Demo Command:-

iverilog -o flipflop dlipflop_tb.v
vvp flipflop -lxt2
gtkwave dflipflop.vcd

Majority Circuit:-

Design:-

```
module MAJORITY_CIRCUIT( input a ,input b,input c,output z);

assign z = (a && b) || (b && c) || (c && a);

endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "majoritycircuit.v"

module MAJORITYCIRCUIT_TEST;

    // Inputs
    reg a;
    reg b;
    reg c;

    // Outputs
    wire z;

    // Instantiate the Unit Under Test (UUT)
    MAJORITY_CIRCUIT uut (
        .a(a),
        .b(b),
        .c(c),
        .z(z)
    );

    initial begin
        $dumpfile("majority.vcd");
        $dumpvars(0,uut);
        // Initialize Inputs
        a = 0;
        b = 0;
        c = 0;

        // Wait 100 ns for global reset to finish
        #100;

        a = 0;
        b = 0;
        c = 1;

        // Wait 100 ns for global reset to finish
        #100;

        a = 0;
        b = 1;
        c = 1;

        // Wait 100 ns for global reset to finish
        #100;

        a = 1;
        b = 0;
        c = 1;

        // Wait 100 ns for global reset to finish
        #100;

        a = 1;
        b = 1;
        c = 1;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end
```

```
endmodule
```

Code Conversion

Binary to BCD:-

Design:-

```
module bintobcd (
    input [3:0]a,
    output [4:0]s
);
    assign s[4] = (a[3]&a[2])|(a[3]&a[1]);
    assign s[3] = (a[3]&~a[2]&~a[1]);
    assign s[2] = (~a[3]&a[2])|(a[2]&a[1]);
    assign s[1] = (a[3]&a[2]&~a[1])|(~a[3]&a[1]);
    assign s[0] = (a[0]);
endmodule
```

Testbench:-

```
`timescale 1ns/1ns
`include "bintobcd.v"
module bin_to_bcd_testbench();
  reg [3:0]a;
  wire [4:0]s;

  bintobcd uut(.a(a),.s(s));
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
    #80 $finish;
  end
  initial begin
    $dumpfile("dump.vcd"); $dumpvars(1);
    a= 4'b0000; #10;
    a= 4'b0010; #10;
    a= 4'b0110; #10;
    a= 4'b0111; #10;
    a= 4'b1001; #10;
    a= 4'b1100; #10;
    a= 4'b1110; #10;
    a= 4'b1111; #10;
  end
endmodule
```

Binary to Ex3:-

Design:-

```
module bintoex3 (
    input [3:0]a,
    output [4:0]y
);
reg [4:0]y;
always @(a) begin
    y = a+3;
end
endmodule
```

Testbench:-

```
`timescale 1ns/1ns
`include "bintoex3.v"
module bintoex3_tb;
reg [3:0]a;
wire [4:0]y;
bintoex3 uut(.a(a),.y(y));
initial begin
$dumpfile("dump.vcd");
$dumpvars(1);
#80 $finish;
end
initial begin
a=4'b0000;#5;a=4'b0001;#5;
a=4'b0010;#5;a=4'b0011;#5;
a=4'b0100;#5;a=4'b0101;#5;
a=4'b0110;#5;a=4'b0111;#5;
```

```
a=4'b1000;#5;a=4'b1001;#5;
a=4'b1010;#5;a=4'b1011;#5;
a=4'b1100;#5;a=4'b1101;#5;
a=4'b1110;#5;a=4'b1111;#5;
end
endmodule
```

Ex3 to Binary:-

Design:-

```
module ex3tobin (
    input [3:0]a,
    output [3:0]y
);
reg [3:0]y;
always @(a)
if(a>2)
begin
    y = a-3;
end
endmodule
```

Testbench:-

```
`timescale 1ns/1ns
`include "ex3tobin.v"
module ex3tobin_tb;
reg [3:0]a;
wire [3:0]y;
ex3tobin uut(.a(a),.y(y));
initial begin
$dumpfile("dump.vcd");
$dumpvars(1);
#80 $finish;
end
initial begin
a=4'b0000;#5;a=4'b0001;#5;
a=4'b0010;#5;a=4'b0011;#5;
```

```
a=4'b0100;#5;a=4'b0101;#5;
a=4'b0110;#5;a=4'b0111;#5;
a=4'b1000;#5;a=4'b1001;#5;
a=4'b1010;#5;a=4'b1011;#5;
a=4'b1100;#5;a=4'b1101;#5;
a=4'b1110;#5;a=4'b1111;#5;
end
endmodule
```

Binary to Gray:-

Design:-

```
module BintoGray(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
```

Testbench:-

```
`timescale 1ns/1ns
`include "bintogray.v"
module test_BG;
  reg [3:0]b;
  wire [3:0]g;
  BintoGray uut(.b(b),.g(g));
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    #100 $finish;
  end
  initial begin
    b = 4'b0000;#10; b = 4'b0001;#10;
    b = 4'b0010;#10; b = 4'b0011;#10;
    b = 4'b0100;#10; b = 4'b0101;#10;
    b = 4'b0110;#10; b = 4'b0111;#10;
    b = 4'b1100;#10; b = 4'b1111;#10;
  end
endmodule
```

Gray to Binary:-

Design:-

```
module graytobin (
    input [3:0]a,
    output [3:0]y
);
reg [3:0]y;
always @(a)begin
if(a<4'b1000)
begin
    assign y[3] = 1'b0;
    assign y[2] = a[3]^a[2];
    assign y[1] = a[2]^a[1];
    assign y[0] = a[1]^a[0];
end
else begin
    assign y[3] = 1'b1;
    assign y[2] = a[3]^a[2];
    assign y[1] = a[2]^a[1];
    assign y[0] = a[1]^a[0];
end
end
endmodule
```

Testbench:-

```
`timescale 1ns/1ns
`include "graytobin.v"
module test;
reg [3:0]a;
wire [3:0]y;
graytobin uut(.a(a),.y(y));
initial begin
```

```
$dumpfile("dump.vcd");
$dumpvars(1);
#80 $finish;
end
initial begin
a=4'b0000;#5;a=4'b0001;#5;
a=4'b0010;#5;a=4'b0011;#5;
a=4'b0100;#5;a=4'b0101;#5;
a=4'b0110;#5;a=4'b0111;#5;
a=4'b1000;#5;a=4'b1001;#5;
a=4'b1010;#5;a=4'b1011;#5;
a=4'b1100;#5;a=4'b1101;#5;
a=4'b1110;#5;a=4'b1111;#5;
end
endmodule
```

Flip Flop

D FlipFlop:-

Design:-

```
module dflip (
  D,CLK,Q,Qbar
);
//D,S,R,J,K are synchronous inputs these influence Q according to clk
//Asynchronous inputs on a flip-flop have control over the outputs (Q and
not-Q) regardless of clock input status.
//These inputs are called the preset (PRE) and clear (CLR). The preset
input drives the flip-flop to a set state
//while the clear input drives it to a reset state.

input D;
//input CLR;
input CLK;
output reg Q;
output reg Qbar;

always @(posedge CLK)
 begin
```

```
  Q=D;
  Qbar=~D;
 end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
`include "dflipflop.v"
module dflip_tb;
  reg D;
 // reg CLR;
  reg CLK;
  wire Q;
  wire Qbar;

  dflip d0(D,CLK,Q,Qbar);

  //  always begin
  //   CLK=1;
  //   forever #1;
  //   CLK= ~CLK;

  // end
  // always #1 CLK=~CLK;

  initial begin
    $dumpfile("dflip.vcd");
    $dumpvars(0,d0);
    CLK = 1'b1;
    D=0;#1;
    D=1;#1;
    D=0;#1;
    D=1;#1;
    //$finish;

  $finish;
  end
  always #1 CLK=~CLK;

endmodule
```

SR FlipFlop:-

Design:-

```
//The reset input is used to get back the flip flop to its original state
from the current state with an output 'Q'.
module srflip(
    s,r,clock,reset,q, qb
    );
    input s,r,clock,reset;
    output qb;
    output reg q;

    assign qb = ~q;

always @ (posedge (clock))
    begin
        if (reset)
            begin
               q <= 0;
               //qb <=1;
            end
        else
            begin
               if (s != r)
                    begin
                    q <= s;
                    //qb <= r;
                    end
               else if (s == 1 && r == 1)
                    begin
                    q <= 1'bZ;
                    //qb <= 1'bZ;
                    end
            end
end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
`include "srflipflop.v"
module srflip_tb;

reg s,r,clock,reset;
wire q,qb;

srflip sr(s,r,clock,reset,q,qb);

// always begin
//     clock=1;
//     forever #1;
//     clock= ~clock;
//   end

  initial begin
    $dumpfile("srflip.vcd");
    $dumpvars(0,sr);
clock=1'b0;
  s=1; r=0; reset=0; #1;
  s=0; r=1; reset=1; #1;
  s=1; r=1; reset=0; #1;
  s=0; r=1; reset=1; #1;
  s=1; r=1; reset=0; #1;
 $finish;
  end
always #1 clock = ~clock;


endmodule
```

JK FlipFlop:-

Design:-

```
module jkflip (
  J,K,CLK,Q,Qb
);
```

```
input J,K,CLK;
output Qb;
output reg Q;
//reg Q,Qb;
assign Qb=~Q;
always @(posedge CLK )
 begin
   case ({J,K})
    2'b00 : Q<=Q;
    2'b01 : Q<=0;
    2'b10 : Q<=1;
    2'b11 : Q<=~Q;
   endcase
end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
`include "jkflipflop.v"
module jkflip_tb;
reg J;
reg K;
reg CLK;
wire Q;
wire Qb;

jkflip jk(J,K,CLK,Q,Qb);
// initial begin
//   CLK=1;
//   forever #2;
//   CLK=~CLK;
// end
initial
 begin
   $dumpfile("jkflip.vcd");
   $dumpvars(0,jk);
   CLK=1'b1;
   J=1;K=0;#1;
   J=0;K=1;#1;
   J=0;K=0;#1;
   J=1;K=1;#1;
   $finish;
 end
 always #1 CLK = ~CLK;
endmodule
```

T FlipFlop:-

Design:-

```
module tflip(
  clk, rstn, t, q
  );
input clk,rstn,t;
output reg q;

  always @ (posedge clk) begin
    if (!rstn)
      q <= 0;
    else
        if (t)
            q <= ~q;
        else
            q <= q;
  end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
`include "tflipflop.v"
module tflip_tb;
  reg clk;
  reg rstn;
  reg t;
  wire q;

  tflip tf ( clk,rstn,t,q );

  // always begin
  //   clk=1;
  //   forever #1;
  //   clk= ~clk;
  // end

  initial begin
    $dumpfile("tflip.vcd");
    $dumpvars(0,tf);
    rstn=1'b0;  //rstn 0 as it starts if !rstn tflip basically works on
previous input
    clk = 1'b1;
    t=1;#1;
    t=1;#1;
    t=0;#1;
    t=1;#1;
   $finish;
  end
  always #1 clk = ~clk;
endmodule
```

Half Adder:-

Design:-

```
module half_adder (
    a,b,s,c
);
    input a,b;
    output s,c;
    assign s=a^b;
    assign c=a&b;

endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "half_adder.v"

module half_adder_tb;
    reg a ,b;
    wire s,c;

    half_adder add1(a,b,s,c);

    initial
        begin
            $dumpfile("half_adder.vcd");
            // $dumpvars(0,half_adder_tb);
            $dumpvars(0,add1);

            a=0;b=0;
            #1;
            a=0;b=1;
            #1;
            a=1;b=0;
            #1;
            a=1;b=1;
            #1;

        end
endmodule
```

Full Adder:-

Design:-

```
module fulladder(
  a, b, cin, s, cout
  );
  input a,b,cin;
  output s,cout;
  assign s=a^b^cin;
  assign cout=a&b|b&cin|cin&a;

endmodule
```

TestBench:-

```
`timescale 1ps/1ps
```

```
module fulladder_tb;
  reg a ,b,cin;
  wire s,cout;

  fulladder add2(a,b,cin,s,cout);

  initial
     begin
      $dumpfile("fulladder.vcd");
      $dumpvars(0,add2);

      a=0;b=0;cin=0;#1;
      a=0;b=0;cin=1;#1;
      a=0;b=1;cin=0;#1;
      a=0;b=1;cin=1;#1;
      a=1;b=0;cin=0;#1;
      a=1;b=0;cin=1;#1;
      a=1;b=1;cin=0;#1;
      a=1;b=1;cin=1;#1;
     end
endmodule
```

Adder/Subtractor:-

Design:-

```
`include "full_adder.v"

module addsub(
  A,B,M,C,S
);

 input [3:0]A;
 input [3:0]B;
 input M;
 output [3:0]S;
 output C;

 wire C1,C2,C3;
```

```verilog
fulladder F1(.a(A[0]),.b(B[0]^M),.cin(M),.cout(C1),.s(S[0]));
fulladder F2(.a(A[1]),.b(B[1]^M),.cin(C1),.cout(C2),.s(S[1]));
fulladder F3(.a(A[2]),.b(B[2]^M),.cin(C2),.cout(C3),.s(S[2]));
fulladder F4(.a(A[3]),.b(B[3]^M),.cin(C3),.cout(C),.s(S[3]));

endmodule
```

TestBench:-

```verilog
`timescale 1ps/1ps
`include "add_sub.v"
module addsub_tb;

reg [3:0]A;
reg [3:0]B;
reg M;
wire C;
wire [3:0]S;

addsub add3(A,B,M,C,S);
initial
  begin
    $dumpfile("addsub.vcd");
    $dumpvars(0,add3);

    A=4'b0100;B=4'b0011;M=0;#1;
    A=4'b0100;B=4'b1000;M=0;#1;
    A=4'b1001;B=4'b1010;M=0;#1;
    A=4'b0111;B=4'b1100;M=1;#1;
    A=4'b0001;B=4'b1010;M=1;#1;
    A=4'b1111;B=4'b0110;M=1;#1;

  end
endmodule
```

3 to 8 Decoder:-

Design:-

```
//3 to 8 Decoder
//3to8decoder.v
module decoder3to8(
    D,enable,A
);
    input [2:0] A;
    input enable;
    output reg [7:0] D;

    always @(*)
    begin
        case (A)
            3'b000: D = 8'b00000001;
            3'b001: D = 8'b00000010;
            3'b010: D = 8'b00000100;
            3'b011: D = 8'b00001000;
            3'b100: D = 8'b00010000;
            3'b101: D = 8'b00100000;
            3'b110: D = 8'b01000000;
            3'b111: D = 8'b10000000;
            default: D = 8'bx;
        endcase
    end
endmodule
```

TestBench:-

```
//3 to 8 decoder testbench
//"3to8decoder_tb.v"
`timescale 1ps/1ps
`include "3to8decoder.v"

module decoder3to8_tb();

reg [2:0] A;
reg enable;
wire [7:0] D;

decoder3to8 uut(D,enable,A);
```

```
initial begin
    $dumpfile("3to8decoder_dump.vcd");
    $dumpvars(0,decoder3to8_tb);
    enable = 1;

    A = 3'b000; #20;
    A = 3'b001; #20;
    A = 3'b010; #20;
    A = 3'b011; #20;
    A = 3'b100; #20;
    A = 3'b101; #20;
    A = 3'b110; #20;
    A = 3'b111; #20;
    $finish;
end

endmodule
```

Priority Encoder:-

Design:-

```
//Priority Encoder
module priorityEncoder(
    input [3:0] Y,
    output [1:0] A
);
    assign A[1] = Y[2]||Y[3];
    assign A[0] = Y[3]||(Y[1]&&(!Y[2]));
endmodule
```

TestBench:-

```
//Priority Encoder Testbench
`timescale 1ps/1ps
`include "priorityEncoder.v"

module priorityEncoder_tb();
reg [3:0] Y;
wire [1:0] A;
```

```
priorityEncoder uut(Y,A);
initial begin
    $dumpfile("priorityEncoder_dump.vcd");
    $dumpvars(0,priorityEncoder_tb);
    // Y = 4'b0000; #10;
    Y = 4'b0001; #10;
    Y = 4'b0110; #10;
    Y = 4'b1001; #10;
    Y = 4'b0011; #10;
end
endmodule
```

Comparator:-

Design:-

```
// Magnitude Comparator
module comparator(a,b,eq,lt,gt);

input [3:0] a,b;

output reg eq,lt,gt;

always @(a,b)
begin
 if (a==b)
 begin
  eq = 1'b1;
  lt = 1'b0;
  gt = 1'b0;
 end
 else if (a>b)
 begin
  eq = 1'b0;
  lt = 1'b0;
  gt = 1'b1;
 end
 else
```

```
   begin
  eq = 1'b0;
  lt = 1'b1;
  gt = 1'b0;
 end
end
endmodule
```

TestBench:-

```
// Magnitude Comparator TestBench
`timescale 1ps/1ps
`include "comparator.v"

module comparator_tst;
  reg [3:0] a,b;
  wire eq,lt,gt;

  comparator UUT (a,b,eq,lt,gt);


  initial
  begin

   $dumpfile("dump.vcd");
   $dumpvars(0,comparator_tst);

   a = 4'b1100;
   b = 4'b1100;
   #7;

   a = 4'b0100;
   b = 4'b1100;
   #7;

   a = 4'b1111;
   b = 4'b1100;
   #7;
    a = 4'b0000;
   b = 4'b0000;
   #7;

  end
endmodule
```

***Design(Prattay’s Version):-**

```
`include "full_adder.v"
module COMPARATOR(  input [3:0]a, input [3:0]b, output l, output e, output
g);
wire s0,s1,s2,s3,c0,c1,c2,c3;
fulladder FA1 (
        .a(a[0]),
        .b(~b[0]),
        .cin(1'b1),
        .s(s0),
        .cout(c0)

    );
fulladder FA2 (
        .a(a[1]),
        .b(~b[1]),
        .cin(c0),
        .s(s1),
        .cout(c1)

    );
fulladder FA3 (
        .a(a[2]),
        .b(~b[2]),
        .cin(c1),
        .s(s2),
        .cout(c2)

    );
fulladder FA4 (
        .a(a[3]),
        .b(~b[3]),
        .cin(c2),
        .s(s3),
        .cout(c3)

    );

assign e = c3 && ~s0 && ~s1 && ~s2 && ~s3;
assign g = c3 && ~e;
assign l = ~c3;

endmodule
```

***TestBench(Prattay's Version):-**

```
`timescale 1ps/1ps
`include "comparator_1.v"
module COMPARATOR_TEST;

    // Inputs
    reg [3:0] a;
    reg [3:0] b;

    // Outputs
    wire l;
    wire e;
    wire g;

    // Instantiate the Unit Under Test (UUT)
    COMPARATOR uut (
        .a(a),
        .b(b),
        .l(l),
        .e(e),
        .g(g)
    );

    initial begin
        $dumpfile("com1.vcd");
        $dumpvars(1);
        // Initialize Inputs
        a = 0000;
        b = 0001;

        // Wait 100 ns for global reset to finish
        #100;

        a = 0010;
        b = 0001;

        // Wait 100 ns for global reset to finish
        #100;


        a = 0010;
        b = 0010;

        // Wait 100 ns for global reset to finish
        #100;

        a = 1000;
        b = 0010;

        // Wait 100 ns for global reset to finish
        #100;
        // Add stimulus here

    end

endmodule
```

Up Down Counter:-

Design:-

```
module updown(
  clk,reset,updown,count
);

input clk,reset,updown;
output [3:0] count;
```

```
reg [3:0] count = 0;

always @(posedge(clk) or posedge(reset))
 begin
   if(reset==1)
   count <= 0;
   else
    if(updown==1)
     if(count==15)//upmode:counter counts from 0 to 15
      count <= 0;
     else
      count <= count+1;//increment
    else//downmode:15 to 0
     if(count == 0)
      count<=15;
     else
      count<=count-1; //decrement
 end
endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "updowncounter.v"
module updown_tb;

reg clk;
reg reset;
reg updown;
wire [3:0]count;


updown ud(clk,reset,updown,count);

initial begin
  $dumpfile("updown.vcd");
  $dumpvars(0,ud);
  clk = 1'b0;
  reset= 0;updown=0;#15;
  updown=1;#15;
  reset =1;updown=0;#15;
  reset=0;
  $finish;
end
always #1 clk=~clk;
endmodule
```

Random Counter:-

Design:-

```
module random(
  clk,reset,count
);
 input clk,reset;
 output reg [3:0]count;

always @(posedge clk)
 begin
   if(reset)
     count <= 4'b0000;
   else if(count==4'b0000)
     count <= 4'b0001;
   else
     begin
        count[3] <= count[3] ^ count[0];
        count[2] <= count[3];
        count[1] <= count[2];
        count[0] <= count[1] ;
     end
 end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
module random_tb;
 reg clk;
 reg reset;
 wire [3:0] count;
 random r2(clk,reset,count);

 initial begin
    $dumpfile("random.vcd");
    $dumpvars(0,r2);

    clk=1;
    reset=1;#15;
    reset=0;#15;

    $finish;
 end
 always #1 clk=~clk;
endmodule
```

Johnson Counter:-

Design:-

```
module johnson(
  clk,reset,count_out
);

input clk,reset;
output [3:0] count_out;
reg [3:0] count_temp;

always @(posedge(clk),reset)
 begin
   if(reset == 1'b1)
   begin
     count_temp = 4'b0001;
   end
   else if (clk==1'b1)
    begin
      count_temp = {count_temp[2:0],~count_temp[3]};
    end
 end
 assign count_out = count_temp;
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
module johnson_tb;

reg clk,reset;
wire [3:0] count_out;

johnson j1(clk,reset,count_out);

initial begin
  $dumpfile("johnson.vcd");
  $dumpvars(0,j1);

  clk=0;
  reset = 1; #15;
  reset = 0; #15;
  reset = 0;
$finish;
end
always #1 clk = ~clk;

endmodule
```

RING COUNTER:-

Design:-

```
module ring(
  clk,reset,count_out
);

input clk,reset;
output [3:0] count_out;
reg [3:0] count_temp;
```

```
always @(posedge(clk),reset)
 begin
   if(reset == 1'b1)
    begin
      count_temp = 4'b0001;
    end
    else if(clk == 1'b1)
     begin
        count_temp = {count_temp[2:0],count_temp[3]};
     end
 end
 assign count_out = count_temp;
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
module ring_tb;
 reg clk,reset;
 wire [3:0] count_out;

 ring r1(clk,reset,count_out);

 initial begin
   $dumpfile("ring.vcd");
   $dumpvars(0,r1);

   clk = 1;
   reset = 1; #15;
   reset = 0; #15;
   $finish;
 end
 always #1 clk=~clk;
endmodule
```

MOD 10 :-

Design:-

```
module mod(
  clk,rstn,out
);

input clk,rstn;
output reg [3:0]out;

always @(posedge clk)
begin
  if(rstn)
  begin
    out <= 0;
  end
  else
  begin
    if(out == 4'b1001)
    out <= 0;
    else
    out <= out + 1;
  end
end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
module mod_tb;
reg clk,rstn;
wire [3:0]out;

mod m1(clk,rstn,out);
initial begin
  $dumpfile("mod.vcd");
  $dumpvars(0,m1);

  clk=1;
  rstn = 1; #10;
  rstn = 0; #10;
  rstn = 1; #10;
  rstn = 0; #10;
$finish;
end
always #1 clk=~clk;
```

```
endmodule
```

GPR:-

Design:-

```
module gpr(
  in,mode,op
);
input [3:0] in;
input mode;
reg [3:0]q;

output reg [3:0]op;
always @(*)
begin
  if(mode==0)
  begin
    q=in;
  end
  else
  begin
    op=q;
  end
end
endmodule
```

Testbench:-

```
`timescale 1ps/1ps
module gpr_tb;
reg [3:0]in;
reg mode;
wire [3:0]op;

gpr g1(in,mode,op);
initial begin
  $dumpfile("gpr.vcd");
  $dumpvars(0,g1);
  mode=1'b0;in=4'b1001;#10;
  mode=1'b1;in=4'b1000;#10;
  mode=1'b0;in=4'b1110;#10;
  mode=1'b1;in=4'b1011;#10;
end
endmodule
```

SISO:-

Design:-

```
module siso(input clk,input si,output so);
reg [3:0]q=0;
always @(posedge clk)
begin
q[3]<=si;
q[2]<=q[3];
q[1]<=q[2];
q[0]<=q[1];


end
assign so = q[0];
endmodule
```

TestBench:-

```
`timescale 1ns/1ps
`include "siso.v"
module siso_test;
reg clk;
reg si;
wire so;

siso uut(clk,si,so);

initial begin
$dumpfile("siso1.vcd");
```

```
$dumpvars(0,uut);
end
initial begin
clk=0;
forever #50 clk=~clk;
end

initial begin

si = 1;
#10;
si=0;
#10;
si = 1;
#10;
si=0;
// #10;
end

endmodule
```

SIPO:-

Design:-

TestBench:-

PISO:-

Design:-

```
module piso(input clk,input mode,input [3:0]data_in,output so);
reg [3:0]q=0;
always @(posedge clk)
begin
if(mode)      //mode =1 for parallel load and mode = 0 serial output
 q<=data_in;
else
```

```
begin
 q[3]<=1'bx;
 q[2]<=q[3];
 q[1]<=q[2];
 q[0]<=q[1];
end
end
assign so=q[0];
endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "piso.v"
module pisotest;
reg clk;
reg mode;
reg [3:0]data_in;
wire so;

piso uut2(clk,mode,data_in,so);

initial begin
$dumpfile("piso.vcd");
$dumpvars(1);
end

initial begin
clk=0;
repeat(96)     //icarus verilog e repeat bhalo kaaj kore nahole kirokom
simulation time puro boro
      #50 clk=~clk;
end

initial begin
data_in=4'b1001;
mode = 1;
#100;

mode = 0;
```

```
end
endmodule
```

PIPO:-

Design:-

```
module pipo(input clk,input [3:0]d,output reg [3:0]q);
always @(posedge clk)
begin
q[3]<=d[3];
q[2]<=d[2];
q[1]<=d[1];
q[0]<=d[0];
end

endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "pipo.v"
module pipotest;
reg clk;

reg [3:0]d;
wire [3:0]q;

pipo uut2(clk,d,q);

initial begin
$dumpfile("pipo.vcd");
$dumpvars(1);
end

initial begin
clk=0;
repeat(96)
      #50 clk=~clk;
end
```

```
initial begin
d=4'b1001;

#100;

d=4'b1011;
end
endmodule
```

Ram:-

Design:-

```
module ram(input clk,input we ,input cs,input [3:0]data_in,input [3:0]add,
output reg [3:0]data_out);
reg [3:0]mem[3:0];
always @(posedge clk) begin
if(cs) begin
  if(~we) begin
    mem[add] <= data_in;
     end
  else  begin
    data_out <= mem[add];
     end
end
end
endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "ram.v"

module ramtest;
reg we,cs; reg clk; reg [3:0]data_in; reg [3:0]add;
wire [3:0]data_out;

ram uut(clk,we,cs,data_in,add,data_out);
```

```

initial begin
$dumpfile("ram.vcd");
$dumpvars(1);
end

initial begin
clk=0;
repeat(96)
      #50 clk=~clk;
end

initial begin
cs=1;we=0;
add=4'b0000;
data_in=4'b0010;
#100;
add=4'b0001;
data_in=4'b0011;
#100;
add=4'b0010;
data_in=4'b1000;
#100;
add=4'b0011;
data_in=4'b1111;
#100;
we=1; add=4'b0000;#100;
we=1; add=4'b0001;#100;
we=1; add=4'b0010;#100;
we=1; add=4'b0011;
end
endmodule
```

HRAM:-

Design:-

```
`include "ram.v"
```

```
module hram(input clk,input we,input cs,input [7:0]data_in,input
[3:0]add,output [7:0]data_out);

ram r1(clk,we,cs,data_in[3:0],add,data_out[3:0]);
ram r2(clk,we,cs,data_in[7:4],add,data_out[7:4]);
endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "hram.v"
module hram_test;
reg clk; reg we;
reg cs;
reg [7:0]data_in;
reg [3:0]add;
wire [7:0]data_out;

hram uut(clk,we,cs,data_in,add,data_out);

initial begin
$dumpfile("hram1.vcd");
$dumpvars(1);
end
initial begin
clk =0;
forever #5 clk=~clk;
end

initial begin
cs = 1; we = 0;
add =4'b0000;
data_in=8'b00000011;
#100;
add = 4'b0001;
data_in=8'b00010010;
#100;
we=1; add =4'b0000;#100;
we=1; add =4'b0001;
end
```

```
endmodule
```

VRAM:-

Design:-

```
`include "ram.v"
module vram(input clk,input we,input cs,input [3:0]data_in,input
[4:0]add,output [3:0]data_out);

ram r3(clk,we,add[4],data_in,add[3:0],data_out);
ram r4(clk,we,~add[4],data_in,add[3:0],data_out);

endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "vram.v"
module vramtest;
reg we,cs; reg clk; reg [3:0]data_in; reg [4:0]add;
wire [3:0]data_out;

vram uut1(clk,we,cs,data_in,add,data_out);

initial begin
$dumpfile("vram.vcd");
$dumpvars(1);
end

initial begin
clk=0;
repeat(96)
      #50 clk=~clk;
end

initial begin
```

```
we=0;
add=5'b00000;
data_in=4'b0010;
#100;
add=5'b10010;
data_in=4'b0011;
#100;
add=5'b00101;
data_in=4'b1000;
#100;
add=5'b10110;
data_in=4'b1111;
#100;
we=1; add=4'b0000;#100;
we=1; add=4'b0010;#100;
we=1; add=4'b0101;#100;
we=1; add=4'b0110;
end


endmodule
```

ALU:-

Design:-

```
module ALU(input [4:0]a,input [4:0]b,input cin,output reg
[7:0]result,input [3:0]alusel);
always @(*)
begin
   case(alusel)
   4'b0000 : result = a - b;
   4'b0001 : result = a + b;
   4'b0010 : result = a + b + cin;
   4'b0011 : result = a - b + (~cin);
   4'b0100 : result = a ^ b;
   4'b0101 : result = ~b;
   4'b0110 : result = a | b;
   4'b0111 : result = a & b;
   default : result = a + b;
   endcase
end
endmodule
```

TestBench:-

```
`timescale 1ps/1ps
`include "alu.v"
module alu_test;
reg [4:0]a;
reg [4:0]b;
reg  cin;
wire [7:0]result;
reg [3:0]alusel;

ALU uut(a,b,cin,result,alusel);

initial begin
$dumpfile("alu.vcd");
$dumpvars(1);
end

initial begin
alusel = 4'b0000;
a = 4'b0011;
b = 4'b0001;
cin = 1'b0;
#100;

alusel = 4'b0001;
a = 4'b0001;
b = 4'b0011;
cin = 1'b0;
#100;

alusel = 4'b0010;
a = 4'b0001;
b = 4'b0111;
cin = 1'b1;
```

```
#100;

alusel = 4'b0111;
a = 4'b0111;
b = 4'b0111;
cin = 1'b0;
#100;

alusel = 4'b0110;
a = 4'b0001;
b = 4'b0111;
cin = 1'b0;
#100;

end
endmodule
```

MUX4:1:-
Design:-

```
module mux(
  s0,s1,i0,i1,i2,i3,f
);
input s0,s1;
input i0,i1,i2,i3;
output f;

assign f=((~s1)&(~s0)&i0)|((~s1)&(s0)&i1)|((s1)&(~s0)&i2)|(s1&s0&i3);

endmodule
```

Testbench:-

```
`timescale 1ps/1ps

module mux_tb;
reg s0,s1;
reg i0,i1,i2,i3;
wire f;

mux a0(s0,s1,i0,i1,i2,i3,f);
```

```
initial
begin
  $dumpfile("mux.vcd");
  $dumpvars(0,a0);

  i0=1;i1=1;i2=0;i3=0;s0=0;s1=0;#1;
  i0=1;i1=1;i2=0;i3=0;s0=0;s1=1;#1;
  i0=1;i1=1;i2=1;i3=1;s0=0;s1=0;#1;
  i0=1;i1=1;i2=0;i3=1;s0=0;s1=0;#1;

end
endmodule
```

8:1 MUX:-
Design:-

```
`include "mux.v"
module mux8(
  s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7,f
);

input s0,s1,s2;
input d0,d1,d2,d3,d4,d5,d6,d7;
output f;

wire ii0,ii1;

mux m1(.s0(s0),.s1(s1),.i0(d0),.i1(d1),.i2(d2),.i3(d3),.f(ii0));
mux m2(.s0(s0),.s1(s1),.i0(d4),.i1(d5),.i2(d6),.i3(d7),.f(ii1));
mux m3(.s0(s2),.s1(1’b0),.i0(ii0),.i1(ii1),.i2(1'b0),.i3(1'b0),.f(f));

endmodule
```

Testbench:-

```
`timescale 1ps/1ps
module mux8_tb;
   reg s0,s1,s2;
   reg d0,d1,d2,d3,d4,d5,d6,d7;
   wire f;
   mux8 a1(s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7,f);
   initial
   begin
     $dumpfile("mux8.vcd");
     $dumpvars(0,a1);

   s0=0;s1=0;s2=0;d0=1;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;#1;
   s0=0;s1=0;s2=1;d0=1;d1=1;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;#1;
   s0=0;s1=1;s2=0;d0=1;d1=1;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;#1;
   s0=1;s1=0;s2=0;d0=1;d1=0;d2=0;d3=1;d4=1;d5=1;d6=1;d7=0;#1;
   end
endmodule
```

ROM:-
DESIGN:-

```
module rom(input clk,input rd,input [3:0]add,output reg [3:0]out);
reg [3:0]rom[15:0];
initial begin
rom[4'b0001]=4'b0010;
rom[4'b0010]=4'b0100;
rom[4'b0011]=4'b0111;
rom[4'b0111]=4'b0101;
rom[4'b1001]=4'b1111;
end
always @(rd,add)
begin
 if(rd)
   begin
    out <= rom[add];
   end
end
endmodule
```

Romtest

```
module romtest;
reg clk;
reg rd;
reg [3:0]add;
wire [3:0]out;
```

```
rom uut3(clk,rd,add,out);

initial begin
$dumpfile("rom.vcd");
$dumpvars(1);
end

initial begin
clk=0;
repeat(96)
    #50 clk=~clk;
end

initial begin
rd=0;#100;
rd=1;add=4'b0001;#100;
rd=1;add=4'b0010;#100;
rd=1;add=4'b0011;#100;
rd=1;add=4'b0111;#100;
rd=1;add=4'b1001;
end
endmodule
```