Assignments-1

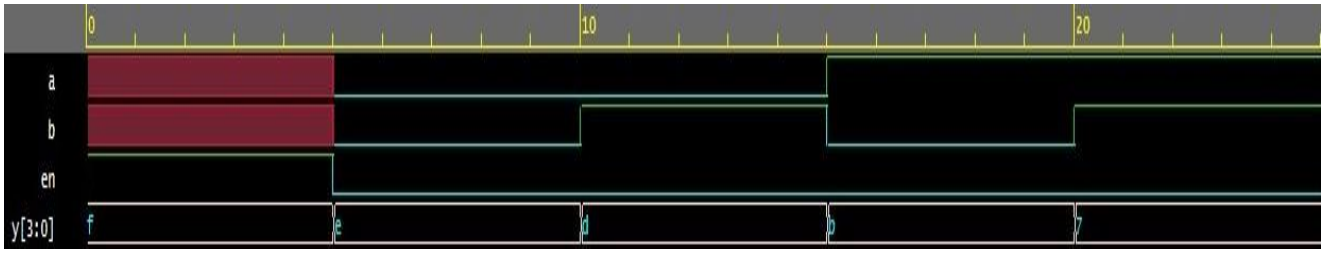
1.Write a Verilog code for 2X4 decoder.

Program :-

```
module tb;


  reg a,b,en;


  wire [3:0]y;


  decoder24_behaviour dut(en,a,b,y);

  initial
    begin
      $dumpvars(1);
      $dumpfile("dump.vcd");
      $monitor("en=%b a=%b b=%b y=%b",en,a,b,y);

      en=1;a=1'bx;b=1'bx;#5
      en=0;a=0;b=0;#5
      en=0;a=0;b=1;#5
      en=0;a=1;b=0;#5
      en=0;a=1;b=1;#5


      $finish;
    end
endmodule
```

```
module decoder24_behaviour(en,a,b,y);

   input en,a,b;


   output reg [3:0]y;



   always @(en,a,b)
     begin

        if(en==0)
          begin
            if(a==1'b0 & b==1'b0) y=4'b1110;
            else if(a==1'b0 & b==1'b1) y=4'b1101;
            else if(a==1'b1 & b==1'b0) y=4'b1011;
            else if(a==1 & b==1) y=4'b0111;
            else y=4'bxxxx;
          end
        else
         y=4'b1111;
     end
endmodule
```

Output :-



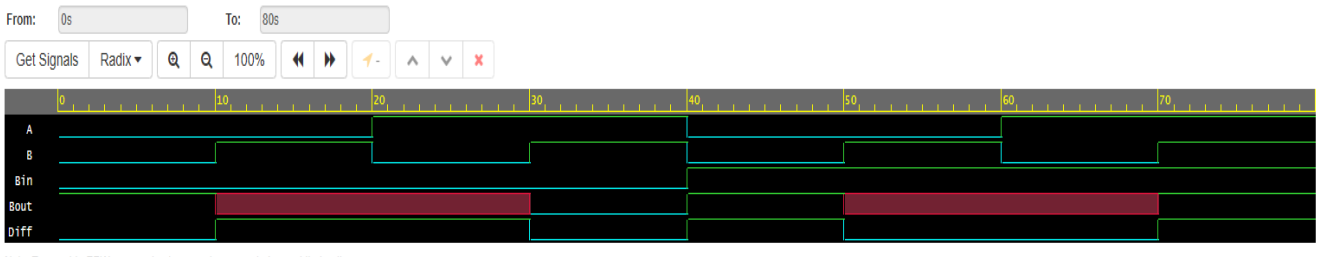
2. Write a Verilog code for Full subtractor.

Program :-

```
module tb_full_subtractor;

    reg A;
    reg B;
    reg Bin;
    wire Diff;
    wire Bout;

    // Instantiate the Full_Subtractor module
    Full_Subtractor uut (
        .A(A),
        .B(B),
        .Bin(Bin),
        .Diff(Diff),
        .Bout(Bout)
    );

    // Stimulus
    initial begin
      $dumpvars(1);
      $dumpfile("dump.vcd");

        $monitor("Time = %0t: A = %b, B = %b, Bin = %b, Diff = %b, Bout = %b",
                 $time, A, B, Bin, Diff, Bout);

        // Test cases
        A = 0; B = 0; Bin = 0; #10;
        A = 0; B = 1; Bin = 0; #10;
        A = 1; B = 0; Bin = 0; #10;
        A = 1; B = 1; Bin = 0; #10;
        A = 0; B = 0; Bin = 1; #10;
        A = 0; B = 1; Bin = 1; #10;
        A = 1; B = 0; Bin = 1; #10;
        A = 1; B = 1; Bin = 1; #10;

        $finish;
    end

endmodule
```

```
module Full_Subtractor(
    input wire A,
    input wire B,
    input wire Bin,
    output wire Diff,
    output wire Bout
);

    wire X1, X2, X3, Y1, Y2, Z;

    // XOR gates
    xor X1_gate(X1, A, B);
    xor X2_gate(Diff, X1, Bin);
    xor X3_gate(X3, ~A, B);

    // AND gates
    and Y1_gate(Y1, ~A, ~B);
    and Y2_gate(Y2, ~X1, Bin);
    and Z_gate(Z, ~X3, ~X2);

    // OR gate
    or Bout_gate(Bout, Y1, Y2, Z);

endmodule

```

Output :-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

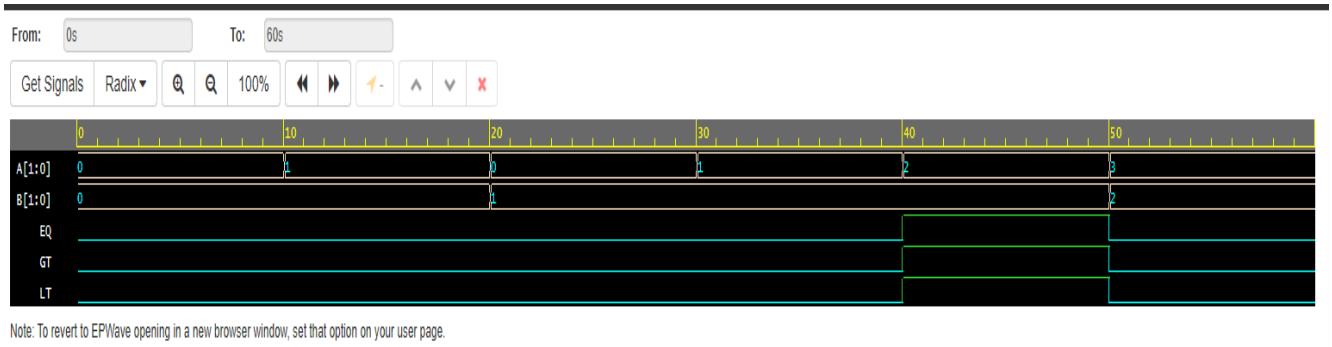
3. Write a Verilog code for 2-bit comparator.

Program :-

```
module tb_comparator_2bit;

    reg [1:0] A;
    reg [1:0] B;
    wire EQ;
    wire GT;
    wire LT;

    // Instantiate the Comparator_2bit module
    Comparator_2bit uut (
        .A(A),
        .B(B),
        .EQ(EQ),
        .GT(GT),
        .LT(LT)
    );

    // Stimulus
    initial begin
      $dumpvars(1);
      $dumpfile("dump.vcd");


        $monitor("Time = %0t: A = %b, B = %b, EQ = %b, GT = %b, LT = %b",
                 $time, A, B, EQ, GT, LT);

        // Test cases
        A = 2'b00; B = 2'b00; #10;
        A = 2'b01; B = 2'b00; #10;
        A = 2'b00; B = 2'b01; #10;
        A = 2'b01; B = 2'b01; #10;
        A = 2'b10; B = 2'b01; #10;
        A = 2'b11; B = 2'b10; #10;

        $finish;
    end

endmodule
```

```
module Comparator_2bit(
    input wire [1:0] A,
    input wire [1:0] B,
    output wire EQ,  // Equal
    output wire GT,  // Greater than
    output wire LT   // Less than
);

    // XOR gates for equality comparison
    assign EQ = (A[0] ^ B[0]) & (A[1] ^ B[1]);

    // AND gate for greater than comparison
    assign GT = (A[1] & ~B[1]) | ((A[1] ^ B[1]) & (A[0] & ~B[0]));

    // AND gate for less than comparison
    assign LT = (~A[1] & B[1]) | ((A[1] ^ B[1]) & (~A[0] & B[0]));

endmodule

```

Output :-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

4. Write a Verilog code for 3 bit binary to gray convertor.

Program :-

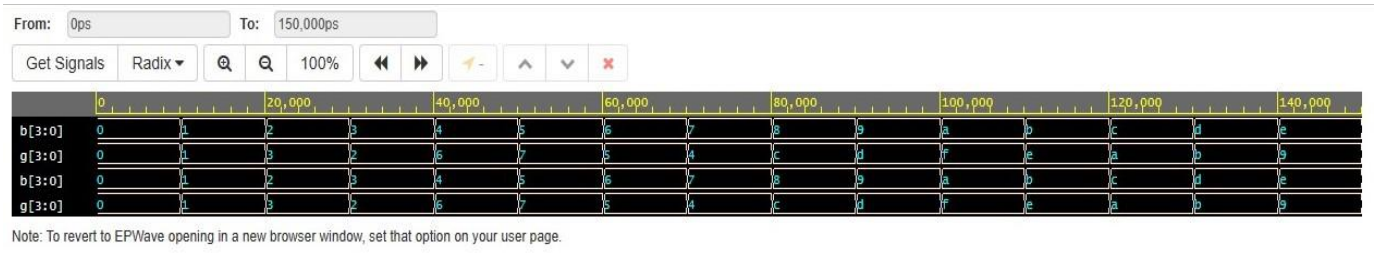
SV/Verilog Testbench

```
//techtbench
`timescale 1ns / 1ps
module Binary_to_Gray_tb;
       reg [3:0]b;
       wire [3:0]g;

       Binary_to_Gray uut (b,g);

       initial begin
         $dumpfile("dump.vcd"); $dumpvars;
       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'b1000;
       #10   b=4'b1001;
       #10   b=4'b1010;
       #10   b=4'b1011;
       #10   b=4'b1100;
       #10   b=4'b1101;
       #10   b=4'b1110;
       #10   b=4'b1111;
       end
endmodule
```

SV/Verilog Design

```
`timescale 1ns / 1ps
module Binary_to_Gray(
    input [3:0] b,
    output [3:0] g
    );
assign g[0]=b[1]^b[0];
assign g[1]=b[2]^b[1];
assign g[2]=b[3]^b[2];
assign g[3]=b[3];
endmodule
```

Output :-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

5. Write a Verilog code for BCD to excess 3 convertors.

Program :-

SV/Verilog Testbench

```
// Code your testbench here
// or browse Examples

module Excess_3;
reg a, b,c,d;
wire w,x,y,z;
  BCD_to_Excess_3
e1(.W(w),.X(x),.Y(y),.Z(z),.A(a),.B(b),.C(c),.D(d));
initial begin
$dumpfile("dump.vcd");
$dumpvars(1);
    a = 0; b=0; c=0; d=0;
#10 a = 0; b=0; c=0; d=1;
#10 a = 0; b=0; c=1; d=0;
#10 a = 0; b=0; c=1; d=1;
#10 a = 0; b=1; c=0; d=0;
#10 a = 0; b=1; c=0; d=1;
#10 a = 0; b=1; c=1; d=0;
#10 a = 0; b=1; c=1; d=1;
#10 a = 1; b=0; c=0; d=0;
#10 a = 1; b=0; c=0; d=1;
#10 a = 1; b=0; c=1; d=0;
#10 a = 1; b=0; c=1; d=1;
#10 a = 1; b=1; c=0; d=0;
#10 a = 1; b=1; c=0; d=1;
#10 a = 1; b=1; c=1; d=0;
#10 a = 1; b=1; c=1; d=1;
#10;
end
endmodule
```

```
// Code your design here

module BCD_to_Excess_3(W,X,Y,Z,A,B,C,D);
input A, B,C,D;
output W,X,Y,Z;

wire xor1,or1,and1;
  not(Z, D);
  xor (xor1, C, D);
  not (Y, xor1);
  or (or1, C, D);
  xor (X,or1,B);
  and (and1, or1, B);
  or (W, and1, A);
endmodule
```

Output :-