Electronic Drawer Organization System

ECE 445: Final Paper

Group # 11

Nathan Marchosky, Michael Stoens, Michael Grawe

Professor: Viktor Gruev

TA: Stasiu Chyczewski

Abstract

This project for ECE 445 Senior Design was an Electronic Drawer System. This aimed to solve organization issues and allow users to keep better track of where they stored their items in drawers. Additional features included the ability to lock one of the three drawers, motorized opening capability and light emitting diode (LED) indication to locate which drawer your items are stored in. Over the course of the semester we designed this system and the end result was a product capable of being operated using only an android phone and plugging the system into a standard 120V wall outlet.

Contents

1 Introduction

1.1 Problem Statement

One of the most important factors in productivity is the level of organization a person has in their workspace. The ease with which someone can find the tools or paperwork they need to do a job has a meaningful impact on the time the actual job takes. Currently, this is an area of the workplace that needs improvement, as 28% of office workers say they would save an hour every day if their workspace was more organized [1].

Perhaps the most common method of office organization involves drawers or cabinets of some kind. The issue with this method occurs when people forget which drawer they put a certain item in. Then, when they need to find it again, they waste time searching through all their drawers. This issue scales exponentially with a large amount of drawers, or a large amount of items in drawers.

1.2 Solution

The electronic solution our group came up with eliminates the need for a person to remember what drawer each item is stored in. This is done by storing the location of the item on a micro-controller (MCU). The user is able to use an Android smartphone application as a way to interface with the micro-controller. On the application, the user is able to select an item that they stored previously in a searchable list, and the micro-controller will then search for that item and notify the user as to which drawer the item is located in. This notification can come in the form of physically opening the correct drawer or lighting an LED mounted on the side of the corresponding drawer.

Some more specifics for the functionality of the drawer system might help make the system a little more clear. The drawer system has three drawers, each of which has a linear motor, a limit switch, and an LED mounted on the side. The drawer will be opened by controlling a linear motor that will push the drawer open. The locking mechanism for the drawer will be a solenoid that will drive a metal pin into a slot in the drawer to lock, and retract to unlock. The LED will be mounted on the side of the drawer system for maximum visibility.

1.3 Visual Aids

Figure 1 shows the visual aid. Figure 2 shows the block diagram. Lastly Figure 3 shows the final assembly of the drawer system.

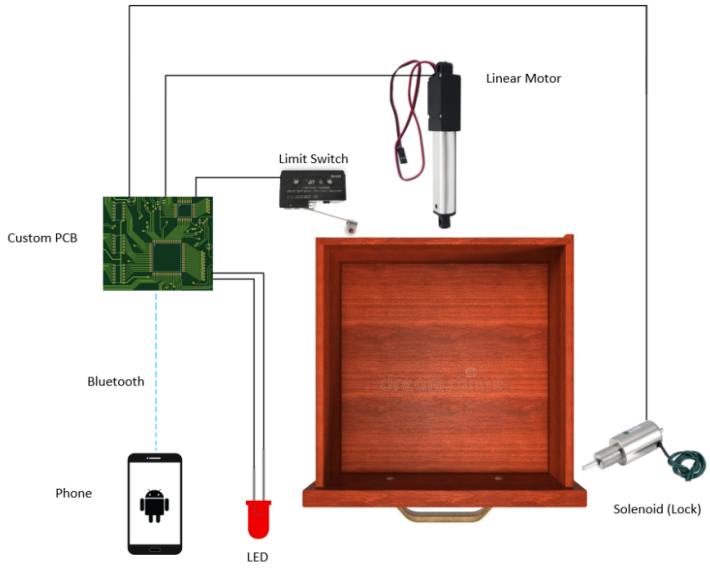


Figure 1: Visual Aid

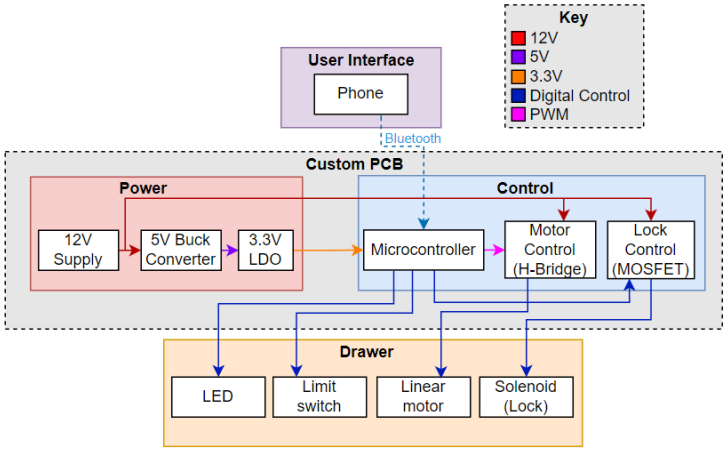


Figure 2: High Level Block Diagram

Figure 3: Drawer Assembly

1.4 Subsystems

1.4.1 Power Subsystem

The Electronic Drawer Organization System drawers will be powered from a wall-outlet using a 120V to 12V AC/DC converter. The 12V supply is then converted to 5V and 3.3V using linear regulators to power the USB and MCU respectively. We sized our 12V power supply to ensure that we would be able to drive two drawers at once. Since our motors require a maximum current of 3A [2], our solenoid requires a maximum of 1.1A [3], and our MCU requires a maximum of 500mA [4], we decided that a 10A power supply would be well suited for our needs.

1.4.2 User Interface Subsystem

The User Interface is a smartphone application that will be connected via Bluetooth to the Control subsystem. This application is on an Android phone. The application gives the user the ability to interact with the MCU to store, remove, and find items stored in the item-drawer pair data structure. It also give the user the ability to determine their indication preferences on the physical drawers, as well as set their desired password for the locking drawer, and view the status of whether or not each of the drawers are open, locked, or the LEDs are on. All the communication between the app and the MCU takes place over Bluetooth.

1.4.3 Control Subsystem

As shown by the high level block diagram in Figure 2 the Control module contains an MCU, an H Bridge motor controller, and a Power MOSFET for lock control. The MCU used for this system is the ESP32. The ESP32 was chosen mainly due to its on-chip Bluetooth capability [4] . The MCU is able to process any input and output signal (IO) necessary to control all auxiliaries in other parts of the control system. The ESP32 has 520KB of static random access memory (SRAM) which is enough to store our program data [4]. The H-bridge, a BD62130AEFJ, which can tolerate input logic signals less than 5.5V, is used for control of the linear actuators. [5]. The DMN2300UFB4-7B is used for our N-type MOSFET (NMOS) that can control our solenoid as it can carry 1.3A needed to run through the solenoid [6].

1.4.4 Drawer Mechanics Subsystem

The drawer mechanical subsystem is responsible for interfacing between the control system and the drawer hardware to open and lock the drawers, as well as to detect if the drawer is closed. The drawer hardware subsection is comprised of three linear actuators (Ok03 model), one solenoid (ROB-11015), three limit switches (MZ-7611), and three LEDs (WP7113ID5V). The linear actuators are controlled by an H-Bridge which enables forward and reverse operation. The solenoids are controlled by an NMOS connected to the MCU. The limit switches provide inputs to the MCU to signal the open or closed status of the drawer. The LEDs are each controlled by the MCU through an NMOS to identify drawers if the user does not want to open the drawer at that time.

2 Design

2.1 PCB Design

2.1.1 Schematic

When completing the schematic, the circuit board was divided into subsections to improve the organization and readability. These subsections include: Power, USB, MCU, and I/O. Shown in Figures 4, 5, 6 and 7. respectively.

The Power subsection is comprised of a barrel jack for connecting the 12V power supply and two linear regulators with filtering capacitors for converting the supply voltage to 5V and 3.3V to power our on board components. Furthermore the requirements and verification tables for the power system is shown in Table 1.

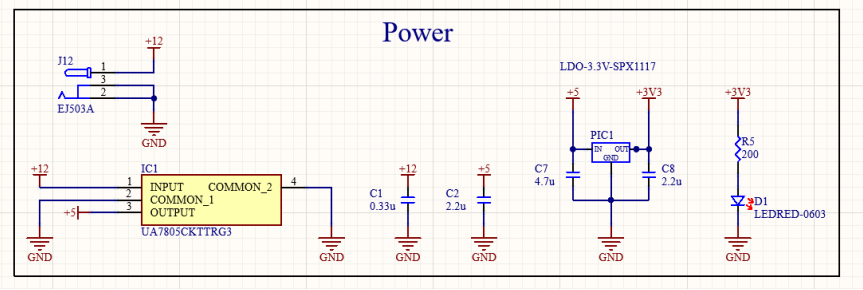


Figure 4: Power Schematic

The USB subsection is responsible for providing the interface between a computer and the ESP32 for programming. This section includes the USB port, two BJTs which control the boot sequence of the ESP32, and the CP2012 USB to UART integrated circuit (IC).

The MCU section holds ESP32 as well as the needed passive components to ensure reliable functionality. The most complex component of designing the MCU schematic was choosing our pin assignments. This involve utilizing the datasheet to identify which pins could be utilized for UART communication and which pins could be used for GPIO [4].

The input/output section holds all of the components to interface with the motors, solenoid, switches and LEDs. The motors are controlled through an H-Bridge IC which enables the motors to move in both directions. The LEDs and solenoid are controlled by N-type MOSFETs. Debouncing circuitry is used to filter the switch signal from the limit switches.

Table 1: Power Subsystem Requirements and Verification Table

Requirement	Verification Procedure
• Supply 12V+/-10% and 10A+/-10% as output from power supply	• Plug in power supply to wall outlet. Using a digital multi-meter in the lab, measure the output voltage and current, making sure it is in the desired range. **Success!**
• 5V LDO supplies 5V+/-10%, 3A+/-10% at output, once it has been connected to 12V power supply.	• Using digital multi-meter in the lab, measure the output voltage of the buck converter after connecting it to the power supply. **Success!** • Using digital multi-meter in the lab, measure the output current of the buck converter. after connecting it to the power supply. **Success!**
• 3V LDO supplies Current of at least 500mA, Voltage of 3V+/-10% to the ESP32 micro-controller.	• Using digital multi-meter in the lab, measure the output voltage of the LDO. **Success!** • Using digital multi-meter in the lab, measure the output current of the LDO. **Success!**

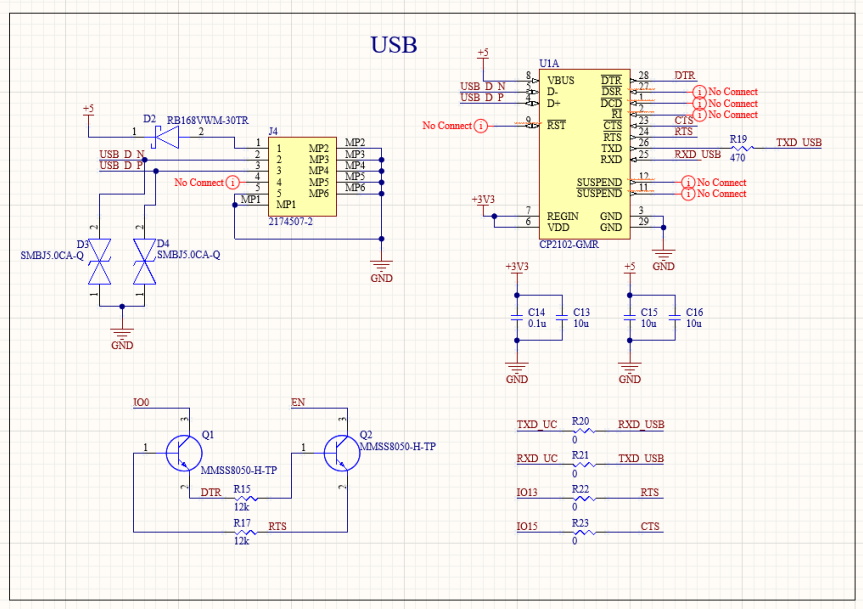


Figure 5: USB Schematic

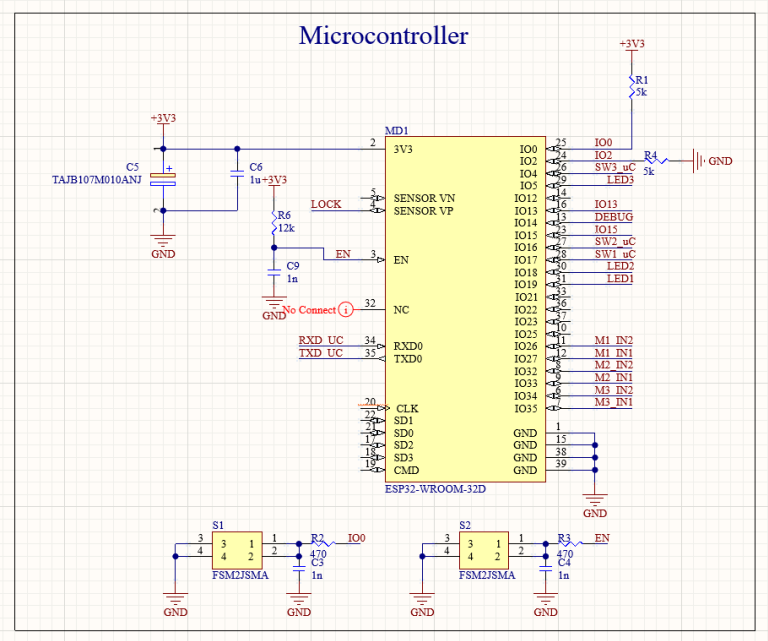


Figure 6: Microcontroller Schematic

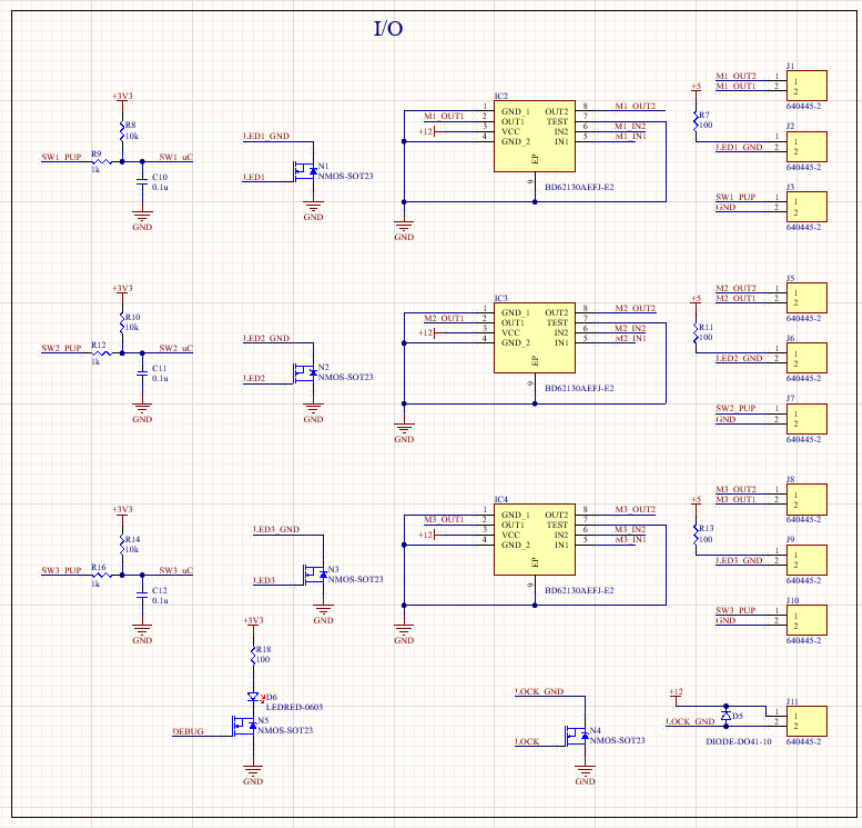
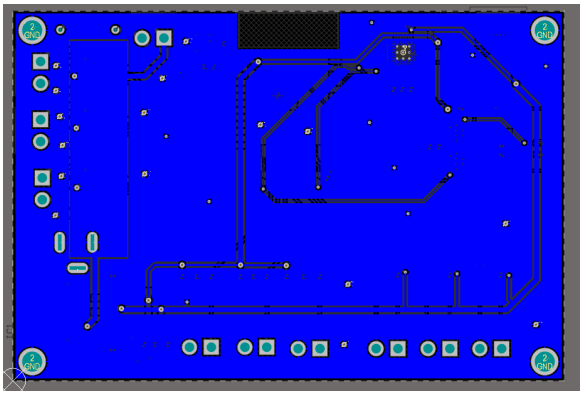


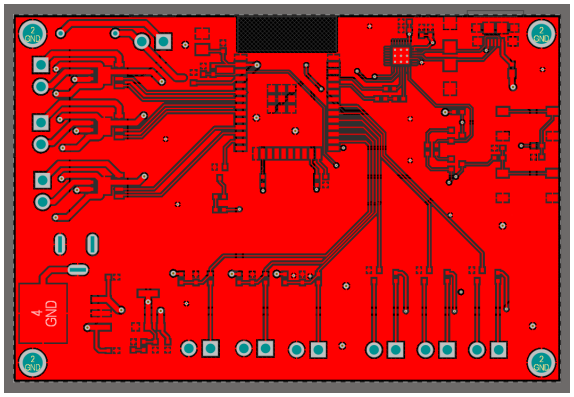
Figure 7: I/O Schematic

2.1.2 Layout

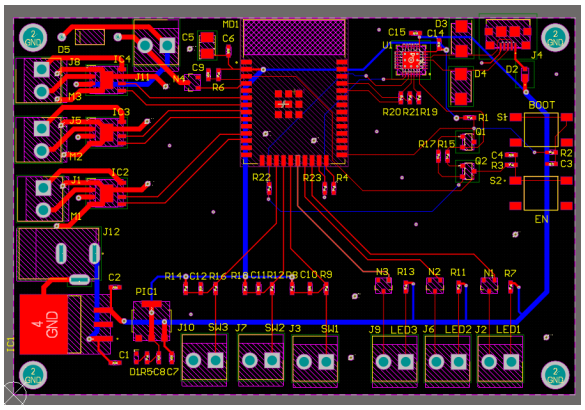
After completing the schematic portion of the printed circuit board (PCB), the next step was completing the layout. We began by placing the most vital components, such as the ESP32 MCU and the USB/UART chip. After the MCU, we began planing the input/output components and identified the optimal placement of components for routing. This involved changing the ESP32 pin assignments for more organized routing. The board was designed in such a way to ensure the power components were kept far away from the USB and communication components. In addition, we ensured the data lines for USB and UART were kept above solid planes for best performance.



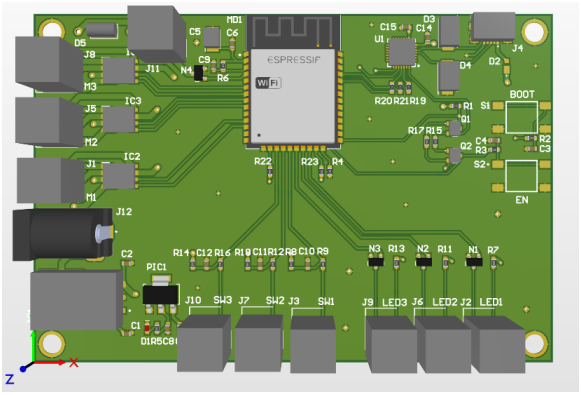
(a) Bottom Layer



(b) Top Layer



(a) Top Layer Tracing



(b) 3D View

2.1.3 Soldering & Testing

After our printed circuit board arrived, we moved on to assembling our board for testing. We first soldered the components needed for programming the board as we believed this was the most important functionality to verify. During our first night of soldering, we soldered the ESP32, the USB/UART chip, USB connector, BJTs and necessary passive components for the previously listed devices. After we finished soldering the components needed for programming the ESP32, we paused the soldering process to test communication with the board. We quickly found that our board was not recognized by out computers. After significant troubleshooting of our system, we identified the problem as a bad connection to our USB/UART chip. To resolve this issue, we modified our PCB to fix the connection and we were then able to connect to the ESP32. Next, we checked the basic functionality of our LEDs, MOSFETs, switches and motor drivers. We verified that all of our components worked as expected except for our motor drivers. We found that we could not control two out of our three motor drivers. Testing with a multimeter, we found that we were not getting an output from the ESP32. After consulting the datasheet, we realized the general purpose input/ouput (GPIO) pins we were using on the ESP32 were input only [4]. We once again modified our PCB to utilize different GPIO which solved our issues. Lastly, we had an issue with our motors getting stuck upon trying to retract. After significant debugging with the oscilloscope to measure our input power to the motor drivers, we identified there was a large inrush current when the motors would change directions. This caused the motor drivers to shut down due to over-current protection. To resolve this issue, we placed a small 1.1Ω resistor in series with each motor to limit the current. After this modification, our motors and overall project functioned flawlessly.

2.2 Firmware Design

The firmware subsystem is responsible for connecting the hardware and software designs together. It must be able to successfully toggle pins based on the current state of the drawers and motors. In addition it must do this while interacting with Bluetooth commands and ensure that it does not violate any safety concerns. In order to do this, firmware must monitor the state of the switches, the locks, and any messages from the app.

To code the ESP32 the Arduino IDE is used and is setup per the steps [7]. To begin testing we

Table 2: Drawer Mechanics Subsystem Requirements and Verification Table

Requirement	**Verification Procedure**
• Linear actuators are driven by the H-Brige of the Control Subsystem	• Initially verify MCU GPIO output is functioning as expected by using an LED connected to a GPIO pin and control using the app. **Success!** • Remove LED and connect H-Bridge to GPIO. Use a multimeter to verify output of H-Bridge is as expected. **Success!** • Connect linear actuator to output of H-Bridge. Verify actuator rotates as expected. **Success!**
• Solenoids are controlled by the MCU to lock the drawers	• Verify GPIO functionality using the above test procedure. **Success!** • Connect the gate of an NMOS to the GPIO pin of the micro-controller • Connect the negative terminal of the solenoid to the drain of the MOSFET and +12V to the positive terminal of the solenoid.Verify the solenoid actuates as expected when toggling the output of the micro-controller. **Success!**
• Limit switches are able to detect when the drawer is open or closed	• Connect 3.3V to limit switch • Connect limit switch to input of micro-controller • Actuate the switch and monitor the input of the micro-controller to verify expected behavior. **Success!**

used the steps discussed in [8] and in [9]. This successfully allowed LED flashing using the ESP32 WROOM dev-kit. Once Bluetooth LED flashing was successfully established, the next step was to implement the usage of non-volatile memory for storage of drawer components. This was done with the data structure found in preferences.h whose operation is discussed in [10].

The firmware subsystem ensures that no illegal states are possible by checking all of the states on our drawers before driving the motors or solenoids. It does this using simple if and else if commands. In addition it runs on a simple system of having a list of possible commands that can be sent from the MCU. For example find item commands, add item commands, unlock drawer commands etc.

This is all run as shown in the block diagram in Figure 10. Once a power reset occurs, all the ESP32 pins are set to safe states. Namely, the solenoid is locked upon turn on, all the motors are deactivated, all the LED's are turned off, and all variables are set to the proper state. Then the loop begins and we check for Bluetooth commands. If there are any, we then store the string and

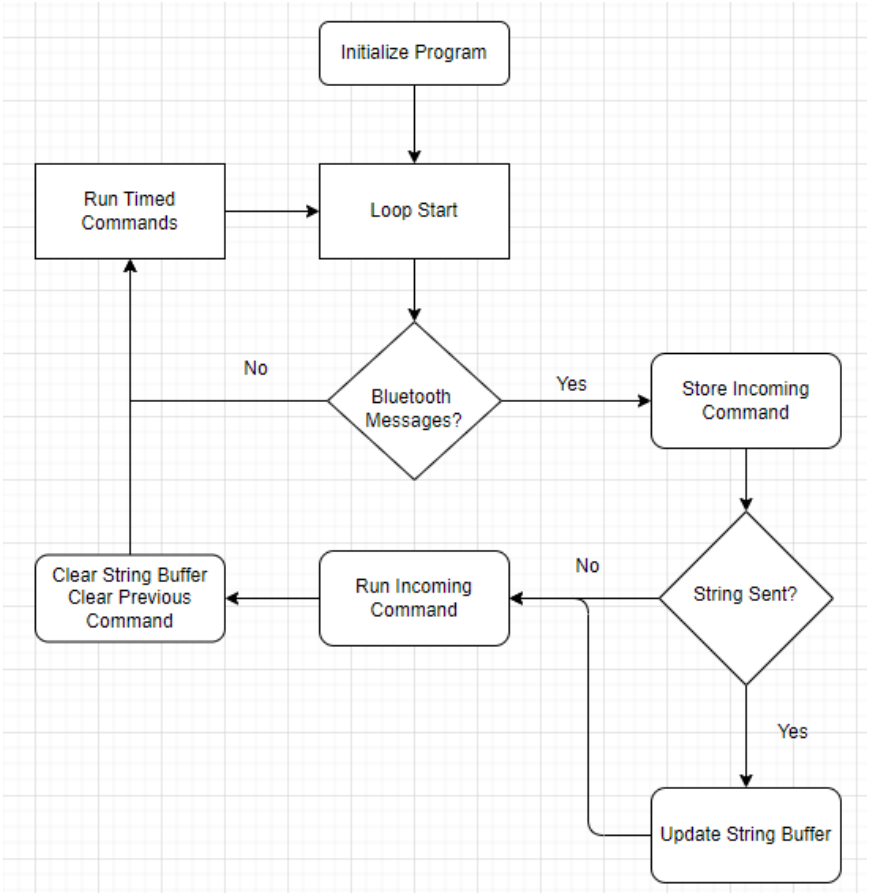


Figure 10: ESP32 Firmware Block Diagram

any corresponding commands. Then, we run the command and clear the string buffer. Finally, we run any callback commands like blinking off an LED after one second.

Lastly to verify that the firmware system operated as intended the requirements and verification shown in Table 3. As shown everything in the table operates successfully.

2.3 App Design

To complete the project design, it was necessary to design an Android application to function as the user interface. The design for the app involved the usage of different screens to house different functional components of to app. This introduced the first difficulty of the app development process, as the Bluetooth connection between the phone and MCU was severed every time a screen was switched on the app. This problem was solved with help from the online resource and YouTube channel Robojax. This channel showed an app implementation that utilized Bluetooth connection

Table 3: Control Subsystem Requirements and Verification Table

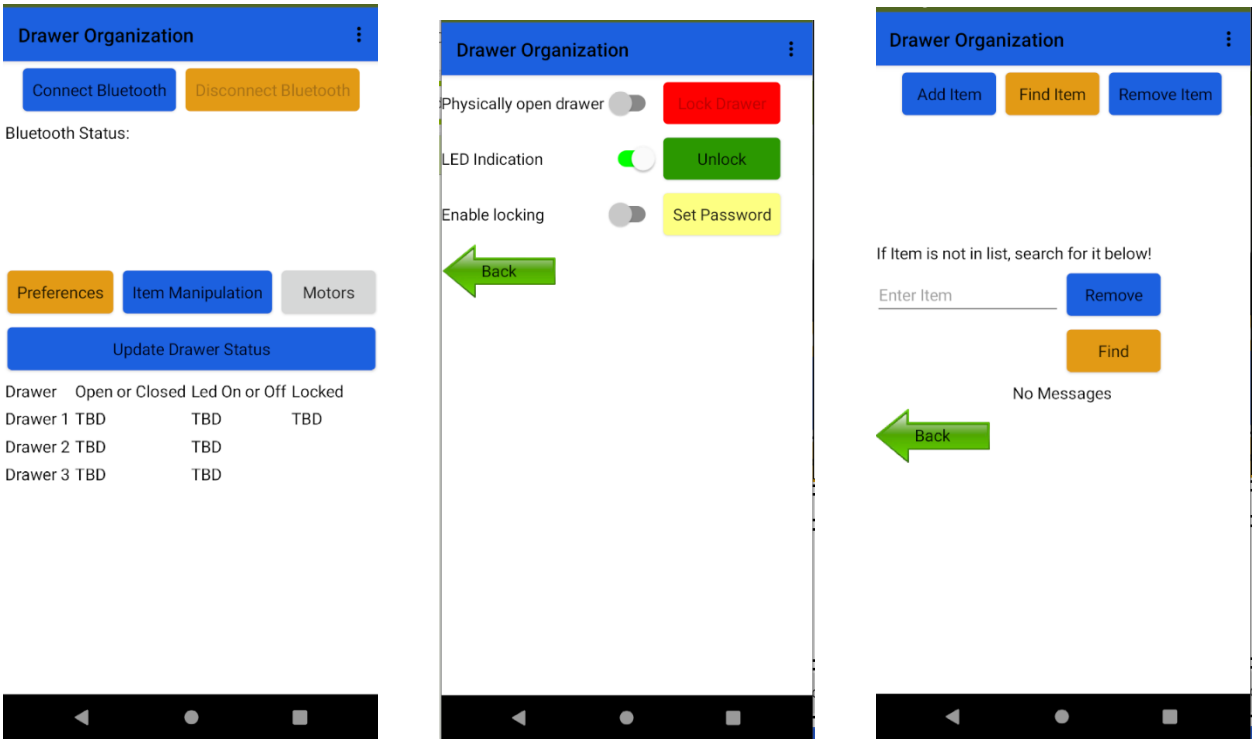
Requirement	Verification Procedure
• Receive command from user interface via Bluetooth	• When command to light LED is sent from the phone, the microcontroller must turn on the LED. **Success!**
• Micro-controller can determine using sensors if a drawer is open or closed and relay that information to the phone app.	• If drawer is closed, it appears as closed on the app, with some potential delay (1 second maximum). **Success!** • If drawer is open it appears as opened on app with some potential delay (1 second maximum). **Success!**
• Micro-controller can toggle its output pins	• Flash Micro-controller with simple program that drives pins and measure the voltage at those pins with a multi-meter. **Success!** • Setup the pins output a PWM waveform and verify with an Oscilloscope that the output matches the desired PWM signal. **Success!**

between Android and ESP32 with a virtual screens method that only visually changed the screen by triggering a change in the visibility of different app components when a given button was pressed [9]. In this way, It was ensured that the app would run smoothly in conjunction with the MCU, without breaks in the Bluetooth connection, since the actual screen was never actually changed.

Screen 1 in Figure 11a acts as a home screen, and provides a way for the user to navigate to the other screens, as well as connect to other Bluetooth devices. This home screen also has a button that reads the MCU pins connected to the LEDs, switches, and solenoid to print the status of these different circuit elements on the screen for the user.

Screen 2 in Figure 11b acts as a preferences or settings screen. The user is able to flip switches to determine their preferred method of drawer indication when an item is searched. On this screen, there is also an option for the user to choose whether they want to have locking capability for the bottom drawer, as well as set the necessary password in the case that locking is enabled.

Screen 3 in Figure 11a contains the functionality of the app to add, remove, and find items from within the MCUs dictionary of drawer-item pairs. To add an item, the user first clicks the "Add Item" button. Then, a text box will appear where the user can type the name of the item they wish to add and use the drawer list picker to select which drawer the item will be stored in. To find or remove an item, the user can either enter the name of the item in the text box at the bottom



(a) Main Menu (b) Open Settings (c) Item Manipulation Menu

Figure 11: App Screens

of the screen and click the corresponding button or use the "Find Item" and "Remove Items" list pickers at the top of the screen. These list pickers are buttons that show the user a searchable list of the items currently stored in the drawers. The list is updated every time that an item is added or removed.

The interaction between the ESP32 and Android app via Bluetooth when an item operation is attempted in the app is detailed in the flowchart below. When the button to add, find, or remove an item is clicked, the app first determines which operation it is trying to perform. If that operation is find, the app first must communicate with the MCU to find which drawer the item is stored in. If the item is found in drawer 3, the bottom locking drawer, the app checks whether the drawer is locked. If it is locked, it will prompt the user to enter their password and send an unlock command. If the command is either find or remove, the app will simply send the command to the MCU, and indicate whether the Bluetooth has timed out or there was an error/duplication in removing or adding the item. The overall functionality of our App is verified in Table 4.

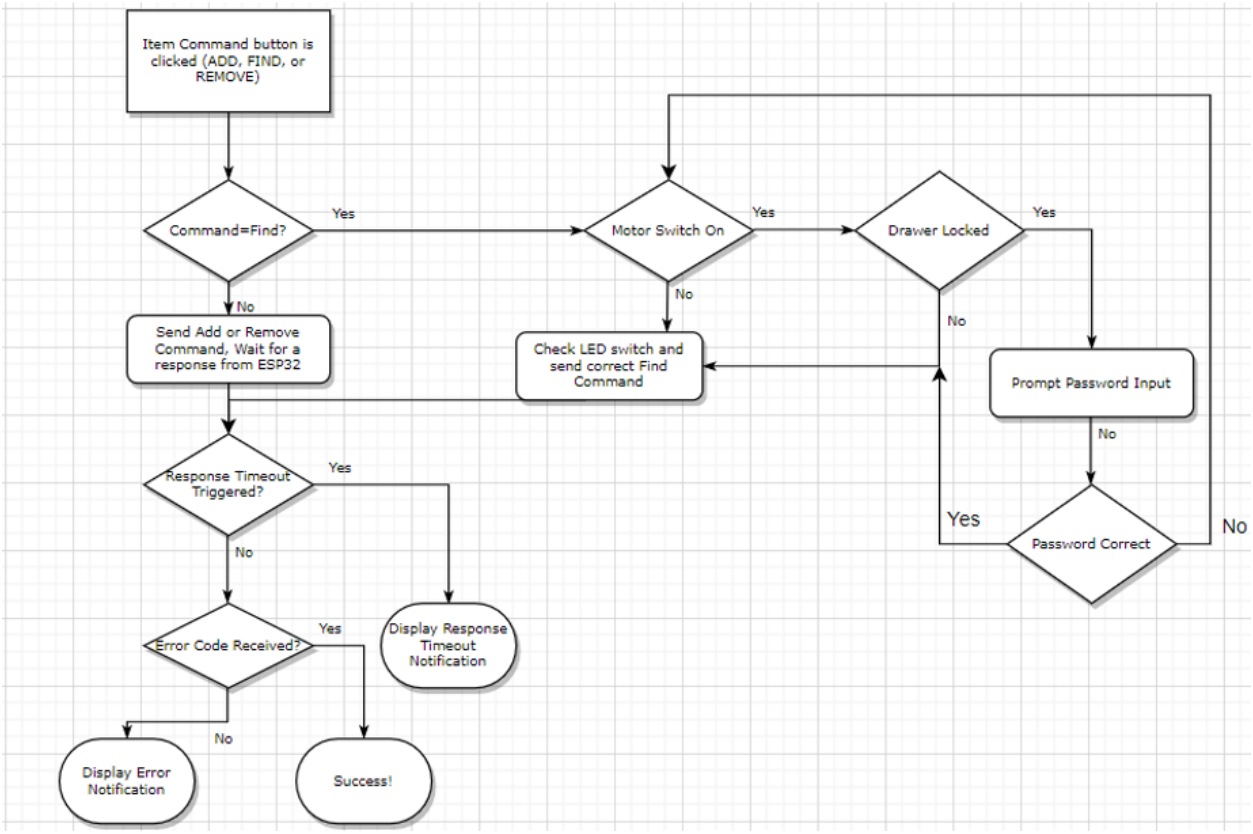


Figure 12: Flowchart for Item Manipulation in App

2.4 Putting it all Together and High-Level Verification

Once the firmware, software, and hardware were brought to a reasonably workable stage, the rest of the design process involved combining all of these different components to create a cohesive project. Due to some testing that had been previously done using the ESP-32 development kit in conjunction with the Android application, we were confident in the functionality of the FW/SW when isolated from the hardware. Thus, the majority of the final design process involved modifying the firmware to support the PCB. The debugging steps taken to solve these issues are detailed above in the individual sub-system that they pertain to. Overall, this process was extremely successful, and all of our High-Level Requirements were met, as detailed in Table 5.

Table 4: User Interface Requirements and Verification Table

Requirement	**Verification Procedure**
• Send and receive data to and from the ESP32 micro-controller via Bluetooth	• Test data sending capability by using an LED connected to a GPIO pin on the micro-controller. Send a digital signal from phone telling MCU to light the LED. Verify LED lights up.**Success!** • Test data receiving capability by sending a known string of Bluetooth data to the phone app. Have the app echo the received data and verify it matches the sent string.**Success!**
• Micro-controller responds to user input within 1 second.	• Verify this capability using the same test procedure as above, with an LED and a GPIO pin on the ESP32. Using a stopwatch, time the response between LED turning on/off and the request for LED to turn on/off. **Success!** • If a stopwatch is too slow to time accurately, then our latency must be low enough.
• User interface responds to micro-controller status within 1 second	• Test this capability using the same test procedure as above, sending a Bluetooth signal from the micro-controller telling the phone app that a certain drawer is closed or opened. Use a stopwatch to verify that the time between the data send and app update is less than 1 second. **Success!** • If a stopwatch is too slow to time accurately, then our latency must be low enough.

Table 5: High Level Requirements and Verification Table

Requirement	**Verification Procedure**	**Result**
• MCU command produces correct binary result on mechanical components (LED, switch, solenoid)	• Using Android App, toggle the LEDs on/off, the motor open/closed, and the solenoid locked/unlocked. Verify correct behavior visually.	• LEDs turn on visually when LED pin is high. **SUCCESS** • When motor pins are configured to extend linear actuator, the linear actuator extends. **SUCCESS** • When motor pins are configured to retract linear actuator, the linear actuator retract. **SUCCESS** • When solenoid pin is set high, the solenoid is in the retracted position. When it is set low, the solenoid is in the extended position. **SUCCESS**
• All operations performed by the drawers must be initiated by the Android application.	• Verify that final product can open drawers, close drawers, find items, add items, remove items, lock/unlock, etc. all through Android application. No separate software should be used.	• All commands for the item data structure and drawer mechanics originate from the Android application. **SUCCESS**
• Bluetooth connection between MCU and Android application should be established up to the largest dimension of an average room.	• The size of an average room is 14x16 feet, thus the Bluetooth connection must be established for at least 21.26 feet [11]. Verify using a tape measure.	• The Bluetooth connection was established up to a distance of 41.2 feet. **SUCCESS**

3 Cost & Schedule

3.1 Cost

The total cost for parts as seen below in Table 6 before shipping is $168.20. 5% shipping cost adds another $8.41 and 10% sales tax adds $16.82. We can expect a salary of $40/hr*2.5hr*60 = $6000 per team member. We need to multiply this amount with the number of team members, $6000* 3 = $18,000 in labor cost. This comes out to be a total cost of $18,193.43.

Table 6: Cost Analysis Table of Components

Description	**Manufacturer**	**Quantity**	**Total Price**
MCU ESP-WROOM-32	EspressIf Systems	1	$4.20
H Bridge BD62130AEFJ-E2	ROHM Semiconductor	3	$5.61
Solenoid Lock ROB-11015	SparkFun Electronics	1	$5.50
Linear Actuator OK03	SZMWKJ	3	$81.54
12V Power Supply LJH128	ALITOVE	1	$35.00
LDO LM3940	Texas Instruments	1	$0.59
LED WP7113ID5V	Kingbright	3	$0.20
Micro Limit Switch MZ-7811	Moujen	3	$3.82
MOSFET N-CHANNEL 20-V	Vishay	4	$2.28
USB to UART Bridge QFN28	Silicon Labs	1	$5.06
LED Side Mount 8mm	Chanzon	3	$8.71
Passive Components	Various	N/A	$15.69
–	–	–	**$168.20**

3.2 Schedule

Table 7: Schedule table

Week	Task	Assignee
Week 1 9/26 - 10/2	Design Check & Met Machine Shop Again to Verified Component Selection	Everyone
	Verified Parts, Started Schematic & Ordered ESP32 Dev-kit	Michael S
	Ordered Android Phone for Group & Researched Android Programming	Michael G
	Researched ESP32 Programming	Nathan
Week 2 10/3 - 10/9	Design Review & PCB Board Reviews	Everyone
	Finished Schematic, Ordered Components, & Started Layout	Michael S
	Began Coding App & Interfaced with ESP32	Michael G
	Began Testing Code on ESP32 & Interfaced With Phone	Nathan
Week 3 10/10 - 10/16	First PCB Way, Teamwork Evaluation & Last Machine Shop Check	Everyone
	Finished Layout & Assisted Testing	Michael S
	Continued App & Started Basic ESP32 Tests	Michael G
	Assisted Michael G With App & Performed Tests	Nathan
Week 4 10/17 - 10/23	Tested/Developed Software & Firmware	Everyone
Week 5 10/24 - 10/30	Started Assembling PCB & Tested Hardware	Everyone
	Tested Hardware & Modified PCB Schematic	Michael S
	Tested App on PCB	Michael G
	Tested ESP32 on PCB	Nathan
Week 6 10/31 - 11/6	Second PCB Way Order	Everyone
	Finished Layout Changes & Tested	Michael S
	Debugged Software	Michael G
	Debugged Firmware	Nathan
Week 7 11/7 - 11/13	Developed Software & Firmware	Everyone
Week 8 11/14 - 11/20	Mock Demo & Debug	Everyone
Week 9 11/28 - 12/4	Final Debug & Demo	Everyone
Week 10 12/5 - 12/8	Final Presentation & Finished Paper	Everyone

4 Conclusion

4.1 Successes and Challenges

Overall, this project was a success. Our hardware, firmware and software all functioned together in the desired manner to produce the result described in our project proposal and design document. Our app was able to add remove and find items in the drawer system via Bluetooth connection to the MCU, and our firmware and hardware was able to handle these requests and perform the desired indication, whether it be opening the drawer, unlocking drawer or lighting LED at the side of the drawer. However, despite the overall success of the project, their were still challenges that had to be overcome along the way. These challenges ultimately helped our group to think in creative ways to debug the issue, which is a skill that is extremely valuable for engineers to have.

4.2 Key Takeaways

One of the biggest takeaways that our group got from this project was gaining experience working with mechanical designs for an electronics project. In typical ECE classes, the focus is on the circuitry and electronic components. In order to be successful on a project like this one, we had to work closely with the machine shop and get creative to mechanically figure out how our system works. Additionally, our team had many opportunities to apply our knowledge from ECE and CS classes in unrestrained and creative ways. This was a valuable opportunity, and a refreshing change of pace from the typical restricted and problem based theoretical applications that are done in a classroom setting. Finally, we gained valuable experience debugging hardware and software issues in a team setting. Initially, our debugging process was a little disorganized and inefficient, as we would end up trying the same things multiple times to fix an issue. To combat this inefficiency, we developed a more systematic approach as a team to debug our problems, which was a big reason for our success.

4.3 Ethics & Safety

In order to have a successful product at the end of the senior design process, creating the project in a safe and ethical manner was paramount.

First and foremost, safety concerns that might be potential issues with the project must be

addressed (IEEE Code of Ethics I.1) [12]. Because the project in question is a drawer system, there are not many inherent safety risks, although some still are present. One concern that exists is a possibility of electrical shock or fire due to poor wiring of circuits or incorrect regulation of current and voltage. This was remedied by making sure that the wires used in the project are in working order, and making sure to check that our currents and voltages don't surpass the maximum rating values for any of our components. Another safety concern is the possibility of injury if a motorized opening drawer is pushed open into a person. This cannot be completely mitigated in the design process, as the user of the product has some responsibility to not put themselves in harm's way. However, by having the linear motor operate at a relatively low speed, the risk of injury from such an event can be minimized.

Ethical concerns for the project must be addressed to ensure that the design process is completed in a manner compliant with IEEE standards. The main concern with ethics with regards to this specific project involves plagiarism (IEEE Code of Ethics II.5) [12]. This project required its designers to perform research on different components and design methods. To make sure that credit is given to the correct people, we made sure that every effort was put forth to make sure the proper sources are cited in the proper manner.

References

[1] S. Borsheim, "Organizing & time management statistics," *Simply Productive*, 23-Apr-2014. [Online]. Available: https://www.simplyproductive.com/2012/03/time-management-statistics/. [Accessed: 12-Sep-2022].

[2] "12V 330lb 1500N Linear Actuator Electric DC Motors for Medical Car Lifting Sofa" *ebay.com.* [Online]. Available: https://www.ebay.com/itm/12V-330lb-1500N-Linear-Actuator-Electric-DC-Motors-for-Medical-Car-Lifting-Sofa-/112752531430. [Accessed: 26-Sep-2022]. a

[3] "Rob-11015: Digi-Key Electronics," *Digi.* [Online]. Available: https://www.digikey.com/en/products/detail/sparkfun-electronics/ROB-11015/6163694. [Accessed: 25-Sep-2022].

[4] "ESP32 series - espressif." [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accessed: 24-Sep-2022].

[5] "BD62130AEFJ," ROHM. [Online]. Available: https://www.rohm.com/products/motor-actuator-drivers/dc-brush-motor/bd62130aefj-product. [Accessed: 24-Sep-2022].

[6] "DMN2300UFB4 - diodes incorporated." [Online]. Available: https://www.diodes.com/assets/Datasheets/DMN2300UFB4.pdf. [Accessed: 25-Sep-2022].

[7] "Installing ESP32 in Arduino IDE (windows, mac OS X, linux)," Random Nerd Tutorials, 19-Apr-2022. [Online]. Available: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/. [Accessed: 01-Nov-2022].

[8] "Esp32 bluetooth classic with Arduino IDE - Getting Started," *Random Nerd Tutorials*, 10-May-2019. [Online]. Available: https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/. [Accessed: 01-Nov-2022].

[9] "ESP32 board: Turn on and off led using Bluetooth from mobile device," Robojax. [Online]. Available: https://robojax.com/learn/arduino/?vid=robojax_ESP32_Bluetooth_LED_blink. [Accessed: 31-Oct-2022].

[10] "Esp32 save data permanently using preferences library," Random Nerd Tutorials, 06-Mar-2021. [Online]. Available: https://randomnerdtutorials.com/esp32-save-data-permanently-preferences/. [Accessed: 01-Nov-2022].

[11] L. Eiler, "The average bedroom size & what to consider when Remodeling yours," *CRD Design Build*, 28-Jul-2021. [Online]. Available: https://www.crddesignbuild.com/blog/average-bedroom-size. [Accessed: 12-Sep-2022].

[12] "IEEE code of Ethics," IEEE. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 12-Sep-2022].

Appendix

Table 8: Common Abbreviations for Electronics Terms

Electrical Term	**Abbreviation**
Input/Output	I/O
Integrated Circuit	IC
Light-Emitting Diode	LED
microcontroller	MCU
N-Type MOSFET	NMOS
Printed Circuit Board	PCB