**HALF ADDER**

**PROGRAM:**

module HalfAdder(a, b, sum, carry);

 input a;

 input b;

 output sum;

 output carry;

 xor g1(sum,a,b);

 and g2(carry,a,b);

endmodule

**OUTPUT**

****

**FULL ADDER**

**PROGRAM:**

module FullAdder(a, b, cin, sum, carry);

 input a;

 input b;

 input cin;

 output sum;

 output carry;

 wire d,e,f;

 xor g1(sum,a,b,cin);

 and g2(d,a,b);

 and g3(e,b,cin);

 and g4(f,cin,a);

 or g5(carry,d,e,f);

endmodule

**OUTPUT**

****

**HALF SUBTRACTOR**

**PROGRAM:**

module HalfSubtractor(a, b, difference, borrow);

 input a;

 input b;

 output difference;

 output borrow;

 xor g1(difference,a,b);

 not g2(c,a);

 and g3(borrow,c,b);

endmodule

**OUTPUT**

****

**FULL SUBTRACTOR**

**PROGRAM:**

module FullSub(a, b, c, diff, borrow);

 input a;

 input b;

 input c;

 output diff;

 output borrow;

 wire d,e,f,g;

 xor g1(diff,a,b,c);

 not g2(d,a);

 and g3(e,d,b);

 and g4(f,d,c);

 and g5(g,b,c);

 or g6(borrow,e,f,g);

endmodule

**OUTPUT**

****

**MULTIPLEXER**

**PROGRAM**

module Mux(d, s, y);

 input [0:3] d;

 input [0:1] s;

 output y;

 wire a,b,c,e,f,g;

 not g1(a,s[0]);

 not g2(b,s[1]);

 and g3(c,d[0],a,b);

 and g4(e,d[1],a,s[1]);

 and g5(f,d[2],s[0],b);

 and g6(g,d[3],s[0],s[1]);

 or g7(y,c,e,f,g);

endmodule

**OUTPUT**

****

**DEMULTIPLEXER**

**PROGRAM**

module DeMux(y, s, d);

 input y;

 input [0:1] s;

 output [0:3] d;

 wire a,b;

 not g1(a,s[0]);

 not g2(b,s[1]);

 and g3(d[0],y,a,b);

 and g4(d[1],y,a,s[1]);

 and g5(d[2],y,s[0],b);

 and g6(d[3],y,s[0],s[1]);

endmodule

**OUTPUT**

****

**ENCODER**

**PROGRAM**

module Encoder(y, a, b);

 input [0:3] y;

 output a;

 output b;

 or g1(a,y[2],y[3]);

 or g2(b,y[1],y[3]);

endmodule

**OUTPUT**

****

**DECODER**

**PROGRAM**

module Decoder(a, b, en, y);

 input a;

 input b;

 input en;

 output [0:3] y;

 wire c,d;

 not g1(c,a);

 not g2(d,b);

 nand g3(y[0],c,d,en);

 nand g4(y[1],c,b,en);

 nand g5(y[2],a,d,en);

 nand g6(y[3],a,b,en);

endmodule

**OUTPUT**

****

**D FLIP FLOP**

**PROGRAM:**

module DFlipFlop(d, clk, clr, q, qbar);

 input d;

 input clk;

 input clr;

 output q;

 output qbar;

 reg q;

 assign qbar=~q;

 always @(posedge clk)

 if(~clr)q=0;

 else q=d;

endmodule

**OUTPUT**



**T FLIP FLOP**

**PROGRAM:**

module TFlipFlop(t, clk, q, qbar);

 input t;

 input clk;

 output q;

 output qbar;

 reg q;

 assign qbar=~q;

 always @ (posedge clk)

 q=~t;

endmodule

**OUTPUT**

