CS/EE 120A
(Section: 021)

Lab #2: Decoders & Muxes

Dan Murphy; SID: 862xxxxxx
dmurp006@ucr.edu

Partner:
Jesse Garcia

Subject Overview

In order to efficiently learn and implement decoders and muxes, the lab was separated into two parts: A and B.

Part A:

Within this section, our goal was to create a Sprinkler Digital Controller System. Fortunately, the schematic was provided for us and we were able to simulate this through the Xilinx IDE. In order to ensure the accuracy of the schematic, a test bench was created. The test bench utilized the template provided from the lab's instructions. After utilizing the test bench, the Simulation Mode allowed us to see and understand the implementation that made the system operate through what its inputs and outputs were.

Part B:

Within this section, we branched out from using the schematic and used a hands-on approach through coding in Verilog. We quickly noticed that there was no way to accept 4 inputs into this system. Through this, we created a module that would be able to accept 4 separate inputs and output them onto designated output wire. Once completed, results were verified through the Verilog module.

New Concepts

1. Describe how this lab built upon previous ones:

This lab did not concretely build upon previous labs as this was our first lab without copying and pasting blocks of code without knowing what exactly they did. However, this lab forced us to get used to using Xilinx and directly saw correlation between inputs and outputs via code and test benches.

2. Describe the most difficult part of this lab for you:

By far, the most difficult part was decrypting the errors given to us through the IDE. For example, some portions of our schematics were "too close" together, causing critical errors which resulted in hours of frustration and confusion. This particular IDE seems to have many quirks and appears to take time to get accustomed to. Not to mention, trying to make a mux did not work for whatever reason, so we had to proceed on the gate level, as shown by the book which is how we were able to get the results we needed.

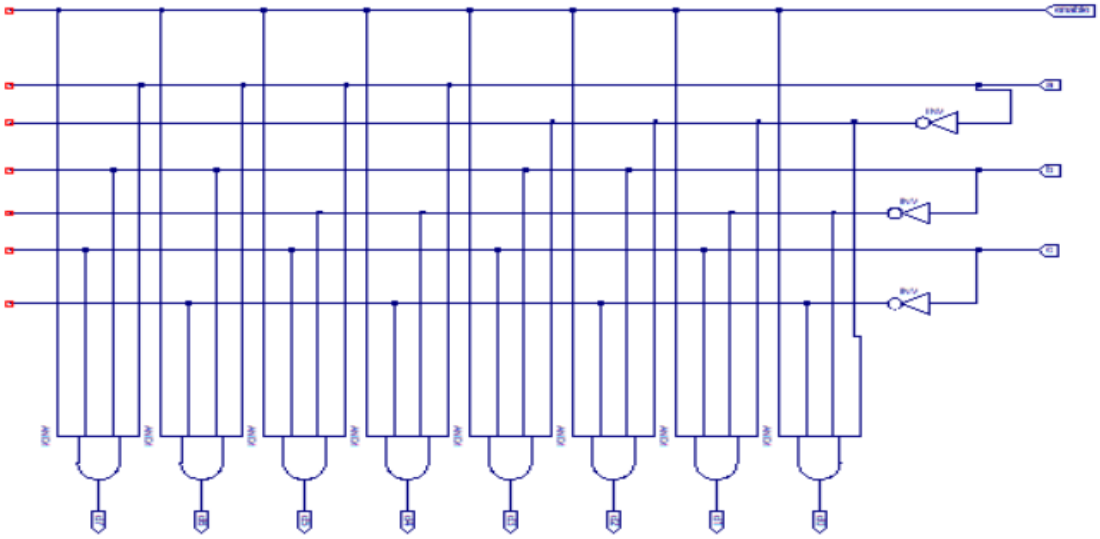
3. Describe problems you faced and how you solved them:

After realizing that we were not the only group suffering from frustration and confusion, we pooled knowledge and debugging-type techniques from various groups and the TA. Eventually, brute force plug-and-chug methods permitted successful implementation.

4. Do you verify that the code included with this report is yours and your partner's original work (yes/no)?

Yes. For the purpose of verifying it is our original code, yes; with the caveat of group collaboration outside of myself and Jesse Garcia. Ideas were shared, code was not. Thus, our code is inherently original.

Decoder Schematics



Decoder Code

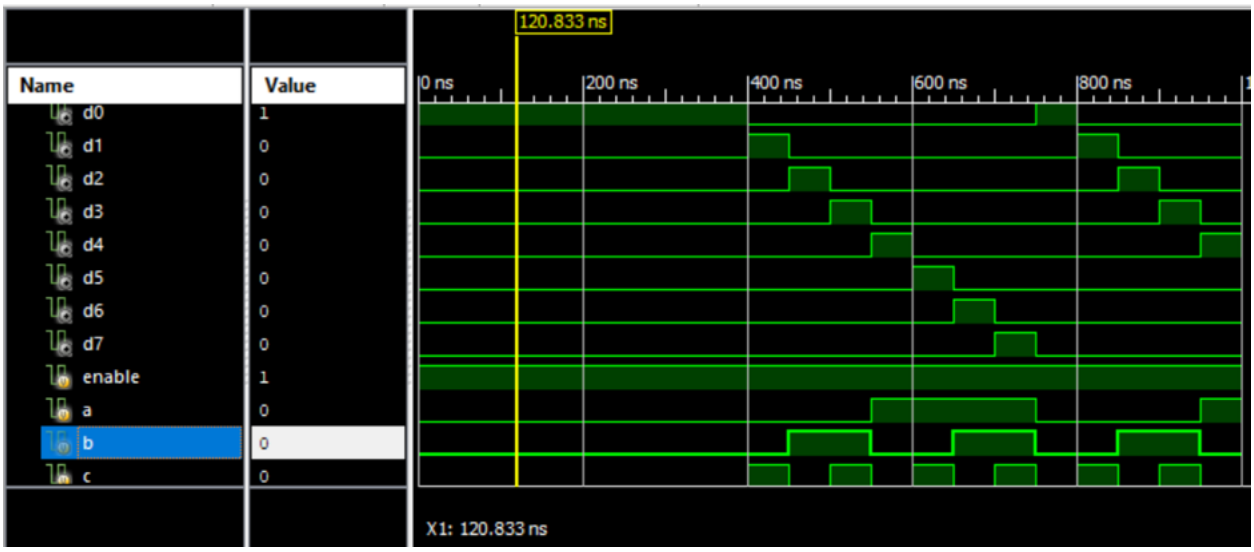
```
module decoder_tb; /* Beginning of module decoder */

// Inputs ---
reg enable; /* Not necessary but is a safety net */
reg a; /* Register a */
reg b; /* Register b */
reg c; /* Register c */

// Outputs ---
wire d0; /* d0 wire (i.e. first wire)   */
```

```
wire d1; /* d1 wire (i.e. second wire)  */
wire d2; /* d2 wire (i.e. third wire)   */
wire d3; /* d3 wire (i.e. fourth wire)  */
wire d4; /* d4 wire (i.e. fifth wire)   */
wire d5; /* d5 wire (i.e. sixth wire)   */
wire d6; /* d6 wire (i.e. seventh wire) */
wire d7; /* d7 wire (i.e. eighth wire)  */

// Instantiate the Unit Under Test (UUT)
        s2 uut(
                .enable(enable),
                .a(a),
                .b(b),
                .c(c),
                .d0(d0),
                .d1(d1),
                .d2(d2),
                .d3(d3),
                .d4(d4),
                .d5(d5),
                .d6(d6),
                .d7(d7)
        ); /* End of UUT --- */

        initial begin
        enable = 1; /* Forces system to be enabled */

        a = 0; /* Initializes register a to zero */
        b = 0; /* Initializes register b to zero */
        c = 0; /* Initializes register c to zero */

/* Here we will have the three different registers a, b, c hold different
values of 0 or 1. */

        #100; /* Wait for 100 ns --- */
        $display("TC11 ");
        if ( d0 != 1'b1 ) $display ("Result is wrong");
                a = 0; /* Sets a to 0 */
                b = 0; /* Sets b to 0 */
                c = 1; /* Sets c to 1 */

        #100; /* Wait for 100 ns --- */
        $display("TC12 ");
        if ( d1 != 1'b1 ) $display ("Result is wrong");
                a = 0; /* Sets a to 0 */
```

```
        b = 1; /* Sets b to 1 */
        c = 0; /* Sets c to 0 */

#100; /* Wait for 100 ns --- */
$display("TC13 ");
if ( d2 != 1'b1 ) $display ("Result is wrong");
        a = 0; /* Sets a to 0 */
        b = 1; /* Sets b to 1 */
        c = 1; /* Sets c to 1 */

#100; /* Wait for 100 ns --- */
$display("TC14 ");
if ( d3 != 1'b1 ) $display ("Result is wrong");
        a = 1; /* Sets a to 1 */
        b = 0; /* Sets b to 0 */
        c = 0; /* Sets c to 0 */

#100; /* Wait for 100 ns --- */
$display("TC15 ");
if ( d4 != 1'b1 ) $display ("Result is wrong");
        a = 1; /* Sets a to 1 */
        b = 0; /* Sets b to 0 */
        c = 1; /* Sets c to 1 */

#100; /* Wait for 100 ns --- */
$display("TC16 ");
if ( d5 != 1'b1 ) $display ("Result is wrong");
        a = 1; /* Sets a to 1 */
        b = 1; /* Sets b to 1 */
        c = 0; /* Sets c to 0 */

#100; /* Wait for 100 ns --- */
$display("TC17 ");
if ( d6 != 1'b1 ) $display ("Result is wrong");
        a = 1; /* Sets a to 1 */
        b = 1; /* Sets b to 1 */
        c = 1; /* Sets b to 0 */

#100; /* Wait for 100 ns --- */
$display("TC18 ");
if ( d7 != 1'b1 ) $display ("Result is wrong");
 a = 0; /* Sets a to 0 */
 b = 0; /* Sets b to 0 */
 c = 0; /* Sets c to 0 */
```

```
        #100; /* Wait for 100 ns --- */
        if ( d0 != 1'b1 ) $display ("Result is wrong");
                a = 0; /* Sets a to 0 */
                b = 0; /* Sets b to 0 */
                c = 1; /* Sets c to 1 */

        #100; /* Wait for 100 ns --- */
        $display("TC12 ");
        if ( d1 != 1'b1 ) $display ("Result is wrong");
                a = 0; /* Sets a to 0 */
                b = 1; /* Sets b to 1 */
                c = 0; /* Sets c to 0 */

        #100; /* Wait for 100 ns --- */
        if ( d2 != 1'b1 ) $display ("Result is wrong");
                a = 0; /* Sets a to 0 */
                b = 1; /* Sets b to 1 */
                c = 1; /* Sets c to 1 */

        #100; /* Wait for 100 ns --- */
        $display("TC14 ");
        if ( d3 != 1'b1 ) $display ("Result is wrong");
                a = 1; /* Sets a to 1 */
                b = 0; /* Sets b to 0 */
                c = 0; /* Sets c to 0 */

        #100; /* Wait for 100 ns --- */

        end /* End of local scope */

endmodule /* End of module --- */
```

Decoder Simulation Output



Questions

1. What is a waveform?

   A waveform is the shape and form of a signal at a particular time. The lab represents waveforms through the change of binary values via input and output.

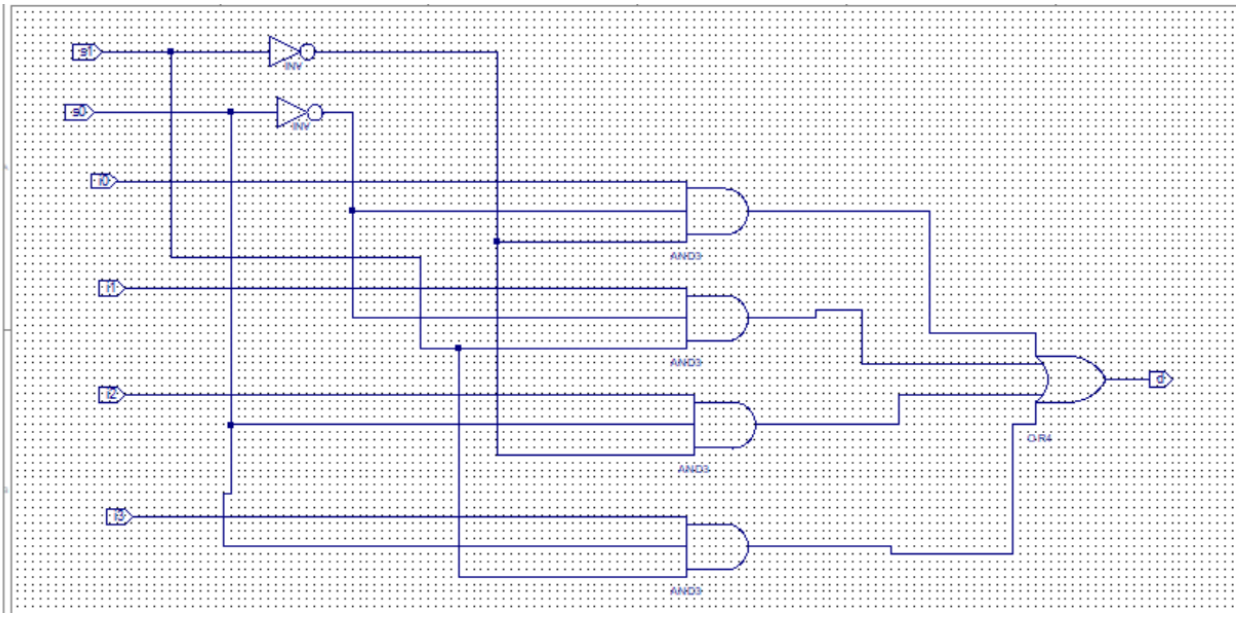
2. What is a testbench?

   A testbench is code written as a safety net which tests edge cases and ensures the code logic is sound. This allows the program to run smoothly and not force human intervention upon a crash (i.e. a program can handle whatever situations it may be dealt).

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

   Of course! Rather than all four inputs forced into the [AND] gate, we can merely divide and conquer; two inputs are directed to an [AND] gate. Its output can be directed to another [AND] gate. Since this is low-level logic, everything is boolean logic, despite it potentially being nested.

Multiplexer Schematics

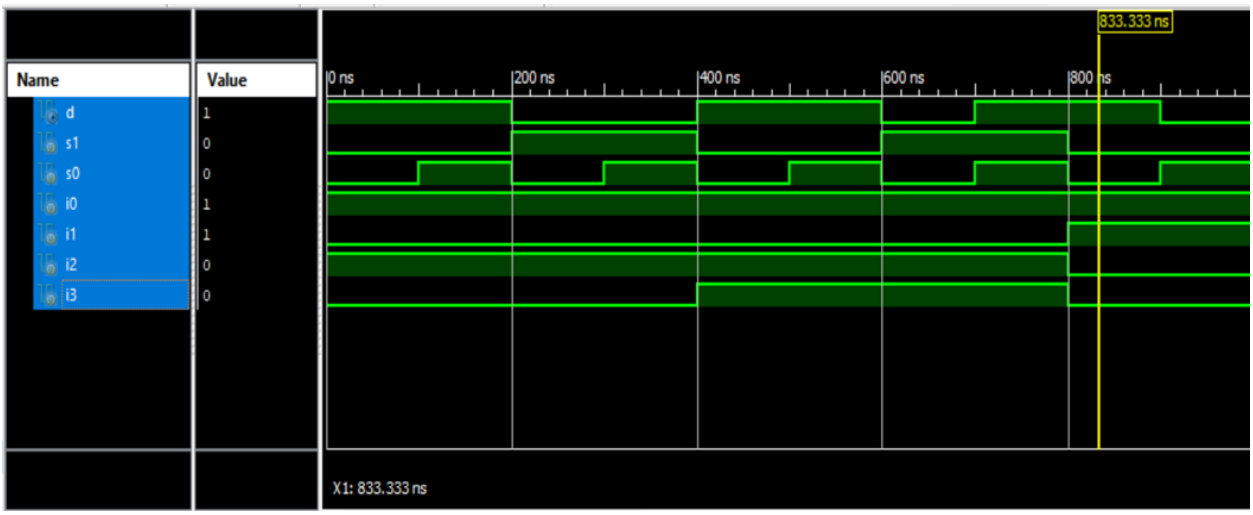


Multiplexer Code

```
module mux_sch_tb; /* Beginning of mux module */

        // Inputs ---

        // 's' registers----------------------
        reg s1; /* Declaring register s1 */
        reg s0; /* Declaring register s0 */

        // 'i' registers ---------------------
        reg i0; /* Declaring register i0 */
        reg i1; /* Declaring register i1 */
        reg i2; /* Declaring register i2 */
        reg i3; /* Declaring register i3 */

        // Outputs ---
        wire d; /* Declaring wire d */

        // Instantiate the Unit Under Test (UUT)
        mux uut (
                .s1(s1),
                .s0(s0),
                .i0(i0),
                .i1(i1),
                .i2(i2),
                .i3(i3),
```

```
        .d(d)
    ); /* End of mux UUT --- */

    initial begin

        /* Initializing registers */

        s1 = 0; /* Initializes s1 to 0 */
        s0 = 0; /* Initializes s0 to 0 */

        i0 = 1; /* Initializes i0 to 1 */
        i1 = 0; /* Initializes i1 to 0 */
        i2 = 1; /* Initializes i2 to 1 */
        i3 = 0; /* Initializes i3 to 0 */


        #100; /* Wait for 100 ns --- */
        $display("TC11 ");
        if ( d != i0 ) $display ("Result is wrong");

        s1 = 0; /* Initializes s1 to 0 */
        s0 = 1; /* Initializes s0 to 1 */
                #100; /* Wait for 100 ns --- */
        $display("TC12 ");
        if ( d != i1 ) $display ("Result is wrong");

        s1 = 1; /* Initializes s1 to 1 */
        s0 = 0; /* Initializes s0 to 0 */
        #100; /* Wait for 100 ns --- */
        $display("TC13 ");
        if ( d != i2 ) $display ("Result is wrong");

        s1 = 1; /* Initializes s1 to 1 */
        s0 = 1; /* Initializes s0 to 1 */
                #100; /* Wait for 100 ns --- */
        $display("TC14 ");
        if ( d != i3 ) $display ("Result is wrong");

    end /* End of local scope */

endmodule /* End of module --- */
```

Multiplexer Simulation Output



Conclusion

In this lab, it showed us two different methods of creating a module and running the simulation. One way, we created a decoder and a multiplexer schematic using AND gates and inverters and made sure they worked through simulations within Xilinx.

After this, simulations were to test for edge cases and complete functionality. By doing this, it allowed us to utilize the board, familiarize ourselves with Xilinx, Verilog and designing circuits to be implemented into functional tech.

Both of our Decoders and Muxes worked correctly. The decoder gave us the correct output only when the enable was set at 1 and the Mux output was dependent upon the setting of S_0 and S_1.

As a whole, the circuit design and Boolean logic allowed us to see the implementation of the decoder and multiplexor in. Once the results expected were yielded, our sprinkler system was successfully completed.