UCS704 EMBEDDED SYSTEMS DESIGN

Submitted To:
Ms. Manisha Singh

Submitted By:
Name: Jatin Kaushal

Roll No: 102103074
Group: 4CO3

Date: 11-Nov-24

Experiment 1 (Truth Table and Logic Gates)
To study and verify the truth table of various logic gates (NOT, AND, OR, NAND, NOR, EX-OR, & EX-N
Code :

```
module logic_gates_test;

  reg a, b;


  wire not_a;
  wire and_ab, or_ab, nand_ab, nor_ab, xor_ab, xnor_ab;

  not U1 (not_a, a);
  and U2 (and_ab, a, b);
  or  U3 (or_ab, a, b);
  nand U4 (nand_ab, a, b);
  nor U5 (nor_ab, a, b);
  xor U6 (xor_ab, a, b);
  xnor U7 (xnor_ab, a, b);
```

```
  initial begin
    $display("A B | NOT AND OR NAND NOR XOR XNOR");
    $display("--------------------------------");



    a = 0; b = 0; #10 display_values();
    a = 0; b = 1; #10 display_values();
    a = 1; b = 0; #10 display_values();
    a = 1; b = 1; #10 display_values();

    $finish;
  end

  task display_values;
    $display("%b %b |  %b   %b   %b   %b    %b   %b    %b", a, b, not_a, and_ab, or_ab, nand_ab, nor_a
  endtask

endmodule
```

OUTPUT :

For NOT gate input A has been considered

Experiment 2 (Half Adder)

To design and verify a half adder using S= (x+y)(x'+y') C= xy

Code :

```
module half_adder (
  input x,         // First input
  input y,         // Second input
```

```
  output S,        // Sum output
  output C         // Carry output
);

  // Implement the sum and carry using the given equations
  assign S = (x | y) & (~x | ~y);  // S = (x + y)(x' + y')
  assign C = x & y;                // C = xy

endmodule
```

The Test Bench Code to verify :

```
module half_adder_tb;

  reg x, y;         // Test inputs
  wire S, C;        // Outputs from the half adder

  // Instantiate the half adder module
  half_adder uut (
    .x(x),
    .y(y),
    .S(S),
    .C(C)
  );

  // Test all possible combinations of x and y
  initial begin
    $display("x y | S C");  // Header for the output table
    $display("-------------");

    x = 0; y = 0; #10 $display("%b %b | %b %b", x, y, S, C);
    x = 0; y = 1; #10 $display("%b %b | %b %b", x, y, S, C);
    x = 1; y = 0; #10 $display("%b %b | %b %b", x, y, S, C);
    x = 1; y = 1; #10 $display("%b %b | %b %b", x, y, S, C);

    $finish;
  end

endmodule
```

Experiment 3 (Full Adder)

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

Code :

```
module full_adder (
  input x,       // First input
  input y,       // Second input
  input z,       // Third input (carry-in)
  output S,      // Sum output
  output C       // Carry output
);

  // Implement the sum and carry using the given equations
  assign S = (~x & ~y & z) | (~x & y & ~z) | (x & ~y & ~z) | (x & y & z);
  assign C = (x & y) | (x & z) | (y & z);

endmodule
```

Test-Bench Code :

```
module full_adder_tb;

  reg x, y, z;   // Test inputs
  wire S, C;     // Outputs from the full adder

  // Instantiate the full adder module
  full_adder uut (
    .x(x),
    .y(y),
```

```
    .z(z),
    .S(S),
    .C(C)
  );

  // Test all possible combinations of x, y, and z
  initial begin
    $display("x y z | S C");  // Header for the output table
    $display("---------------");

    x = 0; y = 0; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 0; y = 0; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 0; y = 1; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 0; y = 1; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 1; y = 0; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 1; y = 0; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 1; y = 1; z = 0; #10 $display("%b %b %b | %b %b", x, y, z, S, C);
    x = 1; y = 1; z = 1; #10 $display("%b %b %b | %b %b", x, y, z, S, C);

    $finish;
  end

endmodule
```

Experiment 4 (Half Subtractor)

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

Code :

```
module half_subtractor (
  input x,       // Minuend
  input y,       // Subtrahend
  output D,      // Difference output
  output B       // Borrow output
);
```

```
  // Implement the difference and borrow using the given equations
  assign D = (~x & y) | (x & ~y);  // D = x'y + xy'
  assign B = ~x & y;               // B = x'y

endmodule
```

TestBench Code

```
module half_subtractor_tb;

  reg x, y;         // Test inputs
  wire D, B;        // Outputs from the half subtractor

  // Instantiate the half subtractor module
  half_subtractor uut (
    .x(x),
    .y(y),
    .D(D),
    .B(B)
  );

  // Test all possible combinations of x and y
  initial begin
    $display("x y | D B");  // Header for the output table
    $display("-------------");

    x = 0; y = 0; #10 $display("%b %b | %b %b", x, y, D, B);
    x = 0; y = 1; #10 $display("%b %b | %b %b", x, y, D, B);
    x = 1; y = 0; #10 $display("%b %b | %b %b", x, y, D, B);
    x = 1; y = 1; #10 $display("%b %b | %b %b", x, y, D, B);

    $finish;
  end

endmodule
```

Experiment 5 (Number Converter)

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

Code :

```
module bcd_to_excess3 (
  input A, B, C, D,   // BCD input
  output W, X, Y, Z   // Excess-3 output
);

  // Implement the Excess-3 outputs using the derived Boolean expressions
  assign W = A | (B & (C | D));      // W = A + B(C + D)
  assign X = (~B & D) | (~B & C) | (B & ~D);  // X = B'D + B'C + BD'
  assign Y = C ^ D;                  // Y = C XOR D
  assign Z = ~D;                     // Z = D'

endmodule
```

Test Bench Code

```
module bcd_to_excess3_tb;

  reg A, B, C, D;          // BCD inputs
  wire W, X, Y, Z;         // Excess-3 outputs

  // Instantiate the converter module
  bcd_to_excess3 uut (
    .A(A),
    .B(B),
    .C(C),
    .D(D),
```

```
    .W(W),
    .X(X),
    .Y(Y),
    .Z(Z)
  );

  // Apply test cases
  initial begin
    $display(" A  B  C  D | W  X  Y  Z ");
    $display("-------------------------");

    A = 0; B = 0; C = 0; D = 0; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 0; C = 0; D = 1; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 0; C = 1; D = 0; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 0; C = 1; D = 1; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 1; C = 0; D = 0; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 1; C = 0; D = 1; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 1; C = 1; D = 0; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 0; B = 1; C = 1; D = 1; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 1; B = 0; C = 0; D = 0; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z
    A = 1; B = 0; C = 0; D = 1; #10 $display(" %b  %b  %b  %b | %b  %b  %b  %b", A, B, C, D, W, X, Y, Z

    $finish;
  end

endmodule
```

Experiment 6 (Multiplexer)

To design and implement a 4:1 multiplexer

Code :

```
module mux_4_to_1 (
  input I0, I1, I2, I3,   // Data inputs
  input S0, S1,           // Selection lines
  output Y                // Output
);

  // Implement the 4:1 MUX logic
  assign Y = (~S1 & ~S0 & I0) | (~S1 & S0 & I1) | (S1 & ~S0 & I2) | (S1 & S0 & I3);

endmodule
```

Test-Bench Code

```
module mux_4_to_1_tb;

  reg I0, I1, I2, I3;   // Data inputs
  reg S0, S1;           // Selection lines
  wire Y;               // Output from the MUX

  // Instantiate the 4:1 MUX
  mux_4_to_1 uut (
    .I0(I0),
    .I1(I1),
    .I2(I2),
    .I3(I3),
```

```
    .S0(S0),
    .S1(S1),
    .Y(Y)
  );

  // Test all combinations
  initial begin
    $display("S1 S0 | I0 I1 I2 I3 | Y ");
    $display("-----------------------");

    I0 = 1; I1 = 0; I2 = 1; I3 = 0;

    S1 = 0; S0 = 0; #10 $display(" %b  %b  |  %b  %b  %b  %b  | %b", S1, S0, I0, I1, I2, I3, Y);
    S1 = 0; S0 = 1; #10 $display(" %b  %b  |  %b  %b  %b  %b  | %b", S1, S0, I0, I1, I2, I3, Y);
    S1 = 1; S0 = 0; #10 $display(" %b  %b  |  %b  %b  %b  %b  | %b", S1, S0, I0, I1, I2, I3, Y);
    S1 = 1; S0 = 1; #10 $display(" %b  %b  |  %b  %b  %b  %b  | %b", S1, S0, I0, I1, I2, I3, Y);

    $finish;
  end

endmodule
```

Experiment 7 (Demultiplexer)

To design and implement a 1:4 demultiplexer.

Code :

```
module demux_1_to_4 (
  input D,          // Data input
  input S0, S1,     // Selection lines
```

```
  output Y0, Y1, Y2, Y3  // Outputs
);

  // Implement the 1:4 DEMUX logic
  assign Y0 = ~S1 & ~S0 & D;
  assign Y1 = ~S1 & S0 & D;
  assign Y2 = S1 & ~S0 & D;
  assign Y3 = S1 & S0 & D;

endmodule
```

Test Bench Code

```
module demux_1_to_4_tb;

  reg D;           // Data input
  reg S0, S1;      // Selection lines
  wire Y0, Y1, Y2, Y3;  // Outputs

  // Instantiate the 1:4 DEMUX
  demux_1_to_4 uut (
    .D(D),
    .S0(S0),
    .S1(S1),
    .Y0(Y0),
    .Y1(Y1),
    .Y2(Y2),
    .Y3(Y3)
  );

  // Test all combinations
  initial begin
    $display("S1 S0 | D | Y0 Y1 Y2 Y3");
    $display("----------------------");

    D = 1;

    S1 = 0; S0 = 0; #10 $display(" %b  %b  | %b | %b  %b  %b  %b", S1, S0, D, Y0, Y1, Y2, Y3);
    S1 = 0; S0 = 1; #10 $display(" %b  %b  | %b | %b  %b  %b  %b", S1, S0, D, Y0, Y1, Y2, Y3);
    S1 = 1; S0 = 0; #10 $display(" %b  %b  | %b | %b  %b  %b  %b", S1, S0, D, Y0, Y1, Y2, Y3);
```

```
    S1 = 1; S0 = 1; #10 $display(" %b  %b  | %b | %b  %b  %b  %b", S1, S0, D, Y0, Y1, Y2, Y3);

    $finish;
  end

endmodule
```

Experiment 8 (Decoder)

To design and verify a 2:4 decoder.

Code :

```
module decoder_2_to_4 (
  input A, B,         // Inputs
  output Y0, Y1, Y2, Y3  // Outputs
);

  // Implement the 2:4 Decoder logic
  assign Y0 = ~A & ~B;
  assign Y1 = ~A & B;
  assign Y2 = A & ~B;
  assign Y3 = A & B;

endmodule
```

Test Bench Code :

```
module decoder_2_to_4_tb;

  reg A, B;          // Inputs
  wire Y0, Y1, Y2, Y3;  // Outputs

  // Instantiate the 2:4 Decoder
  decoder_2_to_4 uut (
```

```
    .A(A),
    .B(B),
    .Y0(Y0),
    .Y1(Y1),
    .Y2(Y2),
    .Y3(Y3)
  );

  // Test all combinations
  initial begin
    $display("A B | Y0 Y1 Y2 Y3");
    $display("-----------------");

    A = 0; B = 0; #10 $display("%b %b | %b  %b  %b  %b", A, B, Y0, Y1, Y2, Y3);
    A = 0; B = 1; #10 $display("%b %b | %b  %b  %b  %b", A, B, Y0, Y1, Y2, Y3);
    A = 1; B = 0; #10 $display("%b %b | %b  %b  %b  %b", A, B, Y0, Y1, Y2, Y3);
    A = 1; B = 1; #10 $display("%b %b | %b  %b  %b  %b", A, B, Y0, Y1, Y2, Y3);

    $finish;
  end

endmodule
```

Experiment 9 (Encoder)

To design and implement a 4:2 encoder.

Code :

```
module encoder_4_to_2 (
  input I0, I1, I2, I3,  // Inputs
  output Y0, Y1          // Outputs
);

  // Implement the 4:2 Encoder logic
  assign Y0 = I1 | I3;
  assign Y1 = I2 | I3;

endmodule
```

Test Bench Code

```
module encoder_4_to_2_tb;

  reg I0, I1, I2, I3;  // Inputs
  wire Y0, Y1;         // Outputs

  // Instantiate the 4:2 Encoder
  encoder_4_to_2 uut (
    .I0(I0),
    .I1(I1),
    .I2(I2),
    .I3(I3),
    .Y0(Y0),
    .Y1(Y1)
  );

  // Test all input combinations with only one input high
  initial begin
    $display("I3 I2 I1 I0 | Y1 Y0");
    $display("--------------------");

    I0 = 1; I1 = 0; I2 = 0; I3 = 0; #10 $display("%b  %b  %b  %b  | %b  %b", I3, I2, I1, I0, Y1, Y0);
    I0 = 0; I1 = 1; I2 = 0; I3 = 0; #10 $display("%b  %b  %b  %b  | %b  %b", I3, I2, I1, I0, Y1, Y0);
    I0 = 0; I1 = 0; I2 = 1; I3 = 0; #10 $display("%b  %b  %b  %b  | %b  %b", I3, I2, I1, I0, Y1, Y0);
    I0 = 0; I1 = 0; I2 = 0; I3 = 1; #10 $display("%b  %b  %b  %b  | %b  %b", I3, I2, I1, I0, Y1, Y0);

    $finish;
```

```
  end

endmodule
```