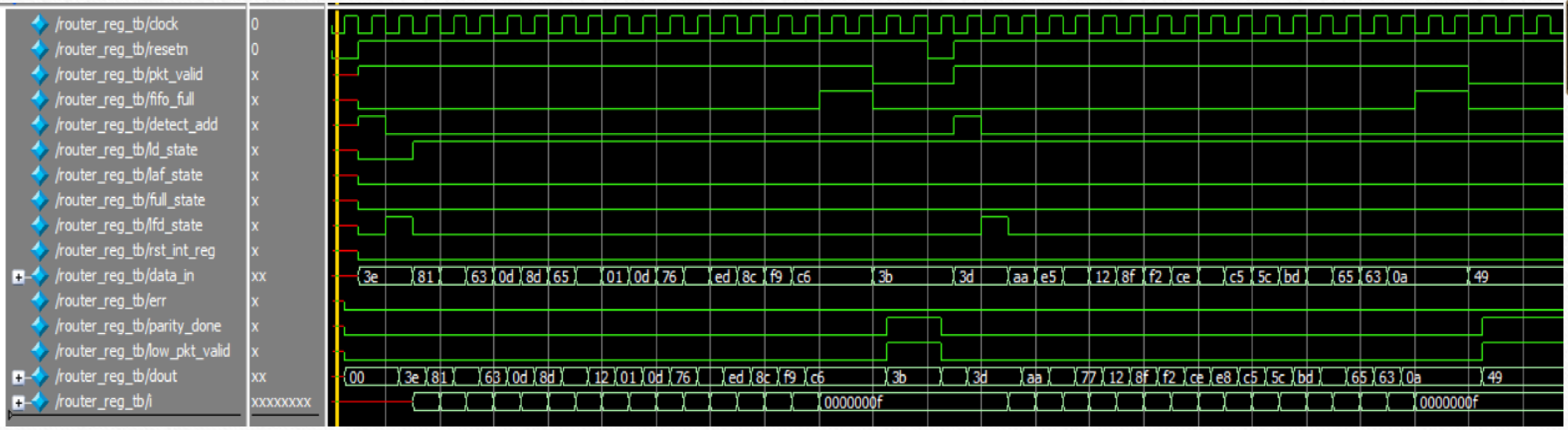


router_fsm:fsm
clk
datain[1..0]
fifo_empty_0
fifo_empty_1
fifo_empty_2
fifo_full
low_packet_valid
packet_valid
parity_done
resetn
soft_reset_0
soft_reset_1
soft_reset_2
busy
detect_add
full_state
laf_state
ld_state
lfd_state
rst_int_reg
write_enb_reg

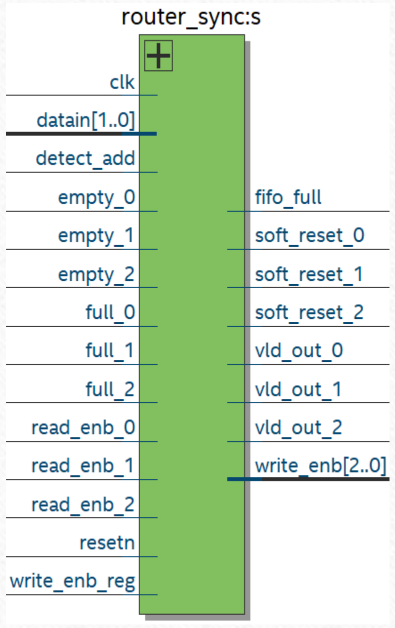
REGISTER

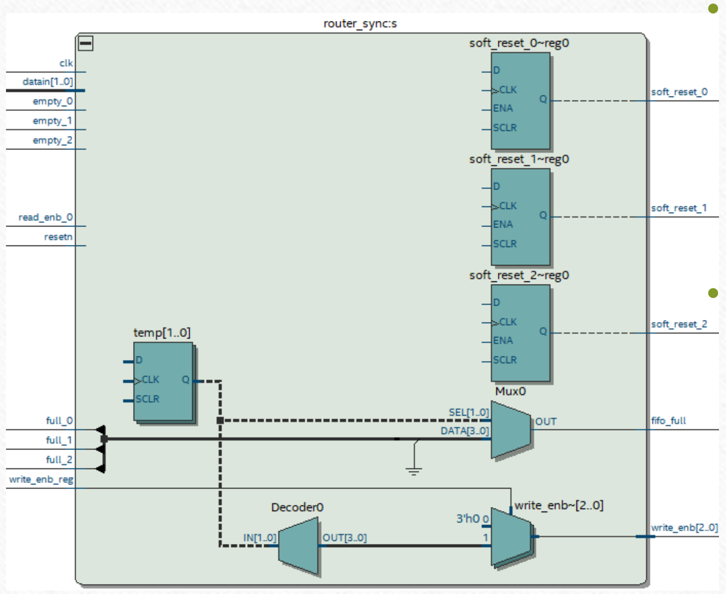

router_reg:r1
clk
datain[7..0]
detect_add
fifo_full
full_state
laf_state
ld_state
lfd_state
packet_valid
resetn
rst_int_reg
dout[7..0]
err
low_packet_valid
parity_done


router_reg:r1
dout[0]~reg[7..0]
clk
datain[7..0]
fifo_full
lfd_state
packet_valid
resetn
D
CLK
SCLR
Q
dout[7..0]
low_packet_valid~reg0
low_packet_valid
parity_done~reg0
parity_done


/router_reg_tb/clock 0
/router_reg_tb/resetn 0
/router_reg_tb/pkt_valid x
/router_reg_tb/fifo_full x
/router_reg_tb/detect_add x
/router_reg_tb/ld_state x
/router_reg_tb/laf_state x
/router_reg_tb/full_state x
/router_reg_tb/lfd_state x
/router_reg_tb/rst_int_reg x
/router_reg_tb/data_in xx
/router_reg_tb/err x
/router_reg_tb/parity_done x
/router_reg_tb/low_pkt_valid x
/router_reg_tb/dout xx
/router_reg_tb/i xxxxxxxx
3e 81 63 0d 8d 65 01 0d 76 ed 8c f9 c6 3b 3d aa e5 12 8f f2 ce c5 5c bd 65 63 0a 49
00 3e 81 63 0d 8d 12 01 0d 76 ed 8c f9 c6 3b 3d aa 77 12 8f f2 ce e8 c5 5c bd 65 63 0a 49
0000000f
0000000f

SYNCHRONIZER





- **Detect_add** and **data_in** signals are used to select a FIFO till a packet routing is over for the selected FIFO.
- Signal **fifo_full** signal is asserted based on full_status of **fifo_0** or **FIFO_1** or **FIFO_2**.

  If **data_in** =2‟b00 then **fifo_full=full_0**

  If **data_in**=2‟b01 then **fifo_full=full_1**

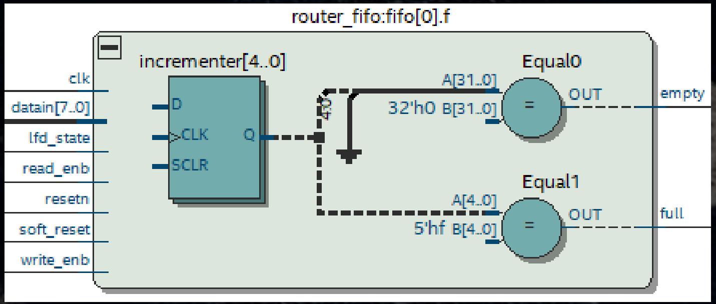
  If **data_in**=2‟b10 then **fifo_full=full_2** else **fifo_full**=0
- The signal **vld_out_x** signal is generated based on empty status of the FIFO as shown below :

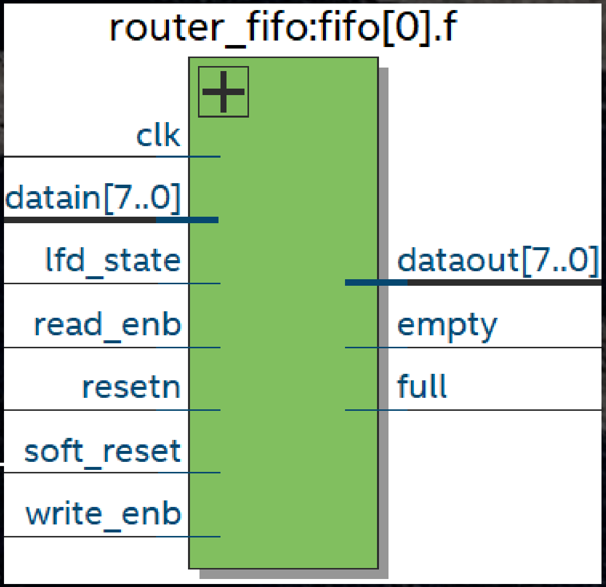
  vld_out_0=~empty_0

  vld_out_1=~empty_1

  vld_out_2=~empty_2
- The **write_enb_reg** signal is used to generate **write_enb** signal for the write operation of the selected FIFO.

FIFO Block:

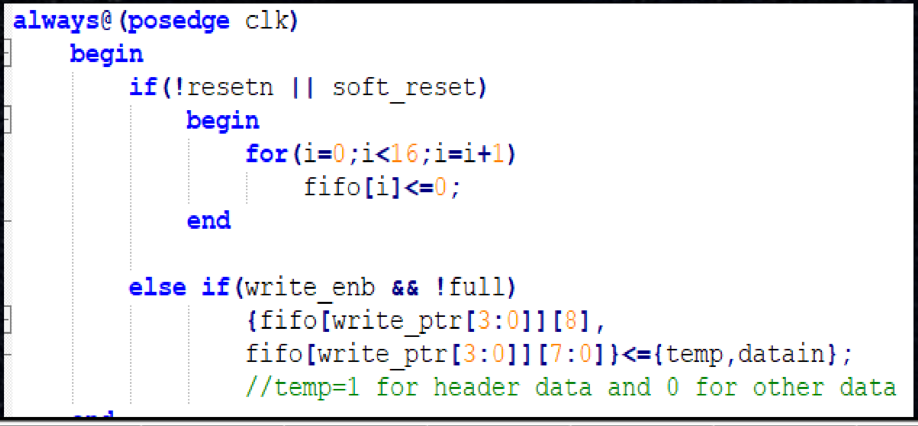




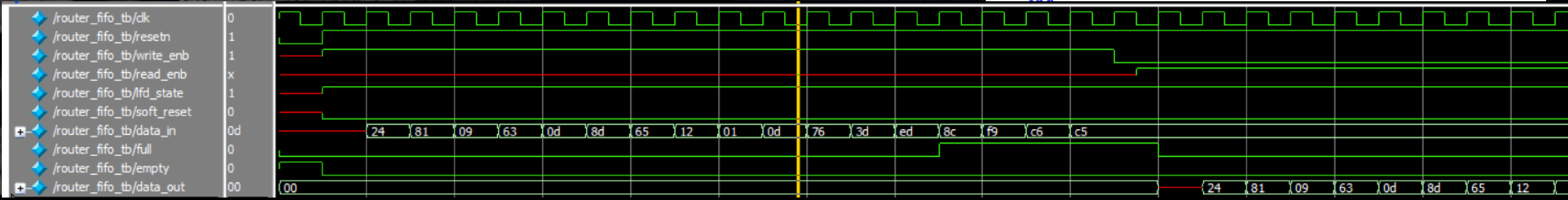
The FIFO works on the system clock and is reset with a synchronizer active low reset.The FIFO is also internally reset by an internal reset signal **soft_reset. Soft_reset** is an active high signal which is generated by the SYNCHRONIZER block during time out state of the ROUTER. If **resetn** is low then full=0, empty=1 and data_out=0.

Write_Operation: Signal **data_in** is sampled at the rising edge of the edge of the clock when **write_enb** is high. Write operation only takes place when FIFO is not full in order to avoid over_run condition.

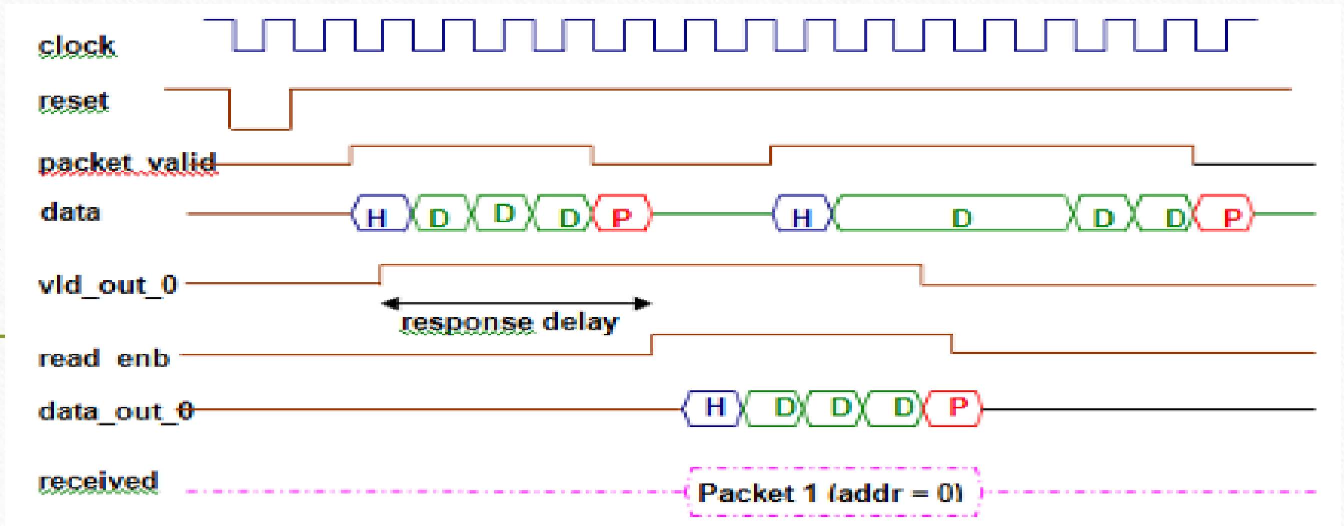
Read_Operation: **Data out** is driven to HIGH impedance state under 2 scenarios: When the fifo m/m is read completely (header+payload+parity). Under the time out condition of the Router. **Full-** FIFO status which indicates that all the locations inside FIFO have been written. **Empty-** FIFO status which indicates that all the locations of the FIFO have been read and made empty. Read and write operation can be done simultaneously.

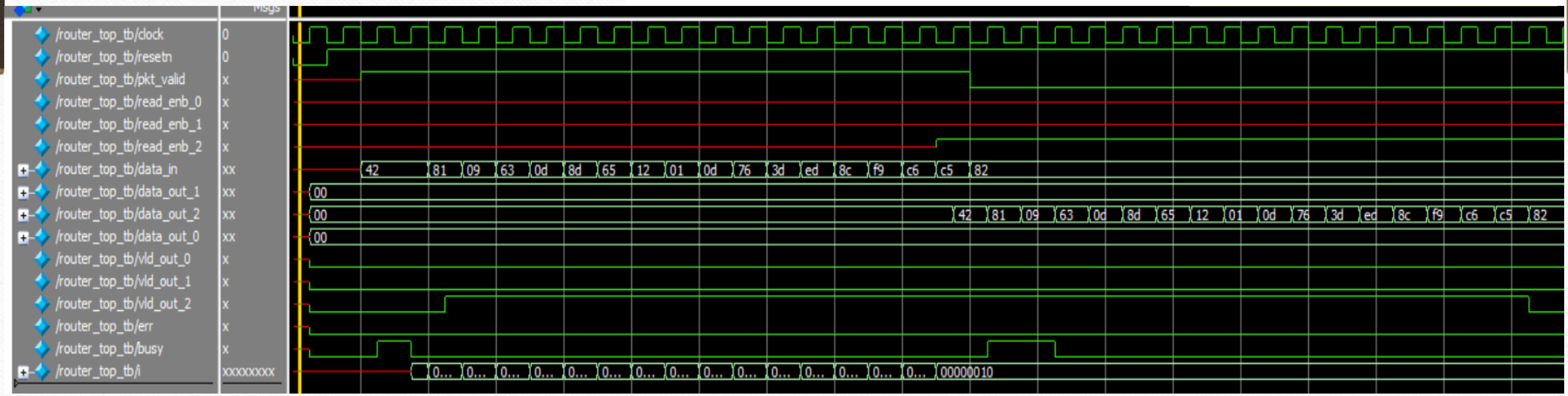


```
always@(posedge clk)
    begin
        if(!resetn || soft_reset)
            begin
                for(i=0;i<16;i=i+1)
                    fifo[i]<=0;
            end

        else if(write_enb && !full)
                {fifo[write_ptr[3:0]][8],
                fifo[write_ptr[3:0]][7:0]}<={temp,datain};
                //temp=1 for header data and 0 for other data
```

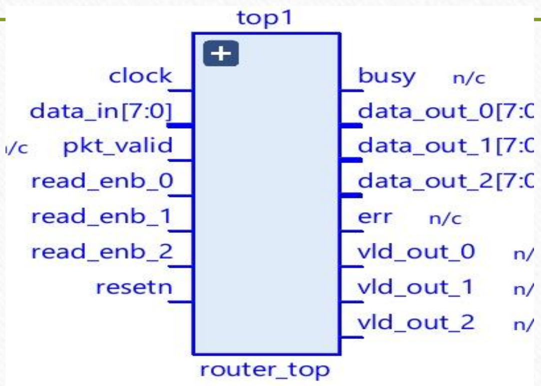


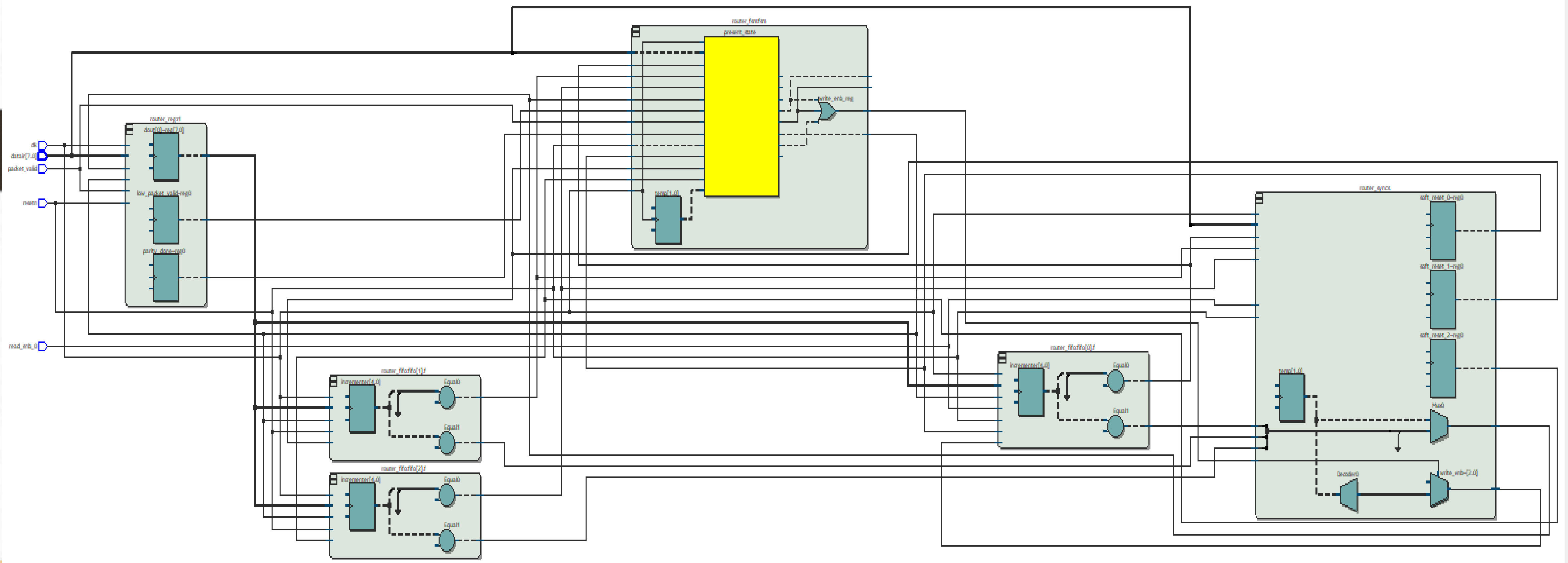
Top Module Output Protocol:




Msgs
/router_top_tb/clock 0
/router_top_tb/resetn 0
/router_top_tb/pkt_valid x
/router_top_tb/read_enb_0 x
/router_top_tb/read_enb_1 x
/router_top_tb/read_enb_2 x
/router_top_tb/data_in xx
/router_top_tb/data_out_1 xx
/router_top_tb/data_out_2 xx
/router_top_tb/data_out_0 xx
/router_top_tb/vld_out_0 x
/router_top_tb/vld_out_1 x
/router_top_tb/vld_out_2 x
/router_top_tb/err x
/router_top_tb/busy x
/router_top_tb/i xxxxxxxx
42 81 09 63 0d 8d 65 12 01 0d 76 3d ed 8c f9 c6 c5 82
00
00 42 81 09 63 0d 8d 65 12 01 0d 76 3d ed 8c f9 c6 c5 82
00
0... 0... 0... 0... 0... 0... 0... 0... 0... 0... 0... 0... 0... 0... 0... 00000010

Top Module


top1
clock
data_in[7:0]
/c
pkt_valid
read_enb_0
read_enb_1
read_enb_2
resetn
busy
n/c
data_out_0[7:0
data_out_1[7:0
data_out_2[7:0
err
n/c
vld_out_0
n/
vld_out_1
n/
vld_out_2
n/
router_top


router_fsm
present_state
write_enb_reg
router_reg1
low_packet_valid~reg0
parity_done~reg0
clk
packet_valid
resetn
read_enb_0
router_fifo:fifo[1].f
router_fifo:fifo[2].f
router_fifo:fifo[0].f
Equal0
Equal1
router_sync
Mux0
Decoder0

Thank you