**UCS704 EMBEDDED SYSTEMS DESIGN**

**LAB ASSIGMENT**

**Experiment 1**

**(Truth Table and Logic Gates)**

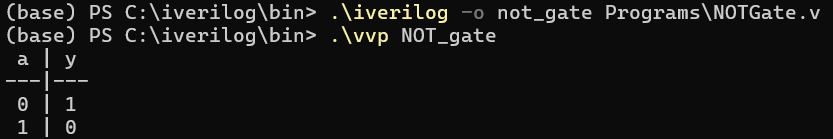
**Question:** To study and verify the truth table of various logic gates (NOT, AND, OR, NAND, NOR, EX-OR, & EX-NOR).

1. **NOT Gate**

**Code:**

**module** not\_gate (  
 **input** a,  
 **output** y  
);  
**assign** y = ~a;   
**endmodule**  
  
**module** testbench\_not;  
**reg** a;  
**wire** y;  
  
not\_gate uut (.a(a), .y(y));  
**initial** **begin**  
 $display(" a | y ");  
 $display("---|---");  
 $monitor(" %b | %b", a, y);  
 a = 0; #10;  
 a = 1; #10;  
 $finish;  
**end**  
**endmodule**

**Output:**

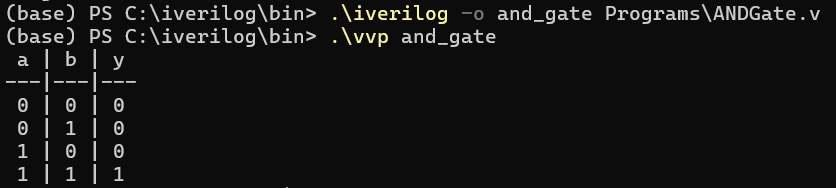
****

1. **AND Gate**

**Code:**

module **and\_gate** (  
 input a,  
 input **b,** output y  
);  
assign y = a & **b;**endmodule  
  
module testbench\_and;  
reg a, **b;**wire y;  
  
**and\_gate** uut (.a(a), .**b(b),** .y(y));  
  
initial **begin** $**display("** a **| b |** y ");  
 $display("---**|---|**---");  
 $monitor(" %**b | %b |** %**b",** a, **b,** y);  
 a = 0; b = 0; #10;  
 a = 0; b = 1; #10;  
 a = 1; b = 0; #10;  
 a = 1; b = 1; #10;  
 $finish;  
end  
endmodule

**Output:**

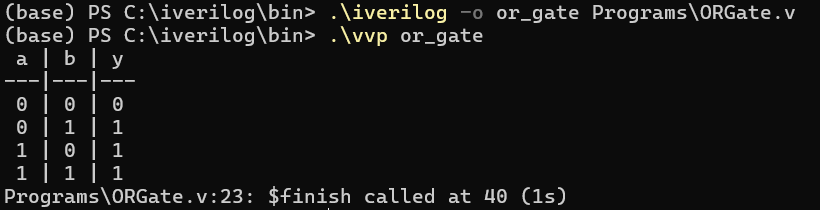
****

1. **OR Gate**

**Code:**

**module** or\_gate (  
 **input** a,  
 **input** b,  
 **output** y  
);  
**assign** y = a | b; // OR operation  
**endmodule**  
  
**module** testbench\_or;  
**reg** a, b;  
**wire** y;  
  
or\_gate uut (.a(a), .b(b), .y(y));  
  
**initial** **begin**  
 $display(" a | b | y ");  
 $display("---|---|---");  
 $monitor(" %b | %b | %b", a, b, y);  
 a = 0; b = 0; #10;  
 a = 0; b = 1; #10;  
 a = 1; b = 0; #10;  
 a = 1; b = 1; #10;  
 $finish;  
**end**  
**endmodule**

**Output:**

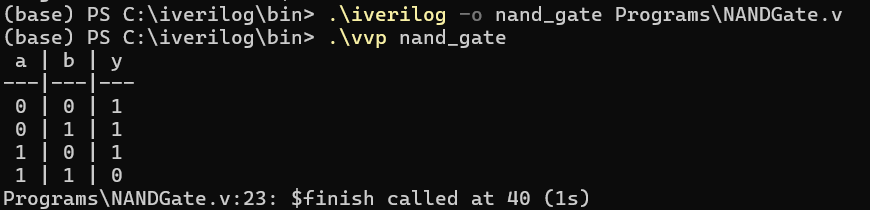
****

1. **NAND Gate**

**Code:**

**module** nand\_gate (  
 **input** a,  
 **input** b,  
 **output** y  
);  
**assign** y = ~(a & b); // NAND operation  
**endmodule**  
**module** testbench\_nand;  
**reg** a, b;  
**wire** y;  
nand\_gate uut (.a(a), .b(b), .y(y));  
  
**initial** **begin**  
 $display(" a | b | y ");  
 $display("---|---|---");  
 $monitor(" %b | %b | %b", a, b, y);  
 a = 0; b = 0; #10;  
 a = 0; b = 1; #10;  
 a = 1; b = 0; #10;  
 a = 1; b = 1; #10;  
 $finish;  
**end**  
**endmodule**

**Output:**

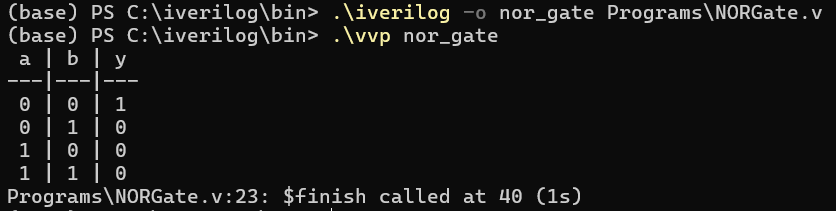
****

1. **NOR Gate**

**Code:**

**module** nor\_gate (  
 **input** a,  
 **input** b,  
 **output** y  
);  
**assign** y = ~(a | b); // NOR operation  
**endmodule**  
  
**module** testbench\_nor;  
**reg** a, b;  
**wire** y;  
  
nor\_gate uut (.a(a), .b(b), .y(y));  
  
**initial** **begin**  
 $display(" a | b | y ");  
 $display("---|---|---");  
 $monitor(" %b | %b | %b", a, b, y);  
 a = 0; b = 0; #10;  
 a = 0; b = 1; #10;  
 a = 1; b = 0; #10;  
 a = 1; b = 1; #10;  
 $finish;  
**end**  
**endmodule**

**Output:**

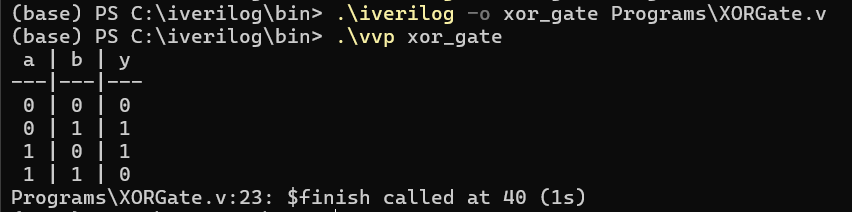
****

1. **EX-OR Gate**

**Code:**

**module** xor\_gate (  
 **input** a,  
 **input** b,  
 **output** y  
);  
**assign** y = a ^ b; // XOR operation  
**endmodule**  
  
**module** testbench\_xor;  
**reg** a, b;  
**wire** y;  
  
xor\_gate uut (.a(a), .b(b), .y(y));  
  
**initial** **begin**  
 $display(" a | b | y ");  
 $display("---|---|---");  
 $monitor(" %b | %b | %b", a, b, y);  
 a = 0; b = 0; #10;  
 a = 0; b = 1; #10;  
 a = 1; b = 0; #10;  
 a = 1; b = 1; #10;  
 $finish;  
**end**  
**endmodule**

**Output:**

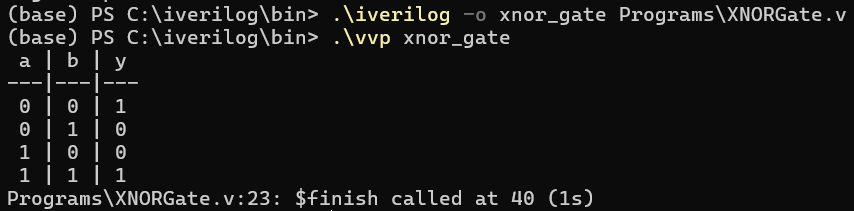
****

1. **EX-NOR Gate**

**Code:**

module xnor\_gate (  
 input a,  
 input **b,** output y  
);  
assign y = ~(a ^ **b);**endmodule  
  
module testbench\_xnor;  
reg a, **b;**wire y;  
  
xnor\_gate uut (.a(a), .**b(b),** .y(y));  
  
initial **begin**   
 $**display("** a **| b |** y ");  
 $display("---**|---|**---");  
 $monitor(" %**b | %b |** %**b",** a, **b,** y);   
 a = 0; b = 0; #10;  
 a = 0; b = 1; #10;  
 a = 1; b = 0; #10;  
 a = 1; b = 1; #10;  
 $finish;  
end   
endmodule

**Output:**

****

**Experiment 2**

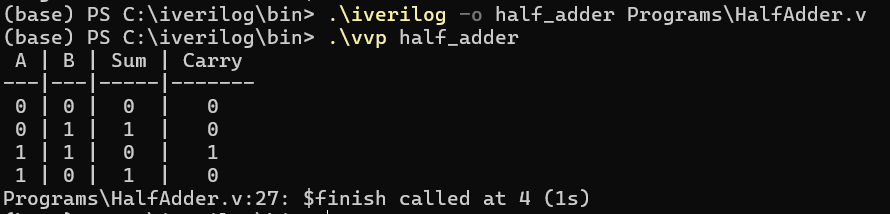
**(Half Adder)**

**Question:** To design and verify a half adder using S= (x+y)(x’+y’) C= xy

**Code:**

**module** halfadder (  
 **input** a, b,  
 **output** sum, carry  
);  
  
**assign** sum = a ^ b;  
**assign** carry = a & b;   
**endmodule**  
  
**module** halfadder\_tb;  
**reg** a, b;   
**wire** sum;  
**wire** carry;  
  
halfadder obj(.a(a), .b(b), .sum(sum), .carry(carry));  
  
**initial** **begin**  
 $display(" A | B | Sum | Carry ");  
 $display("---|---|-----|-------");  
 $monitor(" %b | %b | %b | %b ", a, b, sum, carry);  
 a = 0; b = 0; #1;  
 b = 1; #1;   
 a = 1; #1;   
 b = 0; #1;   
 $finish;  
**end**  
**endmodule**

**Output:**

****

**Experiment 3**

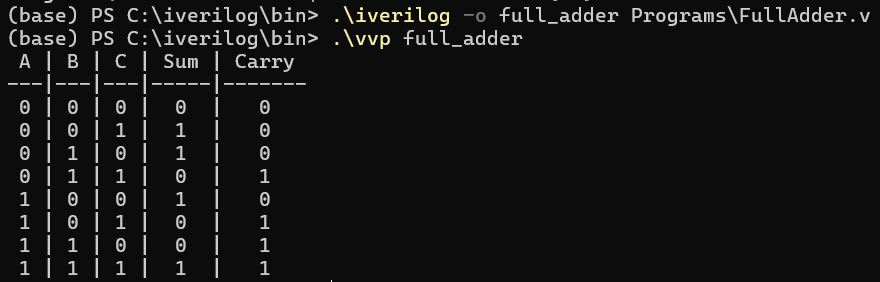
**(Full Adder)**

**Question:** To design and verify a full adder using S = x’y’z+x’yz’+xy’z’+xyz C=xy+xz+yz

**Code:**

module fulladder (  
 input a, b, c,  
 output sum, carry  
);  
assign sum = a ^ b ^ c;  
assign carry = (a & b) | (b & c) | (a & c);  
endmodule  
  
module fulladder\_tb;  
reg a, b, c;  
wire sum, carry;  
fulladder add1(a, b, c, sum, carry);  
  
initial begin  
 $display(" A | B | C | Sum | Carry ");  
 $display("---|---|---|-----|-------");  
 $monitor(" %b | %b | %b | %b | %b ", a, b, c, sum, carry);  
 a = 0; b = 0; c = 0; #5;  
 a = 0; b = 0; c = 1; #5;  
 a = 0; b = 1; c = 0; #5;  
 a = 0; b = 1; c = 1; #5;  
 a = 1; b = 0; c = 0; #5;  
 a = 1; b = 0; c = 1; #5;  
 a = 1; b = 1; c = 0; #5;  
 a = 1; b = 1; c = 1; #5;  
end  
endmodule

**Output:**

****

**Experiment 4**

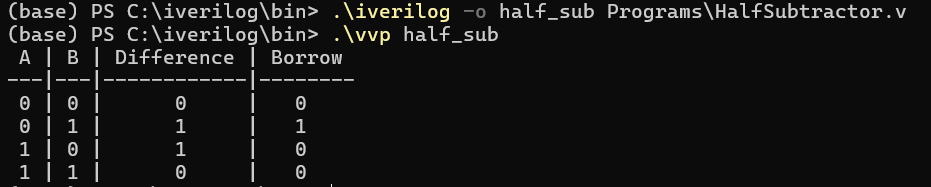
**(Half Subtractor)**

**Question:** To design and verify a half subtractor using D = x’y +xy’ B=x’y

**Code:**

**module** half\_sub (  
 **input** a, b,  
 **output** D, B  
);  
  
**assign** D = a ^ b;  
**assign** B = ~a & b;  
**endmodule**  
  
**module** half\_sub\_tb;  
**reg** a, b;  
**wire** D, B;  
half\_sub hs(a, b, D, B);  
  
**initial** **begin**  
 $display(" A | B | Difference | Borrow ");  
 $display("---|---|------------|--------");  
 $monitor(" %b | %b | %b | %b ", a, b, D, B);  
 a = 0; b = 0; #1;  
 a = 0; b = 1; #1;  
 a = 1; b = 0; #1;  
 a = 1; b = 1; #1;  
**end**  
**endmodule**

**Output:**

****

**Experiment 5**

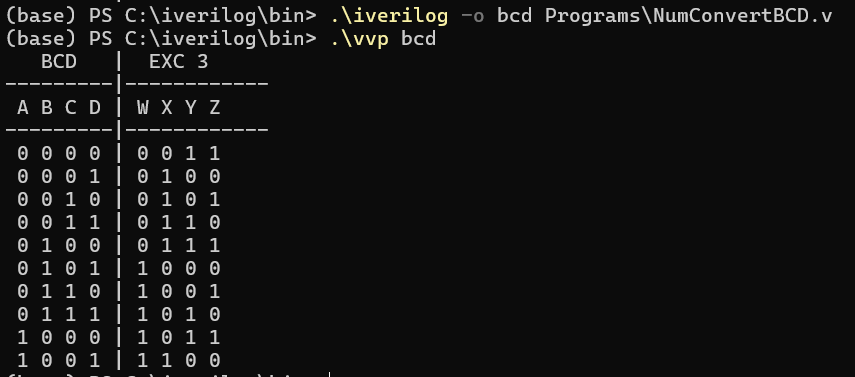
**(Number Converter)**

**Question:** Design a BCD to Excess 3 code converter using combinational circuits.

**Code:**

module BCD2Ex3 (  
 input A, B, C, D,  
 output W, X, Y, Z  
);  
  
assign W = A | (B & C) | (B & D);  
assign X = (~B & C) | (~B & D) | (B & ~C & ~D);  
assign Y = ~(C ^ D);  
assign Z = ~D;  
endmodule  
  
module test\_BCD2Ex3;  
reg A, B, C, D;  
wire W, X, Y, Z;  
  
BCD2Ex3 converter(A,B,C,D,W,X,Y,Z);  
  
initial begin  
 $display(" BCD | EXC 3 ");  
 $display("---------|------------");  
 $display(" A B C D | W X Y Z ");  
 $display("---------|------------");  
 $monitor(" %b %b %b %b | %b %b %b %b", A,B,C,D,W,X,Y,Z);  
 A = 0; B = 0; C = 0; D = 0;  
 #5 A = 0; B = 0; C = 0; D = 0;  
 #5 A = 0; B = 0; C = 0; D = 1;  
 #5 A = 0; B = 0; C = 1; D = 0;  
 #5 A = 0; B = 0; C = 1; D = 1;  
 #5 A = 0; B = 1; C = 0; D = 0;  
 #5 A = 0; B = 1; C = 0; D = 1;  
 #5 A = 0; B = 1; C = 1; D = 0;  
 #5 A = 0; B = 1; C = 1; D = 1;  
 #5 A = 1; B = 0; C = 0; D = 0;  
 #5 A = 1; B = 0; C = 0; D = 1;  
end  
endmodule

**Output:**

****

**Experiment 6**

**(Multiplexer 4:1)**

**Question:** To design and implement a 4:1 multiplexer

**Code:**

module mux (  
 input s1,s2,a,b,c,d,  
 output y  
);  
  
assign y = (~s1 & ~s2 & a) | (~s1 & s2 & b) | (s1 & ~s2 & c) | (s1 & s2 & d);  
endmodule  
  
module test\_mux;  
reg a,b,c,d,s1,s2;  
wire y;  
mux obj(s1,s2,a,b,c,d,y);  
  
initial begin  
 $display(" S1 S2 | A B C D | Y ");  
 $display("-------|---------|---");  
 $monitor(" %b %b | %b %b %b %b | %b", s1,s2,a,b,c,d,y);  
 **a**=0; **b**=0; **c**=0; **d**=0; **s1**=0; **s2**=0;  
 #5 **a**=0; **b**=0; **c**=0; **d**=0; **s1**=0; **s2**=0;  
 #5 **a**=0; **b**=0; **c**=0; **d**=1; **s1**=0; **s2**=1;  
 #5 **a**=0; **b**=0; **c**=1; **d**=0; **s1**=1; **s2**=0;  
 #5 **a**=0; **b**=0; **c**=1; **d**=1; **s1**=1; **s2**=1;  
 #5 **a**=0; **b**=1; **c**=0; **d**=0; **s1**=0; **s2**=0;  
 #5 **a**=0; **b**=1; **c**=0; **d**=1; **s1**=0; **s2**=1;  
 #5 **a**=0; **b**=1; **c**=1; **d**=0; **s1**=1; **s2**=0;  
 #5 **a**=0; **b**=1; **c**=1; **d**=1; **s1**=1; **s2**=0;  
 #5 **a**=1; **b**=0; **c**=0; **d**=0; **s1**=0; **s2**=1;  
 #5 **a**=1; **b**=0; **c**=0; **d**=1; **s1**=0; **s2**=0;  
  
end  
endmodule

**Output:**

****

**Experiment 7**

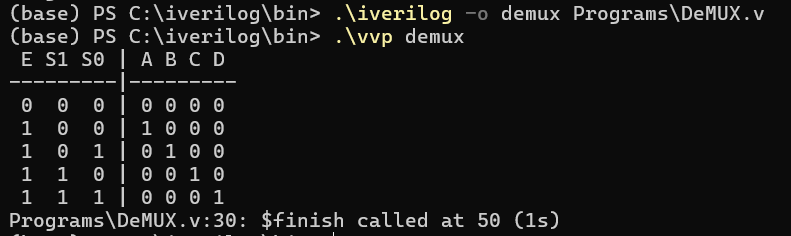
**(Demultiplexer 1:4)**

**Question:** To design and implement a 1:4 demultiplexer.

**Code:**

module demux (  
 input s1, s0, e, i,  
 output a, **b,** c, d  
);  
assign a = i & e & ~s1 & ~s0;  
assign **b** = i & e & ~s1 & s0;  
assign c = i & e & s1 & ~s0;  
assign d = i & e & s1 & s0;  
endmodule  
  
module test\_demux;  
reg s1, s0, e, i;  
wire a, **b,** c, d;  
  
  
demux obj (.s1(s1), .s0(s0), .e(e), .i(i), .a(a), .**b(b),** .c(c), .d(d));  
  
initial **begin** $**display("** E S1 S0 **| A B C D ");  
 $display("---------|**---------");  
 $monitor(" %**b**  %**b**  %**b | %b %b %b %b", e, s1, s0, a, b, c, d);  
  
 // Test cases  
 i = 1; e = 0; s1 = 0; s0 = 0; #10;  
 i = 1; e = 1; s1 = 0; s0 = 0; #10;  
 i = 1; e = 1; s1 = 0; s0 = 1; #10;  
 i = 1; e = 1; s1 = 1; s0 = 0; #10;  
 i = 1; e = 1; s1 = 1; s0 = 1; #10;  
  
 $finish;  
end  
  
endmodule**

**Output:**

****

**Experiment 8**

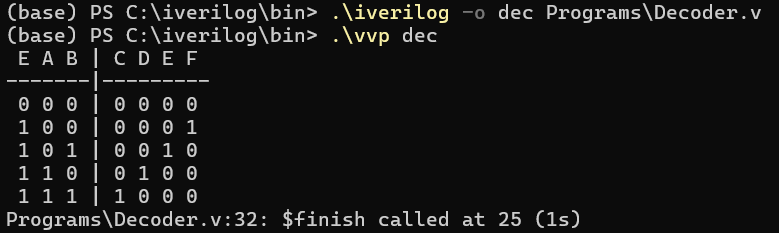
**(Decoder 2:4)**

**Question:** To design and verify a 2:4 decoder.

**Code:**

module decoder (  
 input a, b, E,  
 output c, d, e, f  
);  
  
assign c = E & a & b;  
assign d = E & a & ~b;  
assign e = E & ~a & b;  
assign f = E & ~a & ~b;  
  
endmodule  
  
module test\_decoder;  
reg a, b, E;  
wire c, d, e, f;  
  
// Instantiate the decoder module  
decoder obj (.a(a), .b(b), .E(E), .c(c), .d(d), .e(e), .f(f));  
  
initial begin   
 $display(" E A B | C D E F ");  
 $display("-------|---------");  
 $monitor(" %b %b %b | %b %b %b %b", E, a, b, c, d, e, f);  
  
 // Test cases  
 E = 0; a = 0; b = 0; #5;  
 E = 1; a = 0; b = 0; #5;  
 E = 1; a = 0; b = 1; #5;  
 E = 1; a = 1; b = 0; #5;  
 E = 1; a = 1; b = 1; #5;  
  
 $finish; // End simulation  
end  
  
endmodule

**Output:**

****

**Experiment 9**

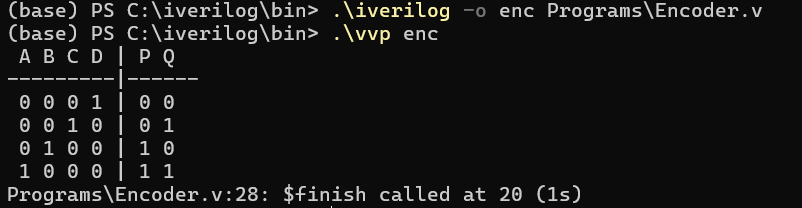
**(Encoder 4:2)**

**Question:** To design and implement a 4:2 encoder.

**Code:**

module encoder (  
 input a, b, c, d,  
 output p, q  
);  
  
assign p = a | b;  
assign q = a | c;  
  
endmodule  
  
module test\_encoder;  
reg a, b, c, d;  
wire p, q;  
  
encoder obj (.a(a), .b(b), .c(c), .d(d), .p(p), .q(q));  
  
initial begin  
 $display(" A B C D | P Q ");  
 $display("---------|------");  
 $monitor(" %b %b %b %b | %b %b", a, b, c, d, p, q);  
  
 // Test cases  
 a = 0; b = 0; c = 0; d = 1; #5;  
 a = 0; b = 0; c = 1; d = 0; #5;  
 a = 0; b = 1; c = 0; d = 0; #5;  
 a = 1; b = 0; c = 0; d = 0; #5;  
  
 $finish; // End the simulation  
end  
  
endmodule

**Output:**

****

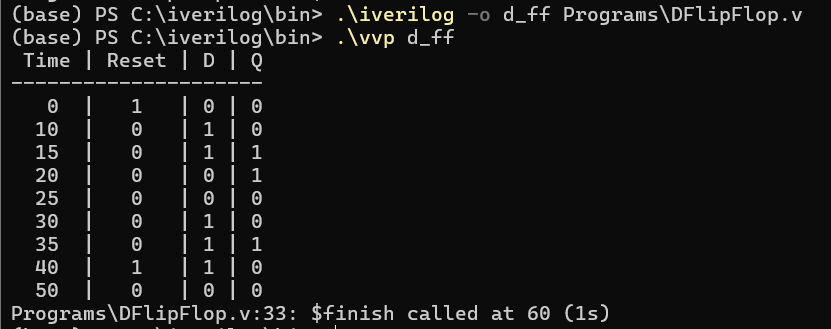
**Experiment 10**

**(D Flip-Flop)**

**Question:** To design and verify the operation of D flip-flops using logic gates

**Code:**

**module** d\_flip\_flop ( **input** D, **input** clk, **input** reset, **output** **reg** Q );  
 **always** @(**posedge** clk **or** **posedge** reset) **begin**  
 **if** (reset)  
 Q <= 0;  
 **else**  
 Q <= D;  
 **end**  
**endmodule**  
  
  
**module** test\_d\_flip\_flop;  
**reg** D, clk, reset;  
**wire** Q;  
  
d\_flip\_flop uut ( .D(D), .clk(clk), .reset(reset), .Q(Q));  
  
**initial** **begin**  
 clk = 0;  
 **forever** #5 clk = ~clk; // Generate a clock with a period of 10 units  
**end**  
  
**initial** **begin**  
 $display(" Time | Reset | D | Q ");  
 $display("---------------------");  
 $monitor("%4t | %b | %b | %b", $time, reset, D, Q);  
  
 reset = 1; D = 0; #10; // Reset the Flip-Flop  
 reset = 0; D = 1; #10; // Apply D=1  
 D = 0; #10; // Change D to 0  
 D = 1; #10; // Change D to 1  
 reset = 1; #10; // Reset again  
 reset = 0; D = 0; #10; // Release reset and set D=0  
 $finish; // End simulation  
**end**  
**endmodule**

**Output:**

**Experiment 11**

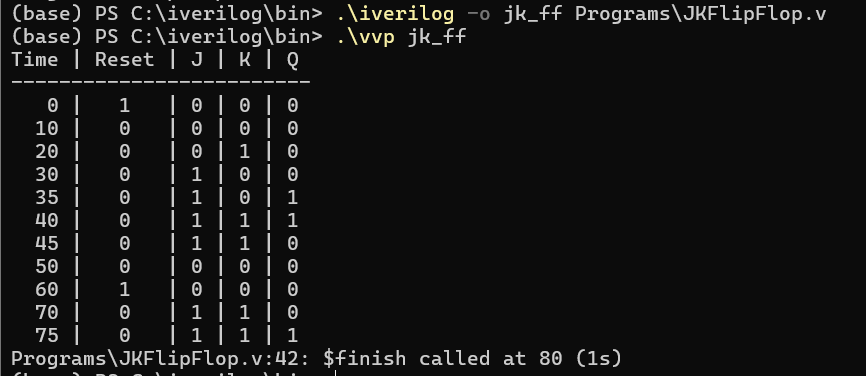
**(JK Flip-Flop)**

**Question:** To design and verify the operation of JK flip-flops using logic gates

**Code:**

module test\_jk\_flip\_flop;  
reg J, K, clk, reset;  
wire Q;  
  
// Instantiate the JK Flip-Flop  
jk\_flip\_flop uut (  
 .J(J),  
 .K(K),  
 .clk(clk),  
 .reset(reset),  
 .Q(Q)  
);  
  
// Clock generation  
initial begin  
 clk = 0;  
 forever #5 clk = ~clk; // Generate a clock with a period of 10 units  
end  
  
// Apply test cases  
initial begin  
 $display("Time | Reset | J | K | Q ");  
 $display("-------------------------");  
 $monitor("%4t | %b | %b | %b | %b", $time, reset, J, K, Q);  
  
 reset = 1; J = 0; K = 0; #10; // Reset the Flip-Flop  
 reset = 0; J = 0; K = 0; #10; // No change  
 J = 0; K = 1; #10; // Reset Q  
 J = 1; K = 0; #10; // Set Q  
 J = 1; K = 1; #10; // Toggle Q  
 J = 0; K = 0; #10; // No change  
 reset = 1; #10; // Reset again  
 reset = 0; J = 1; K = 1; #10; // Toggle Q  
 $finish; // End simulation  
end  
  
endmodule

**Output:**

****

**Experiment 12**

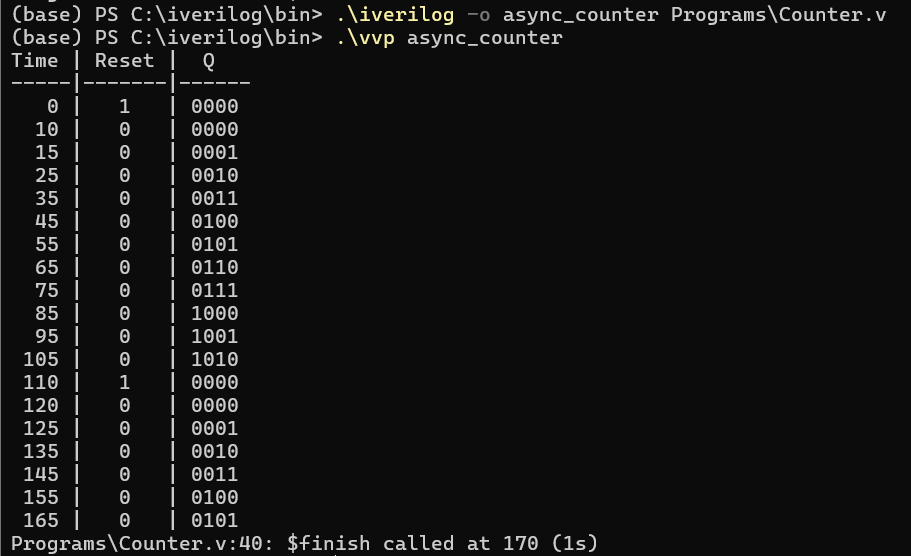
**(Counter)**

**Question:** To verify the operation of asynchronous counter

**Code:**

**module** async\_counter (  
 **input** clk, // Clock input  
 **input** reset, // Asynchronous reset  
 **output** [3:0] Q // 4-bit counter output  
);  
  
**reg** [3:0] count;  
  
**always** @(**posedge** clk **or** **posedge** reset) **begin**  
 **if** (reset)  
 count <= 4'b0000; // Reset counter to 0  
 **else**  
 count <= count + 1; // Increment counter  
**end**  
  
**assign** Q = count;  
**endmodule**  
  
  
**module** test\_async\_counter;  
**reg** clk, reset;  
**wire** [3:0] Q;  
async\_counter uut (.clk(clk), .reset(reset), .Q(Q));  
  
**initial** **begin**  
 clk = 0;  
 **forever** #5 clk = ~clk;  
**end**  
  
// Apply test cases  
**initial** **begin**  
 $display("Time | Reset | Q");  
 $display("-----|-------|------");  
 $monitor("%4t | %b | %b", $time, reset, Q);  
  
 reset = 1; #10; // Reset the counter  
 reset = 0; #100; // Allow counter to increment  
 reset = 1; #10; // Reset again  
 reset = 0; #50; // Allow counter to increment again  
 $finish; // End simulation  
**end**  
  
**endmodule**

**Output:**

****