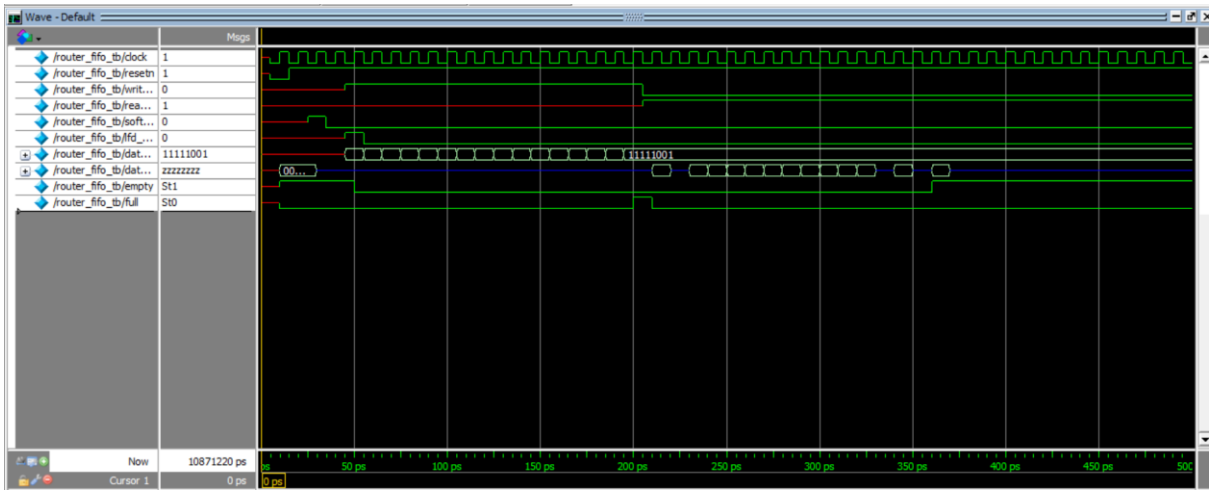
```
module router_fifo ( input clock,
input resetn, input write_enb, input read_enb, input soft_reset,
input [7:0] data_in, input lfd_state,
output reg [7:0] data_out, output empty,
output full
);
integer i;
reg [8:0] mem [0:15]; // 16x8 memory
reg [4:0] wr_pt;
reg [4:0] rd_pt;
reg [6:0] fifo_counter;
reg lfd_state_s;
always@(posedge clock)
begin
if(!resetn)
begin
fifo_counter <= 0;
end
else if(soft_reset)
 begin
fifo_counter <= 0;
end
else if(read_enb & ~empty)
begin
if(mem[rd_pt[3:0]][8] == 1'b1)
fifo_counter <= mem[rd_pt[3:0]][7:2] + 1'b1;
else if(fifo_counter != 0)
fifo_counter <= fifo_counter - 1'b1;
end
end


always@(posedge clock)
begin
if(!resetn) lfd_state_s <= 0;
else
lfd_state_s <= lfd_state;
end


//read operation
always@(posedge clock)
begin
if(!resetn)
data_out <= 8'b00000000;
else if(soft_reset)
data_out <= 8'bzzzzzzzz;
else
begin
if(fifo_counter==0 && data_out != 0)
data_out <= 8'dz;
```

```
else if(read_enb && ~empty)
data_out <= mem[rd_pt[3:0]];
end
end

//write operation
always@(posedge clock)
begin
if(!resetn)
begin
for(i = 0;i<16;i=i+1)
begin
mem[i] <= 0;
end
end
else if(soft_reset)
begin
for(i = 0;i<16;i=i+1)
begin mem[i] <= 0;
end
end
else
begin
if(write_enb && !full)
{mem[wr_pt[3:0]]}<= {lfd_state_s,data_in};
end
end

//logic for incrementing pointer
always@(posedge clock)
begin
if(!resetn)
begin rd_pt <= 5'b00000;
wr_pt <= 5'b00000;
end
else if(soft_reset)
begin
rd_pt <= 5'b00000;
wr_pt <= 5'b00000;
end

else
begin
if(!full && write_enb)
wr_pt <= wr_pt + 1;
else
wr_pt <= wr_pt;
if(!empty && read_enb)
rd_pt <= rd_pt + 1;
else
rd_pt <= rd_pt;
```

```
end
end

assign full= (wr_pt=={~rd_pt[4],rd_pt[3:0]})?1'b1:1'b0;
assign empty=(wr_pt== rd_pt)?1'b1:1'b0;

endmodule


module router_fifo_tb();
 reg clock;
reg resetn;
reg write_enb; reg read_enb; reg soft_reset; reg lfd_state;
reg [7:0] data_in;
wire [7:0] data_out; wire empty;
wire full;
parameter T = 10;


router_fifo dut (
.clock(clock),
.resetn(resetn),
.write_enb(write_enb),
.read_enb(read_enb),
.soft_reset(soft_reset),
.data_in(data_in),
.lfd_state(lfd_state),
.data_out(data_out),
.empty(empty),
.full(full)
);


always begin #(T/2);
clock=1'b0; #(T/2);
 clock =~clock;
end

task sft_dut();
begin
@(negedge clock);
soft_reset=1'b1;
@(negedge clock);
soft_reset=1'b0;
end
endtask

task reset_dut();
begin
@(negedge clock);
resetn=1'b0;
```

```
@(negedge clock);
resetn=1'b1;
end
endtask

task read(input i,input j);
begin
@(negedge clock);
write_enb=i;
read_enb=j;
end
endtask


task write;
reg[7:0]payload_data,parity,header;
 reg[5:0]payload_len;
reg[1:0]addr;
integer k;
begin
@(negedge clock);
payload_len=6'd14;
addr=2'b01;
header={payload_len,addr};
data_in=header;
lfd_state=1'b1;
write_enb=1;
for(k=0;k<payload_len;k=k+1)
begin
@(negedge clock);
lfd_state=0;
payload_data={$random}%256;
data_in=payload_data;
end
@(negedge clock);
parity={$random}%256;
data_in=parity;
end
endtask

initial
begin
reset_dut;
 sft_dut;
write;
read(1'b0,1'b1);
end

endmodule
```




```
module router_sync( input
clock,resetn,detect_add,write_enb_reg,read_enb_0,read_enb_1,read_enb_2,empty_0,empty_1,em
pty_2,full_0,full_1,full_2,
input [1:0]data_in,
output wire vld_out_0,vld_out_1,vld_out_2,
output reg [2:0]write_enb,
output reg fifo_full, soft_reset_0,soft_reset_1,soft_reset_2);

reg [4:0] timer_0, timer_1, timer_2;
reg [1:0] int_addr_reg;

//timer_0 and soft_reset_0 logic
always@(posedge clock)
begin
if(~resetn)
```

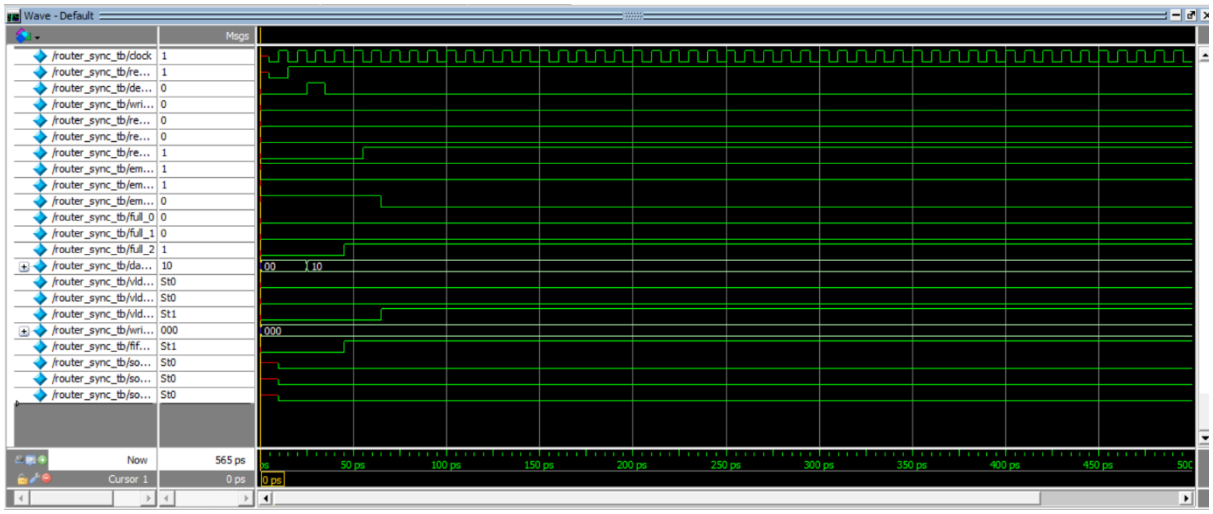
```
begin
timer_0<=0;
soft_reset_0<=0;
end
else if(vld_out_0)
begin
if(!read_enb_0)
begin
if(timer_0==5'd29)
begin
soft_reset_0<=1'b1;
timer_0<=0;
end

else
 begin
begin
soft_reset_0<=0;
timer_0<=timer_0+1'b1;
end
end
end
end
end

//timer_1 and soft_reset_1 logic
always@(posedge clock)
begin
if(~resetn)
begin
timer_1<=0;
soft_reset_1<=0;
end
else if(vld_out_1)
begin
if(!read_enb_1)
begin
if(timer_1==5'd29)
begin
soft_reset_1<=1'b1;
timer_1<=0;
end
else
begin
begin
soft_reset_1<=0;
timer_1<=timer_1+1'b1;
end
end
end
end
```

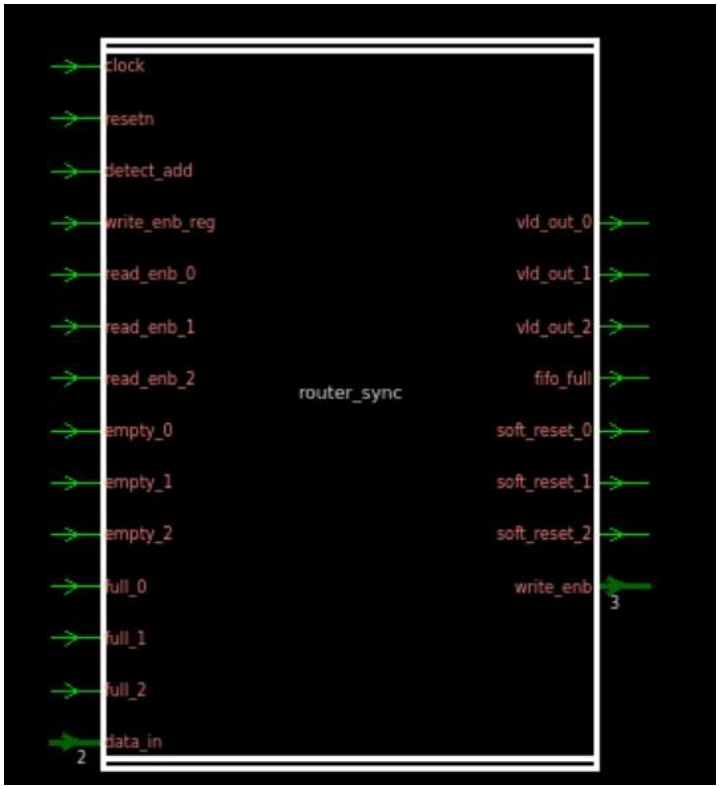
```
end

//timer_2 and soft_reset_2 logic
always@(posedge clock)
begin
if(~resetn)
begin
timer_2<=0;
soft_reset_2<=0;
end
else if(vld_out_2)
begin
if(!read_enb_2)
begin
if(timer_2==5'd29)
begin
soft_reset_2<=1'b1;
timer_2<=0;
end
else
begin
begin
soft_reset_2<=0;
timer_2<=timer_2+1'b1;
end
end
end
end
end

//int_addr_reg logic
always@(posedge clock)
begin
if (~resetn)
int_addr_reg<=0;
else if(detect_add)
int_addr_reg<=data_in;
end

//write enb logic
always@(*)
begin
write_enb = 3'b000;
if(write_enb_reg)
begin
case(int_addr_reg)
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
end

//fifo_full logic
always@(*)
begin
case(int_addr_reg)
 2'b00 : fifo_full = full_0;
2'b01 : fifo_full = full_1;
2'b10 : fifo_full = full_2;
default:fifo_full = 1'b0;
endcase
end

assign vld_out_0 = ~empty_0;
assign vld_out_1 = ~empty_1;
assign vld_out_2 = ~empty_2;
endmodule




module router_sync_tb();
reg clock;
reg resetn;
reg detect_add;
reg write_enb_reg;
reg read_enb_0;
reg read_enb_1;
reg read_enb_2;
reg empty_0;
reg empty_1;

reg empty_2;
reg full_0;
reg full_1;
reg full_2;
reg [1:0] data_in;
wire vld_out_0;
wire vld_out_1;
wire vld_out_2;
wire [2:0] write_enb;
wire fifo_full;
wire soft_reset_0;
wire soft_reset_1;
wire soft_reset_2;

// Instantiate the router_sync
router_sync u_router_sync (
```

```
.clock(clock),
.resetn(resetn),
.detect_add(detect_add),
.write_enb_reg(write_enb_reg),
.read_enb_0(read_enb_0),
.read_enb_1(read_enb_1),
.read_enb_2(read_enb_2),
.empty_0(empty_0),
.empty_1(empty_1),
.empty_2(empty_2),
.full_0(full_0),
.full_1(full_1),
.full_2(full_2),
.data_in(data_in),
.vld_out_0(vld_out_0),
.vld_out_1(vld_out_1),
.vld_out_2(vld_out_2),
.write_enb(write_enb),
.fifo_full(fifo_full),
.soft_reset_0(soft_reset_0),
.soft_reset_1(soft_reset_1),
.soft_reset_2(soft_reset_2)
);
parameter T = 10;

always begin #(T/2);
clock=1'b0; #(T/2);
clock =~clock; end

task reset_dut();
begin

@(negedge clock);
resetn=1'b0;
@(negedge clock);
resetn=1'b1;
end
endtask

//Synchronizer TB
task initialze;
begin
detect_add = 1'b0; data_in = 2'b00;
write_enb_reg = 1'b0;
{empty_0,empty_1,empty_2} = 3'b111;
{full_0,full_1,full_2} = 3'b000;
{read_enb_0,read_enb_1,read_enb_2} = 3'b000;
end
endtask

task addr(input [1:0]m);
```

```
begin
@(negedge clock)
detect_add = 1'b1;
data_in = m;
@(negedge clock)
detect_add = 1'b0;
end
endtask

task write;
begin
@(negedge clock)
write_enb_reg = 1'b1;
@(negedge clock)
write_enb_reg = 1'b0; end
endtask

task stimulus;
begin
@(negedge clock)
{full_0,full_1,full_2} = 3'b001;
@(negedge clock)
{read_enb_0,read_enb_1,read_enb_2} = 3'b001;
@(negedge clock)
{empty_0,empty_1,empty_2} = 3'b110;
end
endtask

initial
begin

initialze;
reset_dut;
addr(2'b10);
stimulus;
#500 $finish;
end

endmodule
```






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

reg [7:0] Header_byte;

reg [7:0] fifo_full_state_byte;
reg [7:0] Packet_parity;
```

```
reg [7:0] Internal_parity;

// dout logic
always @(posedge clock or negedge resetn)
begin if (~resetn)
begin
dout <= 8'b0;
end
else if (lfd_state)
begin
dout <= Header_byte;
end
else if (ld_state && ~fifo_full)
 begin dout <= data_in;
end
else if (laf_state)
begin
dout <= fifo_full_state_byte;
end
end

// Header_byte and fifo_full_state_byte
always @(posedge clock or negedge resetn)
begin
if (~resetn)
begin
Header_byte <= 8'b0;
fifo_full_state_byte <= 8'b0;
end
else
begin
if (pkt_valid && detect_add)
begin
Header_byte <= data_in;
end
else if (ld_state && fifo_full)
begin
fifo_full_state_byte <= data_in;
end
end
 end

// parity_done logic
always @(posedge clock or negedge resetn)
begin
if (~resetn)
 begin
parity_done <= 1'b0;
end
else
begin
```

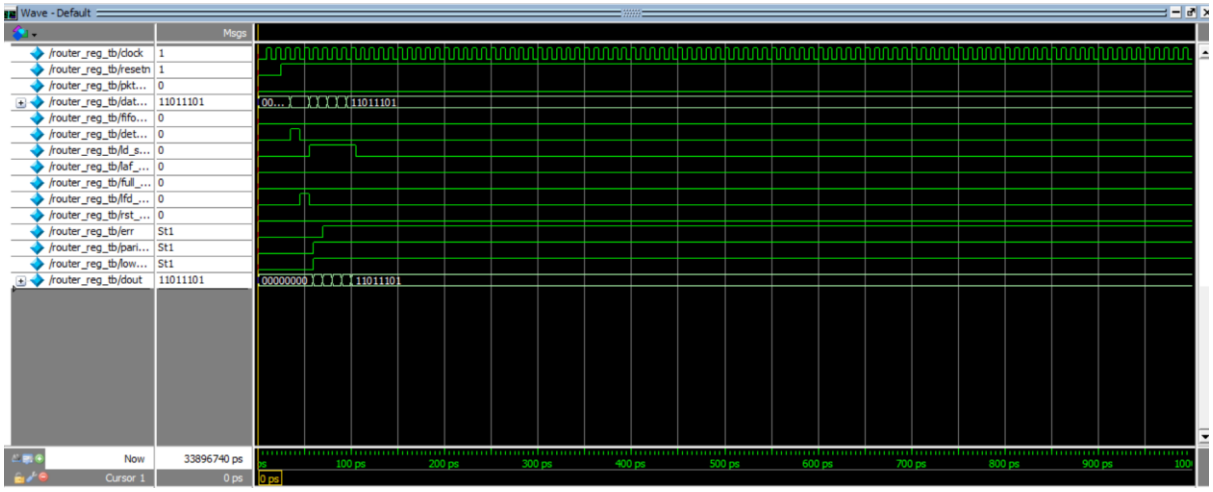
```
if (ld_state && ~pkt_valid && ~fifo_full)
begin
parity_done <= 1'b1;
end
else if (laf_state && ~parity_done && low_pkt_valid)
begin parity_done <= 1'b1;
end
else
begin
if (detect_add)
begin parity_done <= 1'b0;
end
end
end
end

// low_pkt_valid logic
always @(posedge clock or negedge resetn)
begin
if (~resetn)
begin
low_pkt_valid <= 1'b0;
end
else
begin
if (rst_int_reg)
begin
low_pkt_valid <= 1'b0;
end
else if (~pkt_valid && ld_state)
begin low_pkt_valid <= 1'b1;
end
end
end

// Packet_parity logic
always @(posedge clock or negedge resetn)
begin
if (~resetn)
begin
Packet_parity <= 8'b0;
end
else if ((ld_state && ~pkt_valid && ~fifo_full) || (laf_state && low_pkt_valid && ~parity_done))
begin
Packet_parity <= data_in;
end
else if (~pkt_valid && rst_int_reg)
begin Packet_parity <= 8'b0;
end
else
begin
```

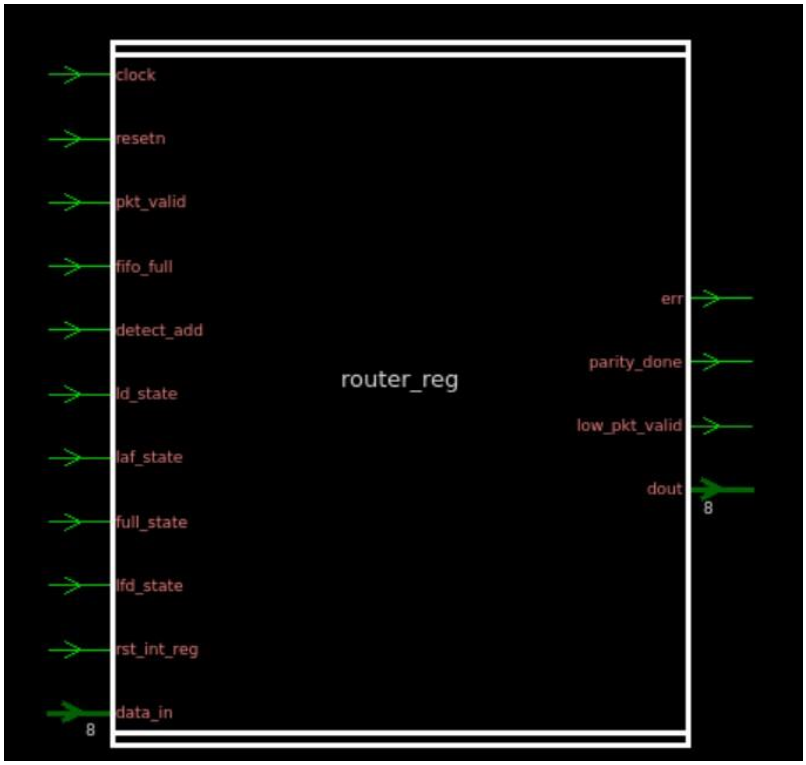
```
if (detect_add)
begin Packet_parity <= 8'b0;
end
end
end

// internal_parity
always @(posedge clock or negedge resetn)
begin
if (~resetn)
begin
Internal_parity <= 8'b0;
end
else if (detect_add)
begin Internal_parity <= 8'b0;
end
else if (lfd_state)
begin
Internal_parity <= Header_byte;
end
else if (ld_state && pkt_valid && ~full_state)
begin Internal_parity <= Internal_parity ^ data_in;
end
else if (~pkt_valid && rst_int_reg)
begin Internal_parity <= 8'b0;
end
end

// error logic
always @(posedge clock or negedge resetn)
begin
if (~resetn)
begin
err <= 1'b0;
end
else
begin
if (parity_done == 1'b1 && (Internal_parity != Packet_parity))
begin
err <= 1'b1;
end
 else
begin
err <= 1'b0;
end

end
end

endmodule
```

```
module router_reg_tb();
reg clock;
reg resetn;
reg pkt_valid;
reg [7:0] data_in;
reg fifo_full;
reg detect_add;
reg ld_state;
reg laf_state;
reg full_state;
reg lfd_state;
reg rst_int_reg;

wire err;
wire parity_done;
wire low_pkt_valid;
wire [7:0] dout;

parameter T = 10;
router_reg dut (
.clock(clock),
.resetn(resetn),
.pkt_valid(pkt_valid),
.data_in(data_in),
.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),
.err(err),
.parity_done(parity_done),
.low_pkt_valid(low_pkt_valid),
.dout(dout)
);

always begin #(T/2);
clock=1'b0; #(T/2);
clock =~clock;

end


task reset_dut();
begin
@(negedge clock);
resetn=1'b0;
@(negedge clock);
resetn=1'b1;
end
```

```
endtask

task packet_generation;
reg [7:0]payload_data,parity,header;
reg [5:0]payload_len;
reg [1:0]addr;
integer i;
begin
@(negedge clock)
payload_len=6'd4;
addr=2'b10;//valid packet pkt_valid=1;
detect_add=1;
header={payload_len,addr}; parity=header;
data_in=header;
@(negedge clock)
detect_add=0;
lfd_state=1;
full_state=0;
fifo_full=0;
laf_state=0;
for(i=0;i<payload_len;i=i+1)
begin
@(negedge clock)
lfd_state=0;
ld_state=1;
payload_data={$random}%256;
data_in=payload_data;
parity=parity^data_in;
end
@(negedge clock)
pkt_valid=0;
data_in=parity;
@(negedge clock)
ld_state=0;
end
endtask

initial
begin
clock = 0;
resetn = 0;
pkt_valid = 0;
data_in = 0;
fifo_full = 0;
detect_add = 0;
ld_state = 0;
laf_state = 0;
full_state = 0;
lfd_state = 0;
rst_int_reg = 0; reset_dut();
```

```
// Start packet generation task
packet_generation();
end
endmodule
```





```
module router_fsm(input clock,resetn,pkt_valid,
input [1:0] data_in, input
fifo_full,fifo_empty_0,fifo_empty_1,fifo_empty_2,soft_reset_0,soft_reset_1,soft_reset_2,parity_done, low_pkt_valid,
output
write_enb_reg,detect_add,ld_state,laf_state,lfd_state,full_state,rst_int_reg,busy);

parameter DECODE_ADDRESS = 3'b000, LOAD_FIRST_DATA = 3'b001, LOAD_DATA = 3'b010, FIFO_FULL_STATE = 3'b011, LOAD_AFTER_FULL = 3'b100, LOAD_PARITY = 3'b101, CHECK_PARITY_ERROR = 3'b110, WAIT_TILL_EMPTY = 3'b111;
reg [2:0] present_state, next_state;
reg [2:0] addr;
//present_state logic
always@(posedge clock)
begin
if(~resetn)
present_state<= DECODE_ADDRESS;
else
if((soft_reset_0 && data_in==2'b00) || (soft_reset_1 && data_in==2'b01) || (soft_reset_2 && data_in==2'b10))
present_state <= DECODE_ADDRESS;
else
present_state <= next_state;
end


//internal variable addr logic
always@(posedge clock)
begin
if(~resetn)
addr<=0;
else if((soft_reset_0 && data_in==2'b00) || (soft_reset_1 && data_in==2'b01) || (soft_reset_2 && data_in==2'b10))
addr <= 0;
else if(detect_add)
 addr <= data_in;
end

//next_state logic
always@(*)
begin
next_state = present_state;
 begin
case(present_state)

DECODE_ADDRESS : if((pkt_valid && (data_in==0) && fifo_empty_0)||(pkt_valid && (data_in==1) && fifo_empty_1) ||(pkt_valid && (data_in==2) && fifo_empty_2))
next_state=LOAD_FIRST_DATA;
else if((pkt_valid && (data_in==0) && ~fifo_empty_0) ||(pkt_valid && (data_in==1) && ~fifo_empty_1) ||(pkt_valid && (data_in==2) && ~fifo_empty_2))
next_state=WAIT_TILL_EMPTY;
```

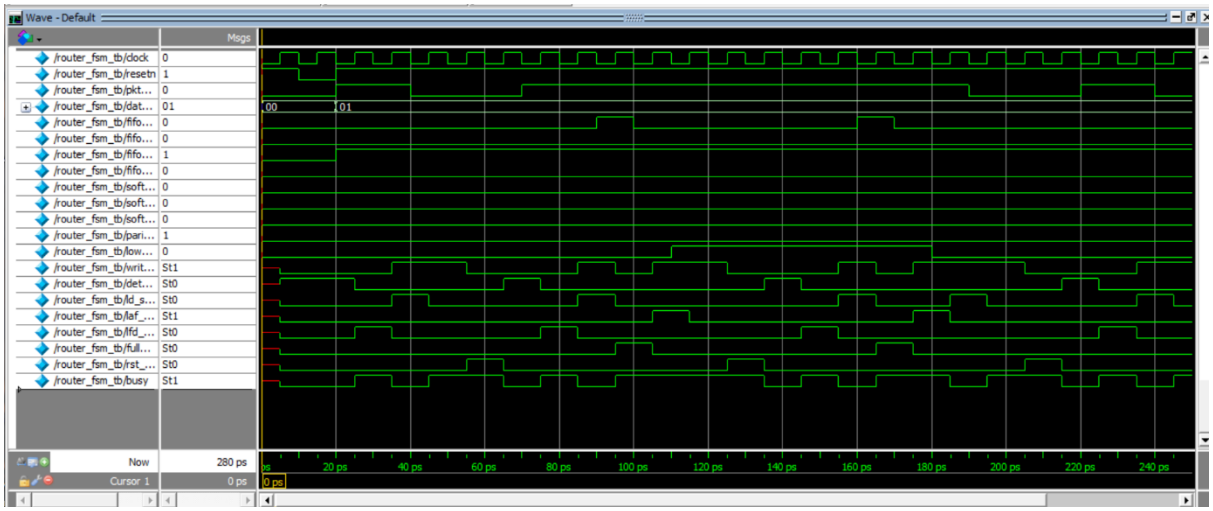
```
else next_state = DECODE_ADDRESS;
LOAD_FIRST_DATA : next_state = LOAD_DATA;
LOAD_DATA : if(fifo_full)
next_state = FIFO_FULL_STATE;
else if(!fifo_full && !pkt_valid) next_state = LOAD_PARITY;
else next_state = LOAD_DATA;
default : next_state = DECODE_ADDRESS;
LOAD_PARITY : next_state = CHECK_PARITY_ERROR;
CHECK_PARITY_ERROR : if(!fifo_full)
next_state = DECODE_ADDRESS;
else
next_state = FIFO_FULL_STATE;
 FIFO_FULL_STATE : if(fifo_full) next_state = FIFO_FULL_STATE;
else
next_state = LOAD_AFTER_FULL;
LOAD_AFTER_FULL : if(!parity_done && !low_pkt_valid) next_state = LOAD_DATA;
else if(!parity_done && low_pkt_valid)
next_state=LOAD_PARITY;
else next_state = DECODE_ADDRESS;
WAIT_TILL_EMPTY : if((fifo_empty_0 && (addr == 0)) || (fifo_empty_1 && (addr ==1)) ||
(fifo_empty_2 && (addr == 2)))
next_state = LOAD_FIRST_DATA;
else next_state = WAIT_TILL_EMPTY;
endcase
end
 end
//output signals
assign detect_add = (present_state == DECODE_ADDRESS) ? 1'b1 : 1'b0;
assign lfd_state =(present_state==LOAD_FIRST_DATA) ? 1'b1 : 1'b0;
assign ld_state=(present_state==LOAD_DATA) ? 1'b1 : 1'b0;
assign full_state=(present_state==FIFO_FULL_STATE) ? 1'b1 : 1'b0;
assign laf_state=(present_state==LOAD_AFTER_FULL) ? 1'b1 : 1'b0;
assign rst_int_reg=(present_state==CHECK_PARITY_ERROR) ? 1'b1 : 1'b0;
assign
write_enb_reg=((present_state==LOAD_DATA)||(present_state==LOAD_AFTER_FULL)||(present_stat
e==LOAD_PARITY)) ? 1'b1 : 1'b0;
assign
busy=((present_state==LOAD_FIRST_DATA)||(present_state==FIFO_FULL_STATE)||(present_state==L
OAD_AFTER_FULL)||(present_state==LOAD_PARITY)||(present_state==CHECK_PARITY_ERROR)||(pr
esent_state==WAIT_TILL_EMPTY)) ? 1'b1 : 1'b0;

endmodule




module router_fsm_tb();
reg clock;
reg resetn;
reg pkt_valid;
reg [1:0] data_in;
```

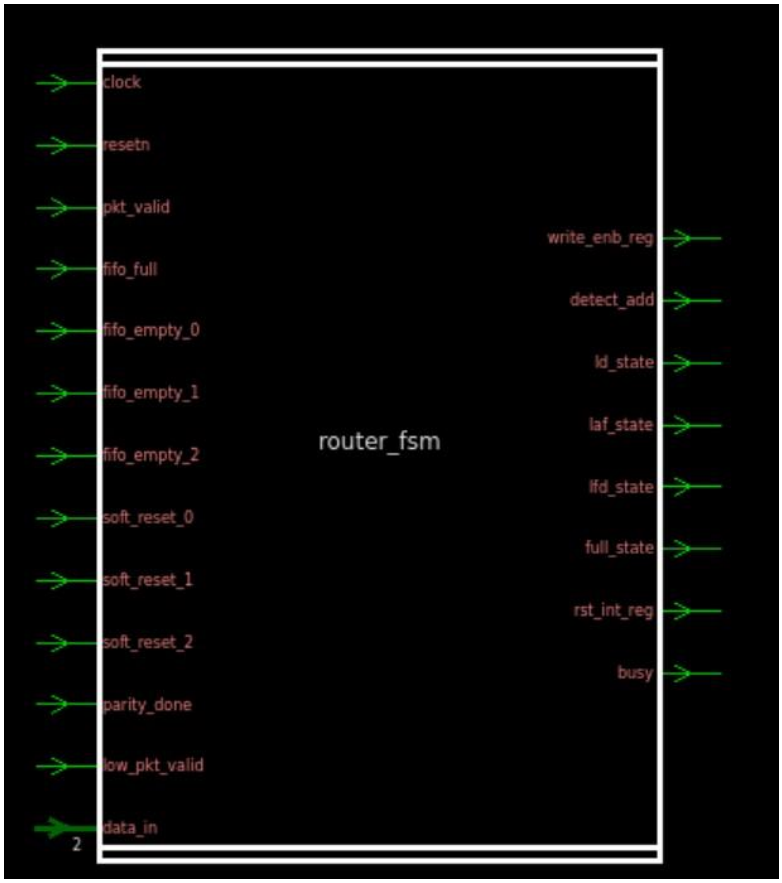
```
reg fifo_full;
reg fifo_empty_0;
reg fifo_empty_1;
reg fifo_empty_2;
reg soft_reset_0;
reg soft_reset_1;
reg soft_reset_2;
reg parity_done;
reg low_pkt_valid;

wire write_enb_reg;
wire detect_add;
wire ld_state;
wire laf_state;
wire lfd_state;
wire full_state;
wire rst_int_reg;
wire busy;

// Instantiate the router_fsm module
router_fsm uut (
.clock(clock),
.resetn(resetn),
.pkt_valid(pkt_valid),
.data_in(data_in),
.fifo_full(fifo_full),
.fifo_empty_0(fifo_empty_0),
.fifo_empty_1(fifo_empty_1),
.fifo_empty_2(fifo_empty_2),
.soft_reset_0(soft_reset_0),
.soft_reset_1(soft_reset_1),
.soft_reset_2(soft_reset_2),
.parity_done(parity_done),
.low_pkt_valid(low_pkt_valid),
.write_enb_reg(write_enb_reg),
.detect_add(detect_add),
.ld_state(ld_state),
.laf_state(laf_state),
.lfd_state(lfd_state),
.full_state(full_state),
.rst_int_reg(rst_int_reg),
.busy(busy)

);

// Clock generation
always
begin
#5 clock = ~clock;
end
```

```
// Initialize signals
initial
begin
clock = 0;
resetn = 0;
pkt_valid = 0; data_in = 2'b00; fifo_full = 0;
fifo_empty_0 = 0;
fifo_empty_1 = 0;
fifo_empty_2 = 0;
soft_reset_0 = 0;
soft_reset_1 = 0;
soft_reset_2 = 0;
parity_done = 0;
low_pkt_valid = 0;

// Reset router_fsm
resetn = 1;
#10 resetn = 0;
#10 resetn = 1;

// Execute the test cases using tasks
t1();
t2();
t3();
t4();

// Add any additional test cases or stimulus here
// ...

// End simulation
$finish;
end

// Task t1
task t1();
begin
@(negedge clock)
pkt_valid = 1'b1;
data_in = 2'b01;
fifo_empty_1 = 1'b1;
 @(negedge clock)
@(negedge clock)
fifo_full = 1'b0;

pkt_valid = 1'b0;
@(negedge clock)
@(negedge clock)
fifo_full = 1'b0;
end
endtask
```

```
task t2();
begin
@(negedge clock)
pkt_valid = 1'b1;
data_in = 2'b01;
fifo_empty_1 = 1'b1;
@(negedge clock)
@(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)
@(negedge clock)
fifo_full = 1'b0;
end
endtask


task t3();
begin
@(negedge clock)
pkt_valid = 1'b1;
data_in = 2'b01;
fifo_empty_1 = 1'b1;
@(negedge clock)
@(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;
 @(negedge clock)
@(negedge clock)
fifo_full = 1'b0;
end
endtask

task t4();
begin
@(negedge clock)
pkt_valid = 1'b1;
data_in = 2'b01;
fifo_empty_1 = 1'b1;
```

```
@(negedge clock)
@(negedge clock)
fifo_full = 1'b0;
pkt_valid = 1'b0;
@(negedge clock)
@(negedge clock)
fifo_full = 1'b1;
@(negedge clock)
fifo_full = 1'b0;
@(negedge clock)
parity_done = 1'b1;
end
endtask
endmodule
```



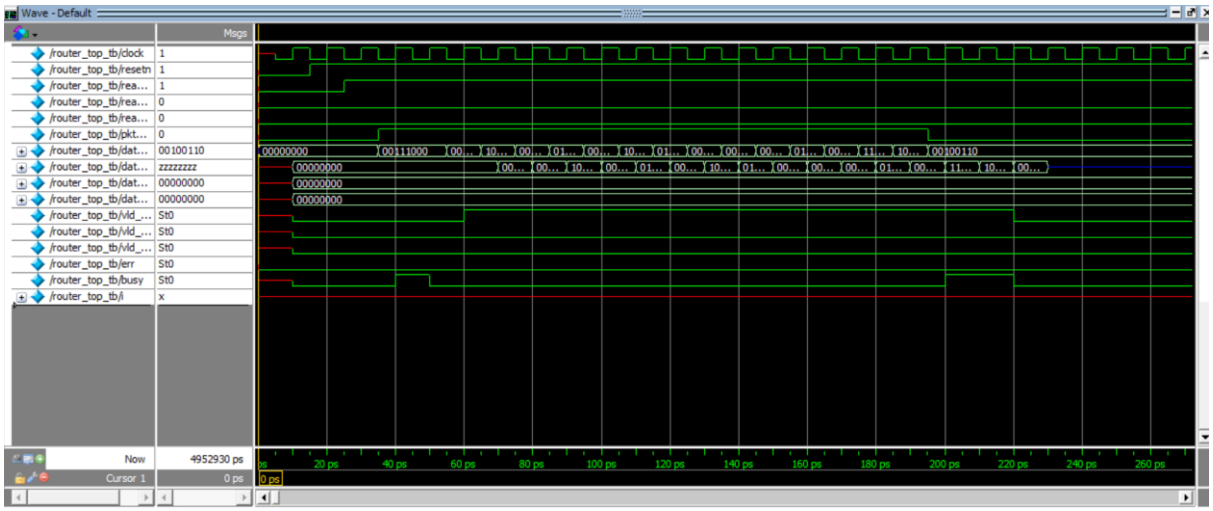


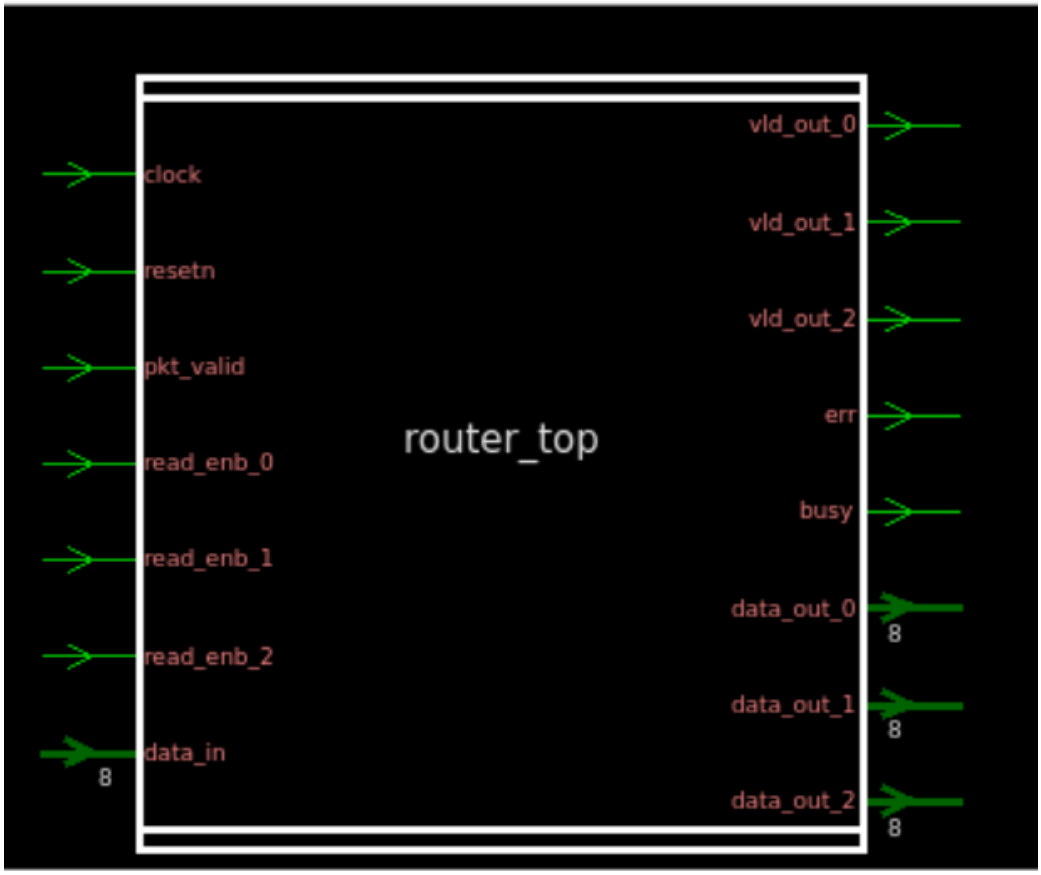

```
module router_top(input clock, resetn, pkt_valid, read_enb_0, read_enb_1, read_enb_2,
input [7:0]data_in,
output vld_out_0, vld_out_1, vld_out_2, err, busy,
output [7:0]data_out_0, data_out_1, data_out_2);

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

router_fifo FIFO_0(.clock(clock), .resetn(resetn), .soft_reset(soft_reset_0),
.lfd_state(lfd_state), .write_enb(write_enb[0]), .data_in(dout), .read_enb(read_enb_0),
.full(full_0), .empty(empty_0), .data_out(data_out_0));

router_fifo FIFO_1(.clock(clock), .resetn(resetn), .soft_reset(soft_reset_1),
.lfd_state(lfd_state), .write_enb(write_enb[1]), .data_in(dout), .read_enb(read_enb_1),
.full(full_1), .empty(empty_1), .data_out(data_out_1));

router_fifo FIFO_2(.clock(clock), .resetn(resetn), .soft_reset(soft_reset_2),
.lfd_state(lfd_state), .write_enb(write_enb[2]), .data_in(dout), .read_enb(read_enb_2),
.full(full_2), .empty(empty_2), .data_out(data_out_2));

router_sync SYNC(.clock(clock), .resetn(resetn), .data_in(data_in[1:0]), .detect_add(detect_add),
.full_0(full_0), .full_1(full_1), .full_2(full_2), .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(empty_0), .empty_1(empty_1), .empty_2(empty_2),
```

```
.vld_out_0(vld_out_0), .vld_out_1(vld_out_1), .vld_out_2(vld_out_2),
.soft_reset_0(soft_reset_0), .soft_reset_1(soft_reset_1),
.soft_reset_2(soft_reset_2), .write_enb(write_enb), .fifo_full(fifo_full));

router_fsm FSM(.clock(clock), .resetn(resetn), .pkt_valid(pkt_valid),
.data_in(data_in[1:0]), .soft_reset_0(soft_reset_0),
.soft_reset_1(soft_reset_1), .soft_reset_2(soft_reset_2),
.fifo_full(fifo_full), .fifo_empty_0(empty_0), .fifo_empty_1(empty_1),

.fifo_empty_2(empty_2),

.rst_int_reg(rst_int_reg),

.ld_state(ld_state),


.parity_done(parity_done), .low_pkt_valid(low_pkt_valid), .busy(busy),

.full_state(full_state), .lfd_state(lfd_state), .laf_state(laf_state),

.detect_add(detect_add), .write_enb_reg(write_enb_reg));


router_reg REG(.clock(clock), .resetn(resetn), .pkt_valid(pkt_valid), .data_in(data_in),
.dout(dout), .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), .err(err),
.parity_done(parity_done), .low_pkt_valid(low_pkt_valid));

endmodule



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 vld_out_0, vld_out_1, vld_out_2, err, busy;
integer i;

router_top DUT(.clock(clock),
.resetn(resetn),
.read_enb_0(read_enb_0),
.read_enb_1(read_enb_1),
.read_enb_2(read_enb_2),
.pkt_valid(pkt_valid),
.data_in(data_in),
.data_out_0(data_out_0),
.data_out_1(data_out_1),
.data_out_2(data_out_2),
```

```
.vld_out_0(vld_out_0),
.vld_out_1(vld_out_1),
.vld_out_2(vld_out_2),
.err(err),
.busy(busy) );
parameter Tp = 10;
always
begin
#(Tp/2) clock = 1'b0;
 #(Tp/2) clock = 1'b1;
end

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

task initialize;
begin
{read_enb_0,read_enb_1,read_enb_2,pkt_valid,data_in}=0;
resetn=0;
end
endtask

task pkt_gen_14;
reg [7:0]payload_data,parity,header;
reg [5:0]payload_len;
reg [1:0]addr;
integer i;
begin
@(negedge clock)
wait(~busy)
@(negedge clock)
payload_len=6'd14;
addr=2'b00;
header={payload_len,addr};
parity=0;
data_in=header;
pkt_valid=1;
parity=parity^header;
 @(negedge clock)
wait(~busy)
for(i=0;i<payload_len;i=i+1)
begin
@(negedge clock)
wait(~busy)
payload_data={$random}%256;
```

```
data_in=payload_data;
parity=parity^data_in;
end
@(negedge clock)
pkt_valid=0;
data_in=parity;
end
endtask

initial
begin
initialize;
 rstn;
 fork
pkt_gen_14;
begin
repeat(2)
@(negedge clock)
 read_enb_0 = 1'b1;
end
join
end
endmodule
```




router_top
clock
resetn
pkt_valid
read_enb_0
read_enb_1
read_enb_2
data_in
8
vld_out_0
vld_out_1
vld_out_2
err
busy
data_out_0
8
data_out_1
8
data_out_2
8