Prince Nnamdi Ajiri

Section 7

CPRE 281

Student ID: 166622517

**Final Project: Simple elevator control system**

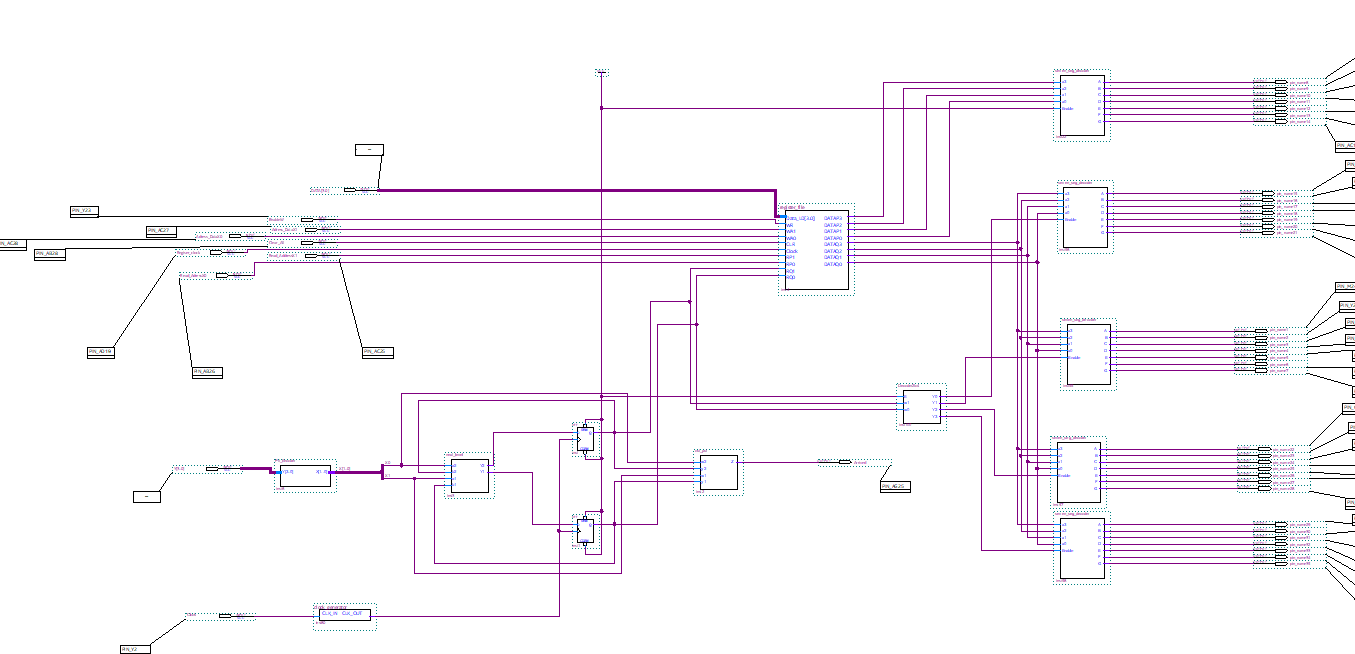
**including level display**

We have learned quite a lot about how digital logic is used to build the systems and devices we use. I used that to implement the control system of a 4-level elevator system. I used a register file to store a 4-bit representation of the different levels. Those values were used to display the level on a 7-segment display. I used a Finite State Machine to transition the user to the level that was inputted, from any given level. The current level of the elevator as it moves is shown on the 7-segment display

**The General Description**

For this 4-level elevator system, I designed a circuit that will take from the user, the binary equivalent of the level he/she would like to go to, and take him/her there. The 7-segment display will display the intermediary levels while taking the user to his/her desired level. I created a register file consisting of 4 4-bit registers in which we will store the 4-bit equivalents of the 4 different levels. I used 1st floor(L1), 2nd floor(L2), 3rd floor(L3), and Basement (B). For this project, L1 (0001), L2 as 2(0010), L3 as (0011), and L4 as B for Basement (1011). It is also important to note that this circuit uses the load functionality of a register file. These levels can be represented by any combination (For example, L1 = A (1010), L2 = B (1011), L3 = C (1100), and L4 = D (1101)).

I created a Finite State Machine that takes, as input, the level the user wants to go to, uses this with the current level he/she is on, and takes this user to his/her desired level. Once the user has arrived at their desired location, an LED light will turn on. Each level is used to get the value in the register that corresponds to this level, then displays it on a particular seven-segment display. The image below shows an overview of the 4-level simple elevator circuit.

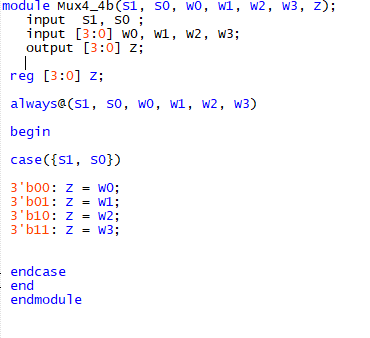


We are going to look individually at how each component of the final design was implemented. We will, first, look at the register file, then the Finite State machine, and finally the display section.

**Register File**

**Multiplexer**

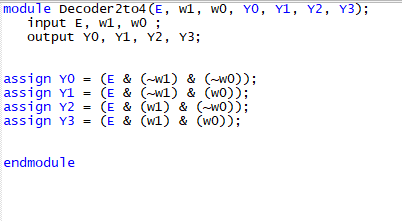
I implement a 4-to-1 multiplexer to manage the outputs of the registers. I wrote the code and created a symbol for my 4-to-1 multiplexer. The image below shows the code I used to get this multiplexer using a multiplexer previous lab.



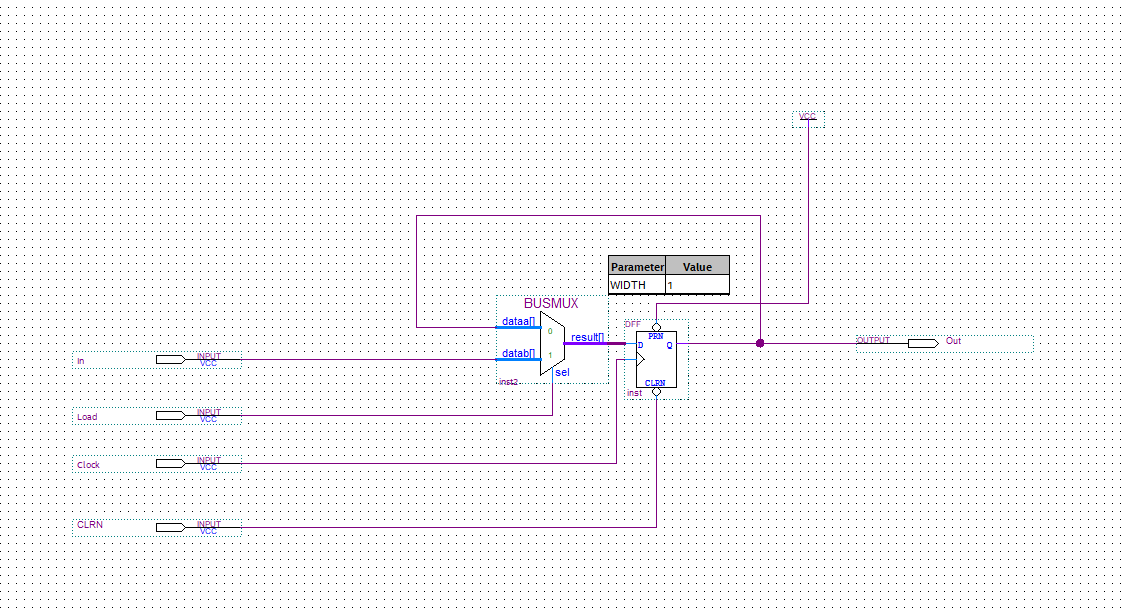
**Decoder**

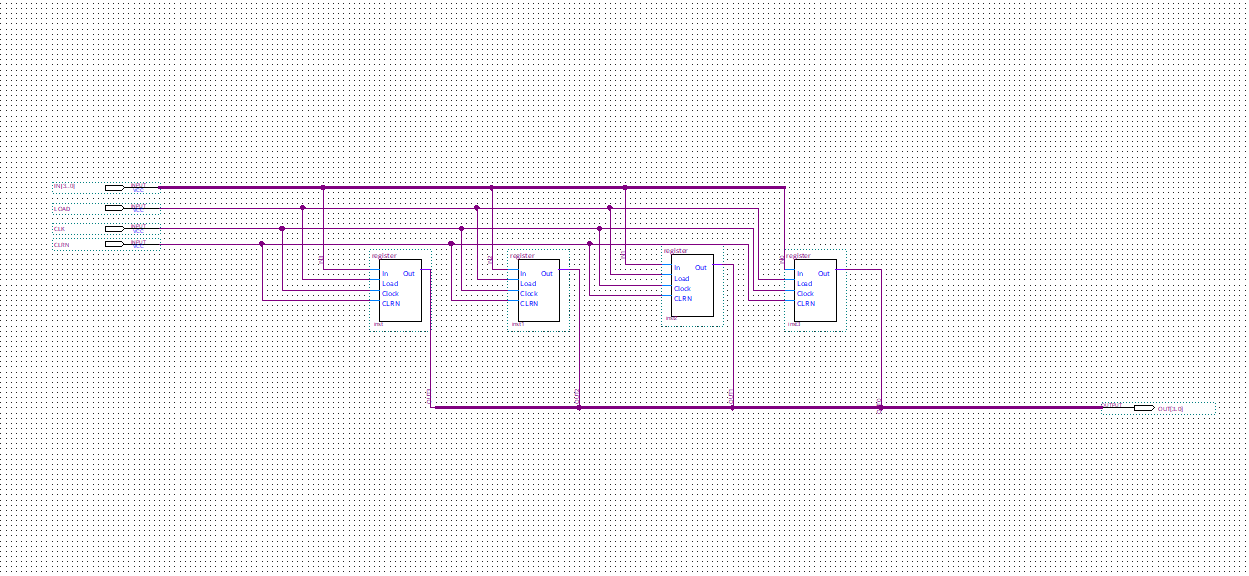
I also implemented a 2 to 4 decoder with enable to indicate where to write in the register file

will create. The image below shows the code used to create this component.

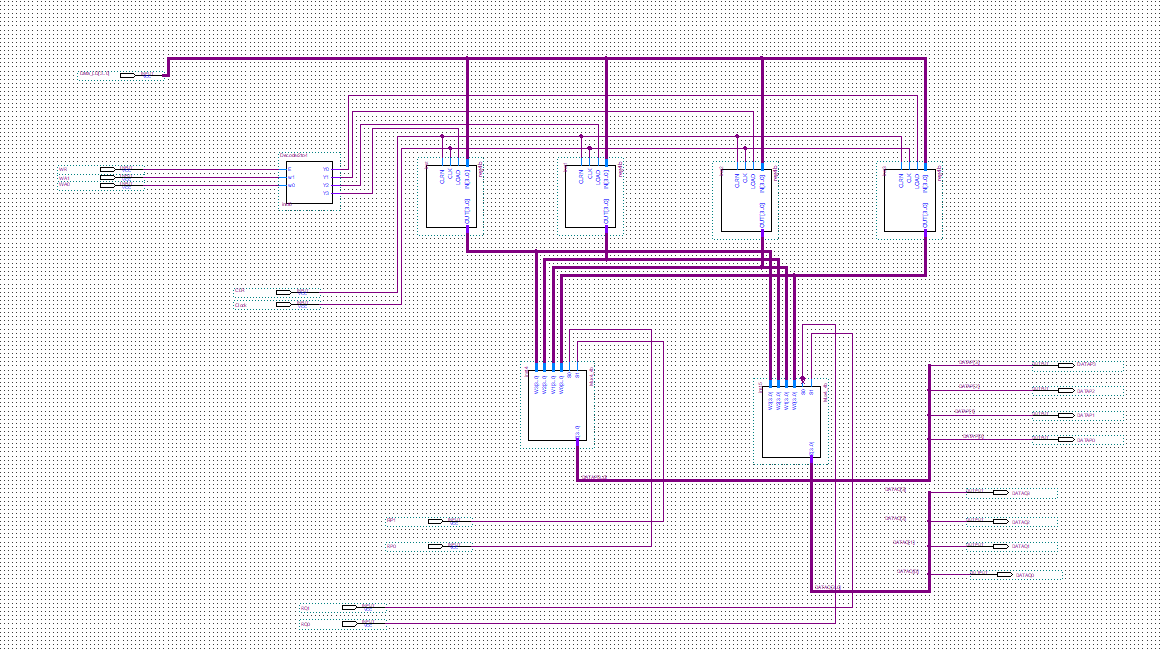


I used a register file of 4 4-bit registers. To start I implemented a one-bit resister with Load and Clear. Then I used 4 of these to create a for bit register with Load and Clear. The images below show this.





I transformed these 4-bit register to symbol, I used 4 of these registers, a 4-to-1 multiplexer, and 2-to-4 decoders to create the register file. I used the 2-to-4 decoder to indicate where to load values. The multiplexer was used to determine which value we needed to read. Below shows the final diagram for the register file.

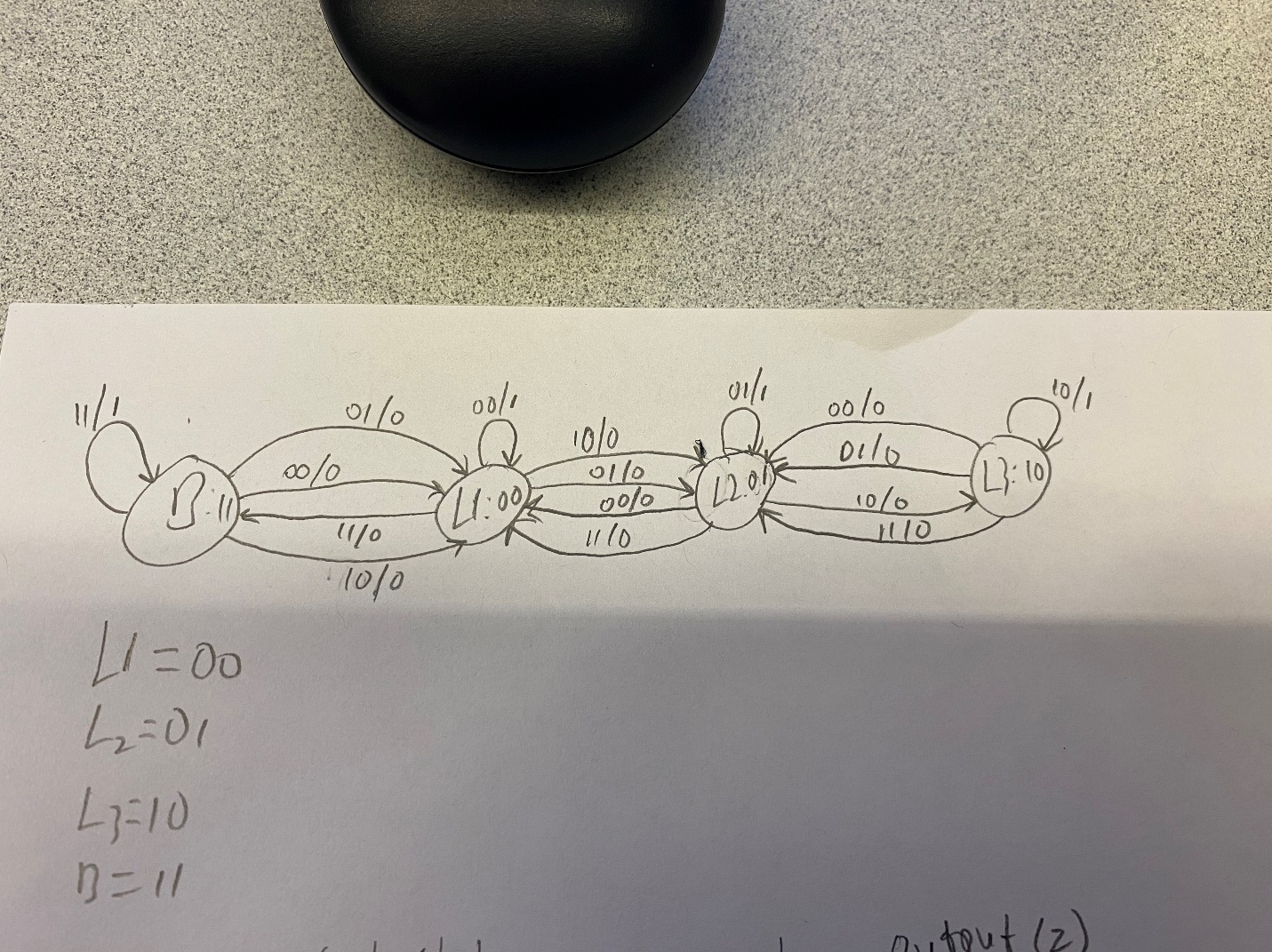


**Finite State Machine**

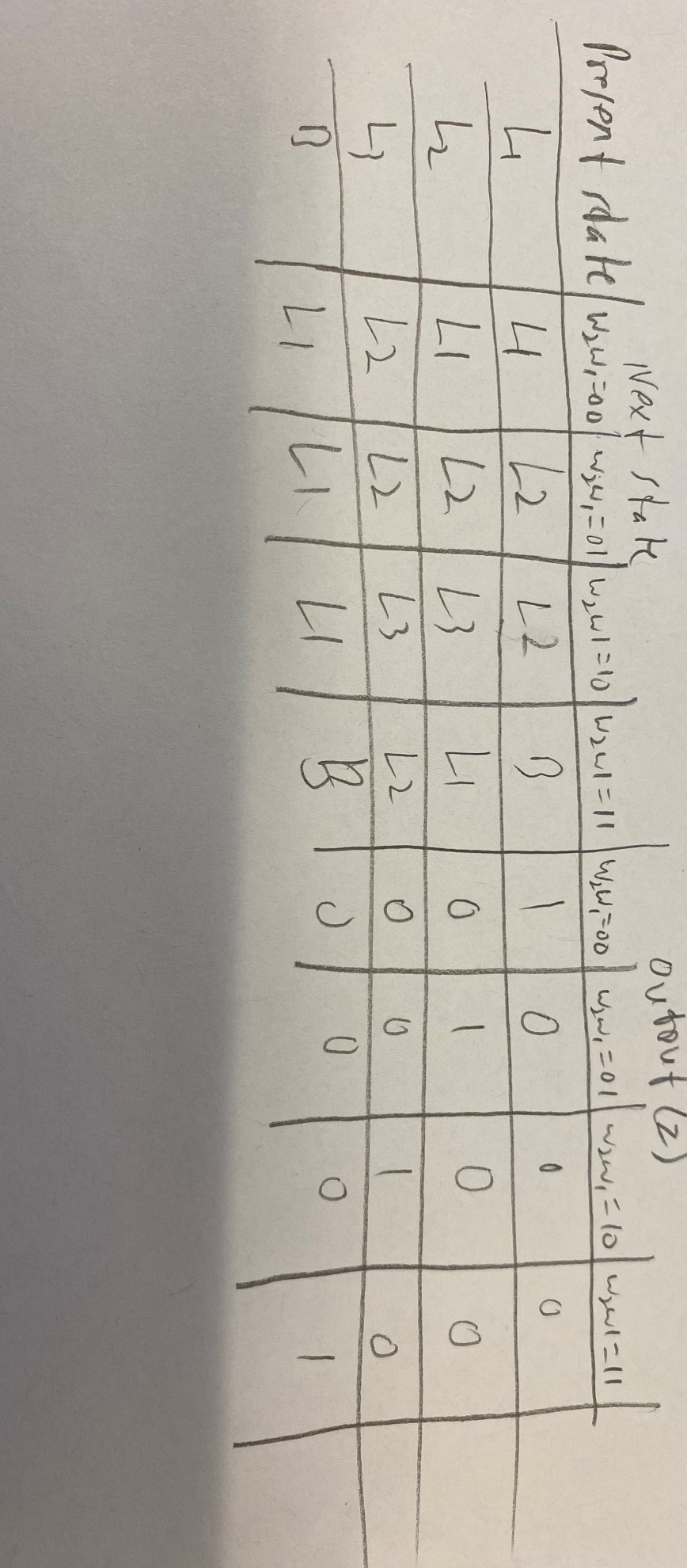
The final state machine is obviously the heart of this project. It helps us navigate from one level to another based on the user input and the level he is on. The elements below will help us understand how this was implemented.

**State Diagram and State Table**

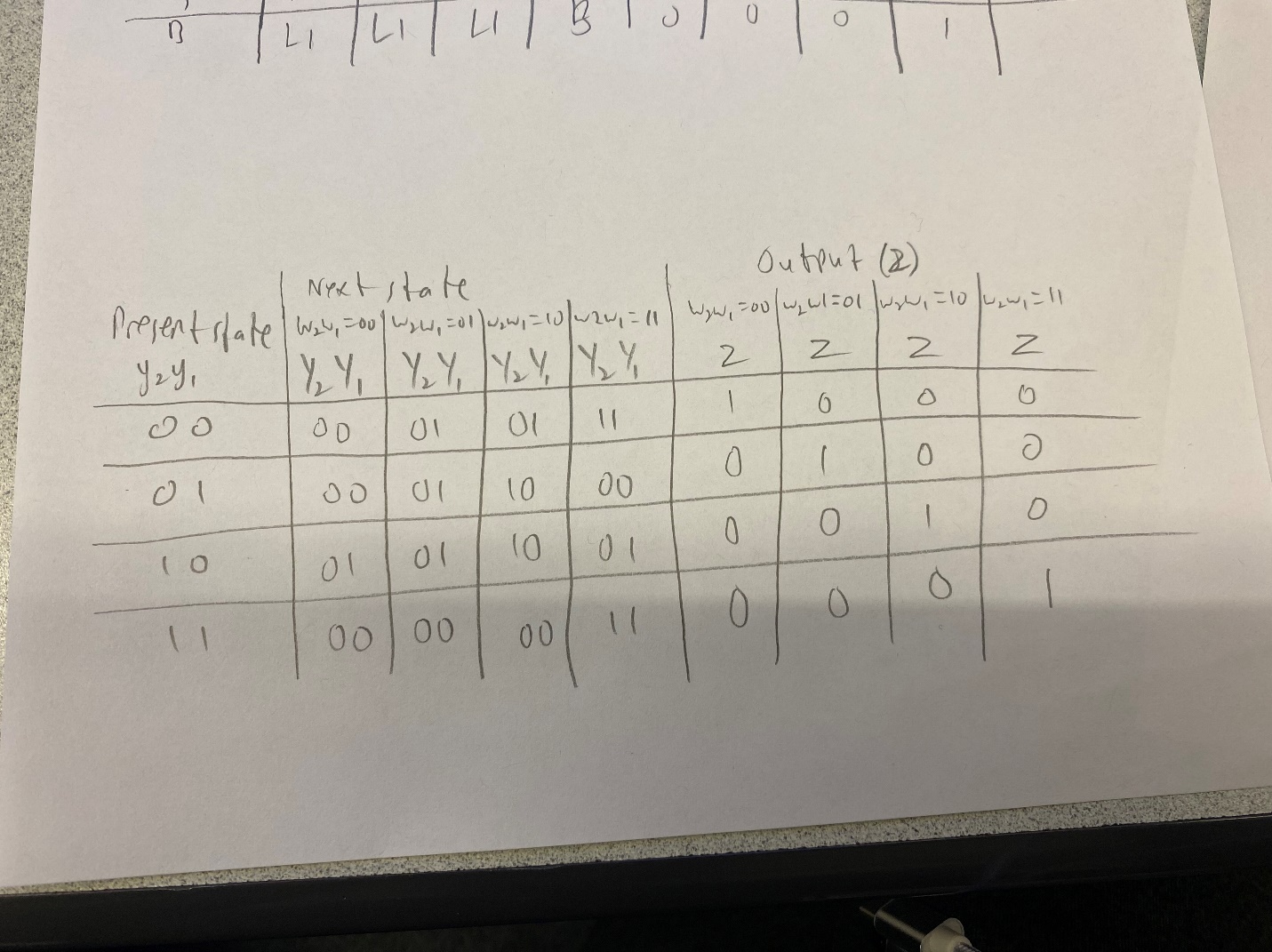
State diagram of the FSM.



Using this state diagram, we get the following state table:

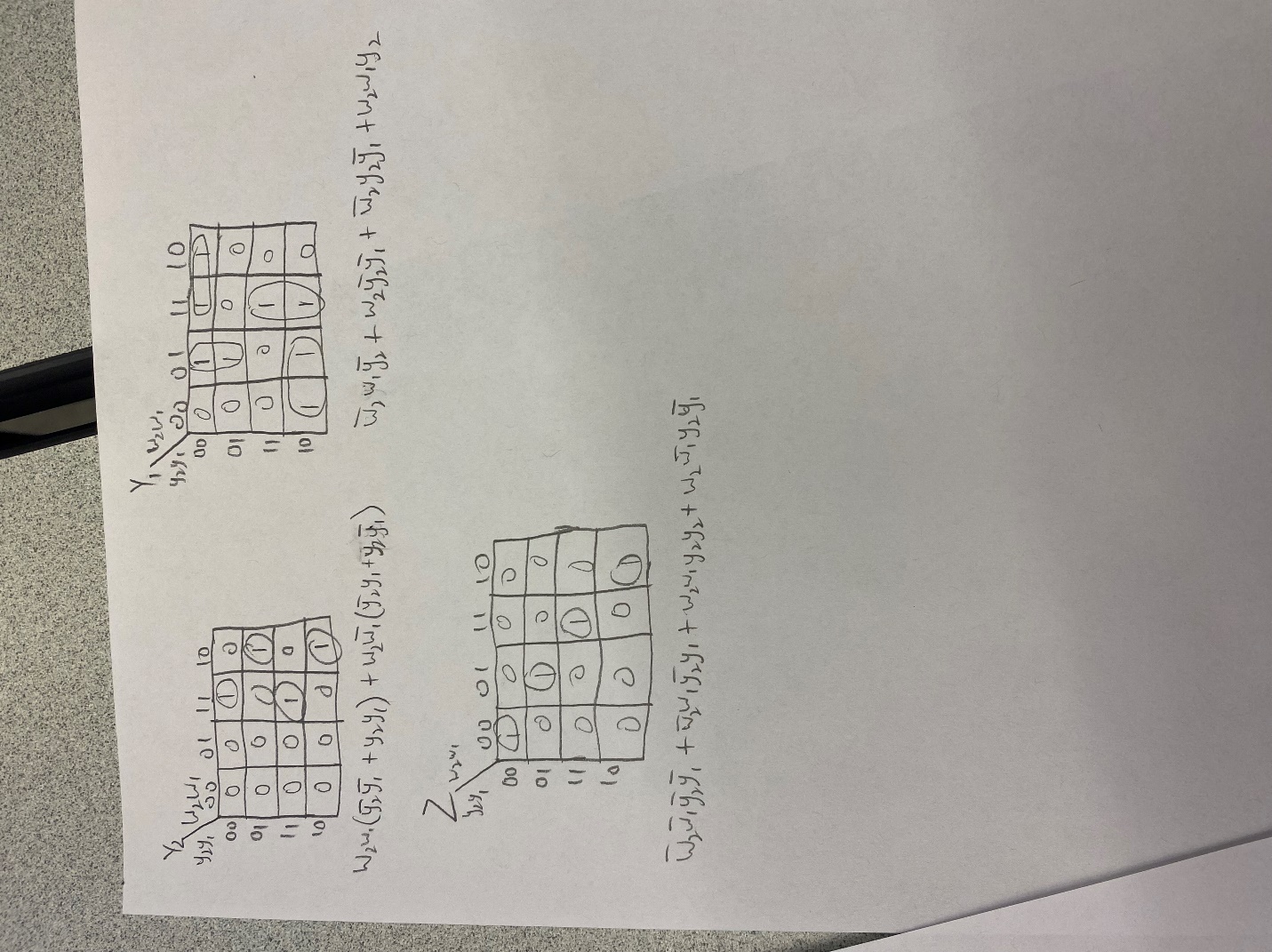


Because we have 4 different states, we represent them using 2 bits. Using the following assignment for each of the 4 states, L1 = 00, L2 = 01, L3 = 10, and B = 11, we get this state assignment table:



From this, we can derive the expressions for Y2, Y1, and Z using K-maps.

**Truth table, K-map, and Logic expressions**



From the K-maps above, the expression for the next state variables Y2, Y1, and of the output Z are given by:

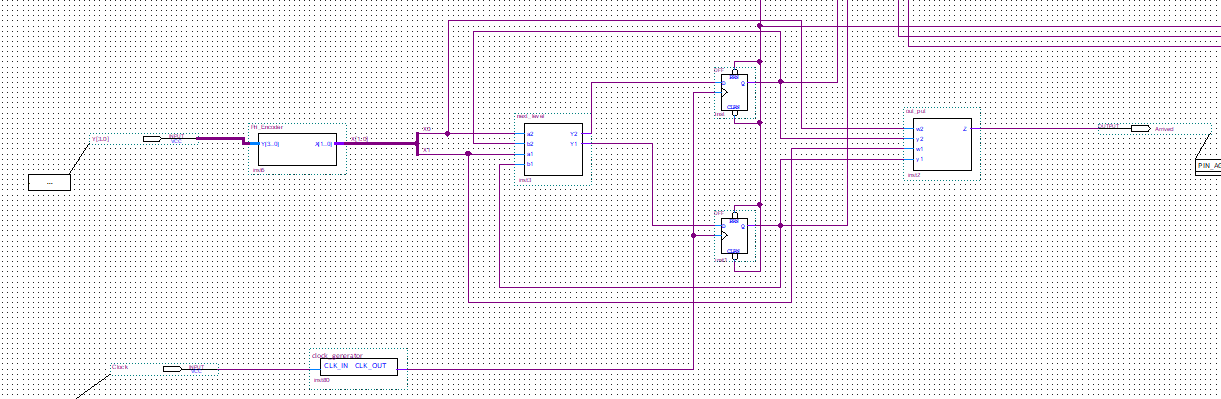
**Y2 =** 𝒘𝟐𝒘𝟏(𝒚̅𝟐𝒚𝟏̅̅̅+𝒚𝟐𝒚𝟏)+𝒘𝟐𝒘̅𝟏(𝒚̅𝟐𝒚𝟏+𝒚𝟏̅̅̅𝒚𝟐)

**Y1 =** 𝒘̅𝟐𝒚𝟐𝒚𝟏̅̅̅+𝒘̅𝟐𝒘𝟏𝒚̅𝟐+𝒘𝟐𝒘𝟏𝒚𝟐+𝒘𝟐𝒚𝟏̅̅̅𝒚𝟐̅̅̅

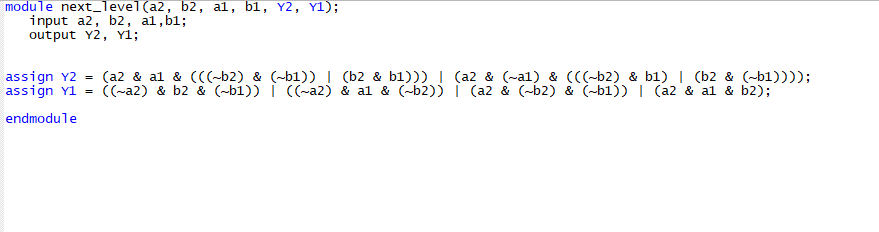
𝒁= 𝒘̅𝟐𝒘̅𝟏𝒚̅𝟐𝒚̅𝟏+𝒘𝟐𝒘̅𝟏𝒚𝟐𝒚̅𝟏+𝒘𝟐𝒘𝟏𝒚𝟐𝒚𝟏+𝒘̅𝟐𝒘𝟏𝒚̅𝟐𝒚𝟏

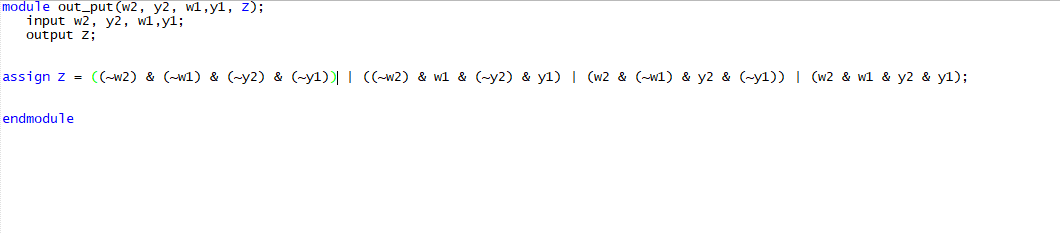
**Implementation: Code and Circuit**

Because we encoded our states with two bits, we will need 2 D flip-flops. The Circuit below is the general circuit of our Finite State Machine.



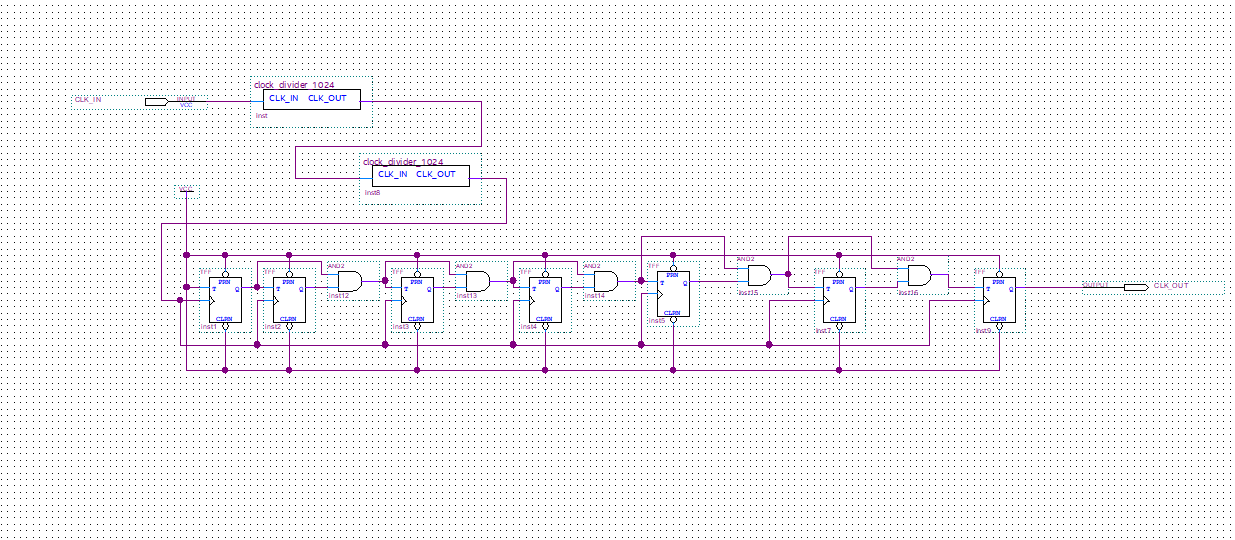
The “next\_level” symbol has the code that gives us the next state from the present state and the input. The symbol “out\_put” has the expression of the output also using w2, w1, y2, and y1 as inputs. Below are the Verilog codes used for each symbol:





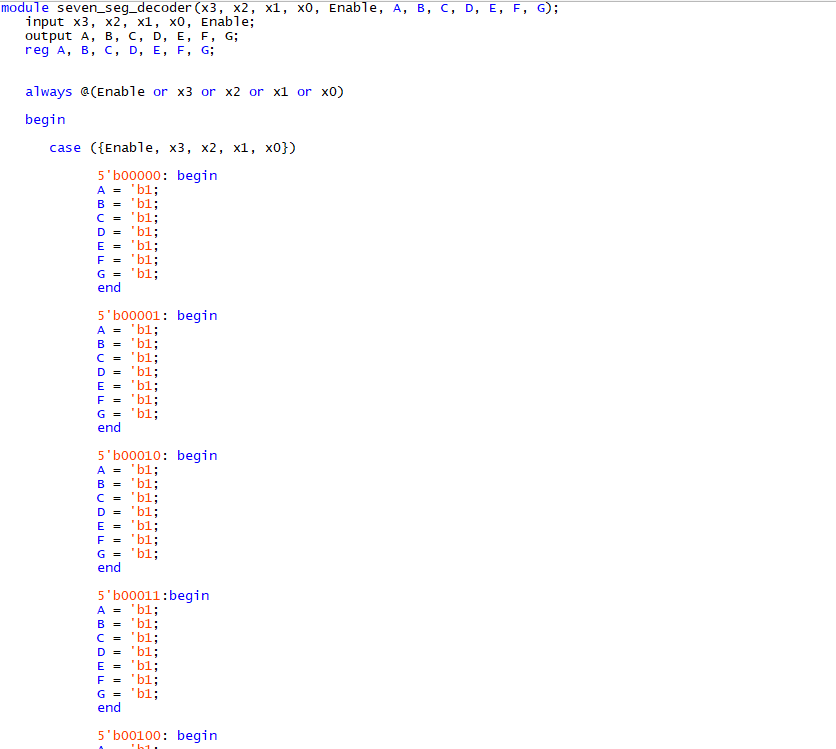
After I generated symbols for these Verilog codes, I used 2 D flip-flops to implement the finite state.

Also, in a bid to make the circuit run smoothly, I used a clock generatorwith the board's clock to make the change of state as slow as that of an actual elevator. I used 27 T-flip-flops to make a clock cycle last for about 2.7s**.** The clock generator is shown in Figure II.2.4.

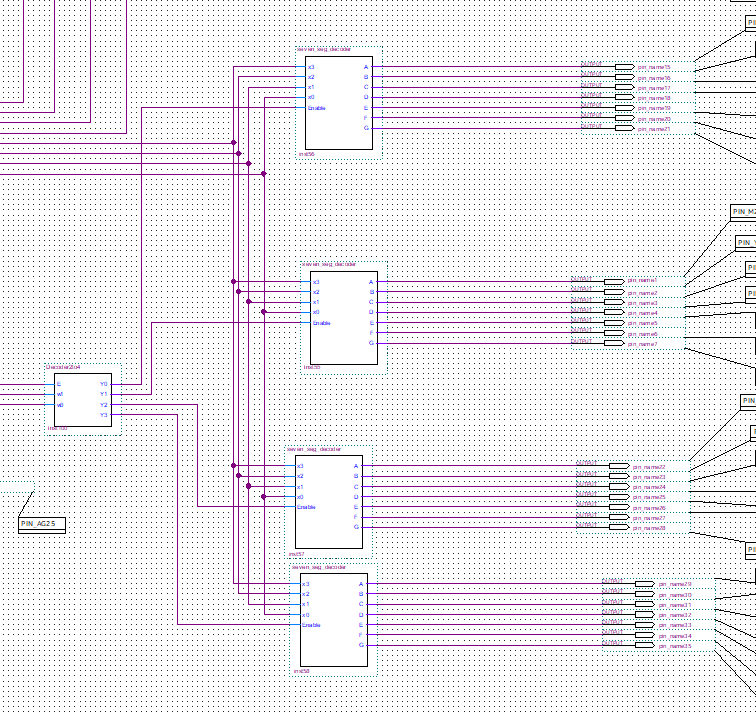


**Display**

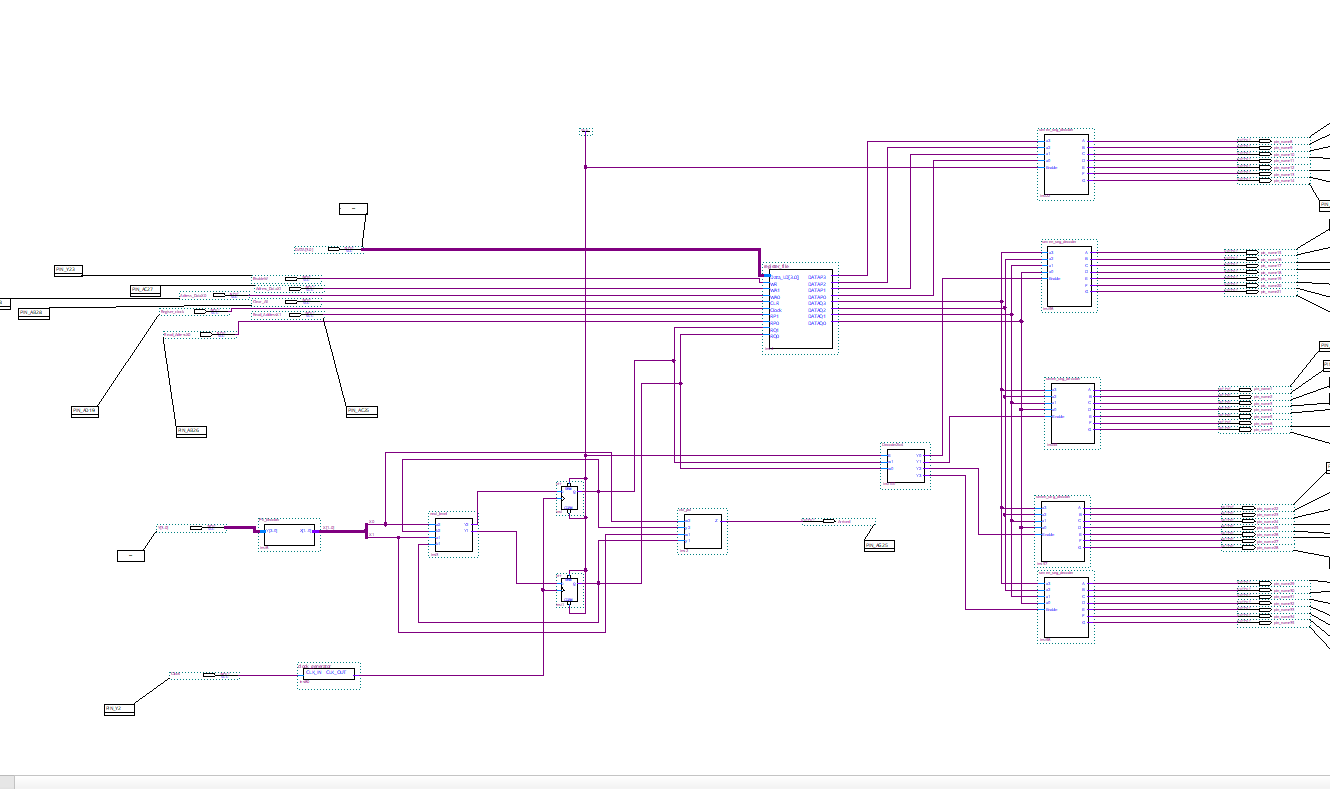
The display consists of 4 seven segments display with enables and a 2-to-4 decoder. The decoder is used to indicate which of the 7-Segment displays should be used to display the particular level, by connecting it to the enable. The code below shows how I added the enable input to the 7-Segment display code we obtained during a previous lab.



By using this 7-Segment display, I implemented the following for the display section:



**Final Circuit, Test, and Observations**



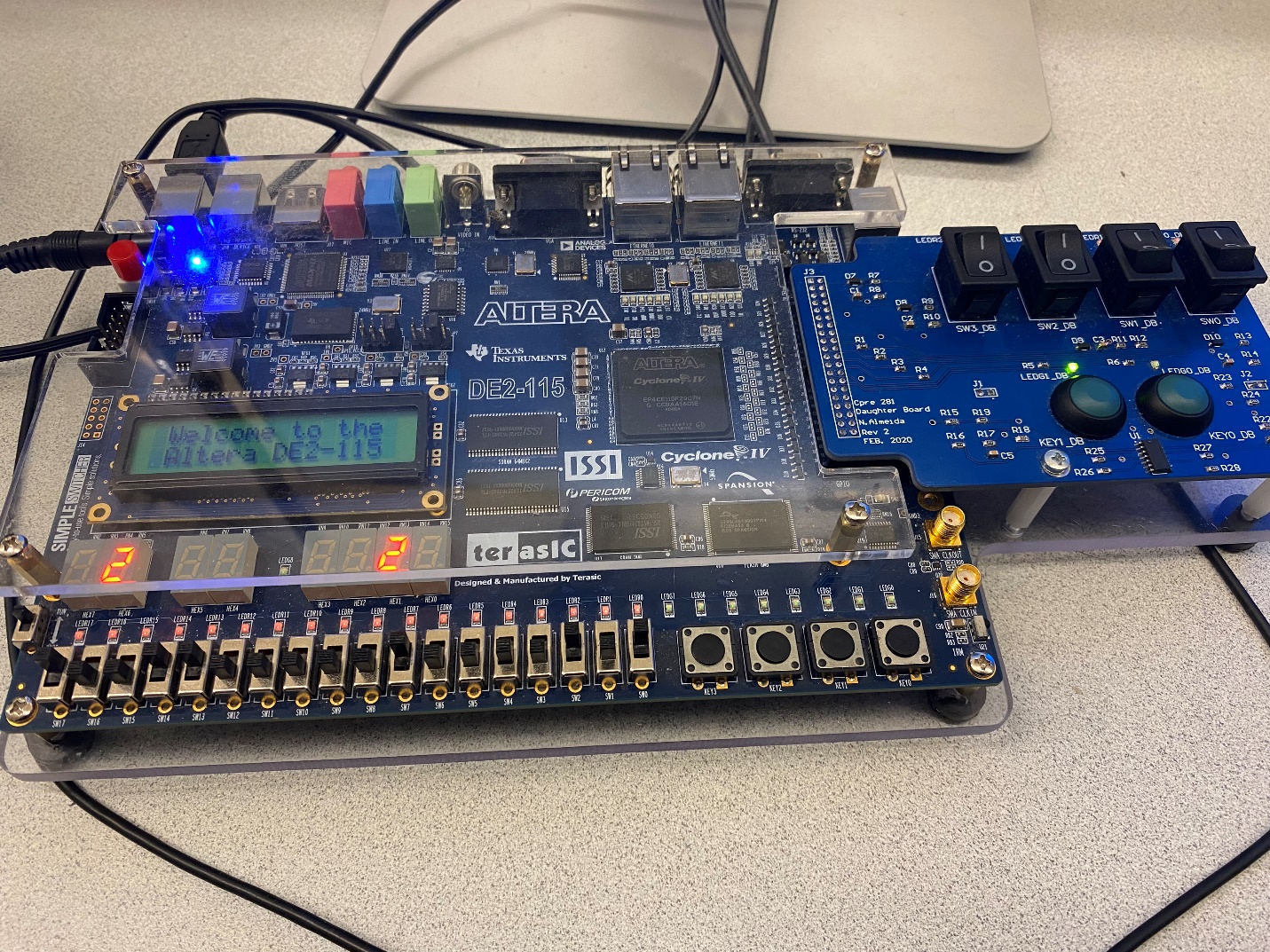
Display

Register

Finite State Machine

To test this, you need to load values into the register file, one at the time ( L1= 1(0001), L2= 2(0010), L3= 3(0011), and B = b(1011)) using WR = 1, and the WA1 and WA0 corresponding to their location ( L1: (WA1= 0, WA0= 0), L2: (WA1= 0, WA0= 1), L3: (WA1= 1, WA0= 0), and B: (WA1= 1, WA0= 1) ). After setting up the level numbers, you can use the elevator by just indicating which level you are going to with two switches on the board using the following code: 00 for L1, 01 for L2, 10 for L3, and 11 for B). Each current level is connected to the RQ of the register file to get the value stored there and display it. Once you get to the level you want to go to, a green LED light on the board will turn on.

The image below is an annotated image of the board during a test that will help you test this circuit.



Load data

Write

RP1

RP0

WA1

WA0

Clear

L2 display

L1 display

Basement display

L3 display

Register clock

Led arrived indicator

Level input w1

Level input w2

From the state of the board in the image above, if we enter Level input w2 = 1,and w1 = 0,after about 2s, L2 Displaywill turn off and L3 Displaywill turn on and display “3”. If we don’t change the Level input values, Display L3 will stay on and 2s later, the ‘led Arrived indicator’will turn green. If you change the Level input values, the same thing will happen until you arrived at the desired location and the led Arrived indicator willturn on.