K VINIL KUMAR

vinilk48@gmail.com

MS/23-24/2864

BPD21

FIFO_RTL(SUB_BLOCK_1)

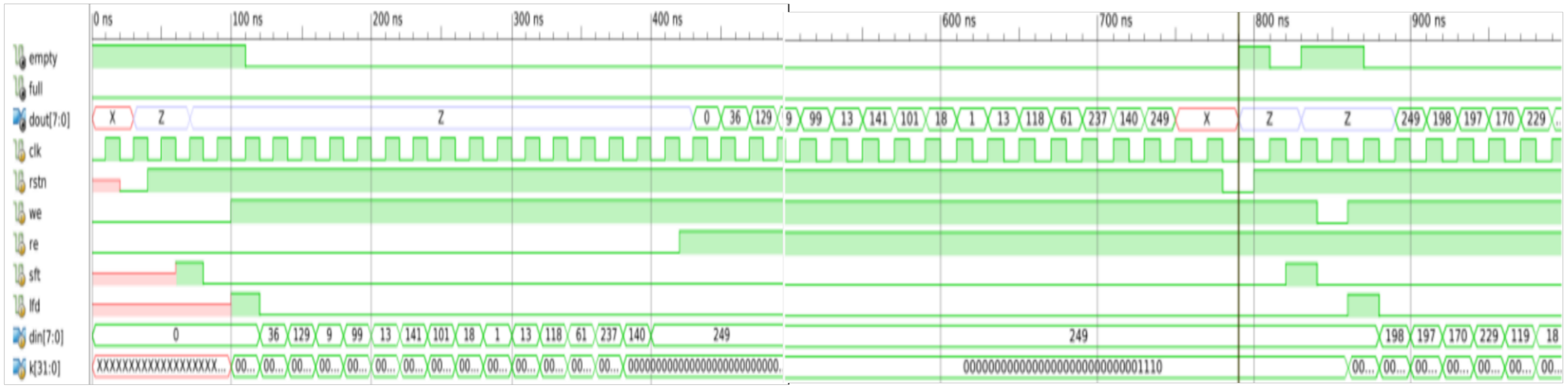
```
module fifo(input clock, resetn, write_en, read_en, soft_reset, lfd_state,
input [7:0] data_in,
output empty, full,
output reg [7:0] data_out);

reg [7:0] wptr = 4'b0;
reg [7:0] rptr = 4'b0;
reg [8:0]mem[15:0];
integer i;
reg [7:0] temp_variable;

assign empty = (rptr == wptr)? 1'b1: 1'b0;
assign full = (wptr == 8'b1 && rptr == 8'b1)? 1'b1: 1'b0;

always @ (posedge clock) begin
if(!resetn)begin
for(i = 0; i<16 ; i= i+1)
mem[i] <= 8'b0;
wptr <= 8'b0;
end
else begin
if(soft_reset)begin
for(i = 0; i<16 ; i= i+1)
mem[i] <= 8'b0;
wptr <= 8'b0;
end
else begin
if(write_en == 1'b1 && full == 1'b0)begin
mem[wptr] <= {lfd_state,data_in};
wptr <= wptr + 1;
end
else
wptr <= wptr;
end
end

end

always @ (posedge clock) begin
if(!resetn)begin
data_out <= 1'bz;
rptr <= 8'b0;
end
else begin
if(soft_reset)begin
```

```
data_out <= 8'bz;
rptr <= 8'b0;

end
else begin
if(read_en == 1'b1 && empty == 1'b0)begin
data_out <= mem[rptr];
rptr <= rptr + 1;
end
// temp_variable <= temp_variable -1;
else if(!temp_variable)begin
data_out <= 8'bz;
rptr <= 8'b0;
end
else
rptr <= rptr;
end
end
end

always @(posedge clock)begin
if(lfd_state)begin
temp_variable <= 1+ mem[wptr][7:2];
end
else begin
temp_variable <= temp_variable -1;
end
end

endmodule
```

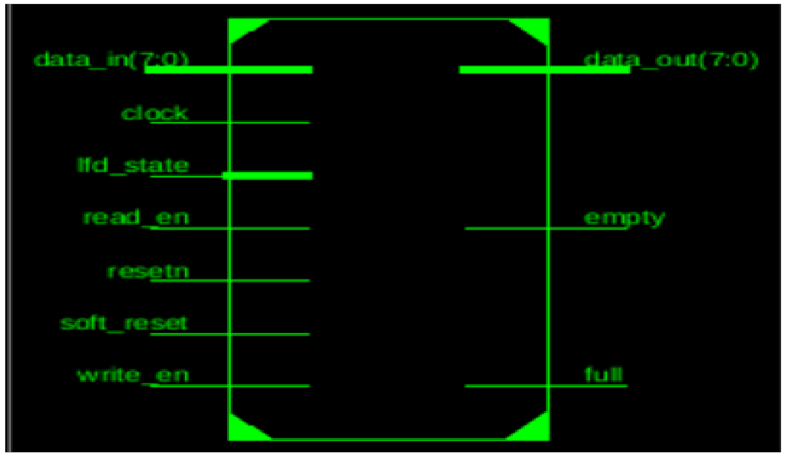
FIFO_TB(SUB_BLOCK_1)

```
module fifo_tb();

reg clk, rstn, we, re, sft, lfd;
reg [7:0] din;
wire empty, full;
wire [7:0] dout;
integer k ;

fifo UUT(clk,rstn,we,re,sft,lfd,din,empty,full,dout);

initial begin
clk = 1'b0;
forever #10 clk = ~clk;
end

task start;
begin
din = 8'b0;
we  = 1'b0;
re  = 1'b0;
end
endtask

task RESET;
begin
@ (negedge clk) rstn = 1'b0;
@ (negedge clk) rstn = 1'b1;
end
endtask

task SRESET;
begin
@ (negedge clk) sft = 1'b1;
@ (negedge clk) sft = 1'b0;
end
endtask

task write;
reg [7:0] PLD, parity, header;
reg [5:0] PLD_len;
reg [1:0] addr;
begin
@ (negedge clk)
PLD_len = 6'd14;
```

```
addr = 2'd1;
header = {PLD_len, addr};
lfd = 1'b1;
we = 1'b1;
for( k = 0; k<PLD_len ; k = k+1) begin
@(negedge clk)
lfd = 1'b0;
PLD = {$random}%256;
din = PLD;
end
@(negedge clk)
parity = $random%256;
din = parity;
end
endtask

task read;
begin
@(negedge clk)
re = 1'b1;
end
endtask

initial begin
start;
RESET;
SRESET;
write;
read;
#350
RESET;
SRESET;
we = 1'b0;
fork
write;
read;
join
#1000 $finish;
end

endmodule
```



0 ns
100 ns
200 ns
300 ns
400 ns
600 ns
700 ns
800 ns
900 ns
empty
full
dout[7:0]
clk
rstn
we
re
sft
lfd
din[7:0]
k[31:0]



data_in(7:0)
clock
lfd_state
read_en
resetn
soft_reset
write_en
data_out(7:0)
empty
full

SYNCRONISER_RTL(SUB_BLOCK_2)

```
module sync( input
clk,resetn,detect_add,write_enb_reg,read_enb_0,read_enb_1,read_enb_2,empty_0,empty_1,empty
_2,full_0,full_1,full_2,
input [1:0]datain,
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 [1:0]temp;
reg [4:0]count0,count1,count2;

always@(posedge clk)
begin
if(!resetn)
temp <= 2'd0;
else if(detect_add)
temp<=datain;
end

always@(*)
begin
case(temp)
2'b00: fifo_full=full_0;
2'b01: fifo_full=full_1;
2'b10: fifo_full=full_2;
default fifo_full=0;
endcase
end

always@(*)
begin
if(write_enb_reg)
begin
case(temp)
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

assign vld_out_0 = !empty_0;
assign vld_out_1 = !empty_1;
assign vld_out_2 = !empty_2;

always@(posedge clk)
begin
if(!resetn)
count0<=5'b0;
else if(vld_out_0)
begin
if(!read_enb_0)
```

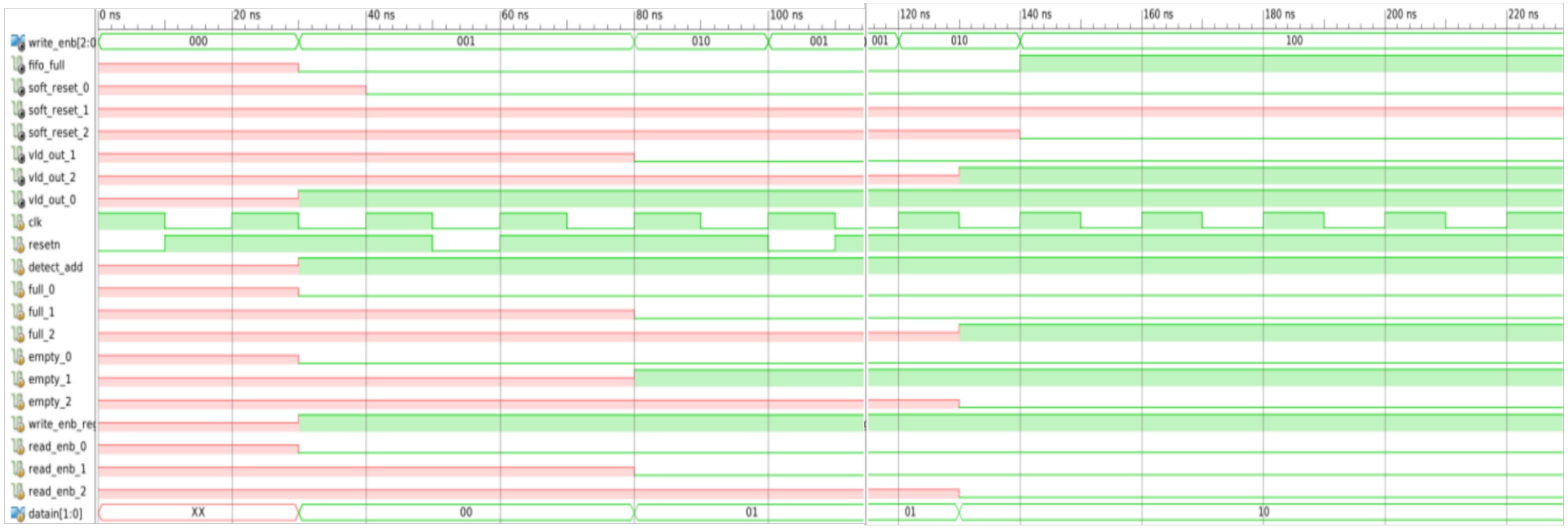
```
begin
if(count0==5'b11110)
begin
soft_reset_0<=1'b1;
count0<=1'b0;
end
else
begin
count0<=count0+1'b1;
soft_reset_0<=1'b0;
end
end
else count0<=5'd0;
end
else count0<=5'd0;
end

always@(posedge clk)
begin
if(!resetn)
count1<=5'b0;
else if(vld_out_1)
begin
if(!read_enb_1)
begin
if(count1==5'b11110)
begin
soft_reset_1<=1'b1;
count1<=1'b0;
end
else
begin
count1<=count1+1'b1;
soft_reset_1<=1'b0;
end
end
else count1<=5'd0;
end
else count1<=5'd0;
end

always@(posedge clk)
begin
if(!resetn)
count2<=5'b0;
else if(vld_out_2)
begin
if(!read_enb_2)
begin
if(count2==5'b11110)
begin
soft_reset_2<=1'b1;
count2<=1'b0;
end
else
begin
count2<=count2+1'b1;
```

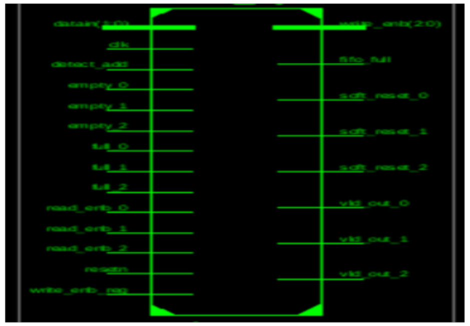
```
soft_reset_2<=1'b0;
end
end
else count2<=5'd0;
end
else count2<=5'd0;
end


endmodule
```

SYNCRONISER_TB(SUB_BLOCK_2)

```
module sync_tb();
reg clk, resetn, detect_add, full_0, full_1, full_2, empty_0, empty_1, empty_2, write_enb_reg,
read_enb_0, read_enb_1, read_enb_2;

reg [1:0]datain;
wire [2:0]write_enb;
wire fifo_full, soft_reset_0, soft_reset_1, soft_reset_2, vld_out_1, vld_out_2, vld_out_0;

sync DUT(.clk(clk),
                                        .resetn(resetn),
                                        .detect_add(detect_add),
                                        .full_0(full_0),
                                        .full_1(full_1),
                                        .full_2(full_2),
                                        .empty_0(empty_0),
                                        .empty_1(empty_1),
                                        .empty_2(empty_2),
                                        .write_enb_reg(write_enb_reg),
                                        .read_enb_0(read_enb_0),
                                        .read_enb_1(read_enb_1),
                                        .read_enb_2(read_enb_2),
                                        .datain(datain),
                                        .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),
                                        .vld_out_1(vld_out_1),
                                        .vld_out_2(vld_out_2),
                                        .vld_out_0(vld_out_0));

initial
begin
clk = 1;
forever
#10 clk=~clk;
end

task rstn;
begin
resetn=1'b0;
#10;
resetn=1'b1;
end
endtask

initial
begin
rstn;
#20;
datain=2'b00;
detect_add = 1'b1;
write_enb_reg = 1'b1;
empty_0 = 1'b0;
```

```
full_0 = 1'b0;
read_enb_0 = 1'b0;
#20;
rstn;
#20;
datain=2'b01;
detect_add = 1'b1;
write_enb_reg = 1'b1;
empty_1 = 1'b1;
full_1 = 1'b0;
read_enb_1 = 1'b0;
#20;
rstn;
#20;
datain=2'b10;
detect_add = 1'b1;
write_enb_reg = 1'b1;
empty_2 = 1'b0;
full_2 = 1'b1;
read_enb_2 = 1'b0;
#100 $finish;
end


endmodule
```


0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
write_enb[2:0]
000
001
010
001
001
010
100
fifo_full
soft_reset_0
soft_reset_1
soft_reset_2
vld_out_1
vld_out_2
vld_out_0
clk
resetn
detect_add
full_0
full_1
full_2
empty_0
empty_1
empty_2
write_enb_reg
read_enb_0
read_enb_1
read_enb_2
datain[1:0]
XX
00
01
01
10


datain[1:0]
clk
detect_add
empty_0
empty_1
empty_2
full_0
full_1
full_2
read_enb_0
read_enb_1
read_enb_2
resetn
write_enb_reg
write_enb[2:0]
fifo_full
soft_reset_0
soft_reset_1
soft_reset_2
vld_out_0
vld_out_1
vld_out_2

FSM_RTL(SUB_BLOCK_3)

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

parameter Decode_address =  4'b0000,Load_First_Data = 4'b0001,Load_Data =
4'b0010,Wait_Till_Empty = 4'b011,Fifo_Full_State = 4'b0100,Load_After_Full = 4'b0101,Load_Parity =
4'b0110,Check_Parity_error = 4'b0111;

reg [3:0] present, next;
reg [1:0] temp;

always@(posedge clk)
begin
if(!resetn)
temp<=2'b00;
else if(detect_add)
temp<=data_in;
end

always@(posedge clk)
begin
if(!resetn)
present <=Decode_address;
else if (((soft_reset_0) && (temp==2'b00)) || ((soft_reset_1) && (temp==2'b01)) || ((soft_reset_2)
&& (temp==2'b10)))                    //if there is soft_reset and also using same channel so we do here
and opertion
present<=Decode_address;
else
present<=next;
end

always@ (*)
begin
case(present)
Decode_address :begin
if((pkt_valid && (data_in==2'b00) && fifo_empty_0)|| (pkt_valid && (data_in==2'b01) &&
fifo_empty_1)|| (pkt_valid && (data_in==2'b10) && fifo_empty_2))
begin
next <= Load_First_Data;
end
else if((pkt_valid & (data_in == 2'b00) & !fifo_empty_0) | (pkt_valid & (data_in == 2'b01)
& !fifo_empty_1) | (pkt_valid & (data_in == 2'b10) & !fifo_empty_2))
begin
next <= Wait_Till_Empty;
end
else
next <= Decode_address;
end
Load_First_Data : next <= Load_Data;
Load_Data :  begin
if(fifo_full == 1'b1)
begin
next <= Fifo_Full_State;
end
```

```
else
begin
if(!fifo_full && !pkt_valid)
next <= Load_Parity;
else
next <= Load_Data;
end
end
Wait_Till_Empty: begin
if ( (fifo_empty_0 && (temp == 2'b00)) || (fifo_empty_1 && (temp == 2'b01)) || (fifo_empty_2 &&
(temp == 2'b10)))
begin
next <= Load_First_Data;
end
else
begin
next <= Wait_Till_Empty;
end
end
Fifo_Full_State:begin
if (fifo_full == 0)
begin
next <= Load_After_Full;
end
else if (fifo_full == 1)
begin
next <= Fifo_Full_State;
end
end
Load_After_Full:begin
if(!parity_done && !low_pkt_valid)
begin
next <= Load_Data;
end
else if(!parity_done && low_pkt_valid)
begin
next <= Load_Parity;
end
else
begin
if(parity_done == 1'b1)
next <= Decode_address;
else
next <= Load_After_Full;
end
end
Load_Parity : begin
next <= Check_Parity_error;
end
Check_Parity_error:begin
if (fifo_full)
begin
next <= Fifo_Full_State;
end
else if(!fifo_full)
begin
next <= Decode_address;
```

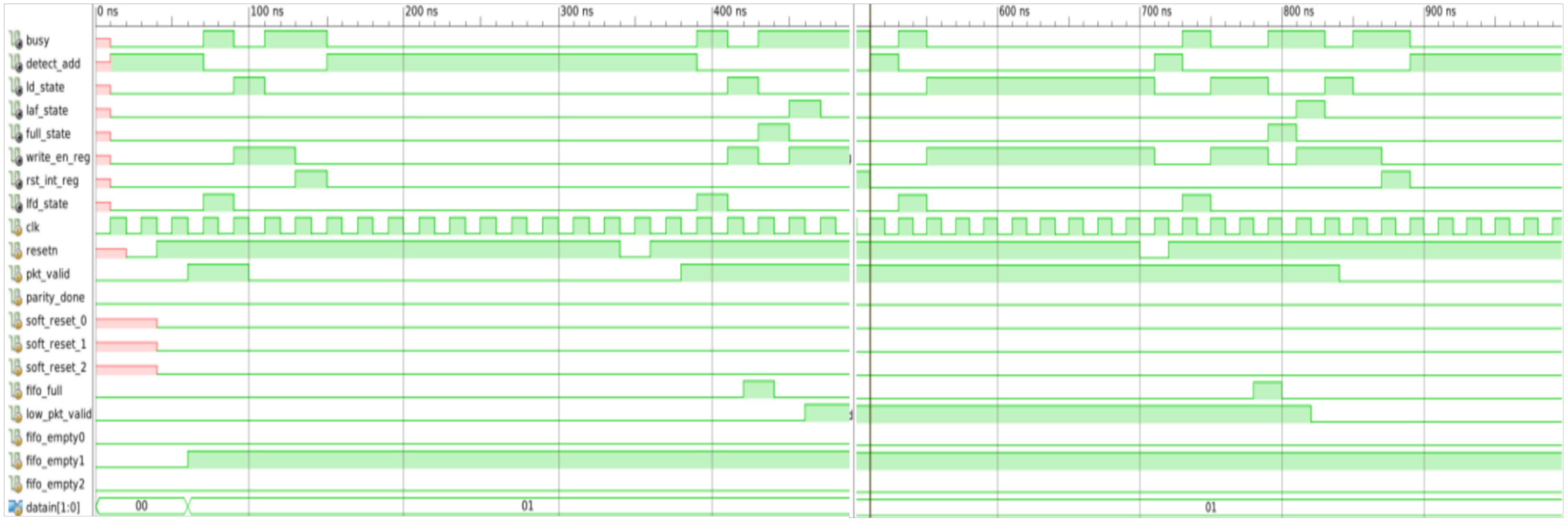
```
end
end
default : next <= Decode_address;
endcase
end

assign busy = ((present==Load_First_Data) || (present==Load_Parity) || (present==Fifo_Full_State) ||
(present==Load_After_Full) || (present==Wait_Till_Empty) || (present==Check_Parity_error)) ?
1'b1:1'b0;
assign detect_add = ((present == Decode_address)) ? 1'b1:1'b0;
assign ld_state = ((present == Load_Data)) ? 1'b1:1'b0;
assign laf_state = ((present == Load_After_Full)) ? 1'b1:1'b0;
assign full_state = ((present == Fifo_Full_State)) ? 1'b1:1'b0;
assign write_enb_reg = ((present == Load_Data) || (present == Load_After_Full) || (present ==
Load_Parity)) ? 1'b1:1'b0;
assign rst_int_reg = ((present == Check_Parity_error)) ? 1'b1: 1'b0;
assign lfd_state = ((present == Load_First_Data)) ? 1'b1:1'b0;

endmodule
```

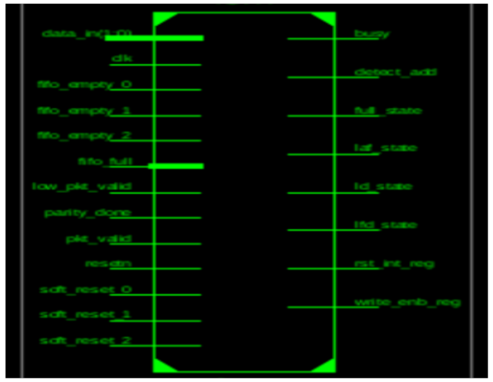
FSM_TB(SUB_BLOCK_3)

```
module fsm_tb();

reg clk, resetn, pkt_valid, parity_done, soft_reset_0,soft_reset_1,soft_reset_2, fifo_full, low_pkt_valid,
fifo_empty0, fifo_empty1, fifo_empty2;
reg [1:0] datain;
wire busy, detect_add, ld_state,laf_state, full_state, write_en_reg, rst_int_reg, lfd_state;

fsm DUT        (.clk(clk),
                                          .resetn(resetn),
                                          .pkt_valid(pkt_valid),
                                          .data_in(datain),
                                          .fifo_full(fifo_full),
                                          .fifo_empty_0(fifo_empty0),
                                          .fifo_empty_1(fifo_empty1),
                                          .fifo_empty_2(fifo_empty2),
                                          .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_en_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));

initial
begin
clk = 1'b0;
forever #10 clk = ~ clk;
end

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

task softreset(input i, input j, input k);
begin
soft_reset_0 = i;
soft_reset_1 = j;
soft_reset_2 = k;
end
endtask

task init;
begin
fifo_full = 1'b0;
pkt_valid = 1'b0;
parity_done= 1'b0;
```

```
low_pkt_valid = 1'b0;
{fifo_empty0, fifo_empty1, fifo_empty2} = 3'b000;
datain = 2'b00;
end
endtask

task cycle1();
begin
@(negedge clk)
pkt_valid = 1'b1;
datain = 2'b01;
fifo_empty1 = 1'b1;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b0;
pkt_valid = 1'b0;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b0;
end
endtask

task cycle2();
begin
@(negedge clk)
pkt_valid = 1'b1;
datain = 2'b01;
fifo_empty1 = 1'b1;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b1;
@(negedge clk)
fifo_full = 1'b0;
@(negedge clk)
parity_done = 1'b0;
low_pkt_valid = 1'b1;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b0;
end
endtask

task cycle3();
begin
@(negedge clk)
pkt_valid = 1'b1;
datain = 2'b01;
fifo_empty1 = 1'b1;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b1;
@(negedge clk)
fifo_full = 1'b0;
@(negedge clk)
parity_done = 1'b0;
low_pkt_valid = 1'b0;
@(negedge clk)
```

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

task cycle4();
begin
@(negedge clk)
pkt_valid = 1'b1;
datain = 2'b01;
fifo_empty1 = 1'b1;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b0;
pkt_valid = 1'b0;
@(negedge clk)
@(negedge clk)
fifo_full = 1'b1;
@(negedge clk)
fifo_full = 1'b0;
@(negedge clk)
parity_done = 1'b1;
end
endtask

initial begin
init;
reset();
softreset(0,0,0);
cycle1();
#200;
reset();
softreset(0,0,0);
cycle2();
#200;
reset();
softreset(0,0,0);
cycle3();
#200;
reset();
softreset(0,0,0);
cycle4();
#1000 $finish;
end

endmodule
```


0 ns
100 ns
200 ns
300 ns
400 ns
600 ns
700 ns
800 ns
900 ns
busy
detect_add
ld_state
laf_state
full_state
write_en_reg
rst_int_reg
lfd_state
clk
resetn
pkt_valid
parity_done
soft_reset_0
soft_reset_1
soft_reset_2
fifo_full
low_pkt_valid
fifo_empty0
fifo_empty1
fifo_empty2
datain[1:0]
00
01
01


data_in[1:0]
clk
fifo_empty_0
fifo_empty_1
fifo_empty_2
fifo_full
low_pkt_valid
parity_done
pkt_valid
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_RTL(SUB_BLOCK_4)

```
module register(input clk,reset,packet_valid,input [7:0] datain,
input fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,
output reg err,parity_done,low_packet_valid,output reg [7:0] dout);
reg [7:0] hb,ffb,ip,ppb;

always@(posedge clk)
begin
if(!reset)
begin
dout<=8'd0;
end
else if (detect_add && packet_valid)
hb<=datain;
else if (lfd_state)
dout<=hb;
else if (ld_state && !fifo_full)
dout<=datain;
else if (ld_state && fifo_full)
ffb<=datain;
else if (laf_state)
dout<=ffb;
else
dout<=8'd0;
end

always@(posedge clk)
begin
if(!reset)
low_packet_valid<=1'b0;
else if(ld_state==1'b1 && packet_valid==1'b0)
low_packet_valid<=1'b1;
else
low_packet_valid<=1'b0;
end

always@(posedge clk)
begin
if(!reset)
begin
parity_done<=1'b0;
end
else if(ld_state && !fifo_full && !packet_valid)
parity_done<=1'b1;
else if (laf_state && !packet_valid)
parity_done<=1'b1;
else
parity_done<=1'b0;
end

always@(posedge clk)
begin
if(!reset)
begin
ip<=8'd0;
end
```

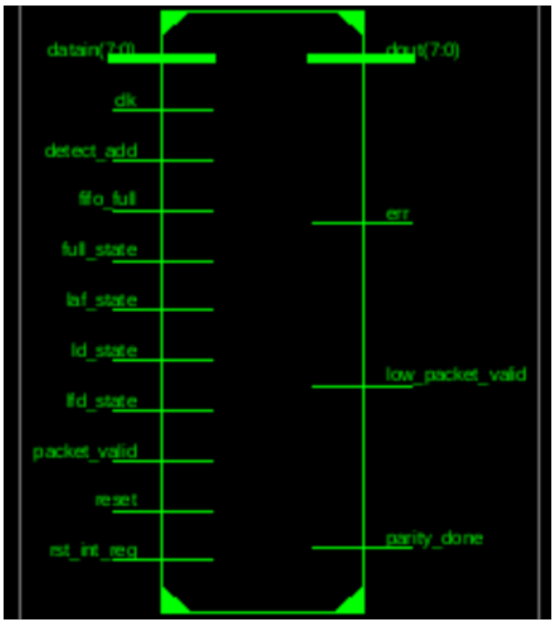
```
else if(rst_int_reg)
begin
ip<=8'd0;
end
else if(lfd_state)
ip<=ip ^ hb;
else if(ld_state && packet_valid && !full_state)
ip<=ip ^ datain;
else
begin
if (detect_add)
ip<=8'b0;
end
end

always@(posedge clk)
begin
if(!reset || rst_int_reg)
ppb<=8'b0;
else
begin
if(!packet_valid && ld_state)
ppb<=datain;
end
end

always@(posedge clk)
begin
if(!reset)
err<=1'b0;
else
begin
if(parity_done)
begin
if(ip!=ppb)
err<=1'b1;
else
err<=1'b0;
end
end
end
endmodule
```

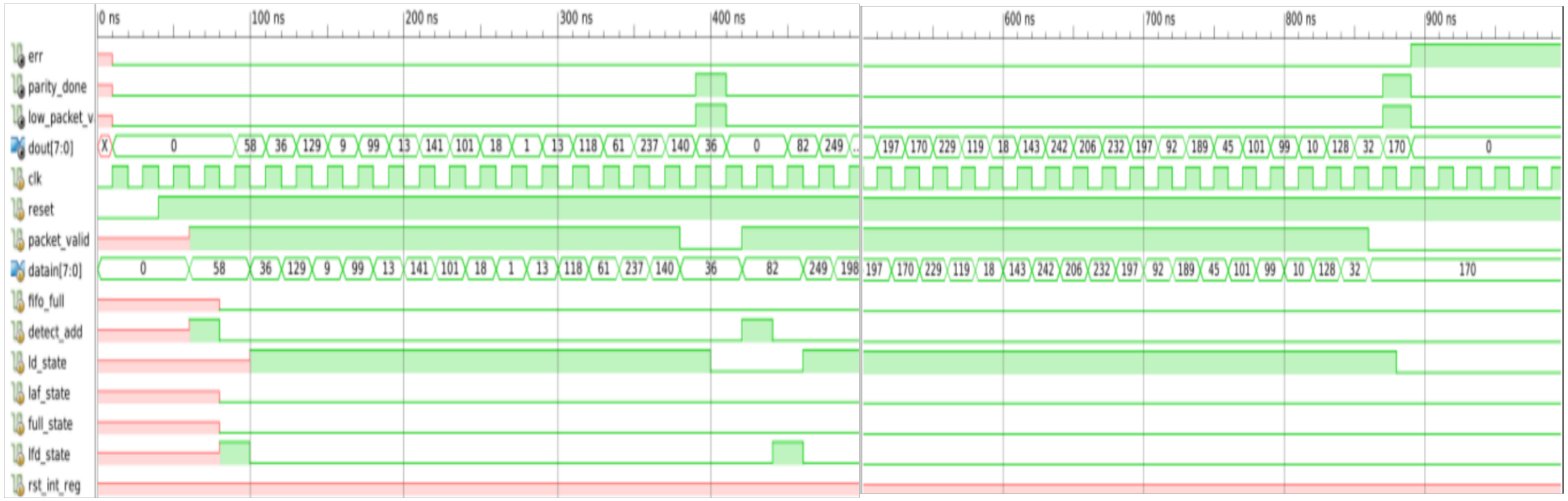
REGISTER_TB(SUB_BLOCK_4)

```
module register_tb();
reg clk,reset,packet_valid;
reg [7:0] datain;
reg fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg;
wire err,parity_done,low_packet_valid;
wire [7:0] dout;

register dut ( clk,reset,packet_valid,datain,
fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_reg,
err,parity_done,low_packet_valid, dout);

initial
begin
clk=1'b1;
forever
begin
clk=~clk;
#10;
end
end

task initialize;
begin
reset = 0;
datain=8'd0;

end
endtask

task RESET();
begin
@(negedge clk) reset = 1'b0;
@(negedge clk) reset = 1'b1;
end
endtask

//good packet
task packet_generation_good;
reg [7:0] payload_data, parity, header;
reg [5:0] payload_len;
reg [1:0] addr;
integer i;
begin
@(negedge clk)
payload_len=6'd14;
addr=2'b10; //valid packet
packet_valid=1;
detect_add=1;
header={payload_len,addr};
parity= 8'd0^header;
datain=header;
@(negedge clk)
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 clk)
lfd_state=0;
ld_state=1;
payload_data={$random}%256;
datain= payload_data;
parity=parity^datain;
end
@(negedge clk)
packet_valid=0;
datain=parity;
@(negedge clk)
ld_state=0;
end
endtask

//bad packet
task packet_generation_bad;
reg [7:0] payload_data, parity, header;
reg [5:0] payload_len;
reg [1:0] addr;
integer j;
begin
@(negedge clk)
payload_len=6'd20;
addr=2'b10; //valid packet
packet_valid=1;
detect_add=1;
header={payload_len,addr};
parity= 8'd0^header;
datain=header;
@(negedge clk)
detect_add=0;
lfd_state=1;
full_state=0;
fifo_full=0;
laf_state=0;
for(j=0;j<payload_len;j=j+1)
begin
@(negedge clk)
lfd_state=0;
ld_state=1;
payload_data={$random}%256;
datain= payload_data;
parity=parity^datain;
end
@(negedge clk)
packet_valid=0;
datain={$random}%256;
@(negedge clk)
ld_state=0;
end
endtask
```

```
initial
begin
initialize;
RESET;
packet_generation_good;
packet_generation_bad;
#500 $finish;
end
endmodule
```




0 ns
100 ns
200 ns
300 ns
400 ns
600 ns
700 ns
800 ns
900 ns
err
parity_done
low_packet_v
dout[7:0]
clk
reset
packet_valid
datain[7:0]
fifo_full
detect_add
ld_state
laf_state
full_state
lfd_state
rst_int_reg

RTL_TOP

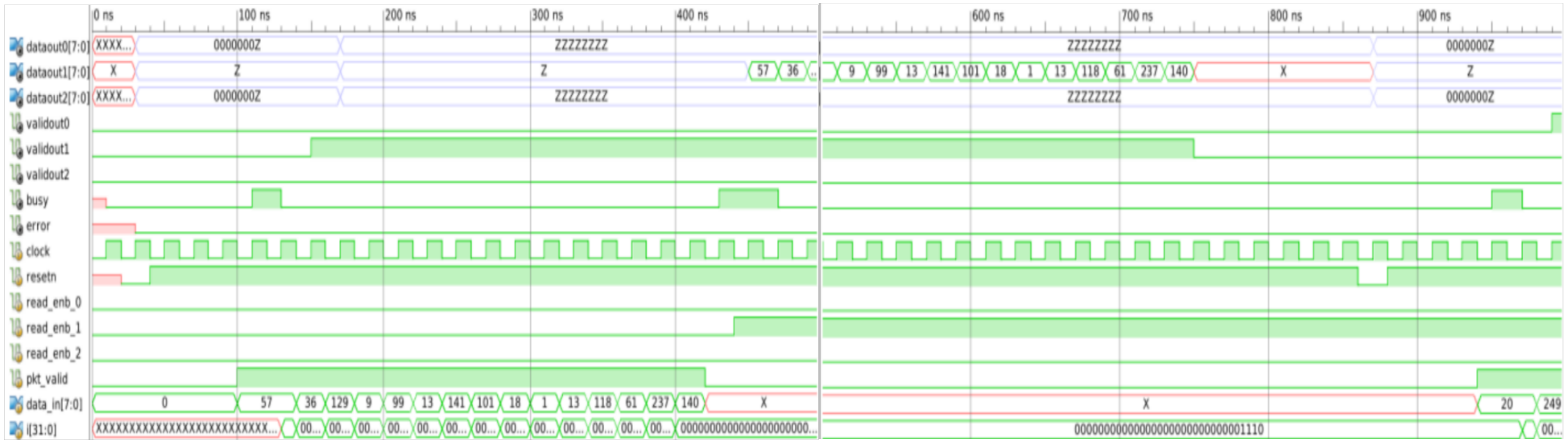
```
module router_top(input clock, resetn, read_enb_0,read_enb_1,read_enb_2, pkt_valid,
              input [7:0] data_in,
              output [7:0]data_out_0, data_out_1, data_out_2,
              output  vld_out_0, vld_out_1, vld_out_2, busy, err);

wire parity_done, soft_reset_0,soft_reset_1, soft_reset_2, fifo_full, low_pkt_valid, empty_0,
empty_1, empty_2,detect_add, ld_state,laf_state,full_state, write_en_reg, rst_int_reg,
lfd_state,full_0,full_1,full_2;
wire [2:0] write_enb;
wire [7:0] data_out;

//FIFO_SUB_BLOCK
fifo fdut0(clock, resetn, write_enb[0], read_enb_0, soft_reset_0, lfd_state,data_out,empty_0,
full_0,data_out_0);
fifo fdut1(clock, resetn, write_enb[1], read_enb_1, soft_reset_1, lfd_state,data_out,empty_1,
full_1,data_out_1);
fifo fdut2(clock, resetn, write_enb[2], read_enb_2, soft_reset_2, lfd_state,data_out,empty_2,
full_2,data_out_2);

//FSM_SUB_BLOCK
fsm  fsdut(clock, resetn, pkt_valid, parity_done, soft_reset_0,soft_reset_1, soft_reset_2, fifo_full,
             low_pkt_valid, empty_0, empty_1, empty_2, data_in[1:0],
             busy, detect_add, ld_state,laf_state, full_state, write_en_reg, rst_int_reg, lfd_state);

//SYNCRONISER_SUB_BLOCK
sync sdut(clock,resetn,detect_add,write_en_reg,read_enb_0,read_enb_1,read_enb_2,
             empty_0,empty_1,empty_2,full_0,full_1,full_2,data_in[1:0],  vld_out_0,vld_out_1,vld_out_2,
             write_enb, fifo_full, soft_reset_0,soft_reset_1,soft_reset_2);

//REGISTER_SUB_BLOCK
register
dut3(clock,resetn,pkt_valid,data_in,fifo_full,detect_add,ld_state,laf_state,full_state,lfd_state,rst_int_
reg,err, parity_done,low_pkt_valid,data_out);


endmodule
```

TOP_TB

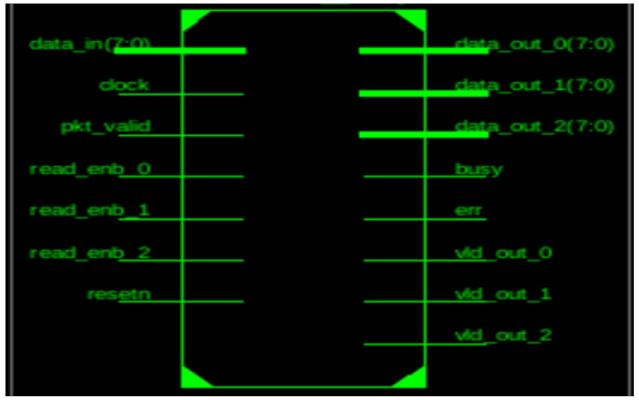
```
module router_top_testbench();

reg clock, resetn,  read_enb_0,read_enb_1,read_enb_2, pkt_valid;
reg [7:0] data_in;
wire[7:0]dataout0, dataout1, dataout2;
wire validout0, validout1, validout2, busy, error;
integer i;

router_top  DUT(clock, resetn, read_enb_0,read_enb_1,read_enb_2, pkt_valid,
        data_in, dataout0, dataout1, dataout2,
        validout0, validout1, validout2, busy, error);
//router_top
dut(clock,resetn,pkt_valid,read_enb_0,read_enb_1,read_enb_2,data_in,data_out_0,data_out_1,data
_out_2,vld_out_0,vld_out_1,vld_out_2,err,busy);

initial begin
  clock = 1'b0;
  forever #10 clock = ~clock;
end

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

task init;
        begin
                data_in = 8'b0;
                {read_enb_0,read_enb_1,read_enb_2, pkt_valid} = 4'b0;
        end
endtask

task pl_14;
reg [7:0]payload,header,parity;
reg [5:0]paylen;
reg [1:0]addr;
        begin
            @(negedge clock)
        wait(~busy) begin
        @(negedge clock)
        paylen = 6'd14;
        addr = 2'b01;
        header = {paylen,addr};
        data_in = header;
        pkt_valid = 1;
        parity = parity ^ header;
        end
        @ (negedge clock)
        wait(~busy)
        begin
        for(i = 0; i<paylen;i=i+1)
          begin
           @ (negedge clock)
           wait(~busy)
```

```
                  payload = {$random}%256;
                  data_in = payload;
                  parity = parity^payload;
                end
             end
              @(negedge clock)
               wait(~busy)begin
               pkt_valid = 0;
               data_in = parity;
               end
             end
endtask

task pl_05;
reg [7:0]payload,header,parity;
reg [5:0]paylen;
reg [1:0]addr;
             begin
                          @(negedge clock)
             wait(~busy)
             @(negedge clock)
             begin
             paylen = 6'd05;
             addr = 2'b00;
             header = {paylen,addr};
             data_in = header;
             pkt_valid = 1;
             parity = parity ^ header;
             end
             @ (negedge clock)
             wait(~busy)
             begin
             for(i = 0; i<paylen;i=i+1)
                begin
                  @ (negedge clock)
                  wait(~busy)
                  payload = {$random}%256;
                  data_in = payload;
                  parity = parity^payload;
                end
             end
              @(negedge clock)
              wait(~busy)begin
              pkt_valid = 0;
              data_in = parity;
              end
             end
endtask
task pl_16;
reg [7:0]payload,header,parity;
reg [5:0]paylen;
reg [1:0]addr;
             begin
                          @(negedge clock)
             wait(~busy)begin
             @(negedge clock)
             paylen = 6'd16;
```

```
                addr = 2'b10;
                header = {paylen,addr};
                data_in = header;
                pkt_valid = 1;
                parity = parity ^ header;
                end
                @ (negedge clock)
                wait(~busy)begin
                @ (negedge clock)
                for(i = 0; i<paylen;i=i+1)
                    begin
                      @ (negedge clock)
                      wait(~busy)
                      payload = {$random}%256;
                      data_in = payload;
                      parity = parity^payload;
                    end
                 end
                 @(negedge clock)
                 wait(~busy) begin
                 pkt_valid = 0;
                 data_in = parity;
                  end
                end
endtask
initial begin
   init;
   reset;
   @(negedge clock)
   pl_14;
   @(negedge clock)
   read_enb_1 = 1;
   wait(~validout1)
   @(negedge clock)
   //read_enb_1 = 0;
   #100;
   reset;
   @(negedge clock)
   pl_05;
   @(negedge clock)
   read_enb_0 = 1;
   wait(~validout0)
   @(negedge clock)
   read_enb_0 = 0;
   #100;
   reset;
   @(negedge clock)
   pl_16;
   @(negedge clock)
   read_enb_2 = 1;
   wait(~validout2)
   @(negedge clock);
   read_enb_2 = 0;
  $finish;
end
endmodule
```



0 ns
100 ns
200 ns
300 ns
400 ns
600 ns
700 ns
800 ns
900 ns
dataout0[7:0]
dataout1[7:0]
dataout2[7:0]
validout0
validout1
validout2
busy
error
clock
resetn
read_enb_0
read_enb_1
read_enb_2
pkt_valid
data_in[7:0]
i[31:0]
0000000Z
ZZZZZZZZ
57
36
129
9
99
13
141
101
18
1
118
61
237
140
20
249
00000000000000000000000000001110



data_in(7:0)
clock
pkt_valid
read_enb_0
read_enb_1
read_enb_2
resetn
data_out_0(7:0)
data_out_1(7:0)
data_out_2(7:0)
busy
err
vld_out_0
vld_out_1
vld_out_2

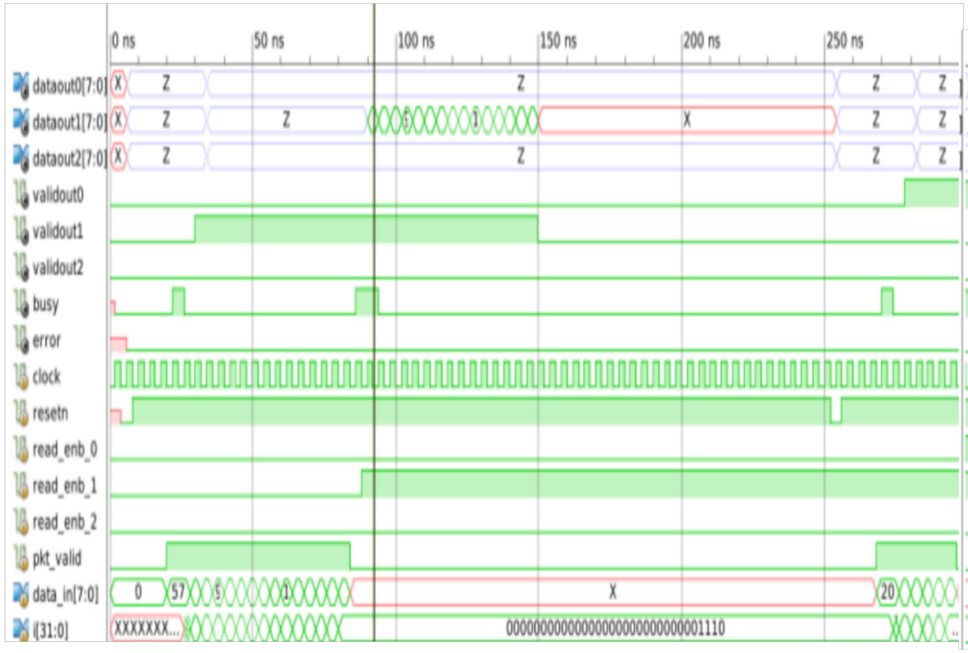
TOP_TB

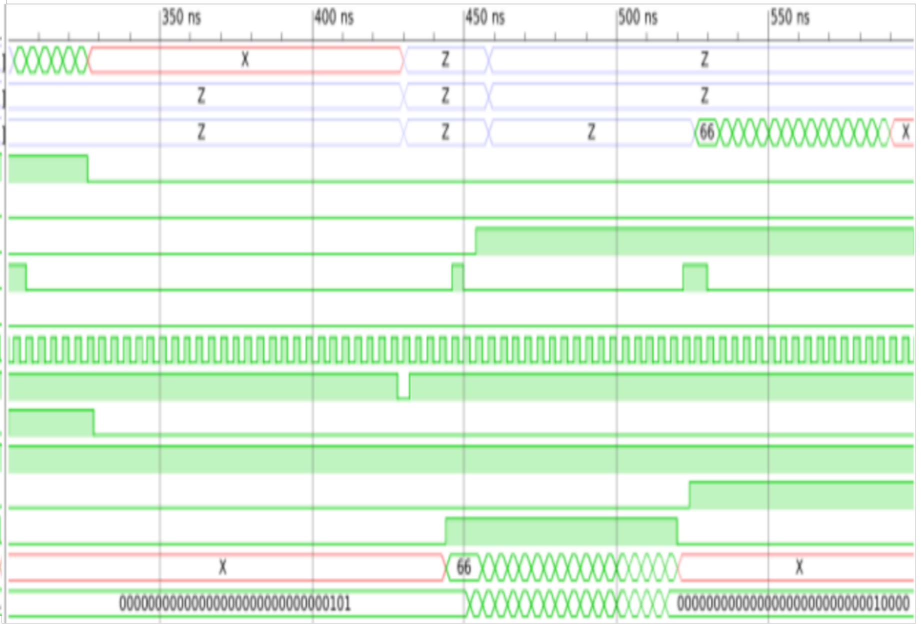
```
module router_top_testbench();

reg clock, resetn,  read_enb_0,read_enb_1,read_enb_2, pkt_valid;
reg [7:0] data_in;
wire[7:0]dataout0, dataout1, dataout2;
wire validout0, validout1, validout2, busy, error;
integer i;

router_top  DUT(clock, resetn, read_enb_0,read_enb_1,read_enb_2, pkt_valid,
            data_in, dataout0, dataout1, dataout2,
            validout0, validout1, validout2, busy, error);
//router_top
dut(clock,resetn,pkt_valid,read_enb_0,read_enb_1,read_enb_2,data_in,data_out_0,data_out_1,data
_out_2,vld_out_0,vld_out_1,vld_out_2,err,busy);

initial begin
  clock = 1'b0;
  forever #2 clock = ~clock;
end

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

task init;
            begin
                        data_in = 8'b0;
                        {read_enb_0,read_enb_1,read_enb_2, pkt_valid} = 4'b0;
            end
endtask

task pl_14;
reg [7:0]payload,header,parity;
reg [5:0]paylen;
reg [1:0]addr;
            begin
                 @(negedge clock)
            wait(~busy) begin
            @(negedge clock)
            paylen = 6'd14;
            addr = 2'b01;
            header = {paylen,addr};
            data_in = header;
            pkt_valid = 1;
            parity = parity ^ header;
            end
            @ (negedge clock)
            wait(~busy)
            begin
            for(i = 0; i<paylen;i=i+1)
               begin
                @ (negedge clock)
                wait(~busy)
               payload = {$random}%256;
               data_in = payload;
               parity = parity^payload;
             end
          end
           @(negedge clock)
            wait(~busy)begin
            pkt_valid = 0;
            data_in = parity;
            end

         end
endtask

task pl_05;
reg [7:0]payload,header,parity;
reg [5:0]paylen;
reg [1:0]addr;
         begin
                  @(negedge clock)
         wait(~busy)
         @(negedge clock)
         begin
         paylen = 6'd05;
         addr = 2'b00;
         header = {paylen,addr};
         data_in = header;
         pkt_valid = 1;
         parity = parity ^ header;
         end
         @ (negedge clock)
         wait(~busy)
         begin
         for(i = 0; i<paylen;i=i+1)
             begin
               @ (negedge clock)
               wait(~busy)
               payload = {$random}%256;
               data_in = payload;
               parity = parity^payload;
             end
         end
          @(negedge clock)
          wait(~busy)begin
          pkt_valid = 0;
          data_in = parity;
          end

         end
endtask

task pl_16;
reg [7:0]payload,header,parity;
reg [5:0]paylen;
reg [1:0]addr;
         begin
                  @(negedge clock)
```

```
                wait(~busy)begin
                @(negedge clock)
                paylen = 6'd16;
                addr = 2'b10;
                header = {paylen,addr};
                data_in = header;
                pkt_valid = 1;
                parity = parity ^ header;
                end
                @ (negedge clock)
                wait(~busy)begin
                @ (negedge clock)
                for(i = 0; i<paylen;i=i+1)
                    begin
                      @ (negedge clock)
                      wait(~busy)
                      payload = {$random}%256;
                      data_in = payload;
                      parity = parity^payload;
                    end
                 end
                 @(negedge clock)
                 wait(~busy) begin
                 pkt_valid = 0;
                 data_in = parity;
                  end
                end
endtask

initial begin
   init;
   reset;
   @(negedge clock)
   pl_14;
   @(negedge clock)
   read_enb_1 = 1;
   wait(~validout1)
   @(negedge clock)
   //read_enb_1 = 0;
   #100;
   reset;
   @(negedge clock)
   pl_05;
   @(negedge clock)
   read_enb_0 = 1;
   wait(~validout0)
   @(negedge clock)
   read_enb_0 = 0;
   #100;
   reset;
   @(negedge clock)
   pl_16;
   @(negedge clock)
   read_enb_2 = 1;
   wait(~validout2)
   @(negedge clock);
   read_enb_2 = 0;
```

```
  $finish;
end

endmodule
```

NOTE: Kindly note that the first TB code I kept delay between clock cycles as 10ns. So was not able to capture the data for all the packets. The second TB code I reduced delay between clock cycles as 2ns and was able to capture the all packets. But data is pixelated as seen.


0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
dataout0[7:0]
dataout1[7:0]
dataout2[7:0]
validout0
validout1
validout2
busy
error
clock
resetn
read_enb_0
read_enb_1
read_enb_2
pkt_valid
data_in[7:0]
i[31:0]
X
Z
5
1
57
0
20
XXXXXXX...
00000000000000000000000000001110


350 ns
400 ns
450 ns
500 ns
550 ns
X
Z
66
00000000000000000000000000000101
00000000000000000000000000010000