Experiment No:1 Date: ____________

Combinational circuits design

Experiment No:1a

DECODER (2:4)

Aim: Write a HDL code to design 2:4 DECODER

Verilog

```
module decoder(I,D);
input [1:0] I;
output [3:0] D;
reg [3:0] D;
always @(I)
begin
if (I==2'B00)  D=4'B0001;
else if (I==2'B01) D=4'B0010;
else if (I==2'B10) D=4'B0100;
else if (I==2'B11) D=4'B1000;
else D=4'BZZZZ;
end
endmodule
```

Test Bench:

Results:

Experiment No:1b Date:____________

ENCODER (8:3)

Aim: Write a HDL code to design 8:3 ENCODER with and without priority.

Without priority Verilog

```
module encoder_wop (I,D);
input [7:0] I;
output [2:0] D;
reg [2:0] D;
always @(I)
begin
case (I)
8'd1 : D=3'd0;
8'd2 : D=3'd1;
8'd4 : D=3'd2;
8'd8 : D=3'd3;
8'd16 : D=3'd4;
8'd32 : D=3'd5;
8'd64 : D=3'd6;
8'd128 : D=3'd7;
default: D=3'BZZZ;
endcase
end
endmodule
```

Test Bench:

With Priority Verilog

```
module encoder_wp (I,D);
input [7:0] I;
output [2:0] D;
reg [2:0] D;
always @ (I)
begin
if (I[7]==1'B1)  D=3'B111;
else if (I[6]==1'B1) D=3'B110;
else if (I[5]==1'B1) D=3'B101;
else if (I[4]==1'B1) D=3'B100;
else if (I[3]==1'B1) D=3'B011;
else if (I[2]==1'B1) D=3'B010;
else if (I[1]==1'B1) D=3'B001;
else if (I[0]==1'B1) D=3'B000;
else D=3'BZZZ;
end
endmodule
```

Test Bench:

Results:

Experiment No: 1c Date:_____________

BINARY TO GRAY

Aim: Write a HDL program to convert 4-bit BINARY TO GRAY code.

Verilog

```
module bin_to_gray(B,G);

input [3:0] B;

output [3:0] G;

assign G[3] = B[3];

assign G[2] = B[3]^B[2];

assign G[1] = B[2]^B[1];

assign G[0] = B[1]^B[0];

endmodule
```

Test Bench:

Results:

Experiment No: 1d Date: ____________

GRAY TO BINARY

Aim: Write a HDL program to convert 4-bit GRAY TO BINARY code.

Verilog

```
module gray_to_bin
        (input [3:0] G, //gray code output
         output [3:0] bin  //binary input );
 assign bin[3] = G[3];
assign bin[2] = G[3] ^ G[2];
assign bin[1] = G[3] ^ G[2] ^ G[1];
assign bin[0] = G[3] ^ G[2] ^ G[1] ^ G[0];
endmodule
```

Test Bench:

Results:

Experiment No: 2 Date:

4 BIT PARALLEL ADDER

Aim: Write a HDL program using generate for 4 bit parallel adder using FPGA implementation.

Verilog

```
module padd (a, b,cin,s,cout); input [3:0]a,b;
input cin; output [3:0]s; output cout; wire [3:1]c;
FAA FA0 (a[0],b[0], cin,s[0],c[1]);
FAA FA1 (a[1],b[1],c[1],s[1],c[2]);
FAA FA2 (a[2],b[2],c[2],s[2],c[3]);
FAA FA3 (a[3],b[3],c[3],s[3],cout);
endmodule
module FAA (a,b,ci,s,co); input a,b,ci;
output s,co; wire vdd, gnd;
assign vdd = 1'b1; assign gnd = 1'b0; assign s = a^b^ci;
assign co = (a&b)|(b&ci)|(ci&a); endmodule
```

Test Bench:

padd_tb.v:

```
module padd_tb;
reg [3:0]A,B;
reg Cin; wire [3:0]S; wire Cout;
padd PA1 (A,B,Cin,S,Cout); initial begin
A=4'b0001; B=4'b0010; Cin=1'b1; #2 A=4'b1111; B=4'b1111; Cin=1'b1; #2 A=4'b1001; B=4'b1010;
Cin=1'b1; #2 A=4'b0001; B=4'b0010; Cin=1'b0; #2
$finish; end
Endmodule
```

Results:

Experiment No: 3 Date:

4 BIT CARRY LOOK AHEAD ADDER

Aim: Write HDL code for Carry look ahead adder using FPGA implementation

Verilog

```
module carry_lookahead_adder_4_bit
(input [3:0]  i_add1,
 input [3:0]  i_add2,
 output [4:0] o_result);

wire [4:0]    w_C;
wire [3:0]    w_G, w_P, w_SUM;

full_adder full_adder_bit_0
  (.i_bit1(i_add1[0]), .i_bit2(i_add2[0]), .i_carry(w_C[0]), .o_sum(w_SUM[0]), .o_carry());

full_adder full_adder_bit_1
  ( .i_bit1(i_add1[1]),.i_bit2(i_add2[1]), .i_carry(w_C[1]), .o_sum(w_SUM[1]),.o_carry() );

full_adder full_adder_bit_2
  ( .i_bit1(i_add1[2]),.i_bit2(i_add2[2]), .i_carry(w_C[2]),.o_sum(w_SUM[2]),.o_carry() );

full_adder full_adder_bit_3
  ( .i_bit1(i_add1[3]), .i_bit2(i_add2[3]), .i_carry(w_C[3]),.o_sum(w_SUM[3]),.o_carry());

// Create the Generate (G) Terms:  Gi=Ai*Bi
assign w_G[0] = i_add1[0] & i_add2[0];
assign w_G[1] = i_add1[1] & i_add2[1];
assign w_G[2] = i_add1[2] & i_add2[2];
assign w_G[3] = i_add1[3] & i_add2[3];

// Create the Propagate Terms: Pi=Ai+Bi
assign w_P[0] = i_add1[0] | i_add2[0];
assign w_P[1] = i_add1[1] | i_add2[1];
assign w_P[2] = i_add1[2] | i_add2[2];
assign w_P[3] = i_add1[3] | i_add2[3];

// Create the Carry Terms:
assign w_C[0] = 1'b0; // no carry input
assign w_C[1] = w_G[0] | (w_P[0] & w_C[0]);
assign w_C[2] = w_G[1] | (w_P[1] & w_C[1]);
assign w_C[3] = w_G[2] | (w_P[2] & w_C[2]);
assign w_C[4] = w_G[3] | (w_P[3] & w_C[3]);

assign o_result = {w_C[4], w_SUM};   // Verilog Concatenation
```

```
endmodule // carry_lookahead_adder_4_bit
```

Test Bench:

Results:

Experiment No: 4 Date:

4-BIT ALU

Aim: Write a HDL code to design 4-bit ALU.

Verilog

```
module alu (a, b, code, aluout );
input [3:0] a, b;
input [2:0] code;
output [7:0] aluout;
reg [7:0] aluout ;
wire[7:0] x, y;
assign x = {4'B0000, a};
assign y = {4'B0000, b};
always @ (x, y, code, a,b)
begin
case (code)
3'd0: aluout = x + y;
3'd1: aluout = x - y;
3'd2: aluout = ~x;
3'd3: aluout = a * b;
3'd4: aluout = x & y;
3'd5: aluout = x | y;
3'd6: aluout = ~(x & y);
3'd7: aluout = ~(x | y);
default: aluout = 8'B00000000;
endcase
end
endmodule
```

Test Bench:

Results:

Experiment No: 5 Date:

FLIP-FLOPS

a) D FLIP-FLOP

Aim: Write a HDL code for D-FF.

Verilog

```
module dff(d, rst, clk, q, qb);
input d, rst, clk;
output q, qb;
reg q, qb;
always @(posedge clk)
begin
if (rst == 1'B1)
begin
q = 1'B0; qb = 1'B1;
end
else
begin
q = d; qb = ~q;
end
end
endmodule
```

Test Bench:

Results:

b) T FLIP-FLOP

Aim: Write a HDL code for T-FF.

Verilog

For Simulation:

```
module tff(t, clk, rstn, q, qb);
input t, rstn, clk;
output q, qb;
reg q, qb;
always @(posedge clk)
begin
if (!rstn)
begin
q <= 0;
qb <= 1;
end
else
if   (t)
begin
q = ~q; qb = ~qb;
end
else
begin
q = q; qb = qb;
end
end
endmodule
```

Test Bench:

Hardware Implementation code:

```
module tff1(t,clk,rstn,q,qb);
input t,rstn,clk;
output q,qb;
reg q,qb;
reg clkdiv;
reg[24:0] div;
always@(posedge clk)
begin
div=div+1'b1;
clkdiv=div[24];
end
always@(posedge clkdiv)
begin
if(!rstn)
begin
q<=0;
```

```
    qb<=1;
    end
    else
    if(t)
    begin
    q=~q;
    qb=~qb;
    end
    else
    begin
    q=q;
    qb=qb;
    end
    end
endmodule
```

Results:

c) JK FLIP-FLOP

Aim: Write a HDL code for JK-FF.

Verilog

```
module jkff(jk, clk, q, qb);
input clk;
input [1:0]jk;
output q,qb;
reg q,qb;
always @(posedge clk)
begin
case(jk)
2'B00:q=q;
2'B01:q=1'B0;
2'B10:q=1'B1;
2’B11:q= ~q;
default:q=1'BZ;
endcase
qb=~q;
end
endmodule
```

Test Bench:

Note: Hardware implementation code needs to be written.

Results:

Pin assignments:

```
set_location_assignment PIN_P11 -to CLOCK_50
set_location_assignment PIN_A8 -to LEDR[0]
set_location_assignment PIN_A9 -to LEDR[1]
set_location_assignment PIN_A10 -to LEDR[2]
set_location_assignment PIN_B10 -to LEDR[3]
set_location_assignment PIN_D13 -to LEDR[4]
set_location_assignment PIN_C13 -to LEDR[5]
set_location_assignment PIN_E14 -to LEDR[6]
set_location_assignment PIN_D14 -to LEDR[7]
set_location_assignment PIN_A11 -to LEDR[8]
set_location_assignment PIN_B11 -to LEDR[9]
set_location_assignment PIN_C10 -to SW[0]
set_location_assignment PIN_C11 -to SW[1]
set_location_assignment PIN_D12 -to SW[2]
set_location_assignment PIN_C12 -to SW[3]
set_location_assignment PIN_A12 -to SW[4]
set_location_assignment PIN_B12 -to SW[5]
set_location_assignment PIN_A13 -to SW[6]
set_location_assignment PIN_A14 -to SW[7]
set_location_assignment PIN_B14 -to SW[8]
set_location_assignment PIN_F15 -to SW[9]
set_location_assignment PIN_B8 -to KEY[0]
set_location_assignment PIN_A7 -to KEY[1]
set_location_assignment PIN_C14 -to HEX0[0]
set_location_assignment PIN_E15 -to HEX0[1]
set_location_assignment PIN_C15 -to HEX0[2]
set_location_assignment PIN_C16 -to HEX0[3]
set_location_assignment PIN_E16 -to HEX0[4]
set_location_assignment PIN_D17 -to HEX0[5]
set_location_assignment PIN_C17 -to HEX0[6]
set_location_assignment PIN_C18 -to HEX1[0]
set_location_assignment PIN_D18 -to HEX1[1]
set_location_assignment PIN_E18 -to HEX1[2]
set_location_assignment PIN_B16 -to HEX1[3]
set_location_assignment PIN_A17 -to HEX1[4]
set_location_assignment PIN_A18 -to HEX1[5]
set_location_assignment PIN_B17 -to HEX1[6]
set_location_assignment PIN_B20 -to HEX2[0]
set_location_assignment PIN_A20 -to HEX2[1]
set_location_assignment PIN_B19 -to HEX2[2]
set_location_assignment PIN_A21 -to HEX2[3]
set_location_assignment PIN_B21 -to HEX2[4]
set_location_assignment PIN_C22 -to HEX2[5]
set_location_assignment PIN_B22 -to HEX2[6]
set_location_assignment PIN_F21 -to HEX3[0]
set_location_assignment PIN_E22 -to HEX3[1]
set_location_assignment PIN_E21 -to HEX3[2]
set_location_assignment PIN_C19 -to HEX3[3]
set_location_assignment PIN_C20 -to HEX3[4]
set_location_assignment PIN_D19 -to HEX3[5]
set_location_assignment PIN_E17 -to HEX3[6]
set_location_assignment PIN_F18 -to HEX4[0]
set_location_assignment PIN_E20 -to HEX4[1]
set_location_assignment PIN_E19 -to HEX4[2]
set_location_assignment PIN_J18 -to HEX4[3]
set_location_assignment PIN_H19 -to HEX4[4]
set_location_assignment PIN_F19 -to HEX4[5]
set_location_assignment PIN_F20 -to HEX4[6]

set_location_assignment PIN_J20 -to HEX5[0]

set_location_assignment PIN_K20 -to HEX5[1]

set_location_assignment PIN_L18 -to HEX5[2]

set_location_assignment PIN_N18 -to HEX5[3]

set_location_assignment PIN_M20 -to HEX5[4]

set_location_assignment PIN_N19 -to HEX5[5]

set_location_assignment PIN_N20 -to HEX5[6]

set_location_assignment PIN_AA1 -to red[0]

set_location_assignment PIN_V1 -to red[1]

set_location_assignment PIN_Y2 -to red[2]

set_location_assignment PIN_Y1 -to red[3]

set_location_assignment PIN_W1 -to green[0]

set_location_assignment PIN_T2 -to green[1]

set_location_assignment PIN_R2 -to green[2]

set_location_assignment PIN_R1 -to green[3]

set_location_assignment PIN_P1 -to blue[0]

set_location_assignment PIN_T1 -to blue[1]

set_location_assignment PIN_P4 -to blue[2]

set_location_assignment PIN_N2 -to blue[3]

set_location_assignment PIN_N3 -to hsync

set_location_assignment PIN_N1 -to vsync
```