ECE 0202 Term Project
Yahya Farag, Aragya Goyal, Krish Patel, Mitch McKain
Dr. Mohamed Bayoumy

ECE 0202 TERM PROJECT - Elevator Design Report

ywf1@pitt.edu arg195@pitt.edu krp151@pitt.edu mtm120@pitt.edu

Hardware Design

Our group's first decision was to split into two subteams, with one of the subteams becoming the Hardware team. This team housed members Yahya and Mitch. Yahya has experience with PCB design in Altium from working with SOAR so when the proposition of designing our elevator on a PCB instead of a breadboard arrived, everybody was in favor of going this route. The first step in design was breaking the elevator circuitry down into smaller subtasks. These subtasks consisted of the 7-segment display for displaying what floor the elevator is on, the keypad for users selecting which floor they wish to take the elevator to, the stepper motors for opening our elevator and moving our small model of the elevator, pushbuttons for an emergency stop and the admin call, LED's & Floor Switches for calling the elevator to a specific floor and lighting up the respective LED, and finally the STM. All of the schematics for our project were designed in Altium with the schematic designer.

For calling the elevator and indicating the current floor, we used 4 LEDs and 4 momentary push buttons. The pushbuttons are debounced with a 100kΩ resistor and 0.1uF capacitor low pass filter, the LEDs have a 220Ω current limiting resistor from the IO.

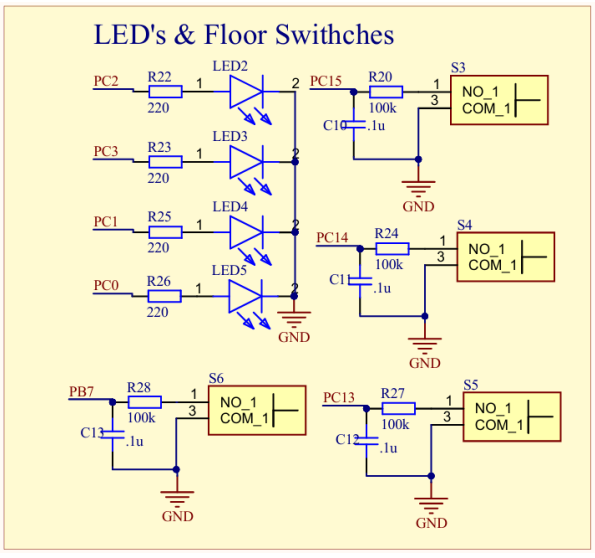

Figure 1: Floor push buttons and LEDs

The 7-segment hex display, which is provided below in Figure 1, has a block for the display driver labeled U1, which is how the hex display will interact with the STM, and a block for the hex display itself DS1, which is how we wire the hex display to the PCB.

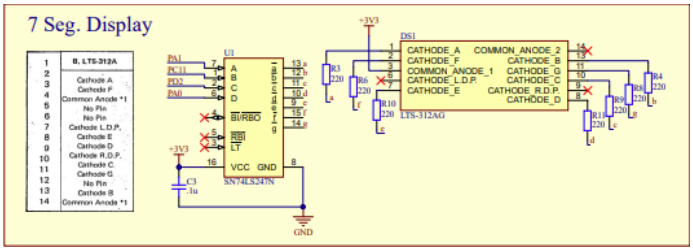

Figure 2: 7-Segment Hex Display.

Another important part of this schematic is the purpose of the 0.1µF capacitor which serves as a high pass filter. The filter will prevent any random voltage spikes from the STM from causing abnormalities on the hex display.

The keypad was easy to design since we were able to reference Lab 4 to see what we needed to do for wiring.

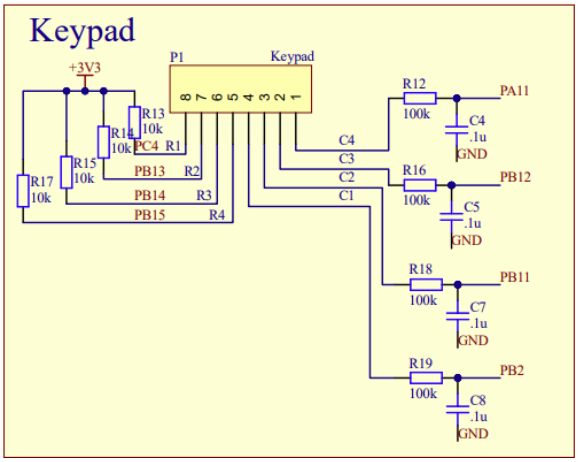

Figure 3: Keypad

We used a 100kΩ resistor and 0.1µF capacitor for the lowpass debounce on the columns for when the user selects what floor to take the elevator to.

The stepper motors are used for opening and closing the elevator door and moving our model up and down to show the motion of the elevator. Instead of using the given stepper motor boards, we isolated the ULN2003A Darlington Transistor Array IC and found the TI datasheet. Then we used the datasheet to wire the correct configuration to drive the stepper motors.

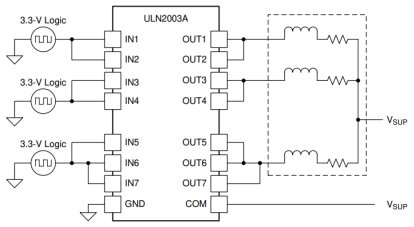

Figure 8-1. ULN2003A Device as Inductive Load Driver

Figure 4: TI Reference Schematic

Our final schematic included decoupling capacitors on the +5V and COM, along with a large bulk capacitor on the 5V line from the STM32 Nucleo board

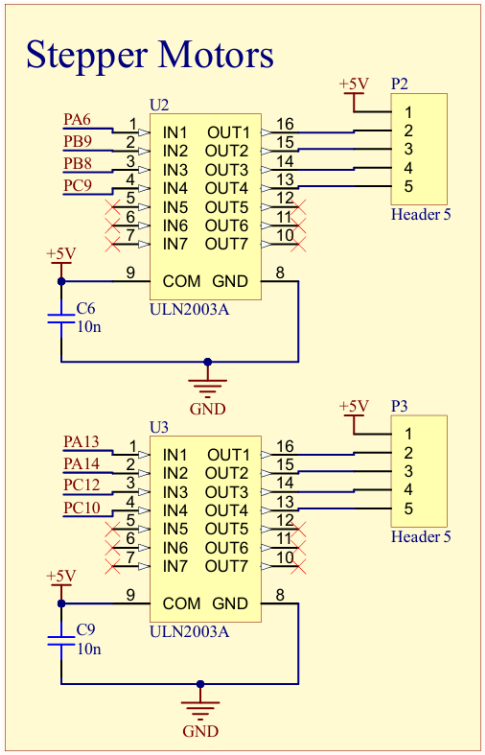

Figure 5: ULN2003A Stepper Motor Control

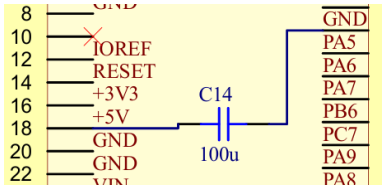

Figure 6: Bulk Capacitor for Motors

We also included some larger tactile push buttons to use as an E-Stop or Override Button, they were pulled up to +3V3 and also had the same debounce circuit on the keypad and floor push switches.

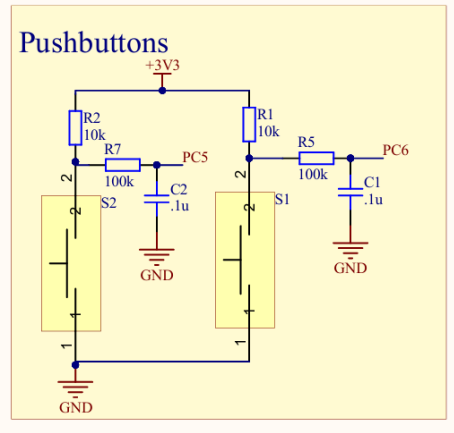

Figure 7: Tactile Pushbutons

For the PCB design layout, we opted for a large 2-layer - 1.6mm PCB. The layout was done in Altium after the schematic was imported, and after arranging the components in their respective positions, we went back to the schematic and assigned the pins that would make the layout as easy as possible.

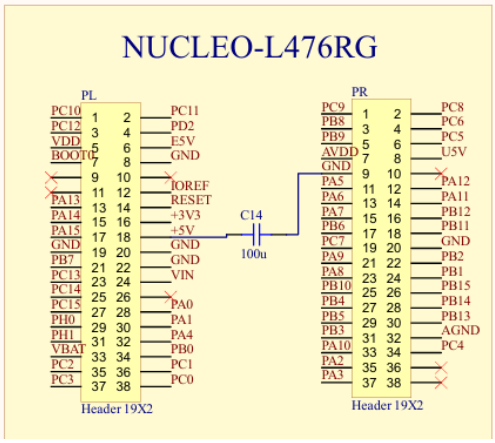

Figure 8: STM32 Nucleo Pinout

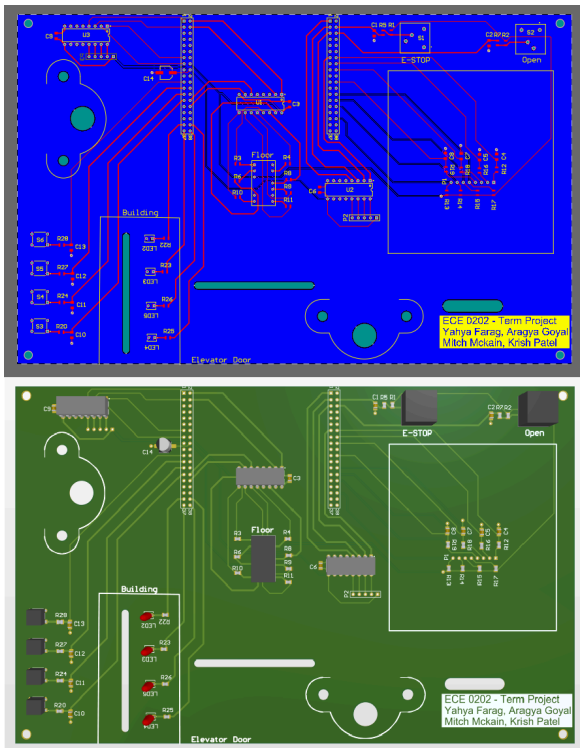

Figure 9: PCB Layout and 3D View

Power for the PCB was routed with 0.5mm traces, while signal / LED traces were routed with 0.25mm traces. The majority of traces were routed on the top layer to avoid using vias. The bottom layer was used as a ground plane, so ground pour was applied to the entire bottom layer. For all ground connections from the top layer, vias with a short trace were used to provide a low-impedance route to ground.

Once the PCB was ordered and it arrived, all the components were soldered. Notably, particular header pins for the STM32 to plug into the PCB. Components such as the LEDs and push buttons were also tested on a breadboard before being soldered to the PCB.

Mechanical Design

Instead of the stepper motors freely spinning, we used the PCB itself as a mechanical base with slots to implement a mechanism to show the elevator door opening and closing, along with the actual movement of the elevator. A 3D printed rack and pinion system was used, with 3D printed rails to stop the pinions from falling out. We imported the PCB as a .STEP component using Altium's PCB exporter. This allowed us to design the mechanical system while the PCB was being manufactured. The Stepper motor models were from grabCad.com. Using the PCB and stepper motor model allowed us to design 3D-printed parts around the PCB slots. The part consisted of a spur gear, a rack of equal module, and sliding holders for the rack to slide along.

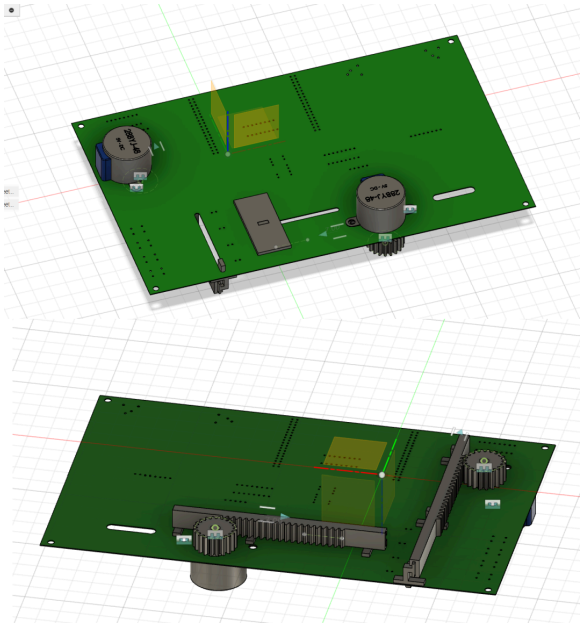

Figure 10: Fusion 360 Model

After designing the complementary components, they were 3D printed in gray PLA plastic. The parts were then physically attached to the physical PCB. We also installed standoffs to permit clearance on the bottom side mechanisms. For the keypad, the cable was routed to the bottom and then glued along with a piece of cardboard to provide rigidity to the design.

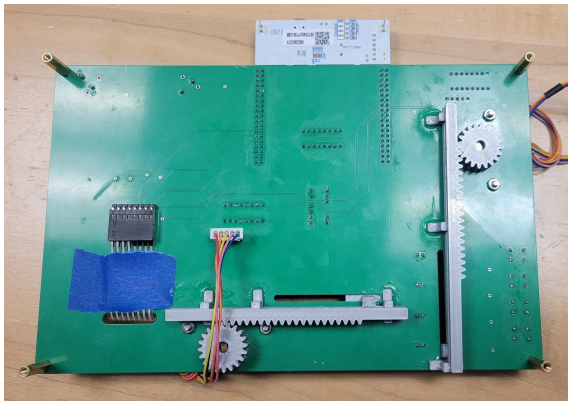

Figure 11: Final Assembly

Software Design

The software team housed Aragya and Krish. The very first objective was to map out the code using a flowchart. This yielded the flowchart pictured below.

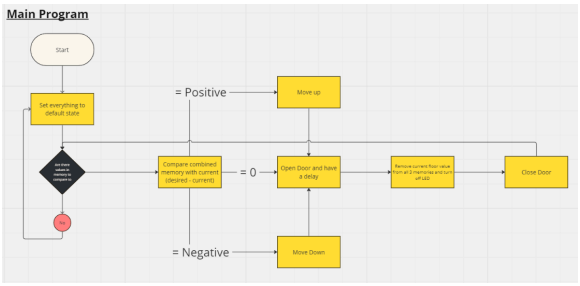

Figure 12: Flowchart of Main

This is the side of the project where project constraints became a big hindrance. The first constraint that was an immediate issue was project specification #7, "The microcontroller should dynamically change the order of which the elevator stops at different levels according to the calls made from the levels." The initial plan was to use interrupts to allow the elevator to stop whenever en route to a destination if the stop is on the way. The flowchart was created below to help keep track of how our interrupts would be handled.

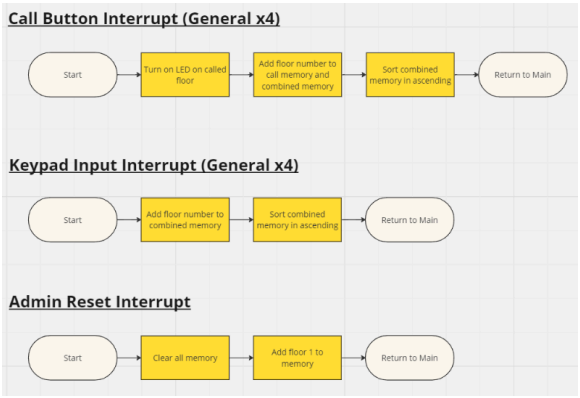

Figure 13: Flowchart of Interrupts

However, this became an issue since interrupts for peripherals group multiple registers together once passed r4, and with the way that we set up our pins, we would not be able to have separate peripheral interrupts for our buttons that call the elevator. This meant that we had to abandon our plan of using peripheral interrupts, and had to switch to using SysTick and polling all of our user inputs during this time. Once inside SysTick_Handler, if a new input is polled, it will be added to the queue, instead of disrupting the current task of the STM.

The next project constraint that posed a challenge was #5, "The user should be able to call for the elevator from at least four different levels by pressing a push button." We decided to do the four floor minimum requirement, to save some space on the PCB and save time on coding. The four floors posed an issue since we had to create logic for all of the scenarios of a call from one floor while the elevator was moving from one floor to another.

The final project constraint that was an issue was one of the motors being defective. We used a breadboard to test our motors before testing the code on our PCB, and everything worked fine. Once we decided to test it on the PCB, the motor spun in the opposite direction that it was supposed to. After further inspection, we found out that the wires were reversed in the motor, so everything was backward from the factory manufacturer. This resulted in us losing time since we had to double-check all of our code for motor operations and replace the motor on the PCB.

A main issue that appeared which wasn't a project constraint, was our decision to use a PCB as opposed to just using a breadboard. The issue with using a PCB is that our pins were set in stone. This wasn't an issue until we had to move away from external interrupts and switch to SysTick. We noticed that we would have to switch some of the pins since two of the pins that we were using for floor call buttons are tied to the crystal oscillator of the microcontroller. Our solution to this problem was running jumper wires underneath the breadboard to the new pins that we assigned for the floor call buttons.

Testing Stages

Issues during the testing stage involved the elevator working dynamically. Early on in the integration process, we had no problem with getting the elevator to go to a call, open the door, and go to the destination that the user requested. The problems started when trying to reroute the elevator if a call was made while the elevator would have moved past a floor. Another issue stemming from this was the LEDs changing properly when a new call was made. We got to a point where we could either answer the calls from the four floors, or we could answer calls from the keypad. The solution to this issue was rewriting how our keypad scanning operation worked

so that the door would remain open until the user selected a floor on the keypad, or when a certain time limit would be reached. If this time limit is reached, then the elevator remains in place, and it can be called from another floor. We never reached a point where the elevator worked dynamically to calls with how we set up our queue, and we had to live with missing out on this important functionality.

During the testing stage, we realized how difficult it was going to be to implement TeraTerm due to us using all of the general purpose registers throughout all of our code. TeraTerm was something that we set aside to be the very last thing that we would implement since we ideally would have the elevator completely functional and then could implement serial communication to TeraTerm within one of our functions. The idea was to have some of TeraTerm in our MotorUp and MotorDown functions so that it would display a message such as “Going Up” or “Going Down” depending on the direction in which the elevator is moving. We could have then implemented more of TeraTerm in our floors function so that once the elevator arrived at a floor, a message such as “Floor 3” would display if the elevator was stopped on the third floor. The issue with us waiting to do this last was that we did not have any other registers available in our motor functions or in our floors function. We also had issues with trying to implement the UART2_Init and USART2_Write functions that were used in lab 1 for our printing. Eventually, a decision was made where we felt that the small model of the elevator was an adequate representation of the current status of the elevator. However, we understand that the point of using TeraTerm was to show that we could effectively use serial communication within our project. This was another missing functionality of our elevator that we would have to live with.

The final result of our project is an elevator that can answer calls from any of four floors so long that only one request is made. The elevator then opens a door and provides a three-second window for the user to enter the floor they wish to go to on a keypad, or else the elevator will not move. The door then closes. While the elevator is moving, a small model of the elevator is illustrated on the PCB where users can see it move up or down to its destination, and a hex display shows what floor the elevator is currently on in real-time. LEDs light up according to where the call is made from outside the elevator. There is an administrative stop button that returns the elevator to the first floor, lights up all of the LEDs, and remains in this state until the STM is reset.