1.1

```
module jk_ff (input j,input k, input rstn, input clk, output reg q);
always @ (posedge clk or negedge rstn)
begin
if(!rstn)
begin
q<=0;
end
else
begin
q <=(j&~q)|(~k&q);
end
end
endmodule
module tb;
reg j,k,rstn,clk;
wire q;
integer i;
reg [2:0] dly;
always #10 clk=~clk;
jk_ff jk_instance ( .j(j),.k(k),.clk(clk),.rstn(rstn),.q(q));
initial
begin
{j,k,rstn,clk}<=0;
#10 rstn<= 1;
for(i=0;i<10;i=i+1)
begin
dly= $random;
#(dly)j<= $random;
#(dly) k<=$random;
end
#20 $finish;
```

```
end
endmodule
```

1.2

```
module t_ff(t,clk,rst,q);
input t,clk,rst;
output reg q;
always@(posedge clk or negedge rst)
begin
if(!rst)
q<=1'b0;
else
begin
if(t==1)
q <=~q;
if(t==0)
q <=q;
end
end
endmodule
module t_ff_tb;
reg t,clk,rst;
wire q;
integer i;
reg [4:0]dly;
t_ff uut(.clk(clk),.rst(rst),.t(t),.q(q));
always #5 clk = ~clk;
initial begin
{rst,clk,t} <= 0;
$monitor("T=%0t,rst = %0b,t=%0d,q=%0d",$time,rst,t,q);
repeat(2)@(posedge clk);
rst <= 1;
```

```
for(i=0;i<=20;i=i+1)
begin
dly = $random;
#(dly)t<=$random;
end
#20 $finish;
end
endmodule
```

1.3

```
module muxbehv(x,y,sel,m);
input x,y,sel;
output reg m;
always@(x or y or sel)
begin
m =0;
if(sel==0)
begin
m = x;
end
else
begin
m = y;
end
end
endmodule
module muxbehav_tb;
reg x,y,sel;
wire m;
muxbehav uut(x,y,sel,m);
initial begin
x = 1'b0; y=1'b0;sel=1'b0;
```

```
#10 x=1'b0;y=1'b0;sel=1'b1;
#10 x=1'b0;y=1'b1;sel=1'b0;
#10 x=1'b0;y=1'b1;sel=1'b1;
#10 x=1'b1;y=1'b0;sel=1'b0;
#10 x=1'b1;y=1'b0;sel=1'b1;
#10 x=1'b1;y=1'b1;sel=1'b0;
#10 x=1'b1;y=1'b1;sel=1'b1;
$finish;
end
endmodule
```

2.1

```
module intra_delay;
reg a,b,c,q;
initial begin
$monitor("[%0t] a=%0b b=%0b c=%0b q=%0b",$time,a,b,c,q);
a<=0;b<=0;c<=0;q<=0;
#5 a<=1;c<=1;
q<=#5 a&b|c;
#30; $finish;
end
endmodule
```

2.2

```
module inter_delay;
reg a,b,c,q;
initial begin
$monitor("[%0t] a=%0b b=%0b c=%0b q=%0b",$time,a,b,c,q);
a<=0;b<=0;c<=0;q<=0;
#5 a<=1;c<=1;
#5 q<=a&b|c;
#10; $finish;
end
```

```
endmodule

3
module tb;
reg [3:0] ctr;
reg clk;
reg rst;
always #10 clk=~clk;
always begin
rst =1;
@(posedge clk)
ctr <=ctr + 1;
end
initial begin
{ctr,clk}<=0;
wait (ctr);
$display("T= %0t Counter reached non zero value 0x%0h",$time,ctr);
wait(ctr==4);
$display("T=%0t counter reached 0x%0h",$time,ctr);
$finish;
end
endmodule

4
module memory(addr, cs, re, we, clk, reset, data_in, data_out);
parameter addr_width = 16;
parameter data_width = 32;
input [addr_width-1:0] addr;
input cs, re, we;
input clk, reset;
input [data_width-1:0] data_in;
output [data_width-1:0] data_out;
```

```
reg [data_width-1:0] data_out;
reg [data_width-1:0] mem[0:2**addr_width-1];
integer i;
always @(posedge clk) begin
if (reset) begin
mem[addr] <= 0;
end else begin
if (cs) begin
if (we) begin
mem[addr] <= data_in;
end
if (re) begin
data_out <= mem[addr];
end
end
end
end
endmodule

module memory_tb();
parameter addr_width = 8;
parameter data_width = 8;
integer count;
reg [addr_width-1:0] addr;
reg [data_width-1:0] data_in;
wire [data_width-1:0] data_out;
reg clk, reset, cs, re, we;
memory #(addr_width, data_width) memory_inst(addr, cs, re, we, clk, reset, data_in,
data_out);
always #5 clk = ~clk;
initial begin
clk = 0;
```

```
reset = 1;
#10 reset = 0;
for (count = 0; count < 'h100; count = count + 1) begin
write_task(count);
read_task(count);
if (data_in != data_out)
$display($time, "FAIL: Data Miss Match for addr=%0h, data_in=%0h,
data_out=%0h\n", count, data_in, data_out);
else
$display($time, "PASS: Data Match for addr=%0h, data_in=%0h, data_out=%0h\n",
count, data_in, data_out);
end
$finish;
end
task write_task;
input [addr_width-1:0] addr_in;
begin
@(posedge clk);
#1;
cs = 1; we = 1; re = 0;
data_in = $random;
addr = addr_in;
@(posedge clk);
#1;
we = 0; cs = 0;
end
endtask
task read_task;
input [addr_width-1:0] addr_in;
begin
@(posedge clk);
#1;
```

```
cs = 1; we = 0; re = 1;
addr = addr_in;
@(posedge clk);
#1;
re = 0; cs = 0;
end
endtask
endmodule
```

5.1

```
primitive mux(out,sel,a,b);
output out;
input sel,a,b;
table
0 1 ? : 1;
0 0 ? : 0;
1 ? 0 : 0;
1 ? 1 : 1;
x 0 0 : 0;
x 1 1 : 1;
endtable
endprimitive

module mux_tb;
reg sel,a,b;
reg [2:0]dly;
wire out;
integer i;
mux u_mux(out,sel,a,b);
initial begin
a<=0;
b<=0;
```

```
$monitor("[T=%0t] a=%0b b=%0b sel=%0b out=%0b",$time,a,b,sel,out);
for(i=0; i <10 ;i=i+1) begin
dly=$random;
#(dly)a<=$random;
dly=$random;
#(dly)b<=$random;
dly=$random;
#(dly)sel<=$random;
end
end
endmodule
```

5.2

```
primitive D_latch(q,clk,d);
output q;
input clk,d;
reg q;
table
// clk d q q+
11 : ? : 1;
10 : ? : 0;
0? : ? : -;
endtable
endprimitive
module Dlatch_tb1;
reg clk,d;
reg [1:0]dly;
wire q;
integer i;
D_latch u_latch(q,clk,d);
always #10 clk=~clk;
initial begin
clk=0;
```

```
$monitor ("[T=%0t] clk=%0b d=%0b q=%0b",$time,clk,d,q);
#10;
for(i=0;i<50;i=i+1)begin
dly=$random;
#(dly)d<=$random;
end
#20 $finish;
end endmodule
```

5.3

```
primitive D_flop(q,clk,d);
output q;
input clk,d;
reg q;
table
//clk d q q+
//obtain output on rising edeg of clk
(01)0 : ? : 0;
(01)1 : ? : 1;
(0?)1 : 1 : 1;
(0?)0 : 0 : 0;
(?0)? : ? : -;
?(??) : ? : -;
endtable
endprimitive

module D_flop_tb;
reg clk,d;
reg [1:0]dly;
wire q;
integer i;
D_flop u_flop(q,clk,d);
always #10 clk=~clk;
```

```
initial begin
clk=0;
$monitor("[T=%0t] clk=%0b d=%0b q=%0b" ,$time,clk,d,q);
for(i=0;i<20;i=i+1)begin
dly=$random;
repeat(dly)@(posedge clk);
d<=$random;
end
#20;
$finish;
end
endmodule
```

6

```
module blocking_nonblocking;
reg[0:7] A,B,C;
initial begin
A=3;
A=A+1;
B=A+1;
$display($time,"Blocking: A=%h B=%h",A,B);
A=8'h3;B=8'h0;
A<=A+'h1;
B<=A+1;
$display($time,"Non-blocking : A=%h B=%h",A,B);
#1;
$display($time,"End of time unit: A=%h B=%h",A,B);
end
endmodule
```

7

```
module counter(clk,reset,up_down,load,data,count);
```

```
//define input and output ports
input clk,reset,load,up_down;
input [3:0] data;
output reg [3:0] count;
//always block will be executed at each and every positive edge of the clock
always@(posedge clk)
begin
if(reset)    //Set Counter to Zero
count <= 0;
else if(load)    //load the counter with data value
count <= data;
else if(up_down)
  //count up
count <= count + 1;
else
//count down
count <= count - 1;
end
endmodule :counter
```

8

```
module BoothMulti(X, Y, Z);
input signed [3:0] X, Y;
output reg signed[7:0] Z;
reg [1:0] temp;
integer i;
reg E1;
reg [3:0] Y1;
always @ (X, Y)
begin
Z = 8'd0;
E1 = 1'd0;
```

```
for (i = 0; i < 4; i = i + 1)
begin
temp = {X[i], E1};
Y1 = - Y;
case (temp)
2'd2 : Z [7 : 4] = Z [7 : 4] + Y1;
2'd1 : Z [7 : 4] = Z [7 : 4] + Y;
default : begin end
endcase
Z = Z >> 1;
Z[7] = Z[6];
E1 = X[i];
end
if (Y == 4'd8)
begin
Z = - Z;
end
end
endmodule
module BoothTB;
reg [3:0] X;
reg [3:0] Y;
wire [7:0] Z;
BoothMulti uut (.X(X),.Y(Y),.Z(Z));
initial begin
X=4'b0110;Y=4'b0101;#10
X=4'b0010;Y=4'b1001;#10
X=4'b0110;Y=4'b0100;#10
X=4'b1110;Y=4'b0110;#10
X=4'b1100;Y=4'b1110;#10
$finish;
end
```

```
endmodule
```

9

```
module async_fifo(
input wclk,rclk,reset,
input w_enable,r_enable,
input [7:0]wdata,
output reg[7:0]rdata,
output full_flag,empty_flag);
parameter FIFO_DEPTH=16, POINTER=4;
reg[7:0]fifo[0:FIFO_DEPTH-1];
reg[POINTER:0]wpointer,rpointer,wpointer_sync,rpointer_sync,Qr,Qw;
assign full_flag = (({~wpointer[POINTER],wpointer[POINTER
1:0]}==rpointer_sync[POINTER:0]))?1:0;
assign empty_flag = (wpointer_sync == rpointer)?1:0;
//write
always@(posedge wclk,posedge reset)
begin
if(reset==1)
begin
wpointer <= 0;
end
else
begin
if(w_enable==1 && full_flag != 1)
begin
fifo[wpointer[POINTER-1:0]] <= wdata;
wpointer <= wpointer+1;
end
end
end
//read
```

```
always@(posedge rclk,posedge reset)
begin
if(reset==1)
 begin
  rpointer <= 0;
 end
 else
 begin
 if(r_enable == 1 && empty_flag != 1)
 begin
  rdata <= fifo[rpointer[POINTER-1:0]];
  rpointer <= rpointer+1;
 end
 end
end

//wr_ptr Synchronization for read side

always@(posedge rclk,posedge reset)
begin
 if(reset==1)
 begin
  Qw <= 0;
  wpointer_sync <= 0;
 end
 else
 begin
  Qw <= wpointer;
  wpointer_sync <= Qw;
 end
end
```

```
//rd_ptr Synchronization



always@(posedge wclk,posedge reset)
begin
 if(reset==1)
 begin
  Qr <= 0;
  rpointer_sync <= 0;
 end
 else
 begin
  Qr <= rpointer;
  rpointer_sync <= Qr;
 end
 end
endmodule
///Testbench///
module async_fifo_tb;
parameter FIFO_DEPTH=16,POINTER=4;
reg wclk_t,rclk_t,reset_t;
reg wenable_t,renable_t;
reg[7:0]wdata_t,data_in;
wire[7:0]rdata_t;
wire full_flag_t,empty_flag_t;
integer count;
async_fifo
#(FIFO_DEPTH,POINTER)async_inst(wclk_t,rclk_t,reset_t,wenable_t,renable_t,wdata_t,rdata_
 t,full_flag_t,empty_flag_t);
//write task
task write_task;
input[7:0]data_in;
```

```
begin
@(posedge wclk_t);
if(!full_flag_t)
begin
wenable_t = 1;
wdata_t = data_in;
@(posedge wclk_t);
wenable_t = 0;
end
end
endtask
//read task
task read_task;
begin
@(posedge rclk_t);
if(!empty_flag_t)
begin
renable_t = 1;
@(posedge rclk_t);
renable_t = 0;
end
end
endtask
//single_wr_rd_test
task single_wr_rd_test();
begin
data_in = $random;
write_task(data_in);
read_task();
end
endtask
//all_wr_rd_test
```

```
task all_wr_rd_test();
for(count=0;count < FIFO_DEPTH; count=count+1)
begin
data_in = $random;
write_task(data_in);
read_task();
end
endtask
//fifo full and empty case
task fifo_full_empty_test();
begin
for(count =0;count < FIFO_DEPTH+1;count=count+1)
begin
data_in = $random;
write_task(data_in);
end
for(count=0; count < FIFO_DEPTH+1;count=count+1)
begin
read_task();
end
end
endtask
//wclk clock gen
always
begin
#5 wclk_t = ~wclk_t;
end
//rclk clock gen
always
begin
#5 rclk_t = ~rclk_t;
end
```

```
//reset and calling the task
initial
begin
wclk_t = 0;
rclk_t = 0;
reset_t = 1;
#15
reset_t = 0;
@(posedge wclk_t);
single_wr_rd_test();
all_wr_rd_test();
fifo_full_empty_test();
#100 $finish;
end
endmodule
```

10

```
module spi_state(

  input wire clk,
  input wire reset,

  input wire [15:0] datain,
  output wire spi_cs_1,
  output wire spi_sclk,
  output wire spi_data,
  output [4:0] counter
 );

  reg [15:0] MOSI;
  reg [4:0] count;
  reg cs_1;
  reg sclk;
  reg [2:0] state;

  always@(posedge clk or posedge reset)
    if (reset) begin
       MOSI <=16’b0;
       count <=5’d16;
       cs_1 <=1’b1;
       sclk <=1’b0;
    end

  else begin
  case (state)

  0:begin
  sclk <=1’b0;
    cs_1 <=1’b1;
    state<=1;
  end



    1:begin
      sclk <=1’b0;
cs_1 <=1’b0;
      MOSI<=datain[count-1];
      count <=count-1;
      state<=2;
    end

    2:begin
      sclk <= 1’b1;
      if (count > 0)
```

```
        state<=1;
        else begin
          count<=16;
         state<=0;
        end
     end



  default:state<=0;
         endcase
 end

  assign spi_cs_1 = cs_1;
  assign spi_sclk = sclk;
  assign spi_data = MOSI;
  assign counter=count;
endmodule
```

Test bench;

```
module tb_spi_state;
  reg clk;
  reg reset;
  reg [15:0]datain;



  wire spi_cs_1;
  wire spi_sclk;
  wire spi_data;
  wire [4:0]counter;

  spi_state dut (
    .clk(clk),
   .reset(reset),
   .counter(counter),
   .datain(datain),
   .spi_cs_1(spi_cs_1),
   .spi_sclk(spi_sclk),
  .spi_data(spi_data)
  );

  initial begin
   clk=0;
   reset =1;
   datain=0;
  end
  always #5 clk=~clk;

  initial begin
   #10 reset=1’b0;

   #10 datain=16’h5569;
   #335 datain=16’h2563;
   #335 datain=16’h9863;
   #335 datain=16’h6561;

  end
endmodule
```