**Embedded LED Matrix**

Final Project

Sage Marks and Matthew Krueger

3360:0001 – Embedded Systems

Professor Beichel

University of Iowa, College of Engineering

5/9/25

# Table of Contents


1. Introduction ..... 3
   a) Motivation and Background ..... 3
   b) Goals and Specifics ..... 3
2. Implementation ..... 3
   a) Overview ..... 3
   b) Hardware Description ..... 3
   c) Software Description ..... 10
   d) 3D Modeling ..... 12
3. Experimental Methods ..... 13
4. Results ..... 14
5. Discussion of Results ..... 15
6. Conclusion ..... 16

Acknowledgments ..... 17

References ..... 17

Appendix – Source Code ..... 18

# 1. Introduction

## a) Motivation and Background

Our project goal was to build something that won't be thrown away - or stripped down to preserve parts - after the class ends. The LED Matrix (and game system) sufficed; it allows for various programs, unique displays, and extendibility beyond this course. Because of the ESP32, this system can be iterated on to support user I/O via a web application. As we are roommates, this system is currently being held in our living room. The Iowa Hawkeye-themed system compliments our room nicely.

## b) Goals and Specifics

The goals of this project were to develop a customizable LED matrix, that can be interacted with by the user. The interaction with this matrix was to be done through buttons, RPGs, and joysticks. The sketch program on the matrix functions similarly to the well-known "Etch A Sketch" game. However, our version mode the user to draw with in full RGB rather than monochrome. Additionally supported at this time is the pixel art mode, where small pixel art can be cycled through using the navigation. Use of the controllers were not fully programmed due to time constraint, so these partial programs were redacted from this submission.

# 2. Implementation

## a) Overview

The Embedded Etch A Sketch was implemented using many hardware and software elements. Various user input options provide ease of access to the user to create any image they can imagine. Various menus and visual feedback tools were also implemented so that the user never feels lost when navigating the system. Through many design iterations and test programs we were able to implement a working solution that provides full sketching and image display functionality while remaining completely customizable by the user.

## b) Hardware Description

There was a good deal of hardware that went into the implementation of our customizable LED matrix. The memory required to operate the LED matrix exceeded the amount provided by the Atmega328P. Knowing this fact, we used an ESP32 for all

interactions with the LED matrix. The Atmega328P served as a microcontroller that operated with input devices only. This meant that the push buttons, RPGs, and joysticks all were connected to the Atmega328P. All these devices were interrupt-driven besides our power button. Periodically the flags these interrupts set were sent over USART to the ESP32 to be processed. Below is an exhaustive list of all the hardware that was included in our project.

| Hardware | Quantity | Description |
|---|---|---|
| Atmega 328P µC | 1 | Programmable µC |
| ESP-WROOM-32 | 1 | Programmable µC |
| Active Low Push Button | 8 | Push button for user inputs |
| Rotary Pulse Generator | 2 | Etch A Sketch drawing user input |
| KY-023 Joystick module | 2 | Joystick for controllers (XY-axis) |
| RGB LED Matrix Panel | 1 | LED display |
| Bidirectional logic level converter | 1 | Communication ESP32-Atmega 328P |
| 1N5189 diode | 2 | Power latch circuit- reverse voltage |
| IRF4905 PMOS | 1 | spikes |
| 2N222A NPN transistor | 1 | Power latch switching |
| | | Power latch |
| 10KΩ Resistor | 12 | Debouncing and power latch circuit |
| 0.01µF Capacitor | 5 | RPG and push button debouncing |

**Figure 1**: Hardware materials list

The specific way in which our connections are made is outlined below in the project schematic. Many wire connections had to be made for this project, especially between the ESP32 and the LED matrix. The number of these connections quickly cluttered the back of the matrix housing. This is one area we can improve on our project in the future.

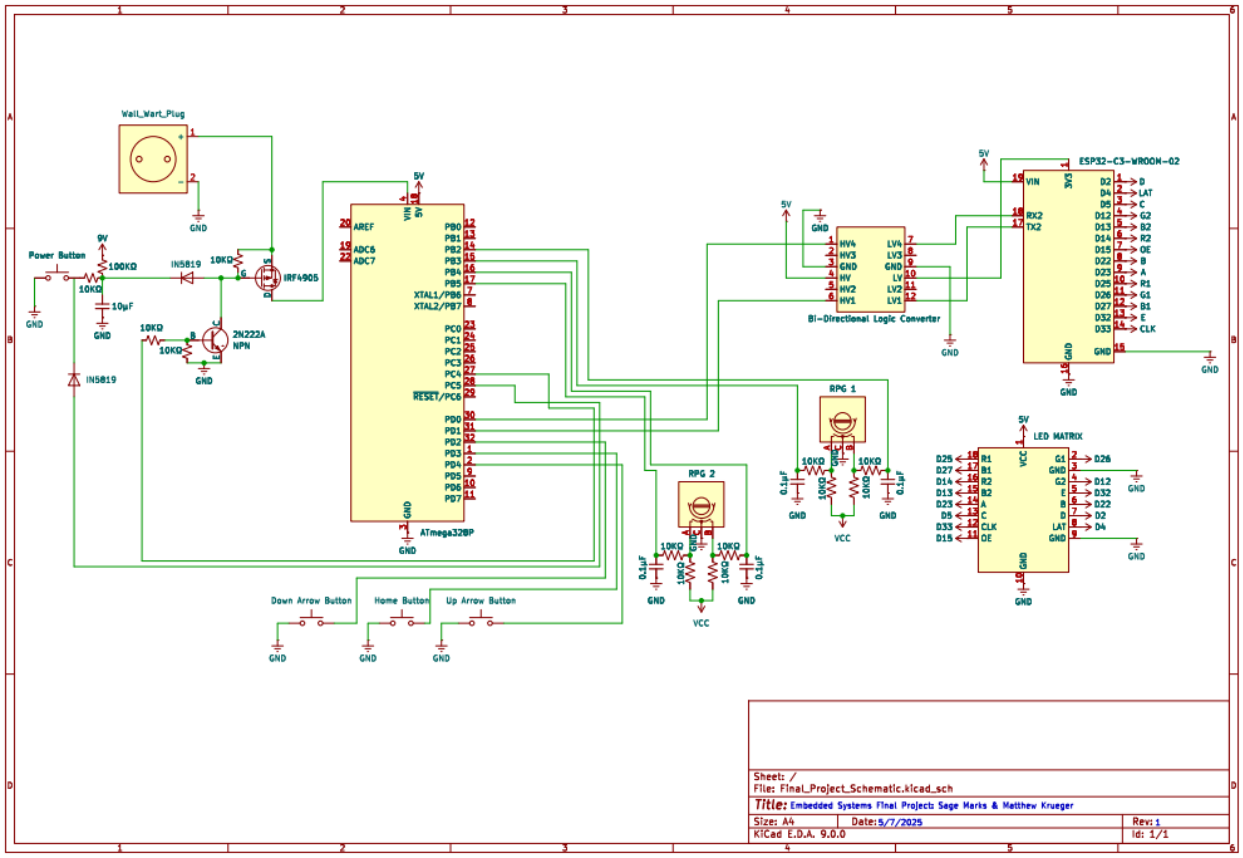


**Figure 2:** Project Schematic

The user inputs integrated into our system consisted of two RPGs and four active low push buttons. The use of our RPGs is for drawing on the matrix, the RPG located on the left side of the matrix is used for moving the cursor and drawing left and right, while the RPG located on the right side of the matrix is used for moving up and down. This implementation is very similar to how the traditional Etch A Sketch works. The push buttons located on the side of the matrix are used for scrolling through menus and selecting menu options. The power button is used for toggling the entire system on and off. Images of these user inputs can be seen below in **Figure 3**.

**Figure 3:** User inputs

All connections were made through soldering and through-hole PCB. This was done to make a more modular solution capable of fitting inside the back of the Matrix assembly. A sizable learning curve was involved with soldering these wires as many different techniques had to be employed to make sound connections. A 5V and ground rail was placed in the middle of the enclosure so that all peripherals could have easy access to a voltage source while maintaining common ground. This common ground is vital to ensuring signal integrity and correct voltage references. An image of the physical wiring and soldered PCBs can be seen below.

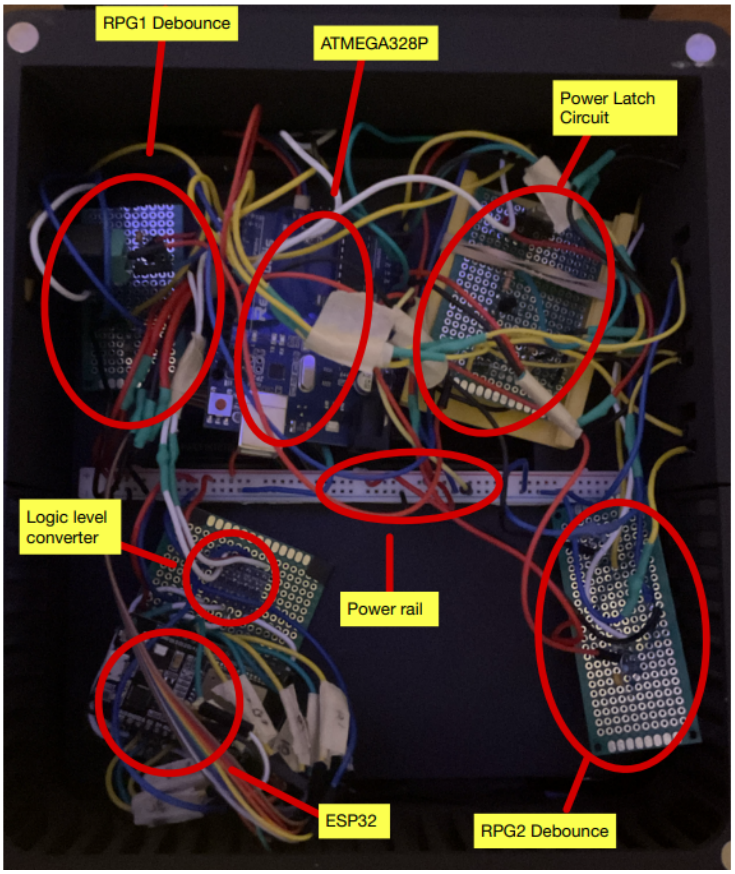


**Figure 4:** Matrix Hardware

To power the system a latch circuit was implanted using a PMOS and NPN transistor. The NPN transistor locks the PMOS in an ON or OFF state when a button press is toggled. There are two GPIO pins utilized with this system. One is used for sensing button inputs while the other is used as an output to the NPN transistor to turn it on and off hence making the latch. Originally, we tried to power the system on a 9V battery. At first, this worked but as we added implementation for colors other than red that draw more current the colors began to display incorrectly or flash. Since the LED matrix has 4096 individual LEDs this is a large current drawing, especially when many LEDs are lit up. Colors such as white or blue require a greater current draw than a color such as red. The matrix can draw up to 4 or more amps. This is not suitable for a 9V battery, it was this fact that led us to switch our power supply to the wall wart plug that was utilized in lab 4 for the PWM fan.

The debounce approach for the RPGs used in drawing was like the method employed in previous labs. For the home, up arrow, and down arrow buttons we used a software debounce approach as we were running low on hardware and the space within

the back of the matrix enclosure was wearing thin. The software debounce approach proved to work just as well as the hardware debounce.

To communicate between the ESP32 and Atmega328P microcontrollers a logic level converter is a must. This is because the ESP32 operates at a 3.3V logic level while the Atmega operates at a 5V logic level. The bidirectional logic level converter ensures that signals that are meant to represent a 1 or high voltage stay recognized as that, and so low logic levels stay in that range. This ensures proper communication. Serial communication via the UART was opted for as it proves as a reliable and easy-to-implement solution. The serial communication experience gained in the previous lab proved to be helpful as the functions and initialization of the UART were already known.

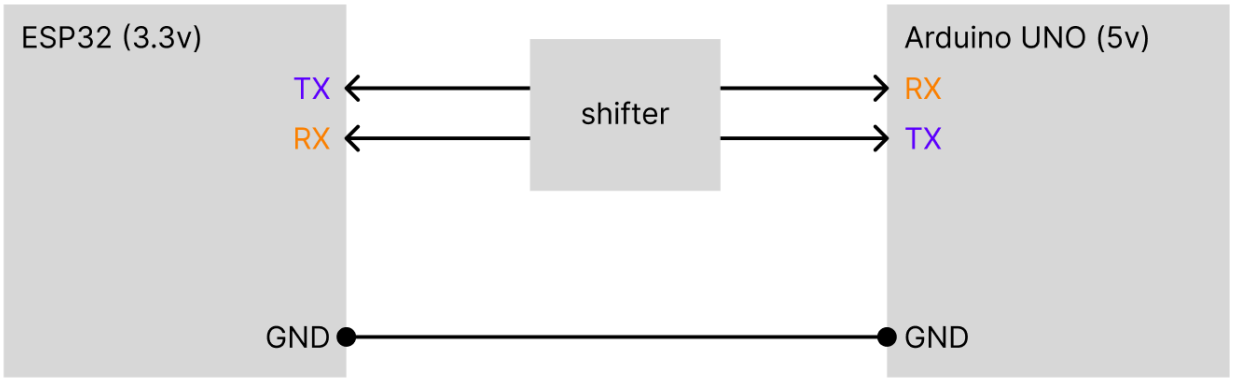


**Figure 6**: Conceptual diagram of level shifter

The centerpiece of the system is the LED matrix, a 64x64 (4096 individual LEDs) matrix. With so many LEDs, the amount of ram needed to store the pixel information (location, rgb, etc) is too much for the 2KB of SRAM of the ATMega328P. Additionally, the matrix would take up most of the GPIO pins on the board with the 16-line HUB75E interface. This is the reason why the ESP32 is a necessity of this project; it has 512MB of RAM and plenty of pins.

The LED matrix is driven using the HUB75(E) (E is for the additional addressing ABCDE used in larger matrices). There are 16 pins including two grounded pins. At one time, 2 rows are lit with R1B1G1 and R2G2B2 describing the information for pixel values. ABCDE are for addressing. 2^5 = 32 which is one row. This information is latched by LAT, and outputted with OE. This is a synchronized process and thus there is a CLK pin. On our board, there are two HUB75(E) interfaces for chaining multiple LED matrices.

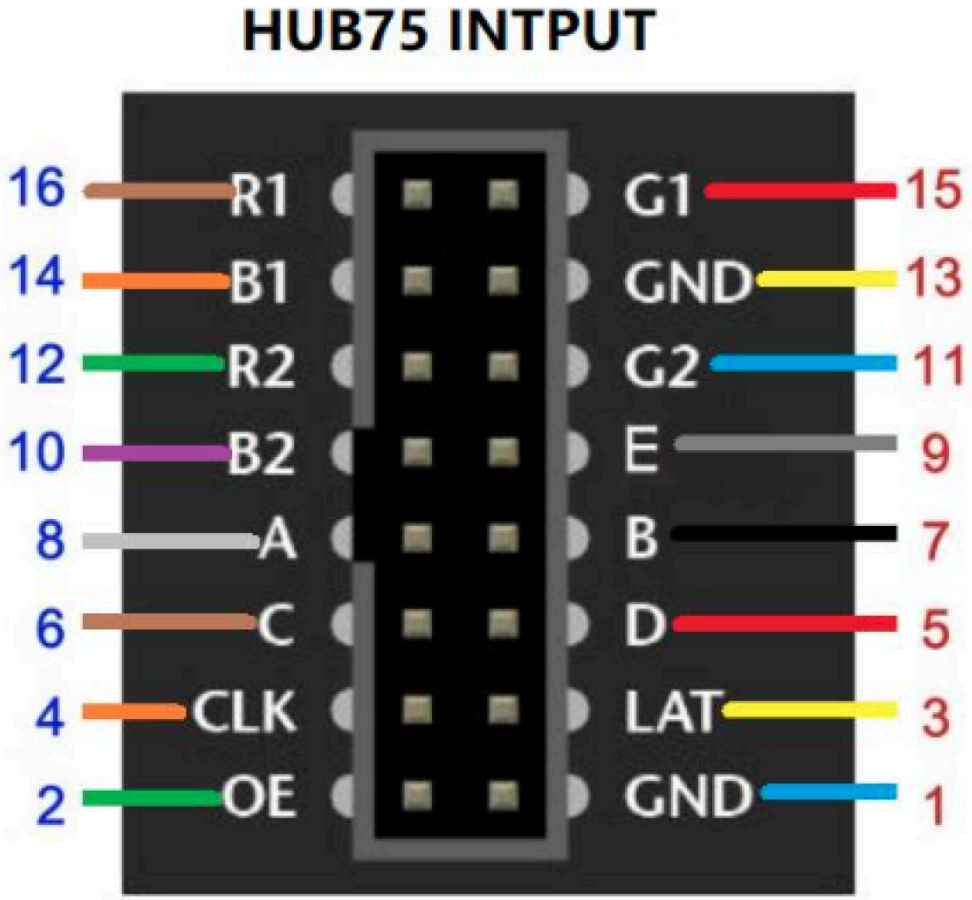


**Figure 7**: HUB75(E) interface

On the matrix itself are over 30 IC chips. We purchased an off-brand clone of the Adafruit 64x64 Matrix to save money, so the following information is incorrect, but applicable to LED Matrices in general. On our specific board is 24 shift registers, 8 demultiplexers for addressing columns, 3 HUB75E buffer chips, and an additional demultiplexer for addressing rows. This hardware correlates with a 2-row scan, meaning that only two rows oof the LED Matrix are lit at one time. However, with extremely refresh rates and human persistence of vision, the human eye cannot perceive the scanlines.

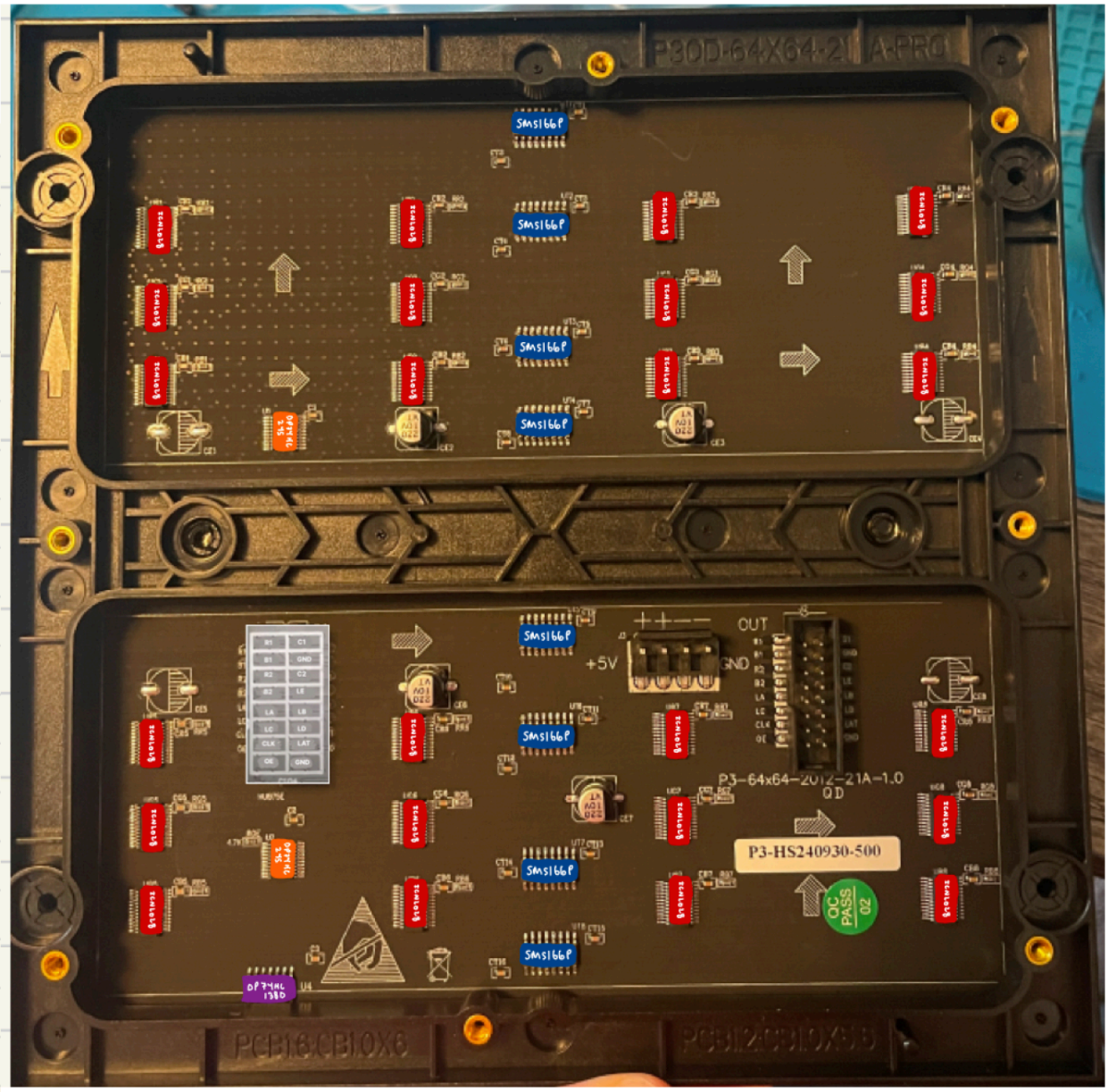


**Figure 8**: LED matrix ICs & interfaces

## c) Software Description

With the use of two devices, we divided our project into two programs. The VSCode PlatformIO extension made this process easy, allowing to isolate environments (libraries, framework, build dependencies) by device – something which the Arduino IDE does not allow. Below is an exhaustive list of the external libraries included by each device

| Device | Library | Description |
|---|---|---|

| | | |
|---|---|---|
| Atmega 328P µC | Arduino.h<br>avr/io.h<br>avr/interrupt.h | Required for Arduino setup() and loop() on PlatformIO<br>GPIO definitions<br>ISR vectors |
| ESP-WROOM-32 | Arduino.h<br>Adafruit GFX<br>FastLED<br>ESP32 HUB75 … | Required for Arduino setup() and loop() on PlatformIO<br>Core graphics<br>General Arduino LED matrix driver<br>HUB75E interface driver |

**Figure 9**: C libraries list

The ATMega328P contained one main.cpp file. This file contains all functionality of the ATMega328P handling the user input from peripherals and the UART. To improve the performance of the system, interrupts were used for the following:

| Interrupt Pin (Arduino Pin) | Peripheral | UART String sent |
|---|---|---|
| PCINT0 (PB0) | Button 'B' (Controller 1) | "btnController1B" |
| PCINT1 (PB1) | Button 'A' (Controller 1) | "btnController1A" |
| PCINT2 (PB2) | RPG ,1 A (LHS) | "rpg1CW/rpg1CCW" |
| PCINT3 (PB3) | RPG 1, B (LHS) | "rpg1CW/rpg1CCW" |
| PCINT4 (PB4) | RPG 2, A (RHS) | "rpg2CW/rpg2CCW" |
| PCINT5 (PB5) | RPG 2, B (RHS) | "rpg2CW/rpg2CCW" |
| PCINT18 (PD2) | Button 'Down' | "btnDownArrow" |
| PCINT20 (PD4) | Button 'Up" | "btnUpArrow" |
| PCINT22 (PD6) | Button 'B' (Controller 2) | "btnController2B" |
| PCINT23 (PD7) | Button 'A' (Controller 2) | "btnController2A" |

**Figure 10**: UART message list

The 'Home' and 'Power' buttons have multiple functions depending on the duration of the press and hold making interrupts unsuitable for these buttons. Additionally, the joystick XY signals are analog and cannot be implemented as an interrupt. For these signals, traditional polling was used inside of the main loop of the program. This creates a very long file. In retrospect, splitting into separate .cpp could have improved readability.

The ESP32 handles the bulk of the system's functionality, both driving the LED matrix and handling program logic. The source code separates the .h files inside of the 'include/' and the .cpp implementations in the 'src/'. PlatformIO handles the compiling and

linkage of these files. The ‘include/’ and ‘src/’ directories are parallel, each including the subdirectories ‘EtchASketch/’ and ‘PixelArt/’.  main.cpp resides in ‘src/’.

The main.cpp file of the ESP32 serves to receive the UART commands coming from the Arduino. The strings listed above are matched and mapped to the wanted action. Pairing these actions with a pseudo-state machine (HOME, COLOR_SELECT, EtchASketch, LOGO_DISPLAY… note current naming convention is clunky & not descriptive), the user can control what is being displayed to the LED matrix. The starting screen is set to HOME.

The ‘EtchASketch/’ directory includes files to program the COLOR_SELECT screen/state – which the user sees when selecting the ‘Sketch’ program from the HOME screen menu. Here, the user selects the color which they want to use for the ‘Sketch’ program. Additionally, the user interface is programed such that the user can change the current drawing color by using the ‘Up’ and ‘Down’ arrows.

The ‘PixelArt/’ directory includes a single file that holds the pixel maps for the supported images and methods to cycle through images in a slideshow fashion. To create a single image, a pixel map with each character representing a pixel was stored in a character array. For example, the character ‘B’ represents a black pixel, ‘b’ a blue pixel, and ‘r’ a red pixel. This was a very tedious process and there is an opportunity to improve it in the future using AI. There are online tools, however, we couldn’t crack how to use them and deduced that because it is such a small matrix, then maybe a tool like this has yet to be designed. More research will be conducted. After creating the array, the characters are scaled to enlarge and displayed row by row to the matrix.

### d) 3D Modeling

This system was designed to be a minimal viable product or proof of concept for a fun game console toy. 3D models for the frame, button caps, and controllers were created using Fusion360 and printed using a BambuLab P1S 3D-printer. The estimated cost (we saved with filament deals) of the frame was calculated using BambuStudio. A breakdown of these costs and the .stl files can be found inside of our repository.

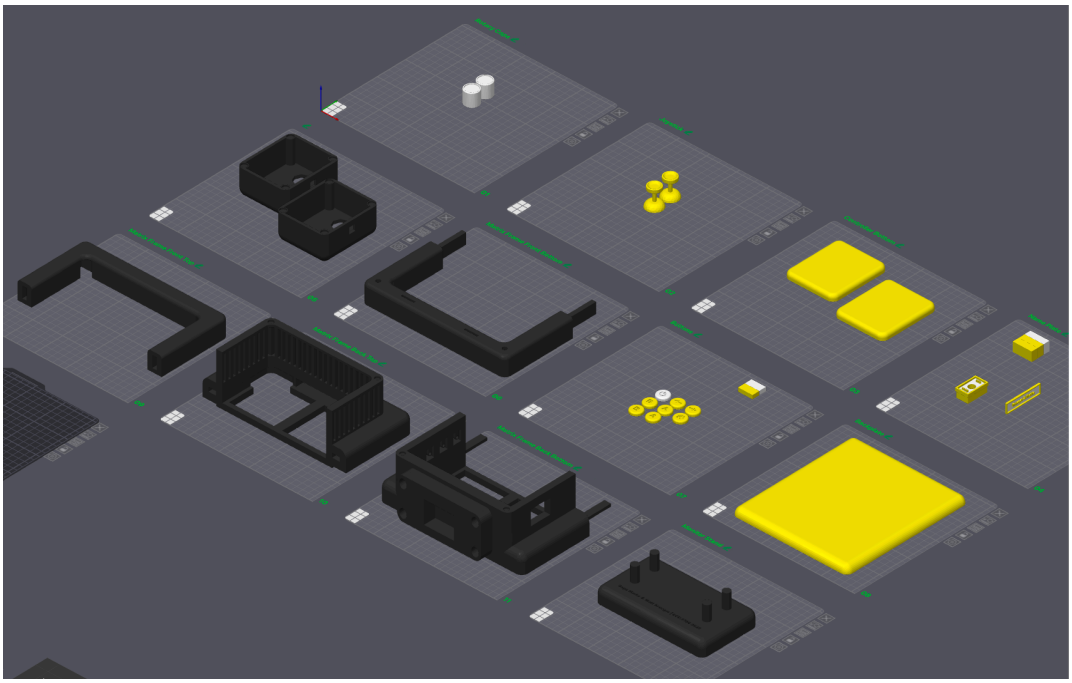

**Figure 11**: 3D models

## 3. Experimental Methods

Throughout the design process, many different test methods were implemented, both on the hardware and software side. Many of the tests not only tested if something worked on the hardware side but also on the software side simultaneously.

One of the first tests conducted was done to see how the LED matrix functioned. This test was carried out by finding the ESP32-HUB75 library and connecting the ESP to the matrix then flashing the test code. By doing this we were able to see what colors and what sections of the matrix functioned correctly. To test serial communication and if we were correctly reading from user inputs the serial monitor functionality of platform IO was utilized. This is a very handy tool to see how the two microcontrollers communicate with one another.

Another test that was conducted was to see how the 9-volt battery operates in the system. This was done by reading the voltage level on the 5V power rail, coming from the regulator on the Atmega328P. When the matrix was getting power from the rail and many LEDs were being displayed, the corresponding readings on the multimeter showed a drop in voltage well below the 5V needed for the matrix to properly function. Using this information, we came to an informed decision to switch to wall power to supply a more stable current and voltage for our system.

The use of experimental and test methods is vital to the success of any project and is one way to ensure that you are on the right path. Without these methods, it is hard to tell if you are progressing towards your final goal. In our case, the test methods helped us know when to change certain design parameters or remove parts of the design altogether as we would not have enough time to finish them before the deadline.

## 4. Results

**Figure 12**: Final product (physical)


**Figure 13**: Final Product (displays)

## 5. Discussion of Results

In the results section above various images of our final product are included. The group was very satisfied with the final version of this project given the time constraints placed upon us. A modular and customizable electronic Etch A Sketch was created that has full color-changing functionality and drawing capabilities. If someone does not feel creatively inclined to make their drawing, pixelated images are provided under the images tab that can be displayed as decoration. These images can be customized from within the code to

display an image you can imagine falling within the 64x64 bit parameters. Easy-to-navigate menus were implemented so that the user could find and create images they wanted to display. Evidence of the functionality of the product can be found in the demo video provided on the references page. One area we wish we could have made more progress in is the controllers and two-player game functionalities of the display. This area will be our focus moving forward with a focus on implementing games such as chess, pong, and tic tac toe. However, the goal of this project was to create a display for someone's household that could be customized, and we knocked it out of the park.

## 6. Conclusion

Ultimately, our design was - and is - predicated on simplicity and ease of use. Its modular frame allows for swappable themes, the simple yet powerful microcontrollers allow for many peripherals, and the intuitive libraries simplify the software overhead. As with any good hardware and software, the design is built to last. The conglomeration of labs resulted in a cohesive system that puts the fun back into engineering. The programs are enjoyable and accessible for users of any age, which is a success in our books.

## Acknowledgments

We would like to thank Professor Beichel for his excellent teaching and guidance throughout the semester. This project would not have been possible without constant learning from him. We would also like to thank each of the Embedded Systems TAs for their unwavering support throughout the class whenever we had issues with labs or our final project. Their knowledge and expertise helped us reach new heights every day. Without this team of support, we find it hard to believe that anyone would have success in this course.

## References

ARDUINO. Arduino GitHub Repository. 2025. <https://github.com/arduino/Arduino>

ADAFRUIT. Adafruit GFX Library – Core graphics library for Adafruit displays. 2025. <https://github.com/adafruit/Adafruit-GFX-Library>

AVR-LIBC. avr/interrupt.h – Interrupt Service Routine (ISR) definitions. 2022. <https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html>

AVR-LIBC. avr/io.h – AVR device-specific I/O definitions. 2022. <https://www.nongnu.org/avr-libc/user-manual/group__avr__io.html>

"Arduino Power Latch." Youtube, uploaded by PKAE Electronics, February 27, 2024, <https://www.youtube.com/watch?v=2yTPqQ3P5pQ>

Beichel, Reinard. *Embedded Systems: C Programming. The University of Iowa, 2025*. <*https://uiowa.instructure.com/courses/248357/files/30138332?module_item_id=8205647* >

Beichel, Reinard. *Embedded Systems: Serial Communication. The University of Iowa, 2025*. <*https://uiowa.instructure.com/courses/248357/files/30164502?module_item_id=8213613* >

Beichel, Reinard. *Embedded Systems, Rotary Pulse Generators and Lab 3. The University of Iowa, 2025* <*https://uiowa.instructure.com/courses/248357/files/29778930?module_item_id=8162158* >

CIRCUITSTATE. DOIT ESP32 DevKit V1 Wi-Fi Development Board – Pinout Diagram & Arduino Reference. 2022. <https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/>

ESPRESSIF SYSTEMS. ESP32 Series – 2.4 GHz Wi-Fi + Bluetooth® + Bluetooth LE SoC. Datasheet Version 4.9. 2025. <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>

FASTLED. FastLED Library – High-performance LED animation for Arduino. 2025. <https://github.com/FastLED/FastLED>

MRCODETASTIC. ESP32-HUB75-MatrixPanel-DMA – ESP32 DMA-driven HUB75 LED matrix library. 2025. <https://github.com/mrcodetastic/ESP32-HUB75-MatrixPanel-DMA>

Pighixxx. *The Definitive Arduino Uno Pinout Diagram*. *May 5, 2013*. <https://uiowa.instructure.com/courses/248357/files/29320694?module_item_id=8042318>

WAVESHARE. RGB-Matrix-P3-64x64 – 64×64 RGB LED Matrix Panel, 3mm Pitch. 2025. <https://www.waveshare.com/wiki/RGB-Matrix-P3-64x64>

## Appendix – Source Code

There are too many files to include in this document. Please refer to our Github page. Additionally, Doxygen was used to produce a local HTML page with all commented code.

- Source Code: https://github.com/mattnkrueger/ECE-3360_Embedded_Systems/tree/main/EmbeddedEtchASketch/src
- Doxygen 'index.html': https://github.com/mattnkrueger/ECE-3360_Embedded_Systems/blob/main/EmbeddedEtchASketch/src/doxygenDocs/html/index.html