LABORATORY_1

Decoders and Muxes

Hannah Hwang | SID : 862419233 | CS120A Section 025
Adithya Chander | SID : 862429692 | CS120A Section 025
Calvin Hong | SID : 862416275 | CS120A Section 025

Overview

In part I of the lab we went over a guided design of a sprinkler valve controller – a 3 x 8 decoder with an “enable” switch. In part A we were given an explanation/diagram of the system created using an MCU with four inputs(E[enable], A, B, C), a decoder with 8 outputs(d0~d7), and a truth table representing the given system. Then went over the construction of a SOP minterm based logic expression for each data output and how to implement it as a gate based logic circuit. In part B, we explored how to implement the sprinkler system in Verilog to practice structural and behavioral modeling. Using the provided code, we added the remaining outputs using the and4 module and generated a waveform through EDA Playground which matched *[figure 1].

In part II of the lab, we went over the design of a 4 x 1 multiplexer. We were given the specifications of the system, having four inputs, two selection inputs, and one output, for a single wired data bus. Then we implemented the behavioral and structural models of the design using Verilog. Then, we made a truth table, algebraic expression, and logic circuit schematic that demonstrates the design of the system and how the inputs correlate to their outputs. The waveforms generated by the implementation can be seen in *[figure 2].

Analysis

[Part 1]

Step - 1

E	A	B	C	d0	d1	d2	d3	d4	d5	d6	d7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Table 1. Truth Table of the Sprinkler System ('x' stands for "don't care")

Step - 2 (equations were derived from rows of the truth table above [Table 1])
d0 = E * A' * B' * C'
d1 = E * A' * B' * C
d2 = E * A' * B * C'
d3 = E * A' * B * C
d4 = E * A * B' * C'
d5 = E * A * B' * C
d6 = E * A * B * C'
d7 = E * A * B * C

Step - 3 explain how the logic gate diagram represents the equations captured in step 1

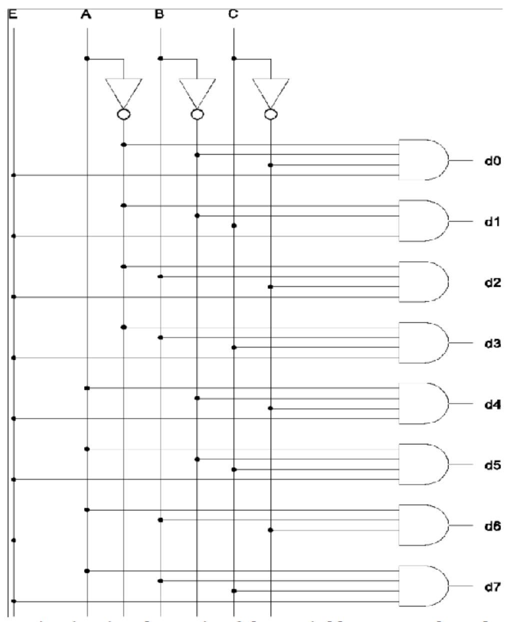


Figure 3. Logic Circuit Schematic of the Sprinkler System based on **Table 1**

The equations from step 1 are represented in the logic gate diagram on the left. There are four input wires: E, which is the enabler input, and A, B, and C, which are the representations of the binary value of whichever sprinkler zone will be active.

These inputs will determine which output (d0-d7) will be selected by the signals for the enabler and combined value of wires ABC, which would in turn activate the valve control and turn on the sprinkler.

[Part 2]

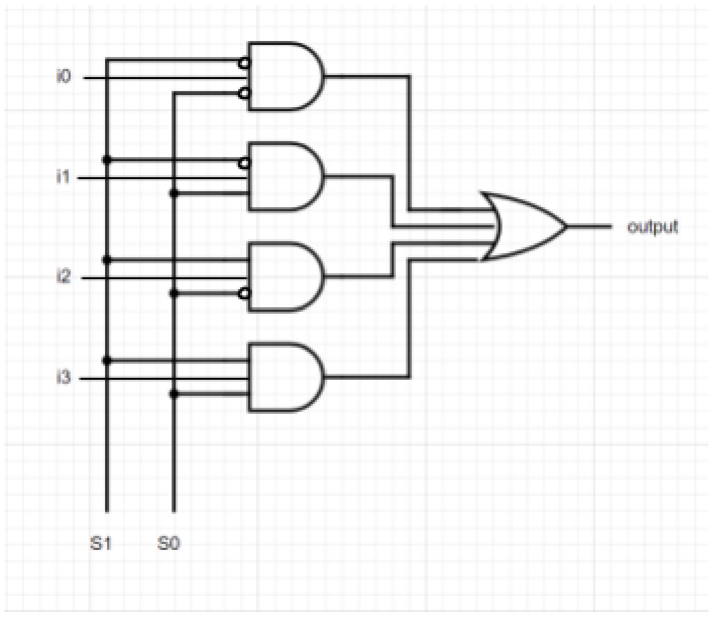
Truth Table

S0	S1	d
0	0	i0
0	1	i1
1	0	i2
1	1	i3

Algebraic Expressions

S0’ * S1’ = i0
S0’ * S1 = i1
S0 * S1’ = i2
S0 * S1 = i3

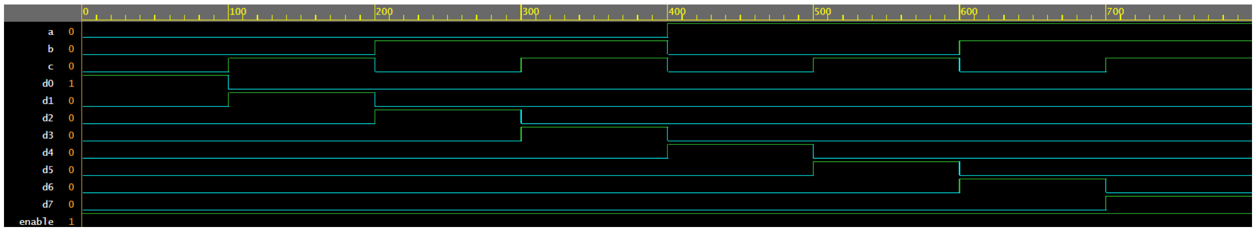
Logic Circuit Schematic



Records

[Part I]

Waveform:



[FIGURE 1] - screenshot of waveform generated by EDA Playground for Part I

Code**

[Testbench]

```
// Code your testbench here
// or browse Examples
module decoder_tb;
        //inputs
        reg enable;
        reg a;
        reg b;
        reg c;
        // Outputs
        wire d0;
        wire d1;
        wire d2;
        wire d3;
        wire d4;
        wire d5;
        wire d6;
        wire d7;
        // Instantiate the Unit Under Test (UUT)
        decoder_st uut ( // change to "decoder_bh" for testing your behavioral model
                .enable(enable),
                .a(a),
                .b(b),
                .c(c),
                .d0(d0),
                .d1(d1),
                .d2(d2),
                .d3(d3),
                .d4(d4),
                .d5(d5),
                .d6(d6),
                .d7(d7)
        );
        initial begin
        $dumpfile("dump.vcd"); $dumpvars;
                enable = 1;
                a = 0;
                b = 0;
                c = 0;
                #100; // Wait for 100 ns
                  $display("TC11 ");

                  if ( d0 != 1'b1 ) $display ("Result is wrong");
                a = 0;
                b = 0;
                c = 1;
                #100;
                $display("TC12 ");
                if ( d1 != 1'b1 ) $display ("Result is wrong");
                a = 0;
                b = 1;
                c = 0;
                #100;
                $display("TC13 ");
                if ( d2 != 1'b1 ) $display ("Result is wrong");
                a = 0;
                b = 1;
                c = 1;
                #100;
                $display("TC14 ");
                if ( d3 != 1'b1 ) $display ("Result is wrong");
                a = 1;
                b = 0;
                c = 0;
                #100;
                $display("TC15 ");
                if ( d4 != 1'b1 ) $display ("Result is wrong");
                a = 1;
                b = 0;
                c = 1;
                #100;
                $display("TC16 ");
                if ( d5 != 1'b1 ) $display ("Result is wrong");
```

```
                    a = 1;
                    b = 1;
                    c = 0;
                    #100;
                    $display("TC17 ");
                    if ( d6 != 1'b1 ) $display ("Result is wrong");
                    a = 1;
                    b = 1;
                    c = 1;
                    #100;
                    $display("TC18 ");
                    if ( d7 != 1'b1 ) $display ("Result is wrong");
                    // Your test cases ******************
          end
endmodule
```

[Design]

```
// Code your design here
module and4(
          input wire enable ,
          input wire a,
          input wire b,
          input wire c,
          output wire r
);
//
          assign r = enable & a & b & c ;
endmodule
```

// **structural model**

module decoder_st(

```
// I/0 ports
          input wire enable ,
          input wire a ,
          input wire b ,
          input wire c ,
          output wire d0,
          output wire d1,
          output wire d2,
          output wire d3,
          output wire d4,
          output wire d5,
          output wire d6,
          output wire d7
);
```

```
// Using the and4 module to set all outputs
        and4 c1(enable, ~a, ~b, ~c, d0) ;
        and4 c2(enable, ~a, ~b, c, d1);
        and4 c3(enable, ~a, b, ~c, d2);
        and4 c4(enable, ~a, b, c, d3);
        and4 c5(enable, a, ~b, ~c, d4);
        and4 c6(enable, a, ~b, c, d5);
        and4 c7(enable, a, b, ~c, d6);
        and4 c8(enable, a, b, c, d7);
        // Your code goes here (7 cases left to implement)
endmodule

// behavioral model
module decoder_bh(
// I/0 ports
        input wire enable ,
        input wire a ,
        input wire b ,
        input wire c ,
        output reg d0,
        output reg d1,
        output reg d2,
        output reg d3,
        output reg d4,
        output reg d5,
        output reg d6,
        output reg d7
);

// Code your design here
module and4(
input wire enable ,
input wire a,
input wire b,
input wire c,
output wire r
);
//

assign r = enable & a & b & c ;
Endmodule

// structural model
module decoder_st(
// I/0 ports
        input wire enable ,
        input wire a ,
        input wire b ,
        input wire c ,
        output wire d0,
        output wire d1,
        output wire d2,
        output wire d3,
        output wire d4,
        output wire d5,
        output wire d6,
        output wire d7
);

// Using the and4 module to set all outputs
and4 c1(enable, ~a, ~b, ~c, d0) ;
and4 c2(enable, ~a, ~b, c, d1);
and4 c3(enable, ~a, b, ~c, d2);
and4 c4(enable, ~a, b, c, d3);
and4 c5(enable, a, ~b, ~c, d4);
and4 c6(enable, a, ~b, c, d5);
and4 c7(enable, a, b, ~c, d6);
and4 c8(enable, a, b, c, d7);
// Your code goes here (7 cases left to implement)

endmodule

// behavioral model
module decoder_bh(
// I/0 ports
        input wire enable ,
        input wire a ,
        input wire b ,
        input wire c ,
        output wire d0,
        output wire d1,
        output wire d2,
        output wire d3,
        output wire d4,
        output wire d5,
        output wire d6,
        output wire d7
);
```

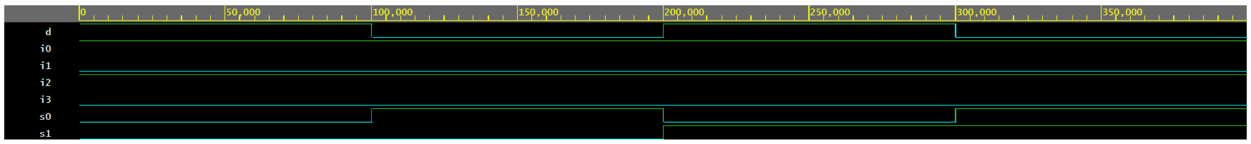
```
// Internal wire
wire [3:0] bundle ;
assign bundle = {enable , a, b, c } ;

// Behavioral description
always @(*) begin
            d0 = 1'b0 ;
            d1 = 1'b0 ;
            d2 = 1'b0 ;
            d3 = 1'b0 ;
            d4 = 1'b0 ;
            d5 = 1'b0 ;
            d6 = 1'b0 ;
            d7 = 1'b0 ;

// Setting the correct output
  case (bundle)
          4'b1000: d0 = 1'b1 ;
          4'b1001: d1 = 1'b1 ;
          4'b1010: d2 = 1'b1 ;
          4'b1011: d3 = 1'b1 ;
          4'b1100: d4 = 1'b1 ;
          4'b1101: d5 = 1'b1 ;
          4'b1110: d6 = 1'b1 ;
          4'b1111: d7 = 1'b1 ;
          default : begin
                    d0 = 1'b0 ;
          end
endcase
end
endmodule
```

[Part 2]

Waveform:



[FIGURE 2]- screenshot of waveform generated by EDA Playground for Part II

Code**
[Design]

```
// Code your design here
module and3(
input wire a,
input wire b,
input wire c,
output wire r
);
assign r = a & b & c ;
endmodule
// structural model
module mux_st(
	// Ports I/O
	input wire s1,
	input wire s0,
	input wire i0,
	input wire i1,
	input wire i2,
	input wire i3,
	output wire d
);
wire r1, r2, r3, r4 ;
and3 c1 ( ~s1,~s0, i0, r1 ) ;
  and3 c2 ( ~s1, s0, i1, r2 ) ;
  and3 c3 ( s1, ~s0, i2, r3 ) ;
  and3 c4 ( s1, s0, i3, r4 ) ;
// my code went here

assign d = r1 | r2 | r3 | r4 ;
endmodule
// behavioral model
module mux_bh(
	// Ports I/O
	input wire s1,
	input wire s0,
	input wire i0,
	input wire i1,
	input wire i2,
	input wire i3,
	output reg d
) ;
always @(*) begin
	d = 1'b0 ;
```

```
        case ( {s1,s0} )
                2'b00 : d = i0 ;
                2'b01 : d = i1 ;
                2'b10 : d = i2 ;
                2'b11 : d = i3 ;
                // your code goes here (3 cases left)
        endcase
end
endmodule
```

[Testbench]

```
// Code your testbench here
// or browse Examples
`timescale 1ns / 1ps
module mux_tb;
        // Inputs
        reg s1;
        reg s0;
        reg i0;
        reg i1;
        reg i2;
        reg i3;
        // Outputs
        wire d;
        // Instantiate the Unit Under Test (UUT)
          mux_st uut ( // change to "mux_st" for testing your structural model
                .s1(s1),
                .s0(s0),
                .i0(i0),
                .i1(i1),
                .i2(i2),
                .i3(i3),
                .d(d)
        );
        initial begin
          $dumpfile("dump.vcd"); $dumpvars;
                i0 = 1;
                i1 = 0;
                i2 = 1;
                i3 = 0;

                s1 = 0;
                s0 = 0;
                #100;
                $display("TC11 ");
                if ( d != i0 ) $display ("Result is wrong");

                s1 = 0;
                s0 = 1;
                #100;
                $display("TC12 ");
                if ( d != i1 ) $display ("Result is wrong");

                s1 = 1;
                s0 = 0;
                #100;
                $display("TC13 ");
                if ( d != i2 ) $display ("Result is wrong");

                s1 = 1;
                s0 = 1;
                #100;
                $display("TC14 ");
                if ( d != i3 ) $display ("Result is wrong");

                // Your test cases
                i0 = 1
                i1 = 1;
                i2 = 1;
                i3 = 1;

                s1 = 0;
                s0 = 0;
                #100;
                $display("TC15 ");
                if ( d != i0 ) $display ("Result is wrong");

                i0 = 0
                i1 = 1;
                i2 = 0;
                i3 = 0;

                s1 = 0;
                s0 = 1;
                #100;
                $display("TC16 ");
                if ( d != i1 ) $display ("Result is wrong");
        end
endmodule
```

Discussion

The systems did work according to the provided specifications.

For Part A, we were expected to use the provided truth tables and functions to design a decoder in Verilog, which worked as intended. The test cases that we needed to implement ourselves ended up being similar in nature to the provided test case, and using that as reference, we were able to create our own and see the waveform visualization. Because of this, no technical issues or problems were encountered that prompted us to make significant changes to the provided code.

For Part B, we were expected to design a multiplexer that took four inputs (i_0, i_1, i_2, and i_3) and chose which input to use and assign that value to the output register, d. It did so by taking two other binary inputs (s_1 and s_0) and using those to determine which of the input wires would be chosen. We didn't encounter any technical difficulties that caused drastic changes to our code or design for the multiplexer, and our final design was similar to the ones discussed in classes. The only realistic way to "improve" the system would be to optimize the amount of transistors by using NAND gates as opposed to AND gates, but apart from that, the multiplexer works as intended.

We did have a mild difficulty in understanding Verilog itself, as none of us are too familiar with the language, so it took some time to decipher what the code itself was attempting to do. This did make writing the test cases take longer than we intended, but after we came to a conclusion on what we were being asked to do, we were able to quickly identify which of the parameters we needed to modify for our test cases.

Conclusion

In this lab, we familiarized ourselves with gate based logic design, designing a decoder and a multiplexer in Verilog. We also utilized truth tables and algebraic expressions to help visualize our logic. These designs serve real life applications, such as a sprinkler system or a computer data bus.

Questions

[Part I]

1. What is a waveform?

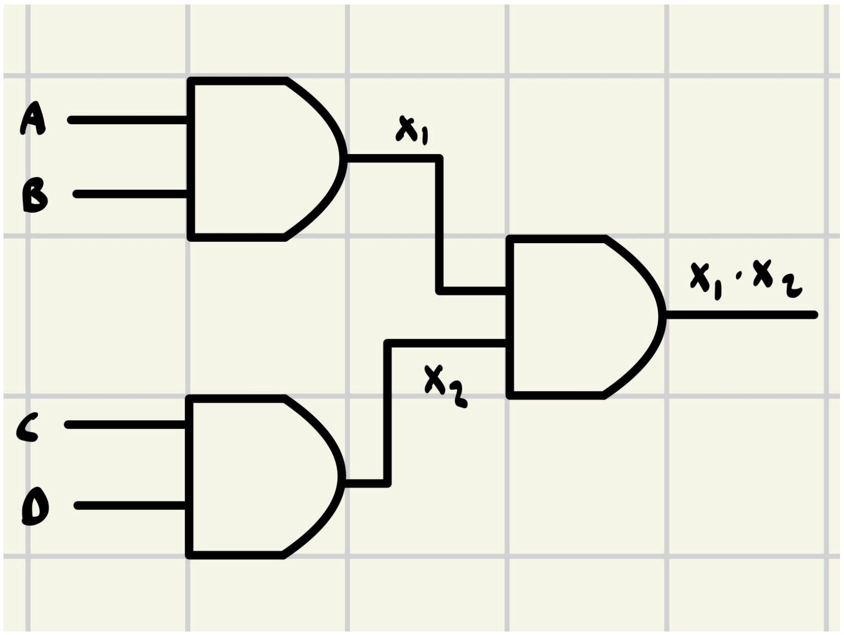
A waveform is a graphical representation of the values going through a signal over time.

2. What is a testbench?

A testbench is a series of tests to make sure that code is producing the correct output. It tests all of the cases.

3. Can we replace the 4-input AND gates in the circuit with the 2-input AND gates? If yes, how?

Yes, we can replace each of the 4-input AND gates with three 2-input AND gates, ANDing the first 2 and the last 2 inputs separately, then putting the outputs of those two AND gates through a third AND gate.



Output = (A*B) * (C*D)
So by Associative Law this is equivalent to a 4 input AND gate Output = (A*B*C*D)