**Oklahoma State University**

**College of Engineering, Architecture and Technology**



**ENSC3213 – Computer-based Systems**

**Instructor:** Dr. Carl Latino

**Fall 2019**

**Final Project Report**

**Lego Elevator**

**Team #13**

**Team members:**

Youssef Dallaly, Michael Thompson, Brady Mayfield, Cole Mitchell

12/5/2019

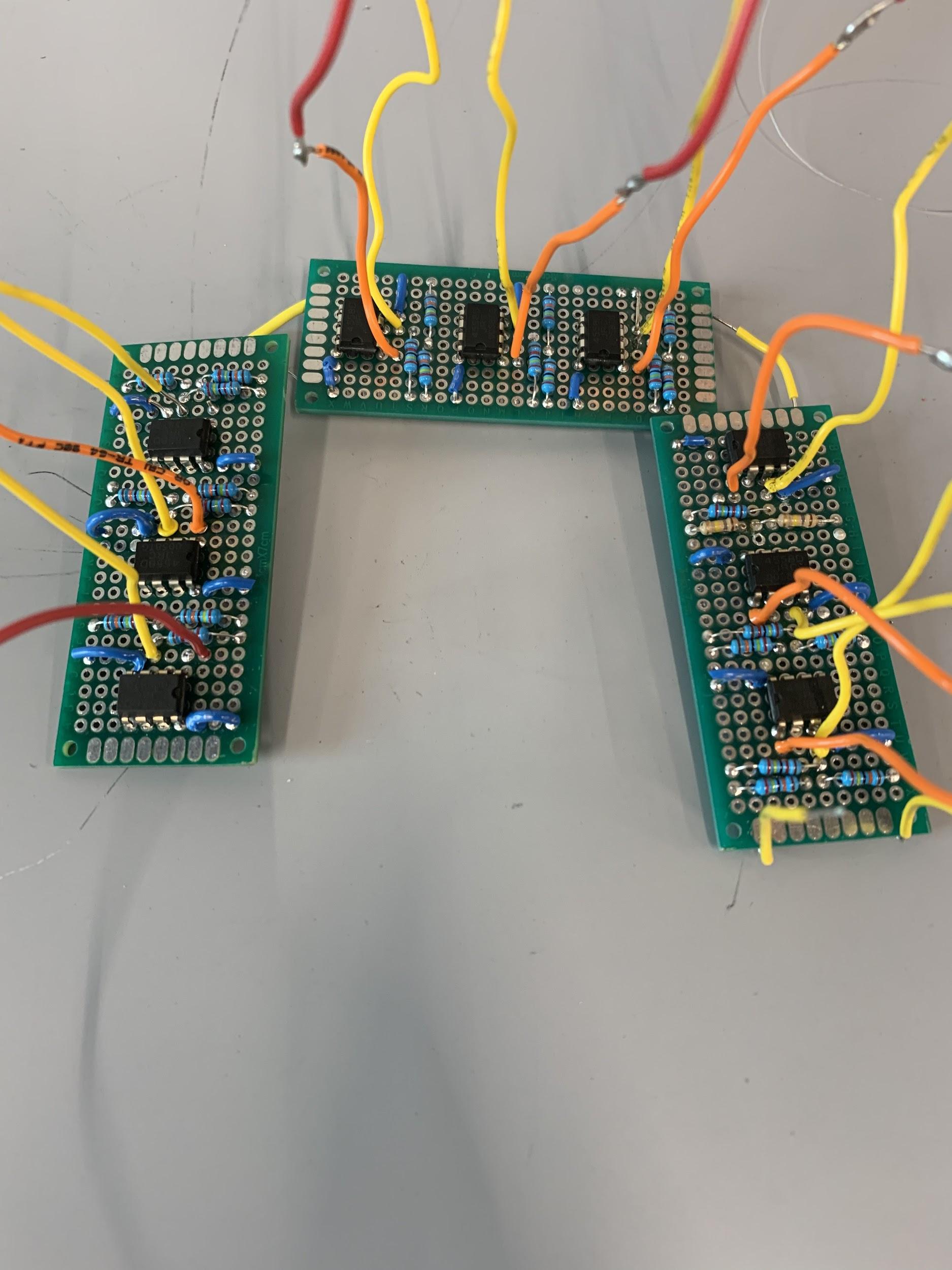
# **Introduction and Motivation**

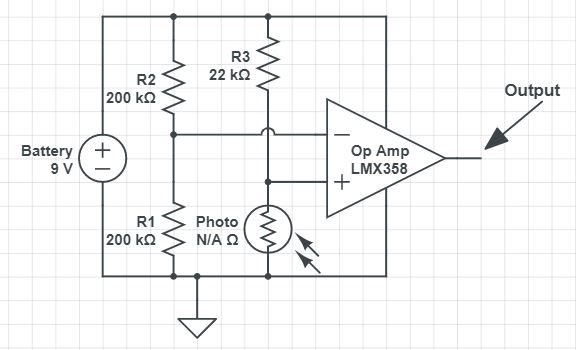
The project goal was to create a small-scale physical, working elevator. The elevator we made was 7 stories tall and constructed out of Legos. We had several objectives with this project, first was to use photoresistors to sense the position of the elevator. Second, we wanted the elevator to use the microprocessor from lab. Three, we wanted our elevator to follow an algorithm to go to the next appropriate floor pending multiple requests. Fourth, we wanted the elevator shaft to be driven and controlled by a motor.

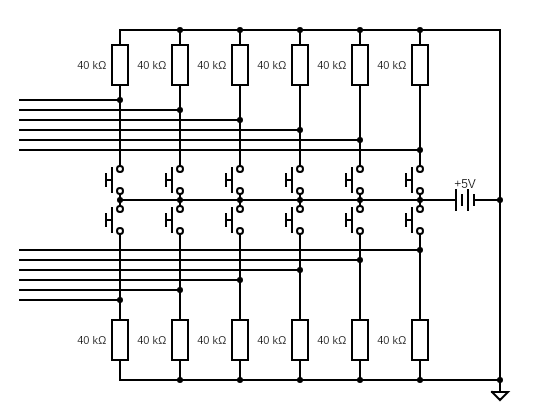
We chose to utilize the microprocessor that was used in Lab, the ARM Cortex M4, with C code to control the elevator. This decision was made because this microprocessor was readily and easily available to us, and we chose to use C code because we believed this would make the coding of the project much simpler.

From these proposed goals, almost all of them were accomplished. In the final implementation, we were not able to use photoresistors to sense the location of the elevator. We attempted to do this, but ran into some issues. This will be discussed more in depth in the hardware section. In lieu of this, we employed a timing base system to keep track of elevator location. We decided that the elevator should take one second to move from one floor to the next, and then made sure this was true using pulse width modulation. Then, we simply had an array that added one every time the motor moved up one floor and subtracted one every time the motor moved down a floor. The next goal was to use the Cortex microprocessor, which we were able to successfully do, and we were also able to develop an algorithm that made the elevator behave as intended. Lastly, we did indeed have the elevator shaft be powered and moved by a motor. Ultimately, the only thing we were not able to implement in some way or another was use of the keypad, which would have acted as the buttons inside of the elevator. We were unable to do this with the way we set up the other methods of input.

# **Hardware Development**

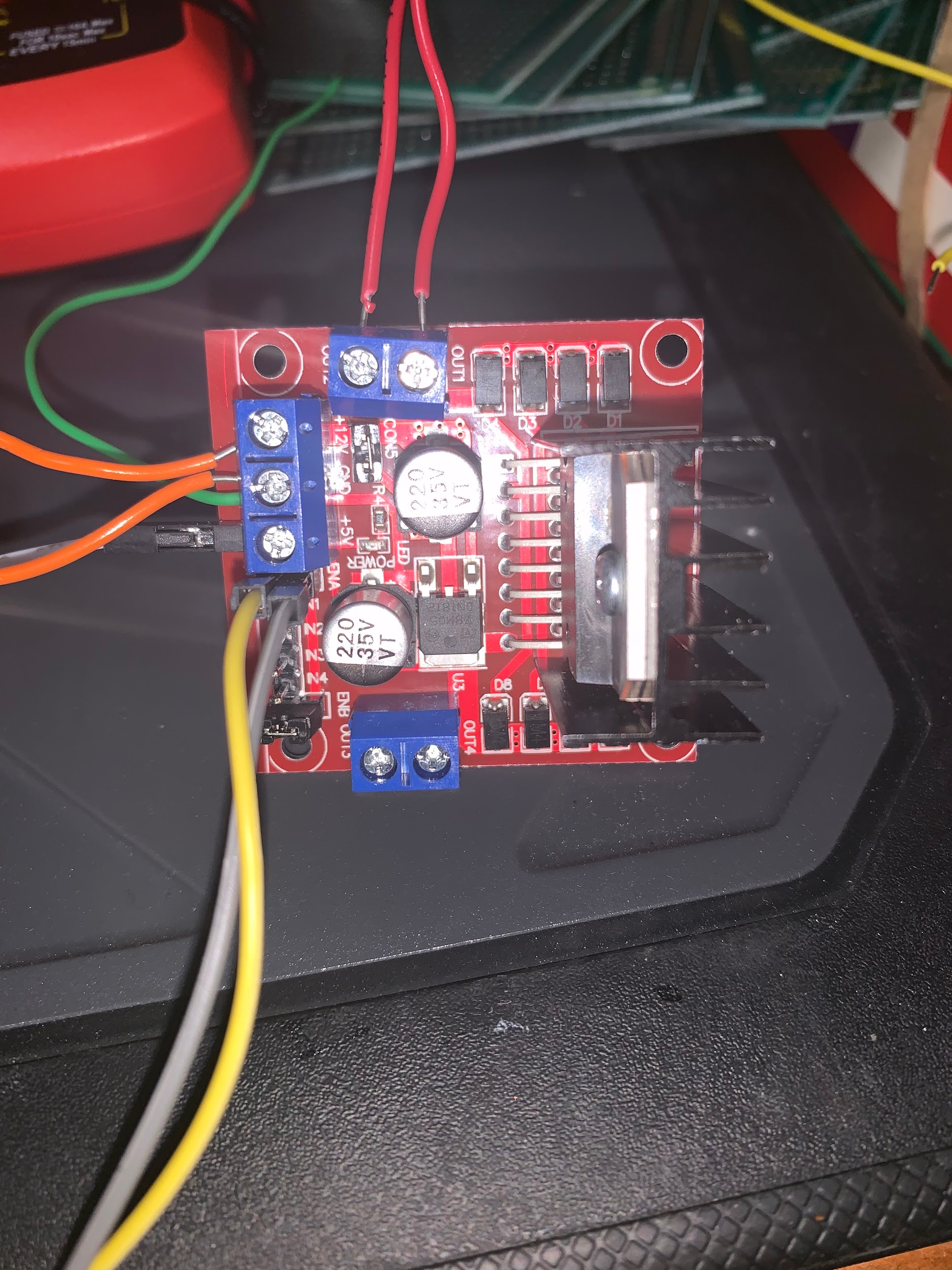
The major hardware designs for this project involved creating sensors for the elevator shaft, button inputs for the floors, a keypad, and a control scheme for the motor. The sensors were needed to detect the position of the elevator and we required a way to change the direction of the motor. We decided to use photoresistors for the sensors and an H-bridge for the motor control. The buttons were set up to directly send an input to the board when pressed. Finally, the keypad was connected as shown in our GPIO diagram. 

The photoresistors took the majority of our time and effort dedicated to hardware. After consideration, we decided to build separate comparator circuits for each photoresistor. In function, these circuits would output a low signal (O volts) when the photoresistor was in the presence of light, and would produce a high signal (5 volts) when the sensor was dark. This happens because when a photoresistor is covered, it’s resistance increases dramatically. Because of this increase, the voltage on the positive terminal of the op-amp to become greater than the negative terminal (which is at a constant 4.5V), causing the op amp to saturate with a high signal. In our case this means the circuit would output a 1 when the elevator car covered a sensor. We used the Cortex 5V power pin for the positive supply on the op amp so that when it saturates the signal produced is 5V. We then built one of the comparator circuits for each of the sensors and connected each power input to a singular 9V battery. What is shown is a picture of the circuit diagram and the completed product. The yellow wire went to the photoresistor and the red was the output connected to the cortex. 

Unfortunately, the sensors and circuits did not work as we intended. During testing in the lab, the circuits worked as we hypothesized. They were tested in groups of three and all sent the correct output when covered. This did not translate to the final project. When implemented inside the elevator shaft, the circuits would not send any output voltages. We never fully pinpointed the direct cause of our problems, but did come to a couple of conclusions on what they might have been. The simple answer could have been that the elevator shaft was too dark, so that the sensors did not notice a change in light when the elevator passed in front of it. Another theory was that the total resistance of all the photoresistors once they were connected was too high. Given the time pressure, we elected to solve this problem with software rather than trying to redesign a bunch of hardware. 

The buttons on each floor of the elevator had a relatively simple design. We wired a common power to each of the buttons and attached a connection to the input terminal of the board and ground. Whenever a button was pressed, the two wires were connected and a voltage was sent to the board to be received as an input. To the right is the schematic for the button circuit.

To control the motor, we ended up using an H-Bridge. The H-Bridge had three pins that received output information from the board. Two of these pins controlled the two poles of the DC motor, and the other was an enable pin to which we sent a PWM signal. The H-bridge was powered by a 9V battery. There was also a power out from the bridge, which we used to power our board in the final product of the elevator.



The keypad was hooked up exactly the same as it was in Lab 6. The four keypad outputs were connected to PE10 through PE13, and the four keypad inputs were PA1, PA2, PA3, and PA5. Also, the keypad inputs were each connected to a 3V power source through 2.2kΩ resistors. This was unable to be fully implemented. The keypad would successfully send inputs to the Cortex, but did not work with the rest of the code. We could not get the program to listen for keypad inputs and other inputs at the same time.

To house the components we created a holding box out of legos in addition to the shaft itself. To hold the buttons and cover the front of the elevator, we elected to CAD a front panel. Given that none of us were super familiar with 3D printing, our friend Weston Taylor was interested in working on a project and he asked if could help with CADing. He modeled the front panel and the holder for the motor. We then used the endeavor printers to print out the CADed items.

For GPIOs, we removed the LCD screen in order to have extra GPIO ports. Ultimately, due to hardware changes, we probably didn’t need to do this. A detailed GPIO pinout, showing how everything was connected, can be seen in the Appendix.

# **Software Development**

The diagram to the left explains how our algorithm works. The intent was that it would give priority to the current direction, and if there were either no floor requests or it was at the top, the direction and priority would switch to their respective opposites and then complete the same task. This offers a situation that is experienced in most elevators, where if someone on the 2nd floor wanted to go up, someone on the 4th floor wanted to go down, and someone on the 5th floor wanted to go up, it would fill the 2nd and 5th floor requests, then fulfill the final one on the 4th floor.

We had originally coded the keypad in Assembly during Lab 6, but for this project we instead used the C code for the keypad that was provided by our TA Francisco Ferdinand.

The rest of the code was fairly straightforward, however it was lengthy because we decided to store requests in various arrays in order to run through them faster and in a more logical fashion. Due to our hardware failures we needed to change our algorithm to, as opposed to handling current floor by comparing the location with sensors, we had to keep track of the current floor in side of the code which did not change our code greatly, but did require us to handle variables such as the global counter differently. With this in mind we decided to use the interrupts of the system to that the whole operation would be calibrated to work within itself, thus making it easier to move the motor in time with itself. Our final code, along with the subfunctions, is described in a flow chart at the end of this report, our workhorse functions being our readButton(), setDirection(), and gotoPing() functions. Each of these is fairly self explanatory, with readButton reading the floor that a request was coming from, setDirection setting the direction in order to keep track of the current floor within the code, and gotoPing actually moving our motor.

Another aspect that we implemented was a PWM (Pulse width modulation) program in order to control the speed of our motor, as its base speed was too high to be accurate. This sub function is found in our code, but only gets used when calibrating the elevator and when gotoPing is used.

The largest problem we ran into with the algorithm was actually not being able to properly use while loops, simply because the elevator needed to be able to read all the requests at the same time as opposed to being able to wait on inputs. This caused us to change our keypad scan so that it would work constantly, scanning at all times and only entering a loop when the button was pressed and exiting when unpressed to avoid odd inputs. These problems ended up being solved by using our timing code as opposed to our sensors, as now we simply check if the proper time had passed. This did create a problem of when requests were handled, because when the motor is on the code is stuck in a timing loop so that the elevator would only move 1 floor, as we solved this problem by having the elevator wait for some time to receive requests to go to other floors, then to begin fulfilling requests.

**Functions Library:**

**readButtons();**

This function works by reading the masks in the GPIO map specifically for each button, then saving that information in their respective arrays to be checked later in our direction and gotoPing functions.

**setDirection();**

This function was needed to store and keep track of the direction of our elevator due to hardware failure. It works by checking the previous floor and comparing it to the current floor to set the direction. It will then reset the direction if there are no more requests, along with resetting the direction on the 7th and 1st floor since their directions can only be down and up respectively.

**motorOnUp, motorOnDown, motorOff;**

These functions control the movement of the elevator. They use our PWM to control the speed of the motor, setting and controlling the enable pins on our H-bridge. By controlling the enable we control speed, and controlling our input pins controls the direction, which motorOnUp and motorOnDown are reversed for each other, and motorOff turns the inputs off completely, thus stopping the motor.

**gotoPing();**

This function checks where requests are coming from, compares them to the current direction as well as if the current floor is a “desired floor” (which comes in from the keypad), and then begins the path, fulfilling all requests that hold the same direction the elevator is currently going, as well as ones they have not passed. Then, once the final request is fulfilled, the same process is repeated, but using the opposite values for direction. The most complex request this can fulfill, and that which is possible, is to be fulfilling a request in one direction, received a request in the opposite direction ahead of it, and then receive a request in the same direction, but behind it. This algorithm will fulfill the current direction, then the opposite direction, then the original direction again. This is the most complex problem available, as everything past this situation is only a copy of itself algorithmically. All data and locations, as well as directions, are controlled with arrays, and manipulated using FOR loops to iterate through everything.

**Initialization;**

The initialization for the GPIOs are all found in the GPIO map. The Systick initialization and the handler are both the same as previous labs, using a 1ms interrupt value, and changing our delay(); function to be able to pass through an integer to control the delay easier.

# **Conclusion**

Our final product was a physical 7 story Lego elevator. When you pressed the button at a respective floor, it would move to that floor. The system was controlled by the Arm Cortex microprocessor, and a 9V DC motor moved the elevator up and down. Each time the elevator reached the appropriate floor, which was selected by the algorithm discussed in the software development section, the elevator waited for user input. The only thing lacking in our final product was the keypad, which would have been used at this point to simulate people walking on to the elevator and choosing what floor they wanted to go to. The algorithm was able to handle this, just not in conjunction with the rest of the project.

We learned a lot with this project. Probably one of the biggest lessons we learned was to test early and often. In other words, test each stage of the project as you are developing. We made the mistake of assuming everything would come together neatly at the final stage, which certainly did not happen. We should have been testing each individual component along the way, not waiting to test until everything was assembled. Moving forward into larger and more complicated projects, this will be a valuable lesson.

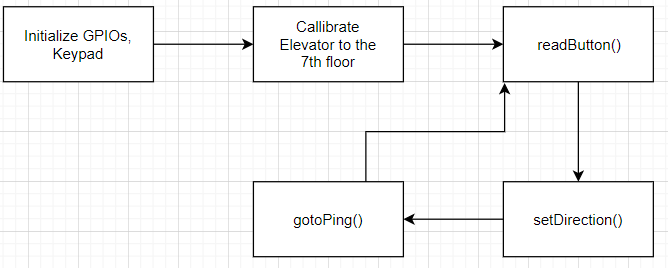
In terms of technical things, we also learned a lot. Hardware wise, we learned how to build a comparator circuit (though it was not implemented in the final design) and how to use a H-Bridge (which was critical in the final design). It was also good experience with how to implement a Finite State Machine in code. We learned about Pulse Width Modulation in an earlier lab, but it was good to practice more with it and see another real world example of PWM. This was also many of our first experiences with combining hardware and software together into an embedded system. This was briefly mentioned before, but we learned a lot about how to interface the software and hardware through GPIO pins.

If we were to do this project again, the one key difference would be with the sensors. If we had more time, we would have chosen Hall Effect Sensors (also known as magnetic sensors). As we understand it, we could have placed a magnet in the elevator, and then when the elevator reached a floor the sensor for that floor would send a signal to the board. This would have saved us from the troubles we faced with the photoresistors, particularly the variability in different environments, and would have eliminated the inconsistencies we faced with the timing method. Difficulties would likely arise with this method as well, but we think that any issues that would come up would have been easier to deal with than the other two methods we used. Additionally, a more optimized elevator shaft and containment, such as adding rails to the elevator car, would have resulted in a much smoother movement. While not crucial for the goals of this project, it would add more polish to the final design.

As far as suggestions for the class are concerned, we had a few. One, it would have been helpful to have clearer expectations. For example, having a grading rubric and detailed project description available from day one would have been very helpful. Another thing that would have been helpful in the class is only using one microprocessor. While it is true that the two different processors were fairly similar, it would’ve been helpful to really get to learn one microprocessor before trying to learn a whole different processor.

# **Appendices**

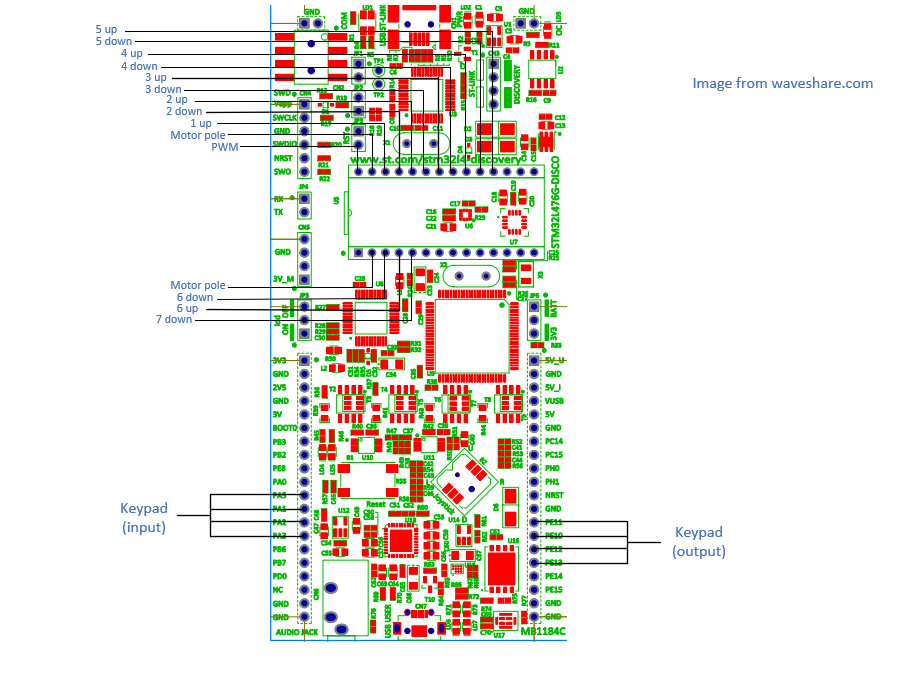
Code Diagram using sub function names:



Final Design:

|  |  |
| --- | --- |
|  |  |

GPIO Map:



Masks for the buttons, used to check in the appropriate Input Data Registers:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1U | 2D | 2U | 3D | 3U | 4D | 4U | 5D | 5U | 6D | 6U | 7D |
| GPIOB | GPIOB | GPIOB | GPIOD | GPIOD | GPIOD | GPIOD | GPIOC | GPIOC | GPIOB | GPIOB | GPIOB |
| 0x1 | 0x1000 | 0x4000 | 0x100 | 0x400 | 0x1000 | 0x4000 | 0x40 | 0x100 | 0x2 | 0x2000 | 0x8000 |

Masks for the motor (all GPIOC):

|  |  |
| --- | --- |
| Up | 0x10 |
| Down | 0x20 |
| Clear / Off | 0x30 |