Experiment 8
Testbenches

Aim: To write testbenches for different Verilog modules.

1) Testbench for a 4:1 multiplexer.

(i) Define a multiplexer module.

```
module mux4to1 (OUTPUT, IO, I1, I2, I3, S1, SO);
output OUTPUT;
input IO, I1, I2, I3;
input S1, SO;
assign OUTPUT = (~S1 & ~SO & IO) | (~S1 & SO & I1) | (S1 & ~SO & I2) | (S1 & SO & I3);
endmodule
```

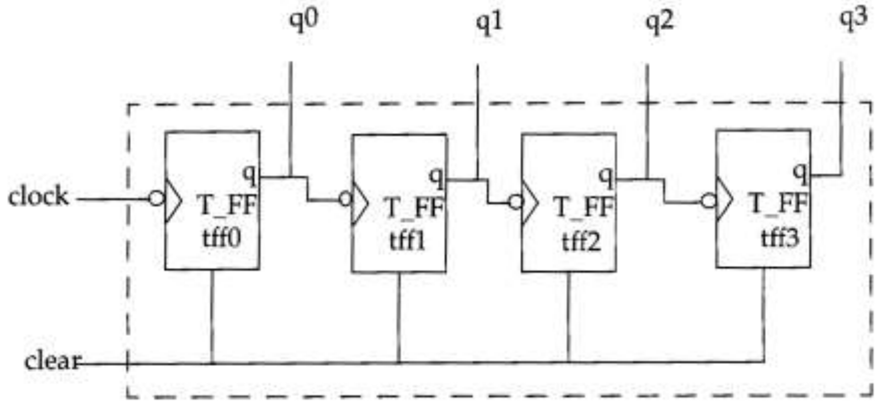
(ii) Write a testbench module which calls the multiplexer module.

```
module stimulus;
reg INO, IN1, IN2, IN3;
reg S1, SO;
wire OUTPUT;
mux4to1 mymux (OUTPUT, INO, IN1, IN2, IN3, S1, SO);
initial
begin
INO = 1; IN1 = 0; IN2 = 1; IN3 = 0;
#5 $display("INO= %b, IN1= %b, IN2= %b, IN3=%b \n”,INO,IN1,IN2,IN3);
S1 = 0; SO = 0;
#5 $display("S1 = %b, SO = %b, OUTPUT = %b \n”, S1, SO, OUTPUT);
S1 = 0; SO = 1;
#5 $display("S1 = %b, SO = %b, OUTPUT = %b \n", S1, SO, OUTPUT);
// choose IN2
S1 = 1; SO = 0;
#5 $display("S1 = %b, SO = %b, OUTPUT = %b \n”, S1, SO, OUTPUT);
// choose IN3
S1 = 1; SO = 1;
#5 $display("S1 = %b, SO = %b, OUTPUT = %b \n”, S1, SO, OUTPUT);
end
endmodule
```

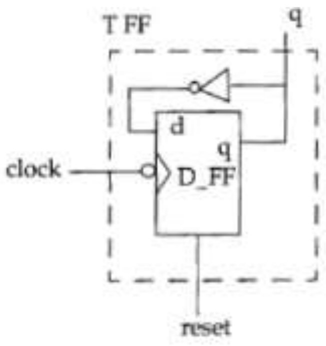
2) Write a testbench for 4–bit Ripple carry adder.

3) Write a testbench for a full subtractor.

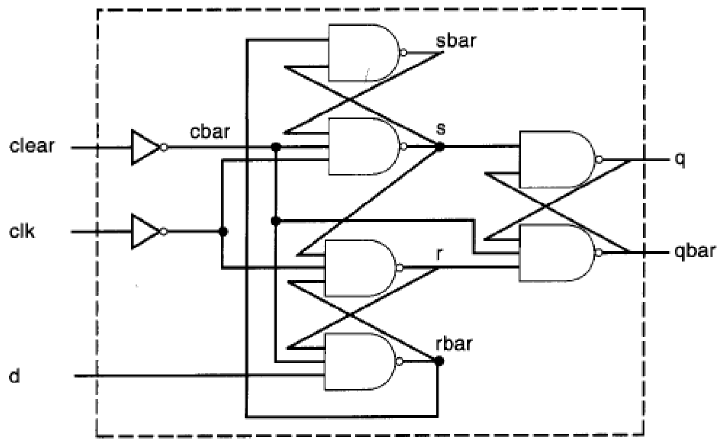
4) Write a testbench for 4 –bit ripple carry counter.



```
// Ripple counter
module counter (Q , clock, clear) ;
output [3:0] Q;
input clock, clear;
// Instantiate the T flipflops
T_FF tffO (Q[0] , clock, clear) ;
T_FF tff1(Q[1], Q[0], clear) ;
T_FF tff2 (Q[2], Q[1], clear) ;
T_FF tff3 (Q[3], Q[2], clear) ;
endmodule
```



```
// Edge-triggered T-flipflop.
module T_FF (q, clk, clear);
output q;
input clk, clear;
edge_dff ff1(q  ,~q, clk, clear);
endmodule
```



```
// Edge-triggered D flipflop
module edge_dff(q, qbar, d, clk, clear);
output q,qbar;
input d, clk, clear;
wire s, sbar, r, rbar, cbar;
assign cbar = ~clear;
assign sbar = ~(rbar & s),
s = ~(sbar & cbar & ~clk),
r = ~(rbar & ~clk & s),
rbar = ~(r & cbar & d);
assign q = ~(s & qbar) ,
qbar = ~(q & r & cbar);
endmodule

module stimulus;
reg CLOCK, CLEAR;
wire [3:0] Q;
counter c1(Q, CLOCK, CLEAR);
initial
begin
CLEAR = 1'b1;
#34 CLEAR = 1'b0;
#200 CLEAR = 1'b1;
#50 CLEAR = 1'b0;
end

initial
begin
CLOCK = 1'b0;
```

```
forever #10 CLOCK = ~CLOCK;
end
initial
begin
#400 $finish;
end
endmodule
```

5) Write a testbench for a clock divider.

Experiment 9
Seven Segment Display Controller

Aim:

- Driving a seven segment display on the Basys3 board.
- Display count on the seven segment display using a 1Hz up counter.
- Display the result of a hexadecimal adder on seven segment display.

Part 1: Verilog Code for Driving A 7 Segment Display

```
module seven_segment (i_binary, o_hexdisplay);
input [3:0] i_binary;
output reg [6:0] o_hexdisplay;
always
case (i_binary)
4'b0000 : o_hexdisplay = 7'b1000000;
4'b0001 : o_hexdisplay = 7'b1111001  ;
4'b0010 : o_hexdisplay = 7'b0100100 ;
4'b0011 : o_hexdisplay = 7'b0110000 ;
4'b0100 : o_hexdisplay = 7'b0011001 ;
4'b0101 : o_hexdisplay = 7'b0010010 ;
4'b0110 : o_hexdisplay = 7'b0000010 ;
4'b0111 : o_hexdisplay = 7'b1111000;
4'b1000 : o_hexdisplay = 7'b0000000;
4'b1001 : o_hexdisplay = 7'b0010000 ;
4'b1010 : o_hexdisplay = 7'b0001000 ;
4'b1011 : o_hexdisplay = 7'b0000011;
4'b1100 : o_hexdisplay = 7'b1000110 ;
4'b1101 : o_hexdisplay = 7'b0100001 ;
4'b1110 : o_hexdisplay = 7'b0000110 ;
4'b1111 : o_hexdisplay = 7'b0001110 ;
default: o_hexdisplay = 7'b0111111;
endcase
endmodule
```

Part2: Verilog Code for A 1 Hz Up Counter and to Display the Count On the Seven Segment Display.

```
module counter ( i_clk_fpga, i_reset, o_seg, o_an);
input i_clk_fpga, i_reset;
output [6:0] o_seg;
output [3:0] o_an;

parameter max_count = 100000000 – 1;
wire counter_en;
reg [26:0] counter_100m;
reg [3:0] counter_10;

assign o_an=4’b1110;
seven_segment disp1(.i_binary(counter_10), .o_hexdisplay(o_seg));

always @(posedge i_clk_fpga, posedge i_reset)
if (i_reset)
        counter_100m <= 0;
else if (counter_100m == max_count)
        counter_100m <= 0;
else
        counter_100m <= counter_100m + 1’b1;

assign counter_en = counter_100m == 0;

always @(posedge i_clk_fpga,posedge i_reset)
if (i_reset)
        counter_10 <= 0;
else if (counter_en)
        if(counter_10 == 9)
          counter_10 <= 0;
        else
            counter_10 <= counter_10 + 1’b1;
endmodule
```

Part3: Implement a Hexadecimal Adder Circuit and display the result on the Seven Segment Display.

```
module hexadder (i_a,i_b,o_out, i_clk, i_rst, o_clk_div, o_segout);
input [3:0] i_a,i_b;
output reg [7:0] o_out;
input i_clk,i_rst;
output reg o_clk_div;
output [6:0] o_segout;
always @(i_a or i_b)
begin
```

```
            o_out= i_a+i_b;
end
localparam constantnumber = 31'd500000;
 reg [31:0] count;
always @ (posedge i_clk, posedge i_rst)
begin
   if (i_rst == 1'b1)
            begin
                 count <= 26'b0;
                o_clk_div= 1'b0;
            end
   else if (count == constantnumber - 1)
            begin
                     count <= 32'b0;
                    o_clk_div = ~o_clk_div;
            end
else
            begin
                    count <= count + 1'b1;
                   o_clk_div=o_clk_div;
            end
end
reg [1:0] activate;
reg [3:0] anode_activate;
reg [3:0] led_bcd;

always @(posedge i_clk)
if (activate == 2'b11)
            activate <= 2'b00;
else
            activate <= activate + 1'b1;
always @(*)
begin
            case(activate)
            2'b00: begin
           anode_activate = 4'b0111;
```

```
         led_bcd = i_a;
           end
       2'b01: begin
          anode_activate = 4'b1011;
          led_bcd =i_b;
           end
       2'b10: begin
          anode_activate = 4'b1101;
          led_bcd =o_out[7:4];
            end
       2'b11: begin
          anode_activate = 4'b1110;
          led_bcd =o_out[3:0];
            end
       endcase
       end
seven_segment hex1(.i_binary(led_bcd), .o_hexdisplay(o_segout));
endmodule
```

Experiment 10
Interfacing Basys3 with Keypad peripheral module

Aim: Interfacing the Basys3 board to a peripheral module PmodKYPD.

Description:

1. Functional Description

Implementation of this program displays the pressed key on the PmodKYPD onto the seven segment display presented on the Basys3 board. The project assumes that the PmodKYPD is connected to one of the 12-pin PMOD (JA, JB, JC) connector on a Basys3 board.

2. Block Description

a) Decoder Behavior

The Decoder determines which key, if any, was pressed on the PmodKYPD by cycling through each column pin with a logic low. After the Decoder sets a column pin low, it checks for a logic low in the row pins. A low in a row pin signifies that a button has been pressed. Once the Decoder has both the row and column of the key, it can determine the corresponding value to output to the Display Controller. The decoder will change columns every 1ms in its cycle.

b) Display Controller Behavior

The Display Controller is used to display the output of the Decoder onto the seven segment display on a Basys3 board. In this project only the rightmost digit on the seven segment display is used. Before any key was pressed, the seven segment display shows a ‘0’ on the rightmost digit. The keypad is represented in hex values. To better differentiate the digits, ‘b’ and ‘d’ are displayed in lowercase.

```
module PmodKYPD(clk,JA,an,seg);
input clk;
inout [7:0] JA;
output [3:0] an;
output [6:0] seg;
wire [3:0] an;
wire [6:0] seg;
wire [3:0] Decode;
Decoder C0(
.clk(clk),
.Row(JA[7:4]),
.Col(JA[3:0]),
.DecodeOut(Decode)
);
DisplayController C1(
.DispVal(Decode),
.anode(an),
.segOut(seg)
);
endmodule
module Decoder(clk,Row,Col,DecodeOut);
input clk;
input [3:0] Row;
output [3:0] Col;
output [3:0] DecodeOut;
reg [3:0] Col;
reg [3:0] DecodeOut;
reg [19:0] sclk;
always @(posedge clk) begin
// 1ms
if (sclk == 20'b00011000011010100000) begin
//C1
Col <= 4'b0111;
sclk <= sclk + 1'b1;
end
// check row pins
else if(sclk == 20'b00011000011010101000) begin
//R1
if (Row == 4'b0111) begin
DecodeOut <= 4'b0001; //1
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b0100; //4
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b0111; //7
```

```
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b0000; //0
end
sclk <= sclk + 1'b1;
end
// 2ms
else if(sclk == 20'b00110000110101000000) begin
//C2
Col<= 4'b1011;
sclk <= sclk + 1'b1;
end
// check row pins
else if(sclk == 20'b00110000110101001000) begin
//R1
if (Row == 4'b0111) begin
DecodeOut <= 4'b0010; //2
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b0101; //5
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b1000; //8
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b1111; //F
end
sclk <= sclk + 1'b1;
end
//3ms
else if(sclk == 20'b01001001001111100000) begin
//C3
Col<= 4'b1101;
```

```
sclk <= sclk + 1'b1;
end
// check row pins
else if(sclk == 20'b01001001001111101000) begin
//R1
if(Row == 4'b0111) begin
DecodeOut <= 4'b0011; //3
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b0110; //6
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b1001; //9
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b1110; //E
end
sclk <= sclk + 1'b1;
end
//4ms
else if(sclk == 20'b01100001101010000000) begin
//C4
Col<= 4'b1110;
sclk <= sclk + 1'b1;
end
// Check row pins
else if(sclk == 20'b01100001101010001000) begin
//R1
if(Row == 4'b0111) begin
DecodeOut <= 4'b1010; //A
end
//R2
else if(Row == 4'b1011) begin
DecodeOut <= 4'b1011; //B
```

```
end
//R3
else if(Row == 4'b1101) begin
DecodeOut <= 4'b1100; //C
end
//R4
else if(Row == 4'b1110) begin
DecodeOut <= 4'b1101; //D
end
sclk <= 20'b00000000000000000000;
end
// Otherwise increment
else begin
sclk <= sclk + 1'b1;
end
end
endmodule
module DisplayController(
DispVal,
anode,
segOut
);
input [3:0] DispVal;
output [3:0] anode;
output [6:0] segOut;
wire [3:0] anode;
reg [6:0] segOut;
assign anode = 4'b0111;
always @(DispVal) begin
case (DispVal)
4'h0 : segOut <= 7'b1000000; // 0
4'h1 : segOut <= 7'b1111001; // 1
4'h2 : segOut <= 7'b0100100; // 2
4'h3 : segOut <= 7'b0110000; // 3
4'h4 : segOut <= 7'b0011001; // 4
4'h5 : segOut <= 7'b0010010; // 5
4'h6 : segOut <= 7'b0000010; // 6
```

```
4'h7 : segOut <= 7'b1111000; // 7
4'h8 : segOut <= 7'b0000000; // 8
4'h9 : segOut <= 7'b0010000; // 9
4'hA : segOut <= 7'b0001000; // A
4'hB : segOut <= 7'b0000011; // B
4'hC : segOut <= 7'b1000110; // C
4'hD : segOut <= 7'b0100001; // D
4'hE : segOut <= 7'b0000110; // E
4'hF : segOut <= 7'b0001110; // F
default : segOut <= 7'b0111111;
endcase
end
```