1. **Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates**

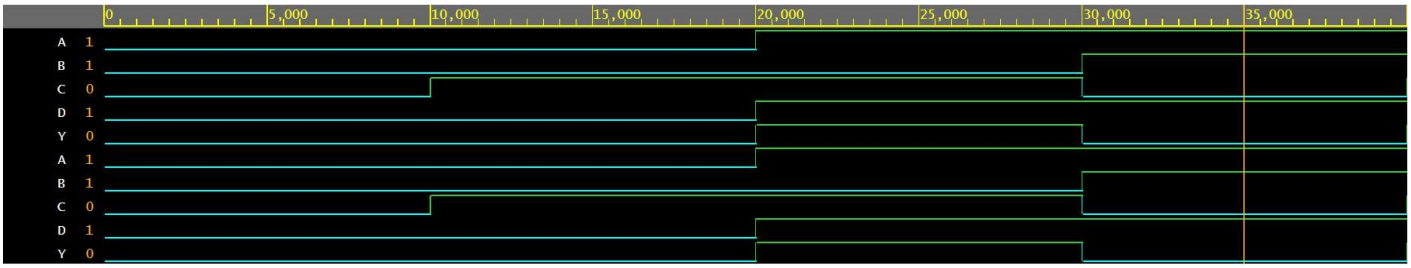
Code

```
module four_var_logic (
        input A,
        input B,
        input C,
        input D,
        output Y
);
        assign Y = (C & D) | (~A & D); // Y = CD + A'D
endmodule
```

TestBench

```
`timescale 1ns / 1ps
module testbench;
  reg A, B, C, D;
  wire Y;
  four_var_logic uut (
    .A(A),
    .B(B),
    .C(C),
    .D(D),
    .Y(Y)
  );

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("A=%b, B=%b, C=%b, D=%b, Y=%b", A, B, C, D, Y);
    A = 0; B = 0; C = 0; D = 0; #10;
    A = 0; B = 0; C = 1; D = 0; #10;
    A = 1; B = 0; C = 1; D = 1; #10;
    A = 1; B = 1; C = 0; D = 1; #10;
    A = 1; B = 1; C = 1; D = 1; #10;
    $finish;
  end

endmodule
```

Output


0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
A 1
B 1
C 0
D 1
Y 0
A 1
B 1
C 0
D 1
Y 0

2. Design a 4 bit full adder and subtractor and simulate the same using basic gates.

Code for Adder

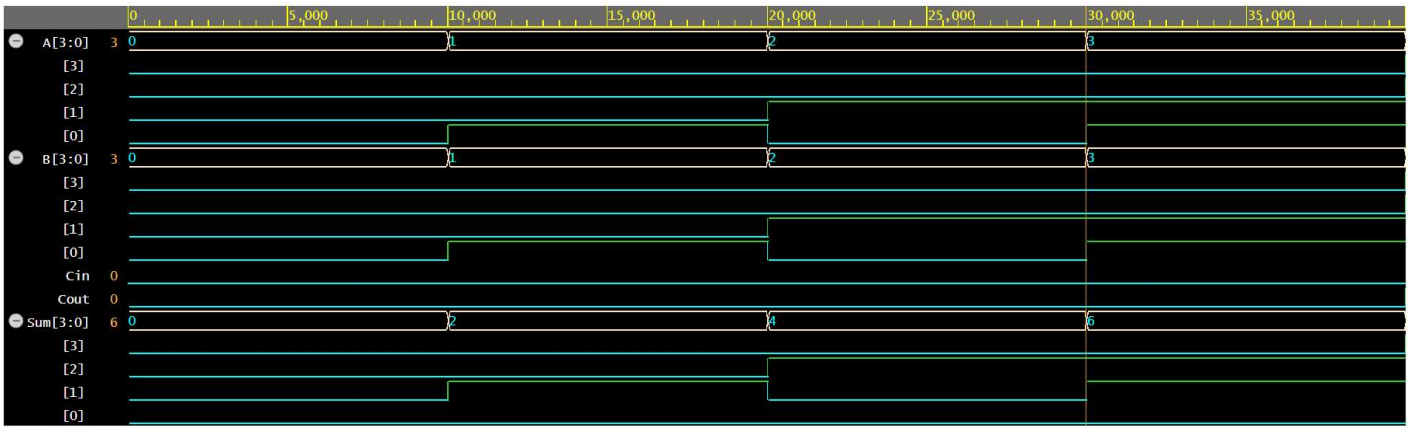
```
module FullAdder4bit (
  input [3:0] A,
  input [3:0] B,
  input Cin,
  output [3:0] Sum,
  output Cout
);
  wire C1, C2, C3;
  // 1st bit
  FullAdder FA1(A[0], B[0], Cin, Sum[0], C1);
  // 2nd bit
  FullAdder FA2(A[1], B[1], C1, Sum[1], C2);
  // 3rd bit
  FullAdder FA3(A[2], B[2], C2, Sum[2], C3);
  // 4th bit
  FullAdder FA4(A[3], B[3], C3, Sum[3], Cout);
endmodule

module FullAdder (
  input A,
  input B,
  input Cin,
  output Sum,
  output Cout
);
  assign Sum = A ^ B ^ Cin;
  assign Cout = (A & B) | (A & Cin) | (B & Cin);
endmodule
```

TestBench

```
module testbench;
  reg [3:0] A, B;
  reg Cin;
  wire [3:0] Sum;
  wire Cout;
  FullAdder4bit uut (
    .A(A),
    .B(B),
    .Cin(Cin),
    .Sum(Sum),
    .Cout(Cout)
  );
```

```
 initial begin
   $dumpfile("dump.vcd");
   $dumpvars(1);
   $monitor("A=%b, B=%b, Cin=%b, Sum=%b, Cout=%b", A, B, Cin, Sum, Cout);

   A = 4'b0000; B = 4'b0000; Cin = 0; #10;
   A = 4'b0001; B = 4'b0001; Cin = 0; #10;
   A = 4'b0010; B = 4'b0010; Cin = 0; #10;
   A = 4'b0011; B = 4'b0011; Cin = 0; #10;
   A = 4'b1111; B = 4'b1111; Cin = 0; #10;

   $finish;
 end

endmodule
```

Output:



Code for Subtractor

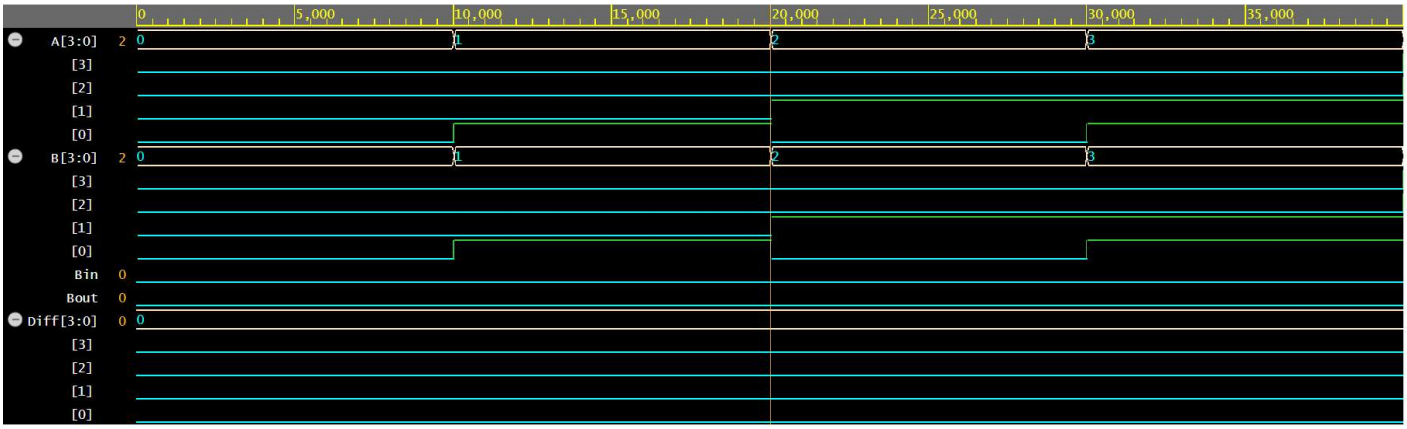
```
module FullSubtractor4bit (
  input [3:0] A,
  input [3:0] B,
  input Bin,
  output [3:0] Diff,
  output Bout
);
  wire B1, B2, B3;
  // 1st bit
  FullSubtractor FS1(A[0], B[0], Bin, Diff[0], B1);
  // 2nd bit
  FullSubtractor FS2(A[1], B[1], B1, Diff[1], B2);
  // 3rd bit
  FullSubtractor FS3(A[2], B[2], B2, Diff[2], B3);
  // 4th bit
  FullSubtractor FS4(A[3], B[3], B3, Diff[3], Bout);
endmodule
module FullSubtractor (
  input A,
  input B,
  input Bin,
  output Diff,
  output Bout
);
  assign Diff = A ^ B ^ Bin;
  assign Bout = (~A & B & Bin) | (A & ~B &Bin);
endmodule
```

TestBench

```
module testbench;
  reg [3:0] A, B;
  reg Bin;
  wire [3:0] Diff;
  wire Bout;

  FullSubtractor4bit uut (
    .A(A),
    .B(B),
    .Bin(Bin),
    .Diff(Diff),
    .Bout(Bout)
  );
```

```
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);
    A = 4'b0000; B = 4'b0000; Bin = 0; #10;
    A = 4'b0001; B = 4'b0001; Bin = 0; #10;
    A = 4'b0010; B = 4'b0010; Bin = 0; #10;
    A = 4'b0011; B = 4'b0011; Bin = 0; #10;
    A = 4'b1111; B = 4'b1111; Bin = 0; #10;
    $finish;
  end
endmodule
```

Output



3. **Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model.**

Code

TestBench

Output

4. **Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.**

Code

```
module half_adder(
  input A,
  input B,
  output sum,
  output carry
 );
  assign sum = A ^ B;
  assign carry = A & B;
endmodule
module full_adder(
  input A,
  input B,
  input Cin,
  output sum,
  output Cout
);
  wire sum1, carry1, carry2;
  half_adder HA1(A, B, sum1, carry1);
  half_adder HA2(sum1, Cin, sum, carry2);
  assign Cout = carry1 | carry2;
endmodule
module half_subtractor(
  input A,
  input B,
  output difference,
  output borrow
);
  assign difference = A ^ B;
  assign borrow = (~A & B);
endmodule

module full_subtractor(
  input A,
  input B,
  input Bin,
  output difference,
  output Bout
);
  wire diff1, bor1, bor2;
  half_subtractor HS1(A, B, diff1, bor1);
```

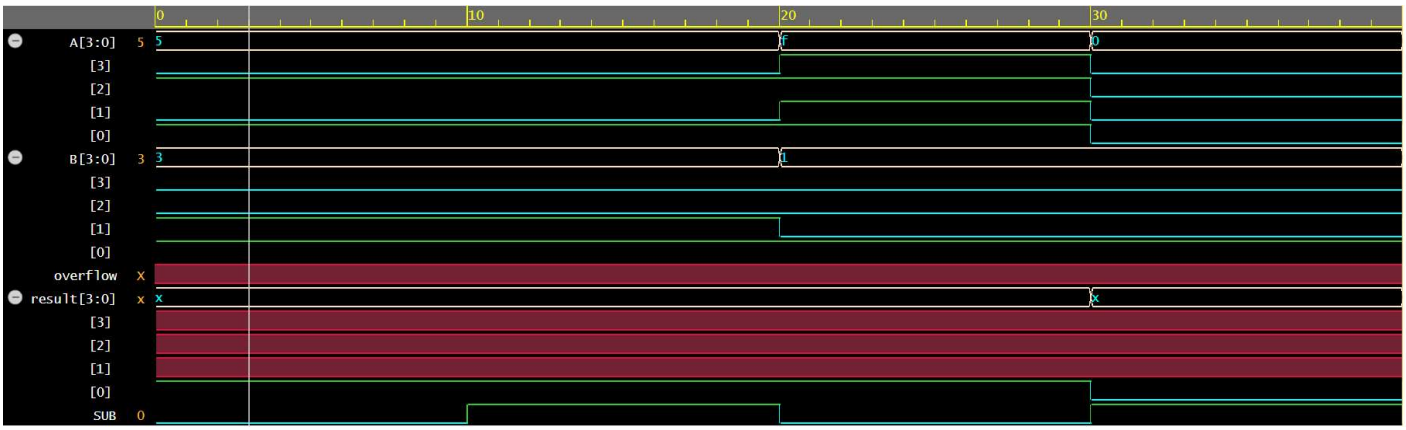
```
  half_subtractor HS2(diff1, Bin, difference, bor2);
  assign Bout = bor1 | bor2;
endmodule

module binary_adder_subtractor(
  input [3:0] A,
  input [3:0] B,
  input SUB,
  output [3:0] result,
  output overflow
);
  wire [3:0] sum;
  wire [3:0] difference;
  wire carry_in, borrow_in, carry_out, borrow_out;
  assign carry_in = ~SUB;
  assign borrow_in = SUB;
  full_adder FA0(A[0], B[0], carry_in, sum[0], carry_out);
  full_adder FA1(A[1], B[1], carry_out, sum[1], carry_out);
  full_adder FA2(A[2], B[2], carry_out, sum[2], carry_out);
  full_adder FA3(A[3], B[3], carry_out, sum[3], carry_out);
  full_subtractor FS0(A[0], B[0], borrow_in, difference[0], borrow_out);
  full_subtractor FS1(A[1], B[1], borrow_out, difference[1], borrow_out);
  full_subtractor FS2(A[2], B[2], borrow_out, difference[2], borrow_out);
  full_subtractor FS3(A[3], B[3], borrow_out, difference[3], borrow_out);
  assign result = (SUB) ? difference : sum;
  assign overflow = carry_out ^ borrow_out;
endmodule
```

TestBench

```
module testbench;
  reg [3:0] A, B;
  reg SUB;
  wire [3:0] result;
  wire overflow;
  binary_adder_subtractor DUT (
    .A(A),
    .B(B),
    .SUB(SUB),
    .result(result),
    .overflow(overflow)
  );
  initial begin
    $dumpfile("dump.vcd");
```

```
    $dumpvars(1);
    $monitor("Time=%0t, A=%b, B=%b, SUB=%b, result=%b, overflow=%b", $time, A, B, SUB, result,
overflow);
    A = 4'b0101; B = 4'b0011; SUB = 0; #10;
    A = 4'b0101; B = 4'b0011; SUB = 1; #10;
    A = 4'b1111; B = 4'b0001; SUB = 0; #10;
    A = 4'b0000; B = 4'b0001; SUB = 1; #10;
    A = 4'b0101; B = 4'b0011; SUB = 0; #10;
  end
endmodule
```

Output



5. Design Verilog HDL to implement Decimal adder.

Code:

```
module DecimalAdder(
   input [3:0] A,
   input [3:0] B,
   output reg [3:0] Sum
);

   reg [3:0] A_binary;
   reg [3:0] B_binary;
   reg [4:0] binary_sum;

   always @* begin
      case(A)
         0: A_binary = 4'b0000;
         1: A_binary = 4'b0001;
         2: A_binary = 4'b0010;
         3: A_binary = 4'b0011;
         4: A_binary = 4'b0100;
         5: A_binary = 4'b0101;
         6: A_binary = 4'b0110;
         7: A_binary = 4'b0111;
         8: A_binary = 4'b1000;
         9: A_binary = 4'b1001;
         default: A_binary = 4'b0000;
      endcase

      case(B)
         0: B_binary = 4'b0000;
         1: B_binary = 4'b0001;
         2: B_binary = 4'b0010;
         3: B_binary = 4'b0011;
         4: B_binary = 4'b0100;
         5: B_binary = 4'b0101;
         6: B_binary = 4'b0110;
         7: B_binary = 4'b0111;
         8: B_binary = 4'b1000;
         9: B_binary = 4'b1001;
         default: B_binary = 4'b0000;
      endcase
   end

   always @* begin
```

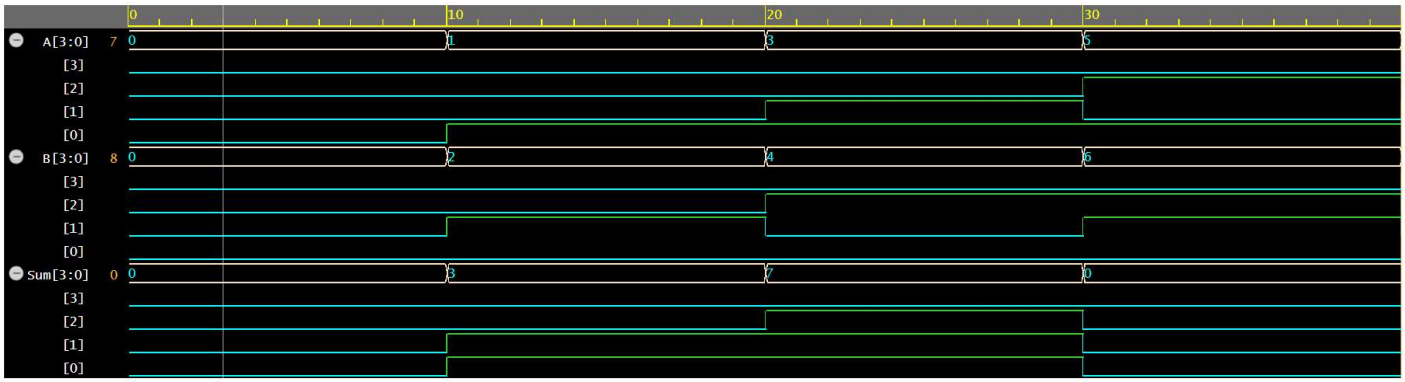
```
    binary_sum = A_binary + B_binary;
  end

  always @* begin
    case(binary_sum)
      0: Sum = 4'b0000;
      1: Sum = 4'b0001;
      2: Sum = 4'b0010;
      3: Sum = 4'b0011;
      4: Sum = 4'b0100;
      5: Sum = 4'b0101;
      6: Sum = 4'b0110;
      7: Sum = 4'b0111;
      8: Sum = 4'b1000;
      9: Sum = 4'b1001;
      10: Sum = 4'b0000;
      11: Sum = 4'b0000;
      12: Sum = 4'b0000;
      13: Sum = 4'b0000;
      14: Sum = 4'b0000;
      15: Sum = 4'b0000;
      default: Sum = 4'b0000;
    endcase
  end
endmodule
```

TestBench:

```
module testbench();
  reg [3:0] A;
  reg [3:0] B;
  wire [3:0] Sum;
  DecimalAdder decimal_adder(
    .A(A),
    .B(B),
    .Sum(Sum)
  );

  initial begin
   $dumpfile("dump.vcd");
   $dumpvars(1);
   $monitor("Time=%0t A=%d B=%d Sum=%d", $time, A, B, Sum);
   A = 0; B = 0; #10;
   A = 1; B = 2; #10;
   A = 3; B = 4; #10;
```

```
    A = 5; B = 6; #10;
    A = 7; B = 8; #10;
    $finish;
  end
endmodule
```

Output:



6. Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.

Code for 2:1:

```
module mux2to1(
 input a,
 input b,
 input sel,
 output y
 );
 assign y = (sel) ? b : a;
 endmodule
```

TestBench:

```
module testbench;
  reg a, b, sel;
  wire y;
  mux2to1 DUT (
    .a(a),
    .b(b),
    .sel(sel),
    .y(y)
  );

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, a=%b, b=%b, sel=%b, y=%b", $time, a, b, sel, y);
    sel = 0; a = 1; b = 0; #10;
    sel = 1; a = 0; b = 1; #10;
    sel = 0; a = 0; b = 0; #10;
    sel = 1; a = 1; b = 1; #10;
    sel = 0; a = 1; b = 1; #10;
  end
endmodule
```

Output:

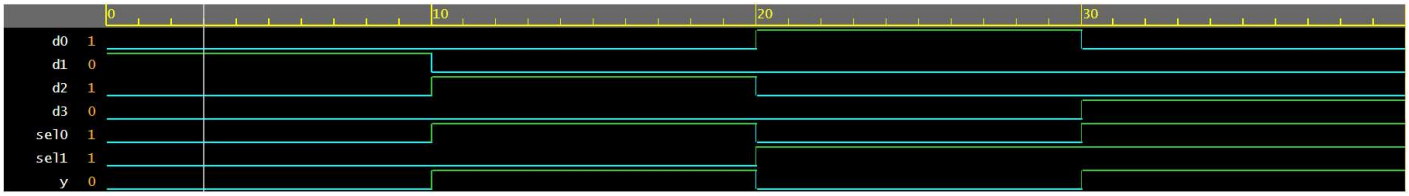
Code for 4:1:

```
module mux4to1(
 input d0,
 input d1,
 input d2,
 input d3,
 input sel1,
 input sel0,
 output y
 );
 assign y = (sel1 == 1'b1) ? ((sel0 == 1'b1) ? d3 : d1) : ((sel0 == 1'b1) ? d2 : d0);
 endmodule
```

Testbench:

```
module testbench;
  reg d0, d1, d2, d3, sel1, sel0;
  wire y;
  mux4to1 DUT (
    .d0(d0),
    .d1(d1),
    .d2(d2),
    .d3(d3),
    .sel1(sel1),
    .sel0(sel0),
    .y(y)
  );
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, d0=%b, d1=%b, d2=%b, d3=%b, sel1=%b, sel0=%b, y=%b", $time, d0, d1, d2, d3,
sel1, sel0, y);
    sel1 = 0; sel0 = 0; d0 = 0; d1 = 1; d2 = 0; d3 = 0; #10;
    sel1 = 0; sel0 = 1; d0 = 0; d1 = 0; d2 = 1; d3 = 0; #10;
    sel1 = 1; sel0 = 0; d0 = 1; d1 = 0; d2 = 0; d3 = 0; #10;
    sel1 = 1; sel0 = 1; d0 = 0; d1 = 0; d2 = 0; d3 = 1; #10;
    sel1 = 1; sel0 = 1; d0 = 1; d1 = 0; d2 = 1; d3 = 0; #10;
  end
endmodule
```

Output:



Code for 8:1:

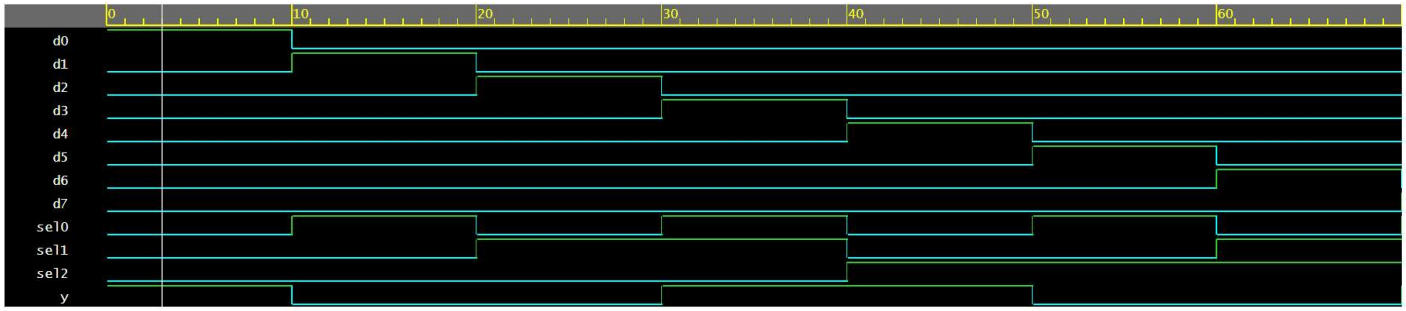
```
module mux8to1(
   input d0,
   input d1,
   input d2,
   input d3,
   input d4,
   input d5,
   input d6,
   input d7,
   input sel2,
   input sel1,
   input sel0,
   output y
);
   assign y = (sel2 == 1'b1) ? ((sel1 == 1'b1) ? ((sel0 == 1'b1) ? d7 : d5) : ((sel0 == 1'b1) ? d6 : d4)) : ((sel1 == 1'b1) ? ((sel0 == 1'b1) ? d3 : d1) : ((sel0 == 1'b1) ? d2 : d0));
endmodule
```

TestBench:

```
module testbench;
  reg d0, d1, d2, d3, d4, d5, d6, d7, sel2, sel1, sel0;
  wire y;
  mux8to1 DUT (
    .d0(d0),
    .d1(d1),
    .d2(d2),
    .d3(d3),
    .d4(d4),
    .d5(d5),
    .d6(d6),
    .d7(d7),
    .sel2(sel2),
    .sel1(sel1),
    .sel0(sel0),
    .y(y)
```

```
  );

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, d0=%b, d1=%b, d2=%b, d3=%b, d4=%b, d5=%b, d6=%b, d7=%b, sel2=%b, sel1=%b,
sel0=%b, y=%b", $time, d0, d1, d2, d3, d4, d5, d6, d7, sel2, sel1, sel0, y);
    sel2 = 0; sel1 = 0; sel0 = 0; d0 = 1; d1 = 0; d2 = 0; d3 = 0; d4 = 0; d5 = 0; d6 = 0; d7 = 0; #10;
    sel2 = 0; sel1 = 0; sel0 = 1; d0 = 0; d1 = 1; d2 = 0; d3 = 0; d4 = 0; d5 = 0; d6 = 0; d7 = 0; #10;
    sel2 = 0; sel1 = 1; sel0 = 0; d0 = 0; d1 = 0; d2 = 1; d3 = 0; d4 = 0; d5 = 0; d6 = 0; d7 = 0; #10;
    sel2 = 0; sel1 = 1; sel0 = 1; d0 = 0; d1 = 0; d2 = 0; d3 = 1; d4 = 0; d5 = 0; d6 = 0; d7 = 0; #10;
    sel2 = 1; sel1 = 0; sel0 = 0; d0 = 0; d1 = 0; d2 = 0; d3 = 0; d4 = 1; d5 = 0; d6 = 0; d7 = 0; #10;
    sel2 = 1; sel1 = 0; sel0 = 1; d0 = 0; d1 = 0; d2 = 0; d3 = 0; d4 = 0; d5 = 1; d6 = 0; d7 = 0; #10;
    sel2 = 1; sel1 = 1; sel0 = 0; d0 = 0; d1 = 0; d2 = 0; d3 = 0; d4 = 0; d5 = 0; d6 = 1; d7 = 0; #10;
    sel2 = 1; sel1 = 1; sel0 = 1; d0 = 0; d1 = 0; d2 = 0; d3 = 0; d4 = 0; d5 = 0; d6 = 0; d7 = 1; #10;
  end

endmodule
```

Output:



7. Design Verilog program to implement types of De-Multiplexer.

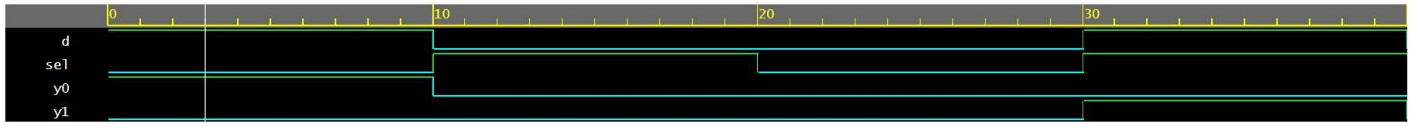
Code 1:2:

```
module demux1to2(
 input d,
 input sel,
 output y0,
 output y1
 );
 assign y0 = ~sel & d;
 assign y1 = sel & d;
 endmodule
```

TestBench:

```
module testbench;
  reg d, sel;
  wire y0, y1;
  demux1to2 DUT (
    .d(d),
    .sel(sel),
    .y0(y0),
    .y1(y1)
  );
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, d=%b, sel=%b, y0=%b, y1=%b", $time, d, sel, y0, y1);
    sel = 0; d = 1; #10;
    sel = 1; d = 0; #10;
    sel = 0; d = 0; #10;
    sel = 1; d = 1; #10;
    sel = 1; d = 0; #10;
  end
endmodule
```

Output:



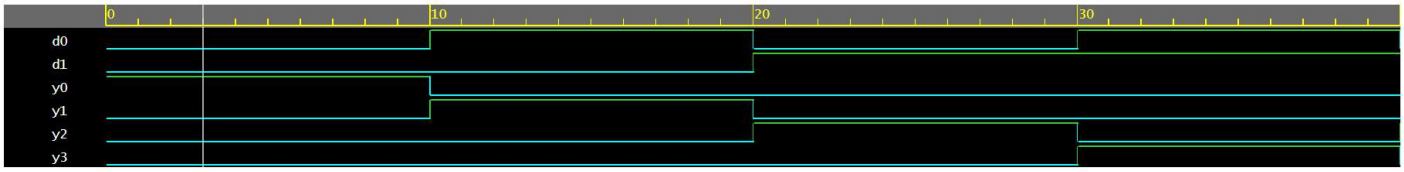
Code 1:4:

```
module demux2to4(
  input d0,
  input d1,
  output y0,
  output y1,
  output y2,
  output y3
);
  assign y0 = ~d1 & ~d0;
  assign y1 = ~d1 & d0;
  assign y2 = d1 & ~d0;
  assign y3 = d1 & d0;
endmodule
```

TestBench:

```
module testbench;
  reg d0, d1;
  wire y0, y1, y2, y3;
  demux2to4 DUT (
    .d0(d0),
    .d1(d1),
    .y0(y0),
    .y1(y1),
    .y2(y2),
    .y3(y3)
  );
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, d0=%b, d1=%b, y0=%b, y1=%b, y2=%b, y3=%b", $time, d0, d1, y0, y1, y2, y3);
    d0 = 0; d1 = 0; #10;
    d0 = 1; d1 = 0; #10;
    d0 = 0; d1 = 1; #10;
    d0 = 1; d1 = 1; #10;
    d0 = 0; d1 = 1; #10;
  end
endmodule
```

Output:



Code 1:8:

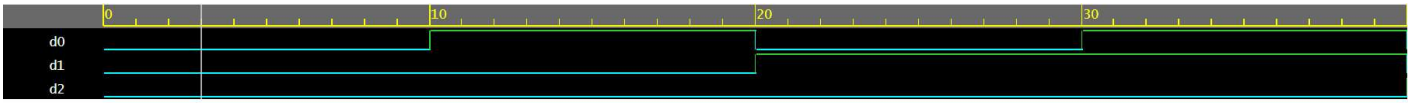
```
module demux3to8(
  input d0,
  input d1,
  input d2,
  output y0,
  output y1,
  output y2,
  output y3,
  output y4,
  output y5,
  output y6,
  output y7
);
  assign y0 = ~d2 & ~d1 & ~d0;
  assign y1 = ~d2 & ~d1 & d0;
  assign y2 = ~d2 & d1 & ~d0;
  assign y3 = ~d2 & d1 & d0;
  assign y4 = d2 & ~d1 & ~d0;
  assign y5 = d2 & ~d1 & d0;
  assign y6 = d2 & d1 & ~d0;
  assign y7 = d2 & d1 & d0;
 endmodule
```

TestBench:

```
module testbench;
  reg d0, d1, d2;
  wire y0, y1, y2, y3, y4, y5, y6, y7;
  demux3to8 DUT (
    .d0(d0),
    .d1(d1),
    .d2(d2),
    .y0(y0),
    .y1(y1),
    .y2(y2),
    .y3(y3),
    .y4(y4),
```

```
   .y5(y5),
   .y6(y6),
   .y7(y7)
  );

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, d0=%b, d1=%b, d2=%b, y0=%b, y1=%b, y2=%b, y3=%b, y4=%b, y5=%b, y6=%b,
y7=%b", $time, d0, d1, d2, y0, y1, y2, y3, y4, y5, y6, y7);
    d0 = 0; d1 = 0; d2 = 0; #10;
    d0 = 1; d1 = 0; d2 = 0; #10;
    d0 = 0; d1 = 1; d2 = 0; #10;
    d0 = 1; d1 = 1; d2 = 0; #10;
    d0 = 0; d1 = 0; d2 = 1; #10;
  end
endmodule
```

Output:



8. Design Verilog program for implementing various types of Flip-Flops such as SR, JK and D.

Code:

```
module sr_ff(
  input clk,
  input S,
  input R,
  output reg Q
);
  always @(posedge clk) begin
  if (S && ~R) begin
    Q<=1'b1;
  end
    else if (~S && R) begin
      Q<=1'b0;
    end
  end
endmodule
```

TestBench:

```
module testbench;
  reg clk, S, R;
  wire Q;
  sr_ff DUT (
    .clk(clk),
    .S(S),
    .R(R),
    .Q(Q)
  );
  always #5 clk = ~clk;
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
    $monitor("Time=%0t, clk=%b, S=%b, R=%b, Q=%b", $time, clk, S, R, Q);
    S = 0; R = 0; #10;
    S = 1; R = 0; #10;
    S = 0; R = 1; #10;
    S = 1; R = 1; #10;
    S = 0; R = 1; #10;
    $finish;
  end
endmodule
```

Output: