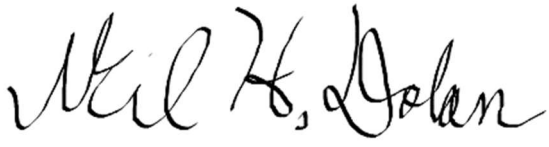
Lounge Improvement Committee/Event Clock for Shared Spaces

Tahsin Kazi, Neil Dolan, William McCollough

Date of Submission: December 13, 2022

Capstone Design ECE 4440 / ECE4991

Signatures



Statement of work:

William

My contributions to the project were focused on the software ran on our clock. I designed the web interface that is used to create and view events. This included the design of the site, the creation of events using the online form, and the editing of clock settings. I also designed the parser that took this information from the web interface and interpreted it, storing in it in a way that could be read and used by the software that runs our LED matrix display. I also designed and wrote the event loop for our display software, which ensures that the correct events are displayed, that upcoming events are stored for future displaying, that the clock can be set to a given time, and that repeated events occur properly. These contributions interacted with the display programming led by Tahsin by providing the display with the correct events at the relevant time. It interacted with the board design, led by Neil, by running off of the PCB designed by him automatically when given power. In addition to these design contributions, I led the creation of documents for the project, such as the Proposal, Midpoint Presentation, and Final Report.

Tahsin

My contributions to the project were focused on the embedded programming that allowed us to control all the components such that we were able to fulfill the project requirements. I also took ownership of the 3D modeled parts to allow for clean product design with an effective display that is visible from a large distance. On top of abiding by NEMA enclosure standards for our enclosure, I made sure that 3D printed components were easily mountable on our backboard using standard M3 screws. I programmed the Raspberry Pi Picos to properly interface with the matrix panel, RTC, and seven segment display. I created the code for printing text onto the matrix panel along with the scrolling feature and a few extra features such as configurable color and speed. I programmed the driver functions so that William's server can just call a select few functions with a standard input and have it properly display text on the matrix display. I came up with my own algorithm for driving the seven segment display and the matrix panels, some of which went through massive design iterations for the most optimized version of the code.

Neil

My contributions to the project focused upon the creation of the physical apparatus to support our microprocessors, and interface them with the displays. This took the form of laying out, assembling, and testing the custom printed circuit board, as well as general construction of a support apparatus for the displays. Abiding by general best practices for the PCB, and minimizing the number of through-hole components to ease mass-manufacture. I soldered the various components, and created the wiring harnesses, in addition to salvaging existing parts from the NI lab and recovering components from our first prototyping board. These contributions interacted with the display programming led by Tahsin, by enabling easier more permanent communication between the displays and the Pico. They further interacted with the server programming led by William by ensuring the server system was properly powered and capable of redundant communication with the LED controlling pico.

Table of Contents

Contents

Table of Figures

Abstract

This project is a networked clock that allows for the visual display of current events occurring at a location. It can be wall mounted and takes AC power, and is designed to be highly visible even from a distance. It utilizes a custom printed circuit board controlled by two Raspberry Pi Pico microcontrollers. The current time is displayed on a 7-segment display in 24-hour format, and events appear on an LED matrix display underneath. Multiple events can occur simultaneously, each displaying their name, start time, and end time. Events are created on a web interface hosted by the device, which allows for the creation of repeatable and single instance events that will then be displayed on the clock at the appropriate time. If no events are currently occurring, the next occurring event will be displayed. Designed for use in academic settings, the clock allows for easy distribution of information on the appropriate use of a shared space at a given time.

Background

Mixed use spaces are a common feature of universities, offices, and myriad other professional spaces. At the University of Virginia, these spaces can be reserved for club meetings, office hours, mixed use open studying, and other academic or social use cases. However, as it currently stands, display methods for the current usage of these spaces are often insufficient, as existing systems are only designed for use with small, single use rooms [1]. As a result, it can often be unclear as to the current use of a shared space, leading to confusion and disruptive use of a space. Additionally, those who have never been to a shared location before and are unsure of if they are in the right location can find it intimidating to ask, as the current usage of a space is often not signposted anywhere visible while in the space, either relegated to online usage portals or not displayed at all.

Unlike other methods of space reservation, which either have only small, hard to view information panels or no information displayed in the space at all, our clock is designed for easy viewing throughout a shared space, allowing for individuals to understand the current acceptable space usage at a glance. Clocks with the ability to display relevant information from a calendar or schedule do exist, however they are primarily designed for individual use, and require the connection of a smartphone or other controller device for use [2]. These clocks are also limited to showing single events at once, with no ability to display multiple concurrent events. By contrast, our device is self-contained, with it able to host a web server to control the creation and editing of events on a local network, thus allowing for the easy creation and altering of events. It also allows for multiple events to be displayed at once, meaning that a shared space with multiple simultaneous use cases can utilize the clock to display all relevant information. Additionally, its large size allows for easy viewing in larger spaces, as opposed to the smaller sizes and reduced visibility of competing products.

The design and construction of the event clock required skills from throughout the engineering courses taken by our group. The construction of the PCB, power control circuitry, and assembly drew from the Fundamentals of Electrical Engineering series of courses (ECE

2630, 2660, 3750). Neil acted as the lead in regards to this aspect of development, as an Electrical Engineering major. The programming required to create the embedded server, and the knowledge of constraints imposed by developing in a microcontroller environment relied on knowledge learned from Introduction to Embedded Computing Systems (ECE 3430) and Embedded Computing and Robotics (ECE 3501, 3502). Tahsin led this aspect of development, utilizing his dual Electrical and Computer Engineering majors. The creation of the web application for the creation of clock events drew from skills learned in Advanced Software Development (CS 3240) and Computer Networks (CS 4457), such as user interface design, the creation of front-end systems, web application hosting, and communication between microcontrollers. William led this aspect of development, as a Computer Engineering major.

Physical Constraints

Design Constraints

The primary constraints on our design regarded the limitations of our chosen microcontroller and the software available for it. As a relatively new device platform, being first released in June of 2022, the Raspberry Pi Pico W is sometimes limited in its software support [3]. This resulted in a number of limitations regarding our design, as only one SDK for web server hosting exists for the Pico W currently, the Pico HTTP Endpoint Wrangler (phew) [4]. Phew currently does not support the ability to modify lists, and as such we were unable to implement one of our design goals, the ability to modify or delete existing events. Phew also operated in a blocking manner when hosting the web server, not allowing other code to run when the server was operating. This issue was solved by using a second Raspberry Pi Pico for managing the LED matrix and 7-segment displays, connecting the 2 via UART. This allowed for the web server and display operation to occur simultaneously, and only increased the cost a minimal amount due to the affordable price of the Pi Pico.

The other major design constraint we ran into involving our microcontroller was the need to use MicroPython as our primary language. The Pi Pico supports development using either C or MicroPython, but only allows for one language to be used at a time. Due to our use of phew for frontend development, this limited the rest of our design to also use MicroPython. However, this did not end up being a major concern, as we were able to achieve appropriate runtimes using MicroPython throughout our project.

A number of constraints on the manufacture of our PCB were imposed by the chosen manufacturer, Advanced Circuits. This included a maximum of 2 layers, a minimum hole diameter of 0.01’’, and a maximum size of 60 square inches. Our project was able to follow these constraints without issue. We similarly did not run into any constraints regarding part availability, as all needed parts were able to be purchased without issue from reputable retailers, including DigiKey and Mouser Electronics [5][6].

Cost Constraints

Overall, we were able to remain within the allocated $500 budget without too much issue. We expected the most expensive component to be our board, which we estimated would end up using roughly 1/3rd of our total budget, with expected costs including the printing of test and

final boards, assembly costs, and component costs. The actual cost was higher than expected, as component compatibility issues, the need for updated components, and the purchasing of spare components in case of damage resulted in additional parts being ordered. Additionally, some of these parts had unexpectedly high shipping costs, which further increased cost. with express This resulted in overall board costs for the board to be roughly $220, or just over 2/5^{ths} of our budget. Our LED matrix and 7-segment display costs were expected to be each roughly 1/10th of our total cost, with $60 being budgeted for the LED matrix and $55 for the 7-segment display. Both were able to be completed under budget, with the LED matrix costing $40 and the 7-segment costing $30. These savings were due to the cost of 3D printing being much lower than expected. Similarly, the enclosure costs were also much less than expected, costing roughly $30 as compared to the expected $50. This was due to a number of components being found as surplus from previous years, instead of having to be purchased new. The remaining costs, such as the power supply and additional Raspberry Pi Pico microcontrollers, were as expected, with a total cost of $40. This resulted in a total cost for the project of roughly $360, which was higher than our initial cost estimate but still well below the $500 budget. We never felt heavily constrained by cost throughout the development process, as we planned for unexpected costs and were able to come under budget for a number of subsystems.

Tools Employed

For programming our microcontrollers, we used the Thonny IDE [7]. While limited in feature set compared to other similar tools, Thonny is pre-configured for writing code on the Raspberry Pi Pico, handling the assignment of communication ports and other settings that have to be manually configured in other IDEs. Thonny caused no major usability issues, and all code used in our project was written in it. We did not use any software for mathematical analysis or similar, as it was not necessary for the design, testing, and assembly of our project.

To create the web GUI used to create events and manage the clock, Bootstrap 4 was used alongside HTML [8]. A tool for injecting CSS to improve the appearance of websites, Bootstrap was used to supplement the HTML forms used on the web server. It allowed for a more scalable web interface that would remain readable on both desktop and mobile devices, and improved the user experience of our interface.

For creating and modifying 3D models, Autodesk Fusion 360 was used. It was chosen due to its feature set and existing familiarity [9]. Models were either designed from scratch in Fusion 360, or modified from existing sources in Fusion 360. Models were then printed using Ultimaker Cura, which was chosen due to its convenience and support by the 3D printing tools available to our group [10].

For creating our PCB KiCad was used [11]. Though lacking a built-in circuit simulator and auto-placement and routing tools, KiCad allowed great flexibility in implementation and portability for files. Said flexibility was most visible in importing the layout footprints of various component parts, as well as custom schematics for the Raspberry Pi Picos, and hierarchical design generally. No issues were encountered in using KiCad and all schematic and PCB work was done within the program.

Societal Impact Constraints

Environmental Impact

The environmental impact of our clock, outside of the manufacturing of the components used, is minimal. The use of AC power over battery power, lack of any moving parts, and high reliability of required components means that the event clock should be able to perform its required function for a long period of time without need for replacement or service. If a component becomes damaged, the device is generally repairable, as it features off the shelf common parts that can easily be replaced to continue operation. Additionally, the power draw requirements of the clock are relatively low, meaning that operating costs and continually environmental impact are minimal.

Sustainability

In the system's ongoing service life, the primary sustainability concerns are centered on the Real Time Clock (RTC) which is dependent upon coin-cell batteries. Coin cell batteries were historically made with heavy metals, such as cadmium or mercury, but are now largely Lithium based. Given the environmental hazards, particularly to human life, that battery waste poses the RTC battery is of particular importance [12]. When the system reaches end of life and must be disposed of there is the consideration of e-waste. While the ultimate disposition of the system will be handled by the consumer, awareness of potential pollutants is vital. Heavy metals and chemicals within PCBs and components can leach into the environment and act as toxins [13]. As such it will be practice to avoid such components where possible, so as to limit potential exposures, and label the system so as to inform users of disposal requirements.

Health and Safety

As our device uses AC wall power with an off the shelf external power brick and communicates through a standard Wi-Fi protocol, there are no notable Health and Safety concerns regarding off the shelf components of our design. Similarly, the lack of moving parts or exposed wires and the way our design uses a NEMA Type 1 standard enclosure mitigates any major Health and Safety concerns of our designed components. [14] Additionally, our device follows prescribed weight recommendations for wall-mounted devices using the mounting kit we selected. Overall, there are no notable Health and Safety concerns for our device.

Ethical, Social, and Economic Concerns

As a device for the distribution of information that does not currently have a human job equivalent, our device does not pose risks to human employment. It similarly has minimal concerns regarding privacy, as it does not feature any sensors that could be used to violate privacy, or require any personal information to use. As a networked device, it poses the risk of being an entry point to a secure network, however the use of standardized network security methods on the device minimizes this risk. Its use as an information distribution tool should have no significant negative effects on society, as it only provides a new vector for distributing information that individuals already have access to in a less convenient form. It also provides a useful tool for those without access to technology such as laptops or cell-phones, as it distributes

information previously only found on these devices over the internet in a manner that is accessible to anyone without vision impairment.

External Considerations

External Standards

A number of external standards were considered and subsequently used in the design of the event clock. As a device that connects to wireless networks, the Infineon CYW43439 wireless chip on the Raspberry Pi Pico W supports single band IEEE 802.11n 2.4GHz Wi-Fi, and meets FCC requirements for such devices [3][15][16][17]. As we followed usage guidelines for Wi-Fi on our device, these standards were met without any special considerations.

For designing and accessing our web interface, the Hypertext Transfer Protocol (HTTP) was used [18]. The protocol was used to generate the web interface, which was hosted on a Raspberry Pi Pico W. Forms hosted on the web interface would then be sent using the HTTP POST request standard, for parsing by the Pico W for use on the event clock. It was chosen as it is the standard protocol for communication on the web.

To communicate between our two microcontrollers, the universal asynchronous receiver-transmitter protocol (UART) was used [19]. UART was selected due to the hardware support for UART featured on both of our microcontrollers, enabling it to be used with minimal code overhead and without fear of connection issues. A baud rate of 9600 was chosen, as we felt it was a sweet spot between the speed required for our transmissions and the reliability of transmission.

For power, we were able to use an off the shelf AC converter, and as such did not have to factor in NEMA AC power standards. However, our PCB enclosure was designed targeting the NEMA Type 1 standard for protection against dust, light, and indirect water splashes [14].

Our PCB was designed following IPC-2221 specifications for low voltage circuits, with 0.1mm minimum clearance for general purpose components and 0.13mm minimum clearance for power conversion components [20].

We used the WS2812B individually addressable LED standard for both our LED matrix display and 7-segment display, using the NeoPixel protocol for communication and the writing of values [21]. Using a shared LED standard in both components reduced the required amount of code and increased reliability, as the same standard could be used in both.

The programing of our microcontrollers was done following over USB, following the USB 2.0 implementation standard [22]. This communication was done between the microcontroller and the windows computers used to program them with a USB-A to USB-Micro-B cable.

Intellectual Property Issues

Overall, we feel that our project would not conflict with existing patents. While similar patents exist for individual components of our project, none follow the exact design and use case paradigms that we have implemented, and the synthesis of ideas in our project is unique. A

patent that uses similar display technology to our LED matrix display was created in 2017, however it specifies connection to a mobile phone through an app and describes the combination of displays, a distinct use case when compared to our LED matrix's usage in the event clock [23]. Similarly, patents on networked clock devices exist, but these patents are primarily concerned with synchronizing multiple devices over a network, rather than being set over a network and then used to handle scheduled events [24]. Our device is independent of this patent as it does not sync between multiple devices, instead receiving events and storing them locally. Finally, similar patents regarding the programming of lighting devices, such as our LED matrix, exist, but cover specific implementations of such a device that are distinct from our implementation [25]. This specific implementation uses RF signals from a remote to control the lighting device, as opposed to the embedded microcontroller and web interface of our device. Because of its distinctiveness when compared to these similar patents and unique implementation methods not covered in other patents, we feel justified in our belief that the idea behind the event clock would be eligible for a patent.

While the ideas and techniques used in the creation of the event clock are distinct and would likely be eligible for a patent, the specific event clock constructed would not be eligible for a patent application, due to the usage of a Non-Commercial Creative Commons licensed design for our 7-segment display [26]. This license allows for the modification and usage of the provided design in noncommercial products, and requires that all subsequent designs follow the same license. As a result, should we wish to attempt to patent our event clock a new design for the 7-segment display would have to be created and implemented into the proposal. Similarly, a new design would have to be designed or a contract negotiated with the designer of the 7-segment display should the device go into commercial manufacturing. However, in its current use case the 7-segment display follows acceptable usage standards.

Detailed Technical Description of Project.

Overview and Block Diagram

The event clock for shared spaces was designed as a tool for spaces that have multiple events occurring simultaneously, such as a collaborative working location that may host multiple office hours or club meetings at the same time. To this end, the system designed for this can be broken down into the following sections:

1. Hardware
    a. Power distribution
    b. Signal Step-up/Conversion
    c. Pico to Pico Communication
    d. RTC Communication
2. Web Server
    a. Event Creation and Processing
    b. Communication Protocol
3. Event Processing Loop and LED Driver
    a. Event Processing Loop
    b. LED Firmware

    c. RTC access
4. Assembly
    a. CAD & 3D print design
    b. Physical assembly

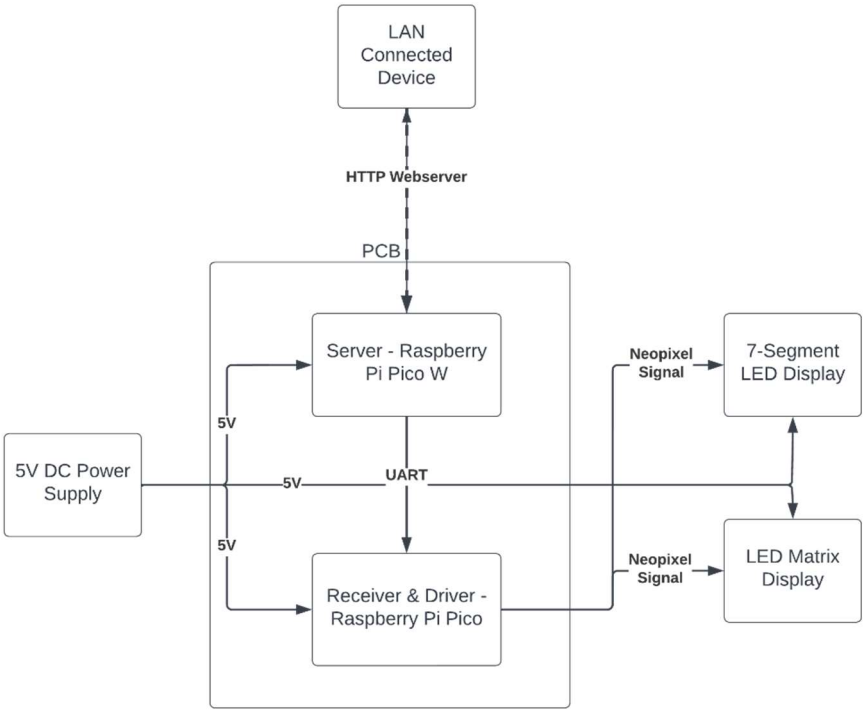


Figure 1: System Block Diagram

Hardware

The hardware system is a printed circuit board (PCB) designed to interface with the LED matrix display, 7-segment Display, Raspberry Pi Pico Microcontrollers, and Real Time Clock (RTC). The PCB contains a power supply and distribution system, signal conversion for the 3.3V GPIO output of the Pico to 5V of the displays, Pico to Pico communication, and communication from the Pico to the RTC. The power supply and distribution system uses a barrel jack to power the whole system from an external DC power source. The signal step-up converts the output of the Pico to meet the required logic levels of the system displays. Pico to Pico communication enables the server Pico to properly interface with the light and RTC operating Pico. RTC communication enables a Pico to interface with RTC to properly set time and recover time after power loss.

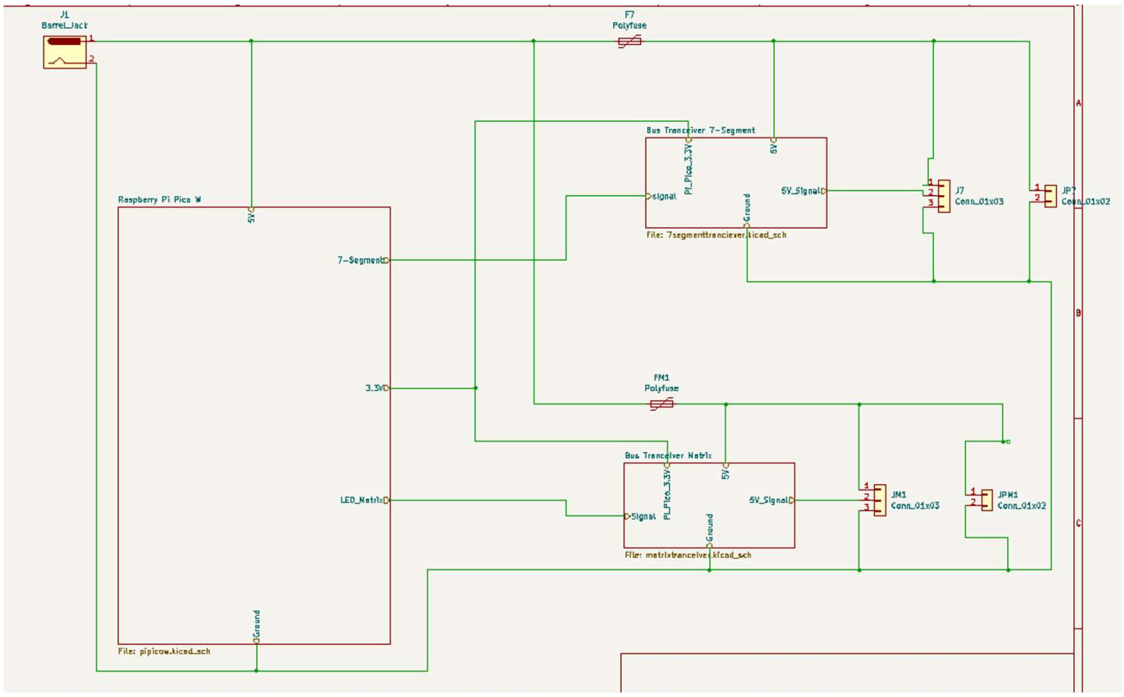


Figure 2: Overview of PCB Hardware

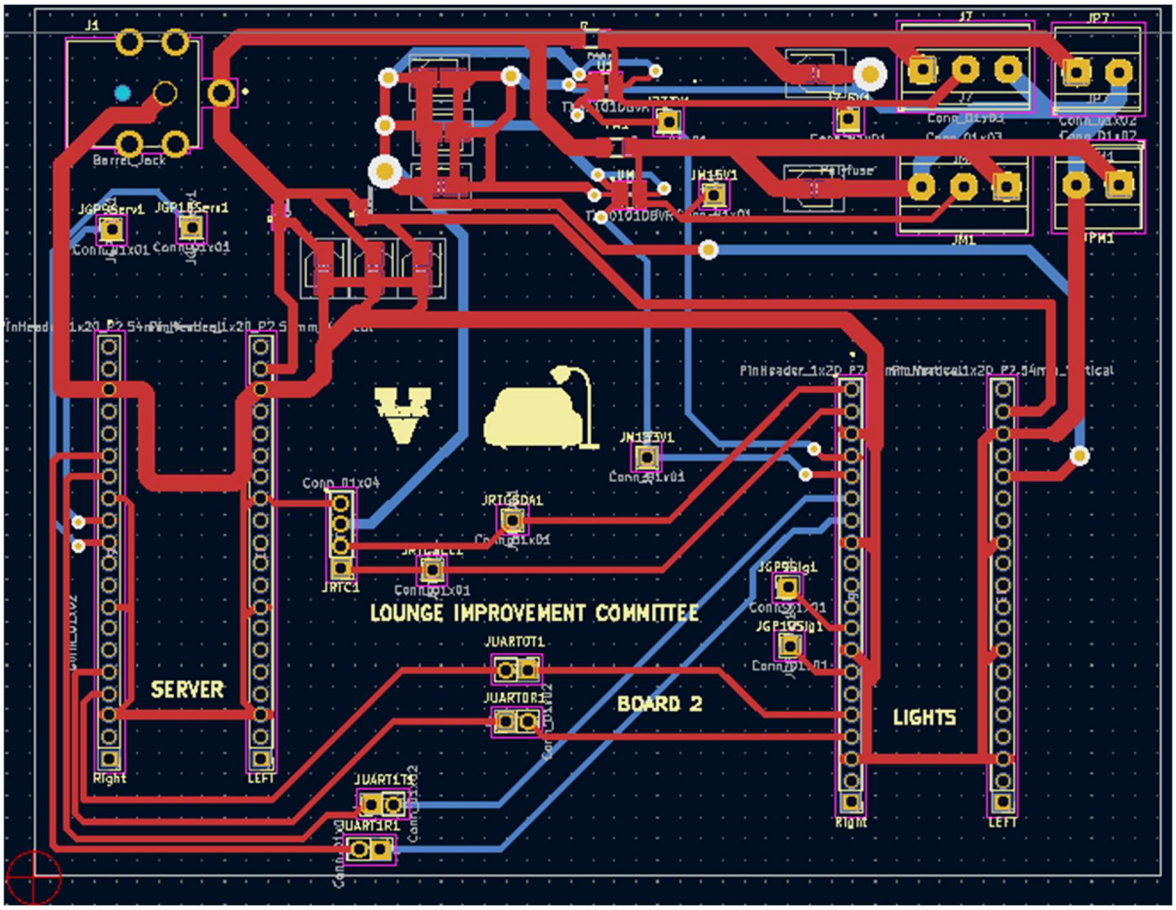


Figure 3: Digital PCB Layout

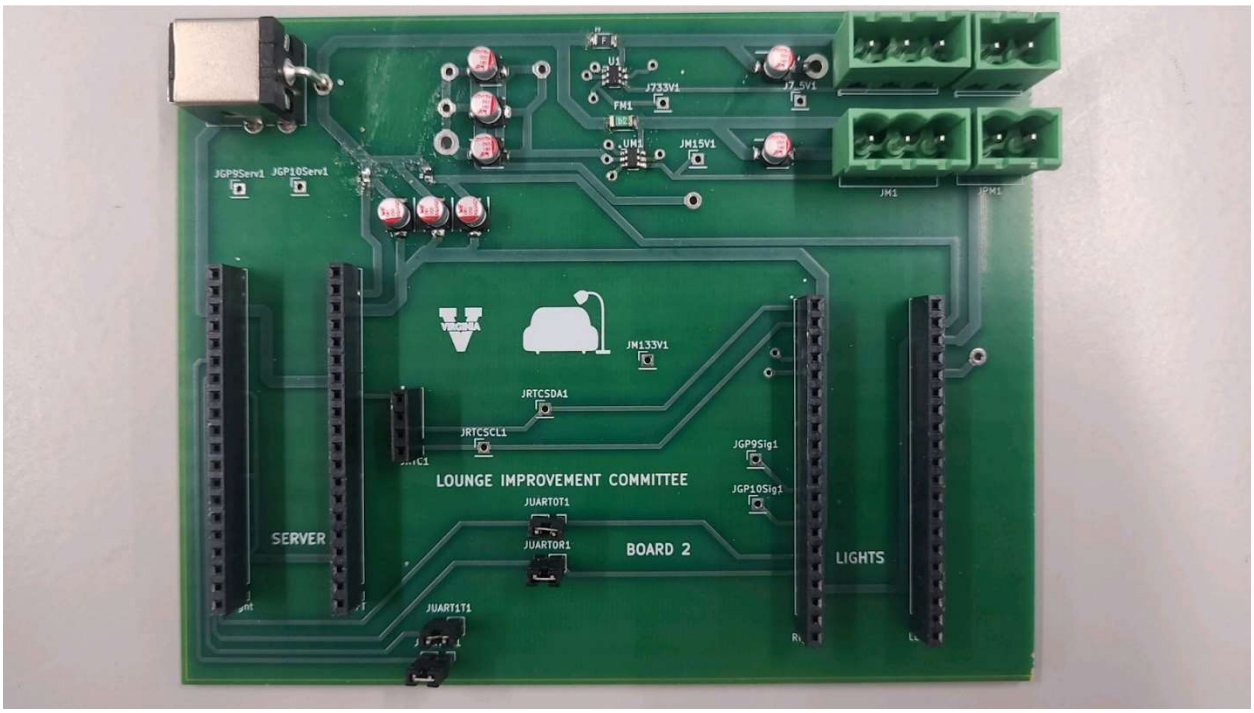


Figure 4: Completed PCB with Populated Components

Power Distribution

The power distribution system uses a barrel jack to take in 5 VDC power from an offboard AC-DC converter. That power is then routed to the 4 primary places from which power is taken: the fuse for the 7-segment display, the fuse for the matrix display, and the two diodes feeding the individual Picos. The fuses act to prevent overcurrent from the displays, with values of 0.5 A and 2 A for the 7 segment and matrix display respectively. The fuses themselves supply power to the relevant level shifters, as well as both the 3-position female header for the main system input for the relevant display, and the 2-position female header for suppling additional power. The barrel jack also serves to provide ground for the whole of the system, routing to all appropriate points including the Picos, level shifters, 2 and 3-position headers, and RTC.

Signal Step/Up

In order to properly address the WS2812B LEDs used in the 7-segment and LED matrix displays, the 3.3V logic signals from the Pico had to be stepped up to 5V. 74LVC1T45W6-7 level shifters, as seen in Figures 5 and 6, were used to do this, as well as to buffer the Pico from the wires themselves and properly communicate with the LEDs. Blocking capacitors were also employed to help ensure smooth operation with the ground of the system and the overall.

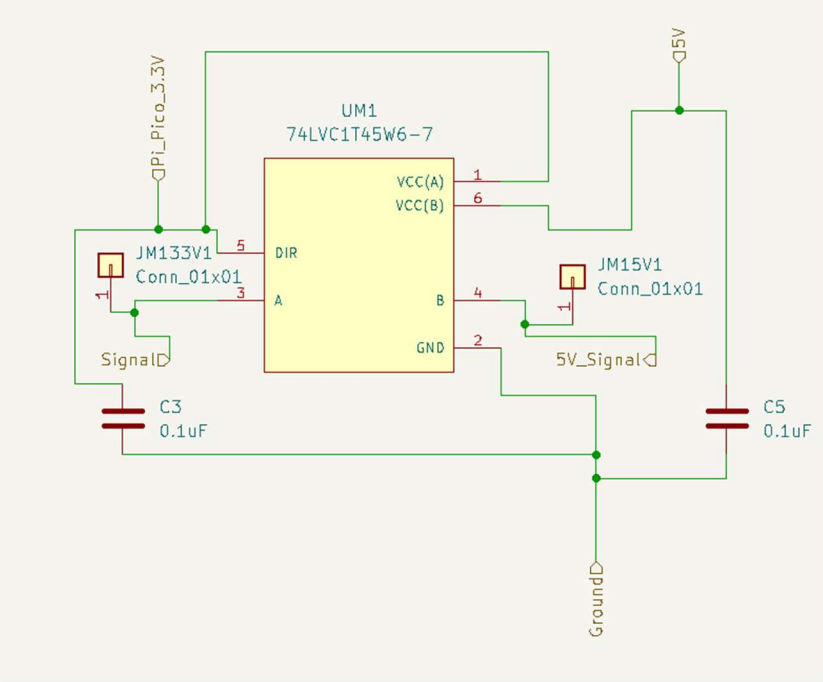


Figure 5: Step Up Transformer for LED Matrix Display Used in PCB

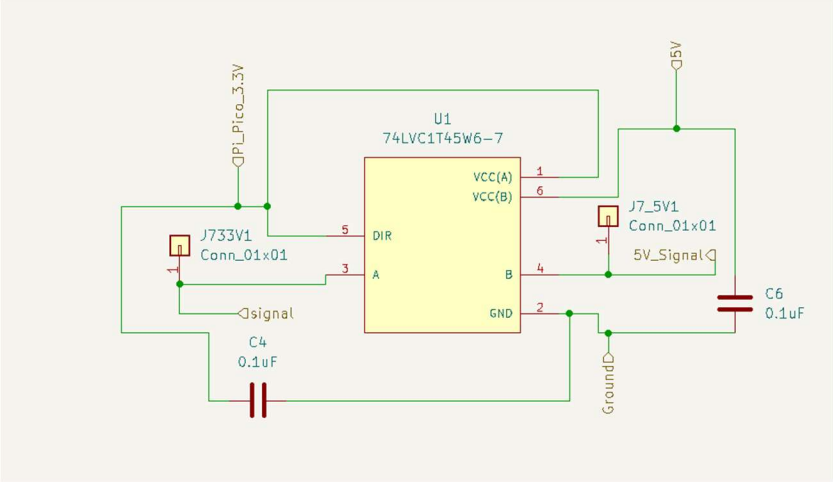


Figure 6: Step Up Transformer for 7-Segment Display Used in PCB

Pico to Pico Communication

The two Picos have to be connected to one another in order to enable the server to communicate with the LED/RTC driver. This connection was based upon the UART capabilities of the Pico, with a transmit and receiving channel required for each single UART connection. Two UARTs were connected, with 2 pin headers for each channel, JUART1R1, JUART1T1, JUART0R1, and JUART0T1 in Figure 7. These headers enabled the channels to either be shunted to the designed destination, directly used as input or test points for analysis, or interchanged to different destinations than initially designed.

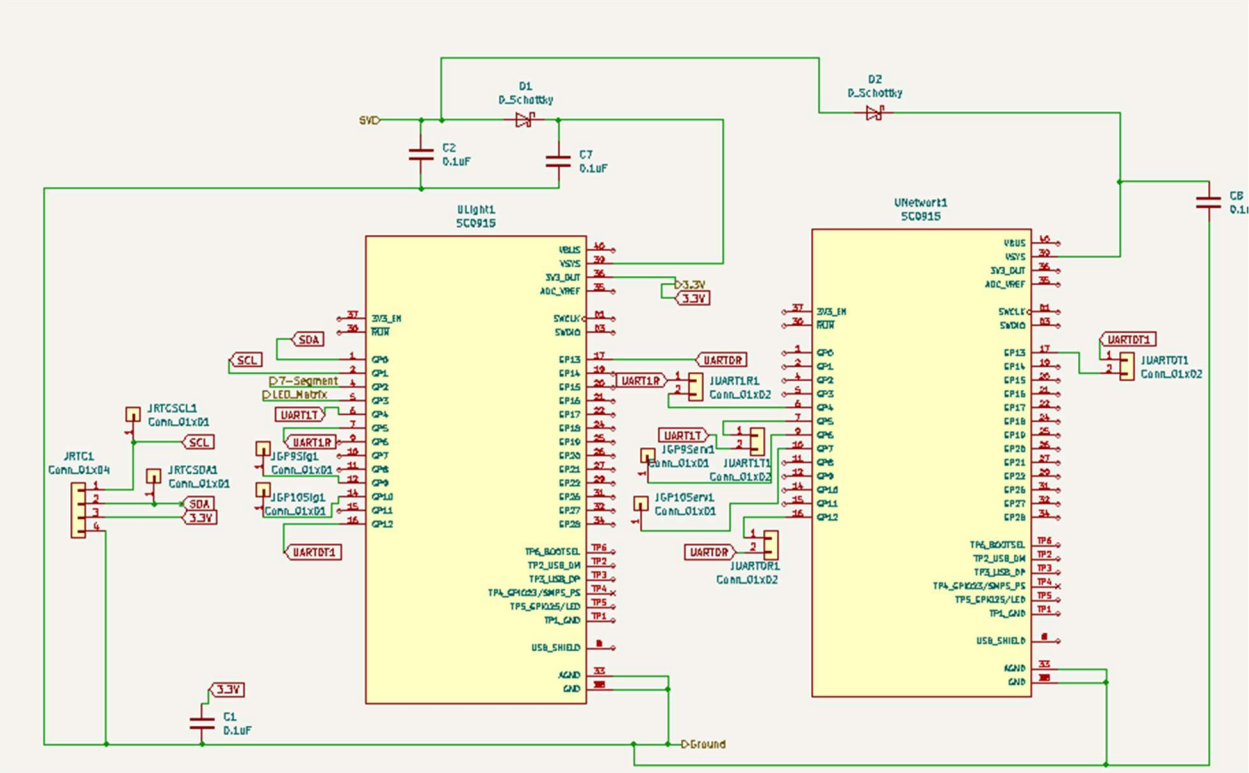


Figure 7: Raspberry Pi Pico Pinouts and Connections on PCB

RTC Communication

The RTC interfaced to the broader board using a 4-pin female header, JRTC1 in figure 7. The RTC requires a power at the same level as it will be accepting logic, a ground, and SDA and SCL channels. The power in this case came directly from the 3.3 V rail upon the Pico itself. The RTC communication was constructed to directly interface with the Pico, with test points, JRTCSDA1 and JRTCSCL1 seen in Figure 7, allowing for the direct evaluation or injection of signals if needed.

Web Server

Event Creation and Processing

The server component of the event clock runs on a Raspberry Pi Pico W microcontroller, using the Pico HTTP Endpoint Wrangler (phew) webserver library [3][4]. When provided with power, the Pico W will attempt to connect to a 2.4GHz Wi-Fi network using a provided username and password, which is hard-coded in a file stored on the Pico W. Pages are rendered by Phew as HTML documents, and use Bootstrap to improve aesthetics and readability [8]. The webpages can be accessed on an IP assigned to the Pico upon a successful connection to a Wi-Fi network, which is sent over UART to the LED matrix for printing and printed on the debug console. The homepage of the hosted website allows for the RTC time to be set and the Event Ending Indicator to be toggled, and is shown in Figure 8.

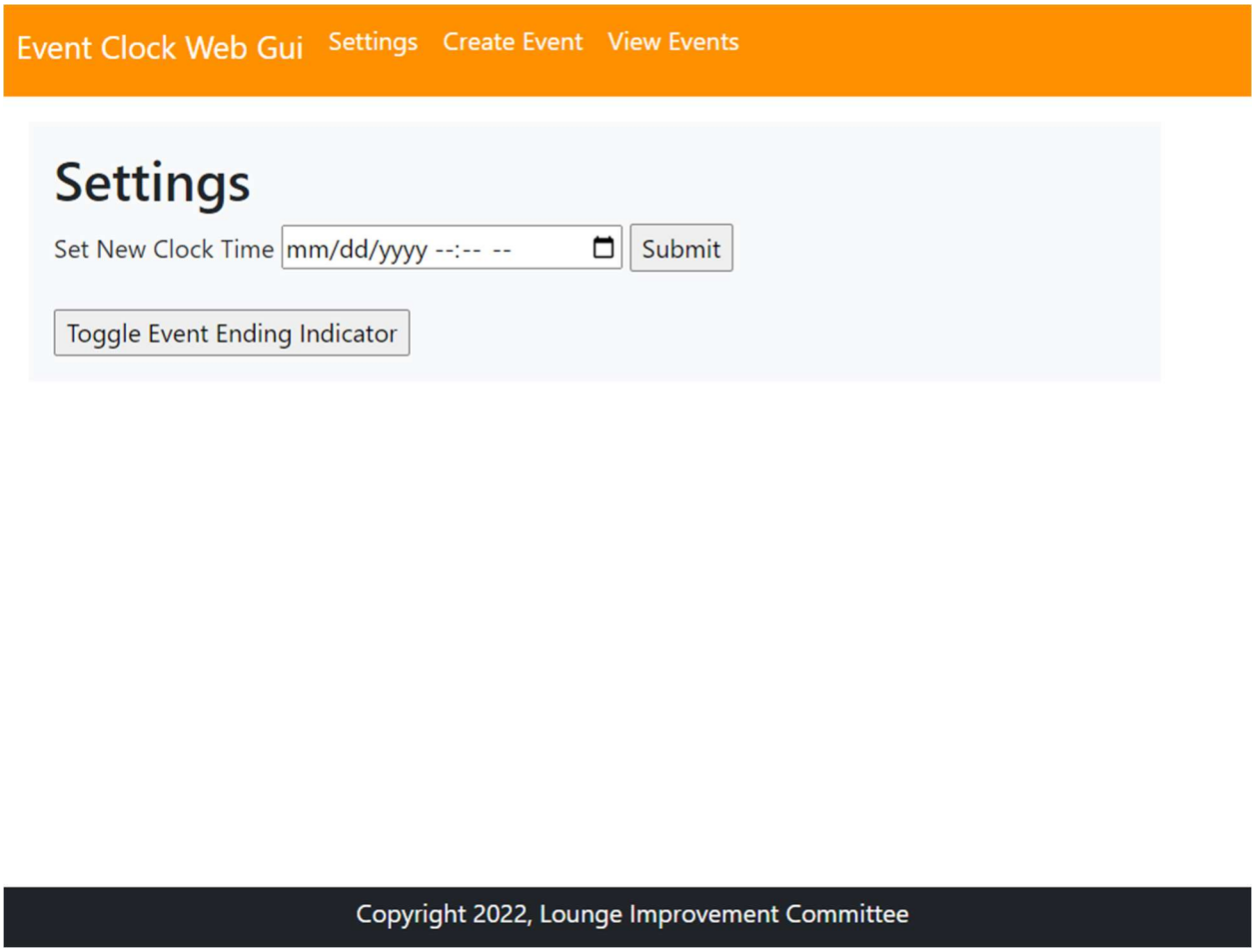


Figure 8: Web Server Settings Interface

Events can be created using the website by selecting the 'Create Event' option on the taskbar at the top of the webpage. Selecting it will take users to the page shown on Figure 9, which allows for the creation of events with a name, defined start time, defined end time, and a number of weeks to repeat the event. The creation of events and the changing of settings are both handled by HTML forms using the POST method. These forms are received by phew upon the selecting of the submission button and then stored for parsing.

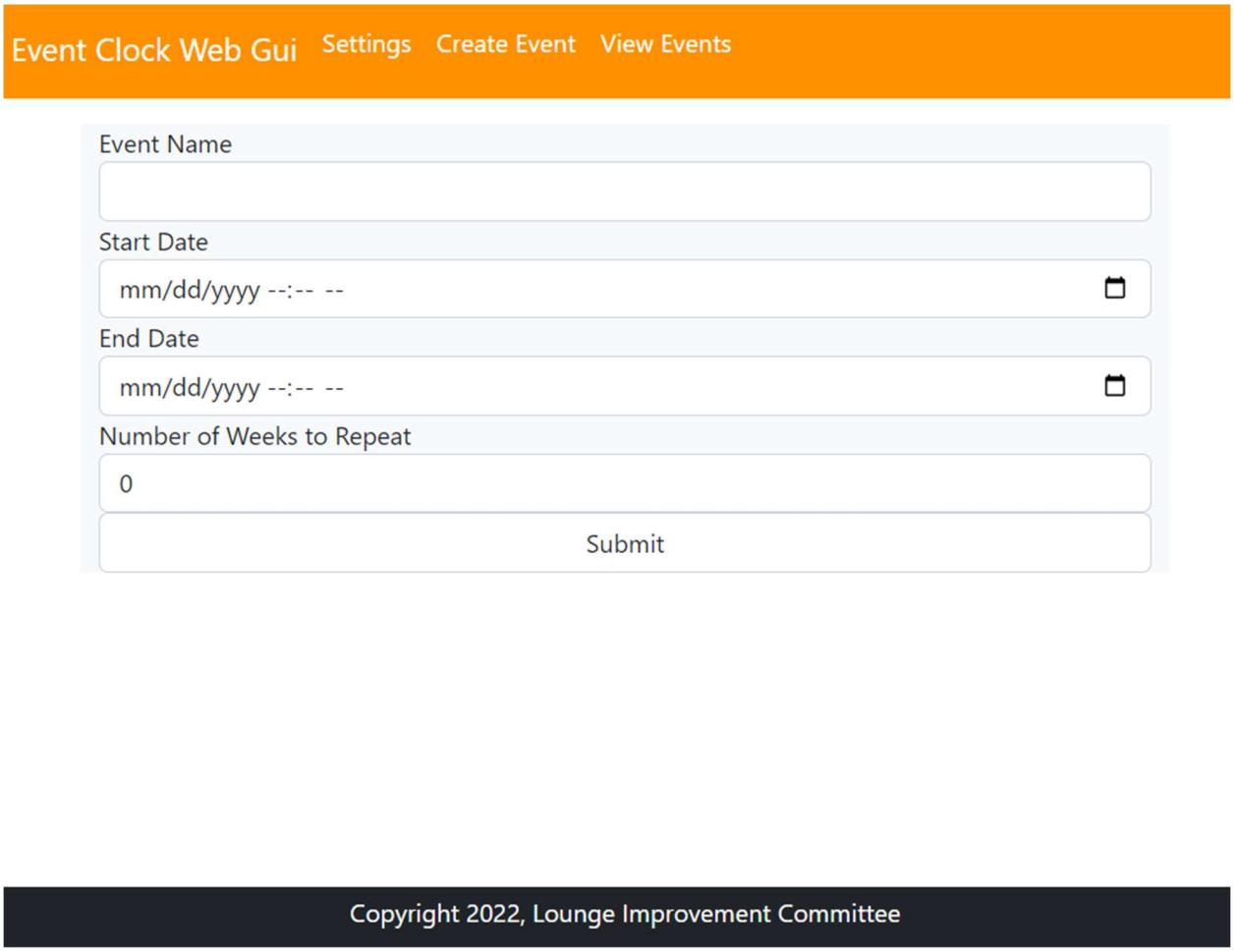


Figure 9: Web Server Event Creation Interface

When an Event Creation form is received by phew upon submission, the values submitted are parsed and stored as an event object. Parsing constitutes the converting of the start date and end date from datetime-local formatted strings to epoch time integers using the time.mktime() MicroPython function [27][28]. Event names are stored as strings, while the number of weeks to repeat is stored as an integer. The event object constructed with these values is then saved locally in a list for use when viewing events. Once saved to the list of events, the event is then sent to the receiver Raspberry Pi Pico over UART. In the case of a settings form being sent, the value of the new RTC time is parsed using the same time.mktime() command, then all updated settings are sent over the UART.

If a user wishes to see a list of currently created events, they can do so using the 'View Events' button on the taskbar at the top of the webpage. This will generate a page using phew and the event list created from submitted events. An example page with placeholder events is shown in Figure 10.

Figure 10: Web Server Example 'View Events' Page

Communication Protocol

To communicate between the Raspberry Pi Pico W running the server component of the clock and the Event Loop Raspberry Pi Pico, hardware UARTs running through GPIO pins 4 and 5 are used on both devices [19]. The UART on the server Pi Pico W is initialized alongside the web server, and is first used to send the IP address assigned by the Wi-Fi network to the UART receiver on the Event Loop Pi Pico for display on the LED matrix.

When a settings HTML form is sent to the server, it is parsed for UART sending by appending a series of characters that are recognized as settings commands by the receiving UART handler to the front of the provided strings. The values are then sent over UART as strings, using a helper class.

When an event creation HTML form is sent to the server, an event object is first created containing all relevant event information. The event is then sent over UART as a string using its toString() method, which includes each field of the event separated by a pattern of characters understood to be separators by the receiver. To ensure that these characters are not used in the name of an event, uncommon characters that cannot be printed by the LED matrix are used. An example debug message for the sending of events over UART is shown in Figure 11.

```
sending message: TEST EVENT 2*-*1670835600*-*1670844600*-*0
```

Figure 11: Example Debug UART Communication Message

Event Processing Loop and LED Driver

A second Raspberry Pi Pico is responsible for the Event Processing Loop of the event clock, as well as the LED Driver. The LED Driver is responsible for writing events to the LED matrix display and the 7-segment display, while the receiver is responsible for receiving information over UART, parsing it, and storing the information. The Receiver and LED Driver are both controlled by an event processing loop, which runs continuously on the Pi Pico and is responsible for all core functions of operation.

On startup, the second Pi Pico first initializes the Real Time Clock (RTC) on GPIO pins 0 and 1, and UART communication on pins 4 and 5. It then prints a diagnostic message using the LED matrix display. Once the diagnostic message has finished printing, the second Pico will check the UART for the IP address provided by the web server Pi Pico W, and will print the address to the LED matrix display, allowing for the user to know how to access the server. Once this has finished printing, the Pico will initialize the 7-segment display with the current time from the RTC, and enter the event processing loop, where it will remain indefinitely.

Event Processing Loop

The event processing loop on the Pi Pico has 4 main stages: receiving from UART, receiving upcoming events, printing current events, and removing finished events. A state machine depiction of the event processing loop can be seen in Figure 12.

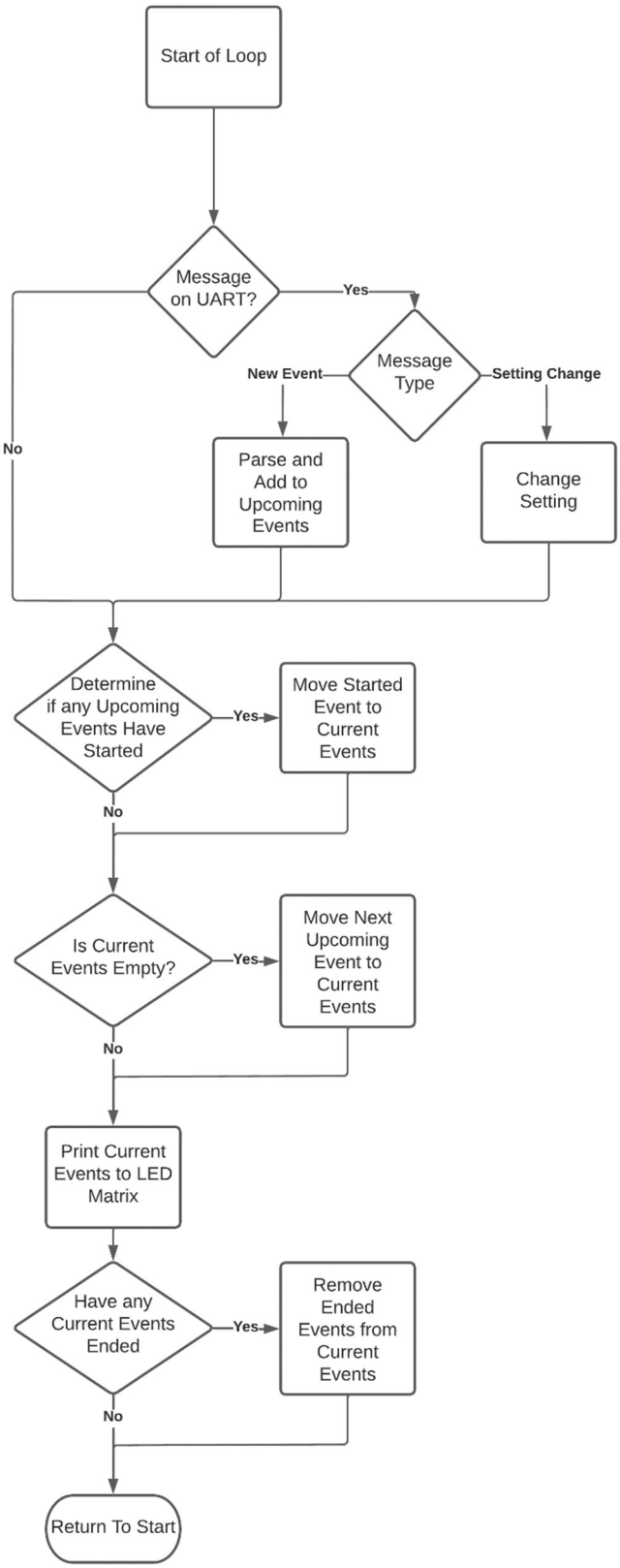


Figure 12: Event Processing Loop

The event loop begins by checking the Hardware UART for any received messages. If none are received, it continues to the next stage. If a message has been received, it is decoded and parsed to determine its type. Setting messages are indicated by a string at the beginning of the message, and are handled according to the setting changed. If the ending indicator setting is read, the flag that determines if the event Ending LEDs will be toggled, becoming true if currently false and vice versa. If the real time clock (RTC) time set setting is read, then the RTC will be set to the value sent in the message. Event messages are indicated by the absence of a setting message string, and are first split into components using a common separator. The event is then added to a list of upcoming events, with repeated events being added as separate events in the list that have had their start and end times pushed back. Once messages have been read and parsed, the event loop continues to the next stage.

The second stage of the event loop is responsible for parsing upcoming events. Each event in the list of current events has its start time compared to the current time of the RTC, with each value stored as an epoch time integer. If the value of the real time clock is greater than that of the start time for the event, then the event is moved to from the upcoming events list to the current events list, as it means the event's start time has been reached. Once all events have been checked and all started events have been moved, the event loop checks the size of the current events list. If the current events list is empty, then the upcoming events list is sorted by start times and the event with the next closest start time is moved to the list of current events. This results in the next upcoming event being printed in the event that no events are currently occurring. Once this check has occurred, the event loop moves onto the next stage.

The third stage of the event loop prints current events on the LED matrix. Each event in the list of current events is printed to the LED separately, fully completing before the next in the list begins to print. If an event will end in the next 5 minutes according to the current time on the RTC and if enabled, an indicator will be printed alongside the event, as detailed in the section on the LED firmware. Once each event has been printed, the event loop moves to the final stage.

The final stage of the event loop is similar to the second, but is responsible for parsing current events rather than upcoming events. Each current event has its end time compared to the current time of the RTC. If the value of the RTC is greater than that of the end time, then it means the event has ended, and the event is removed from the list of current events. Once all current events have been checked, the event loop restarts from the beginning, moving back to the first stage.

Between each stage of the event loop, and in between each current event being printed on the LED matrix, the time displayed on the 7-segment display is checked against the RTC. If a new minute has started according to the RTC, the 7-segment display is updated accordingly. This ensures that the time on the 7-segment display remains accurate.

LED Firmware

The LED Firmware is used to drive both the LED matrix display and the 7-segment display. Each has its own driver implementation that allows the main program to interact with those components. The LEDs in both classes are addressed in a sequential manner with integers. Brightness and color can also be set in both classes. The signal generator for both classes uses modified example code provided by the Raspberry Pi foundation for using the PIO state machines to generate the appropriate signal for the WS2812B LEDs [3].

The 7-segment display is made up of sequential LED strips arranged in a fashion that allows the entire assembly with the 3D print to function as a 7-segment display. The 7-segment class takes in color input and initializes the LEDs with the appropriate signal generating code. The 7-segment class has a function that can be called which allows the user to display any time on the clock. The input for the function takes in two integers, one representing hour and one representing minutes, and converts them to internal variables that can use the specific hour driver for the 7-segment clock and minute driver for the 7-segment clock. The internal minute driver converts the digits to a 2-element list, where a zero is added if the minute is less than 10. This ensures that both minute segments are always properly lit in the manner expected for a 7-segment clock. The hour driver also converts the integer to a 2-element list but it does prepend a zero when displaying hours less than 10. The driver then calls individual functions that address the appropriate LED segments to represent the respective digits. An example of the 7-segment display can be seen in Figure 13, and the assembly of the display is seen in Figure 14.

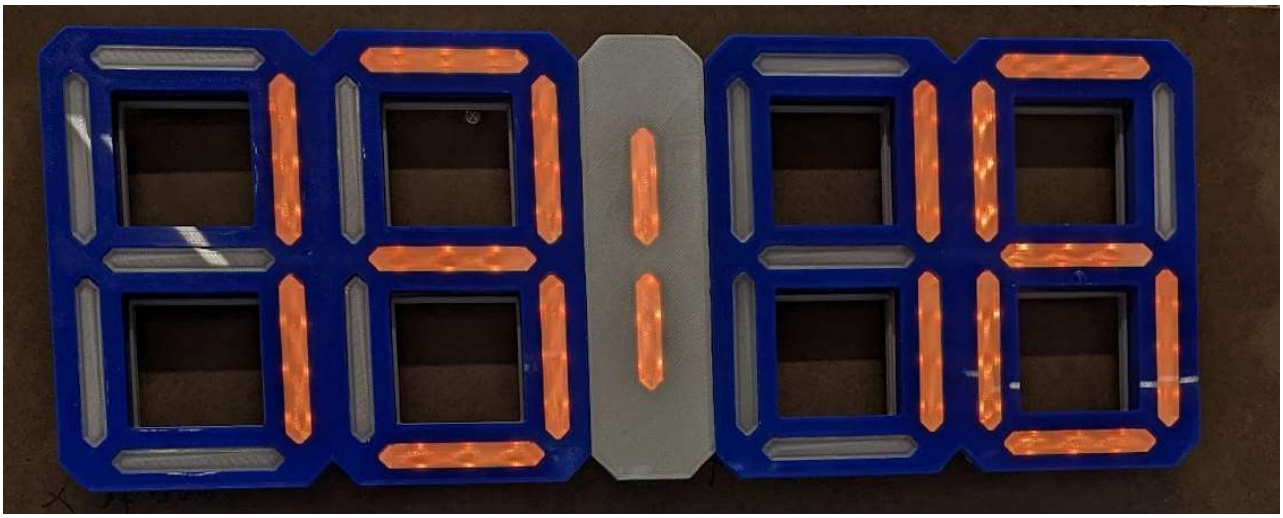


Figure 13: Example 7-Segment Display

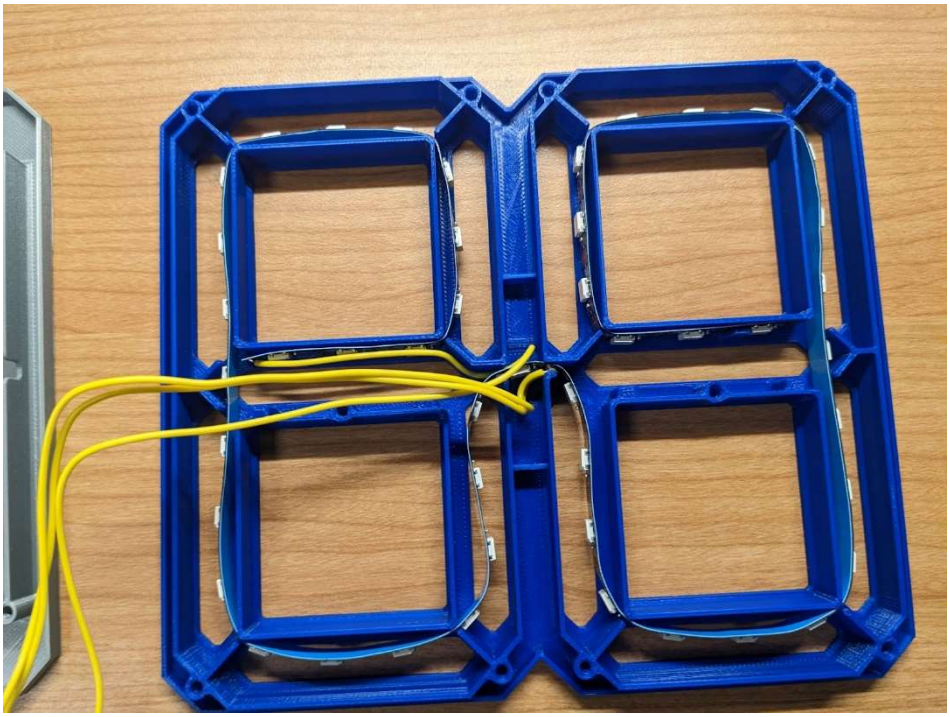


Figure 14: 7-Segment Display Assembly

The matrix display uses the same LED driver class to initialize the LEDs, then uses a 2D integer array to represent the physical location of each LED in the matrix panel. The 2D array allows the core functionality of allowing scrolling text, using a number of MicroPython mathematics functions. The abstracted 2D array allows for the addressing of each individual LED of the 512 in the matrix panel. Each character is represented by a function that addresses the appropriate LEDs to represent the character using 7x5 pixels. Each character function also has a horizontal translation integer that sets its position with respect to the matrix panel and another offset integer that shifts the character on the display based on its position in the input string. The updated horizontal translation integer continues to move the text across the screen and wrap around the edges so that the entire string can be printed. The LEDs hold the last value sent to them, forcing them to be reset with each new value sent. Thus, the printer function clears the entire screen by sending in a blank signal to all of the LEDs before printing the next frame of the display. Due to the wrap around design using the 2D array, there is no hard limit on the length of the string that can be printed. These functions are wrapped in a writing function, which allows for the scroll speed to be set and an optional indicator to be toggled. An example of the event printing normally is shown in Figure 15, and an example with the indicator on is shown in Figure 16.

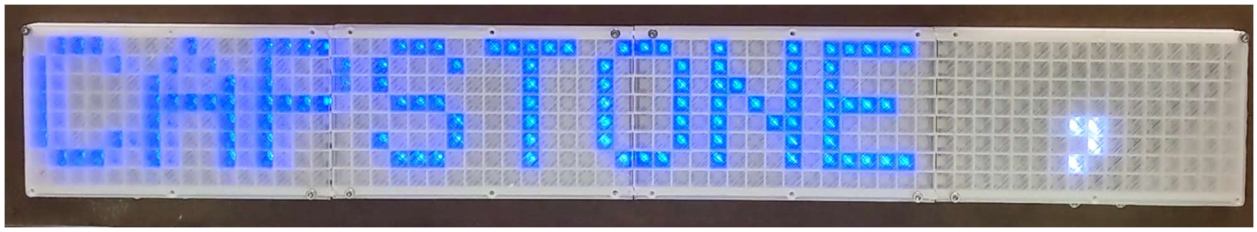


Figure 15: Standard Text Print on Matrix Display

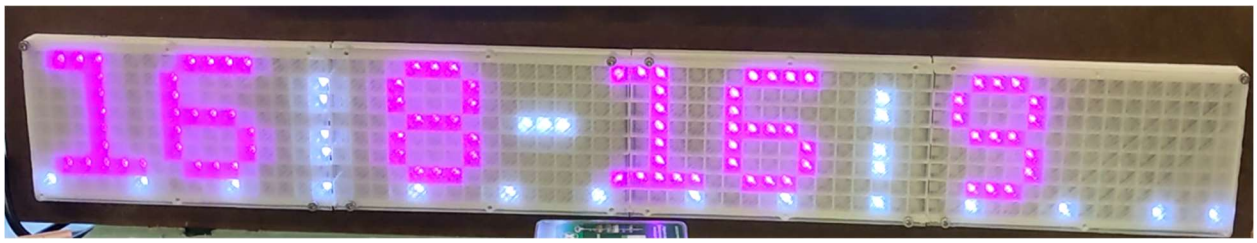


Figure 16: Text Print on Matrix Display with Indicator

RTC Access

The RTC uses standard I2C protocol to communicate with the Raspberry Pi Pico using GPIO 0 and 1. The RTC has a standard format of bytes that it reads as input which allows the user to set the RTC time when programming the first initialization or if the user wants to change the clock's time. The RTC also sends bytes on the I2C bus which the RTC class on the Pico uses to read the current time. The RTC class has a readtime() function that the main file uses to read epoch time. The RTC is frequently called in the main event loop to display the time on the 7-segment display and to keep internal timing for the event handling. A photo of the RTC module in use is shown in Figure 17.

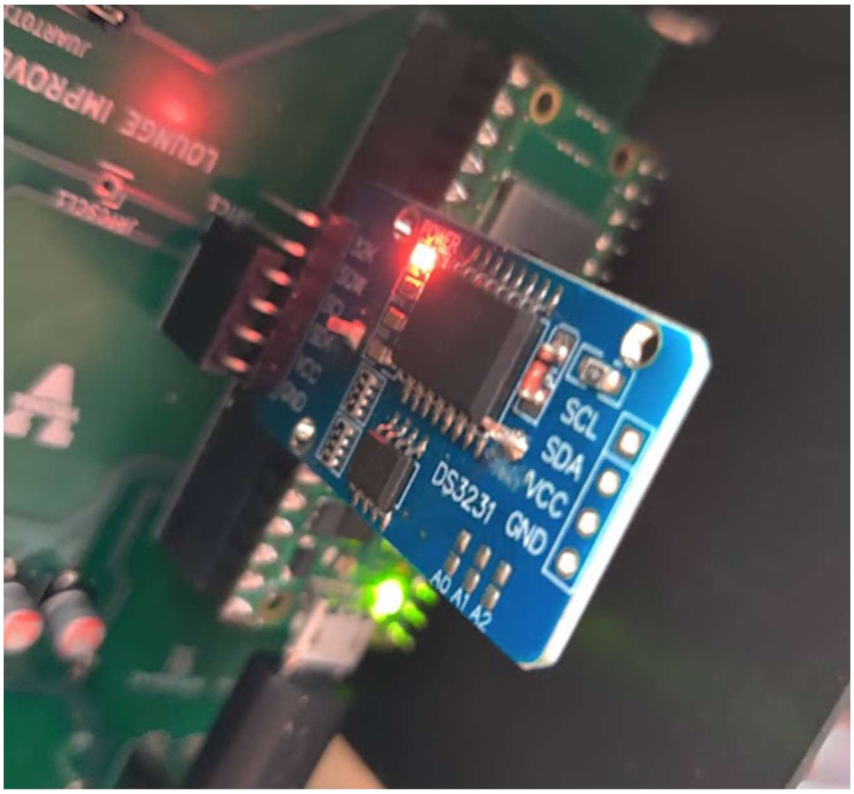


Figure 17: Real Time Clock Module in Use

Assembly

Physical Assembly

Physical assembly of the device used a combination of 3D printed parts and lumber. Parts that required a level of precision were 3D printed, while the frame and wall-mounting hardware used lumber. Lumber was chosen for its ease of modification and its strength, while 3D printing was chosen for its flexibility should an error in design occur. Examples of the fully assembled clock are shown in Figures 18 and 19.

Figure 18: Final Clock Assembly Rear View

Figure 19: Final Clock Assembly Front View

CAD and 3D Print Design

A combination of custom parts and existing creative commons designs were used in the creation of 3D models for the project. Custom 3D models were designed using Autodesk Fusion 360 and sliced using Ultimaker Cura. The 3D models were designed with simple orthogonal shapes with modifications made for M3 standard screw holes for mounting. Examples of a fully custom 3D print, the frames for the LED Matrix displays, are shown in Figures 20 and 21.

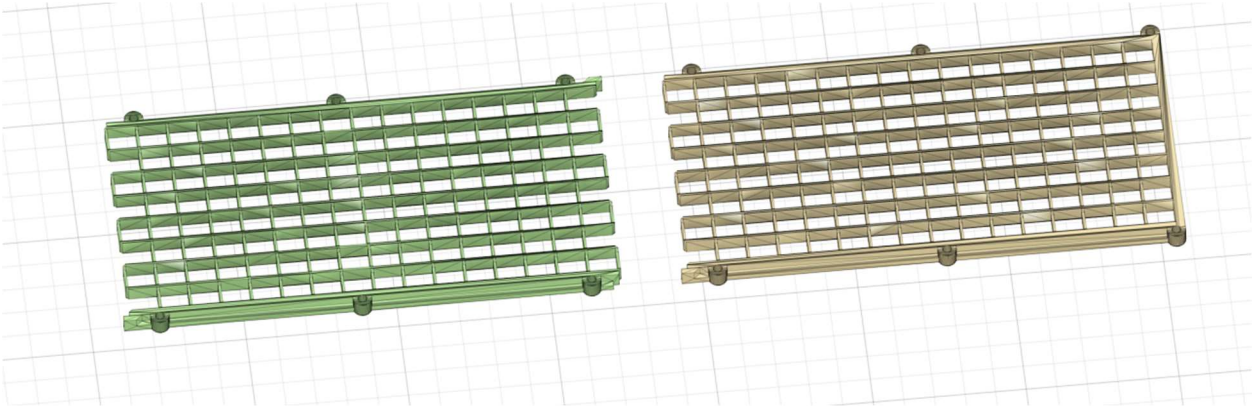


Figure 20: Diffusion Grid for LED Matrix Display

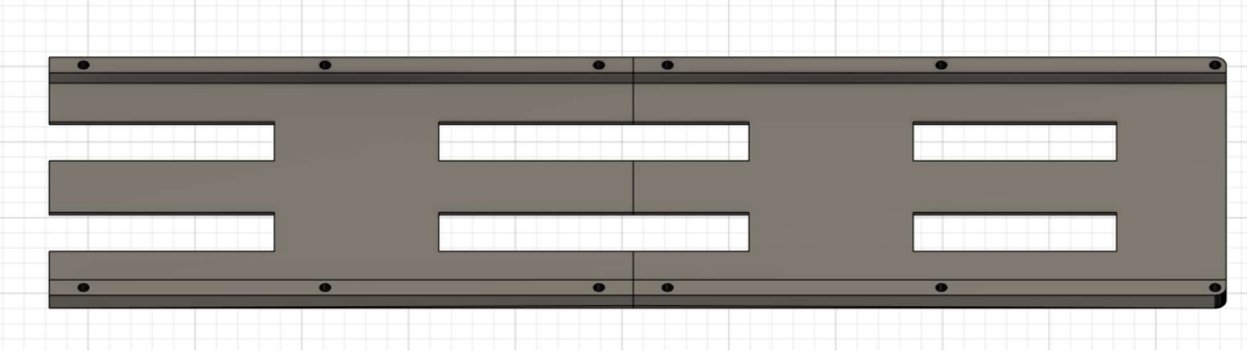


Figure 21: Mounting Frame for LED Matrix Display

The design for the 7-segment display was based on an existing creative commons design, using a standard WS2812B LED strip assembled to allow for individual segments to be lit. A 3D model of one of the segments is shown in Figure 22. The 3D printer used for all prints was the Ultimaker S3, chosen for its high print speed. A variety of colors were used for the prints, and clear PLA plastic was used for the printing of diffusers to increase the visibility of the LED Matrix and 7-segment displays.

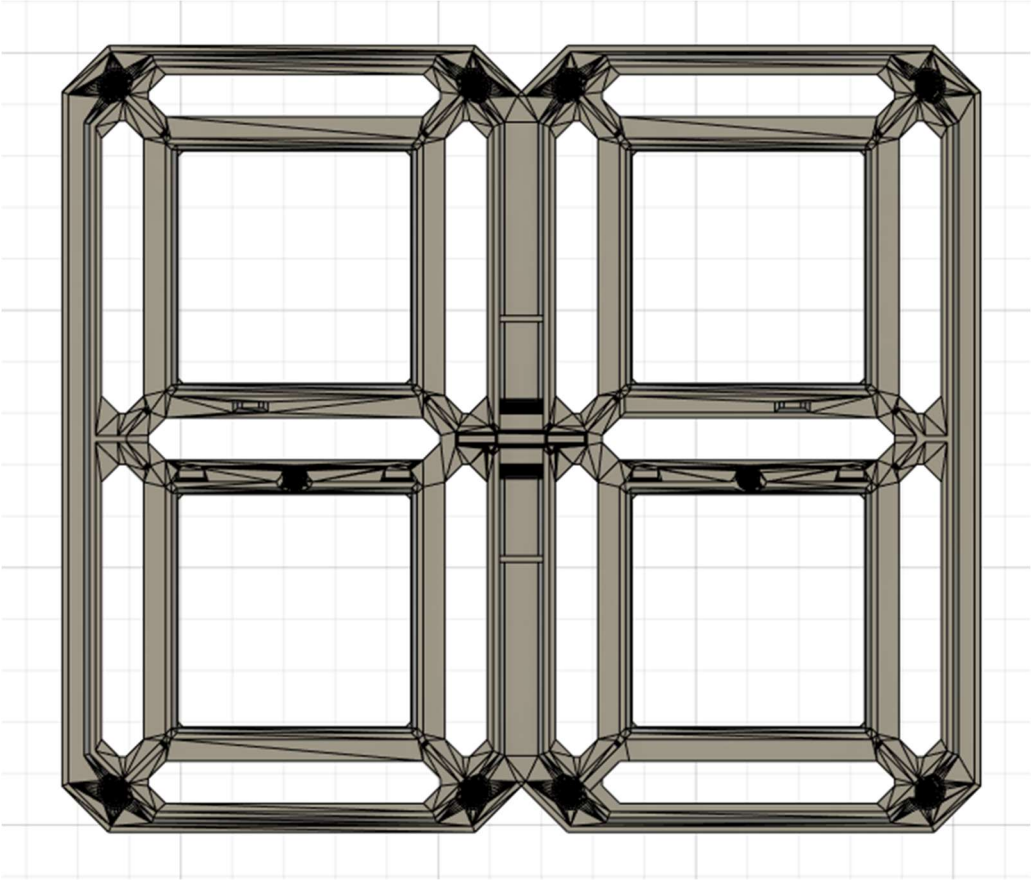


Figure 22: 7-Segment Display Component

Project Time Line

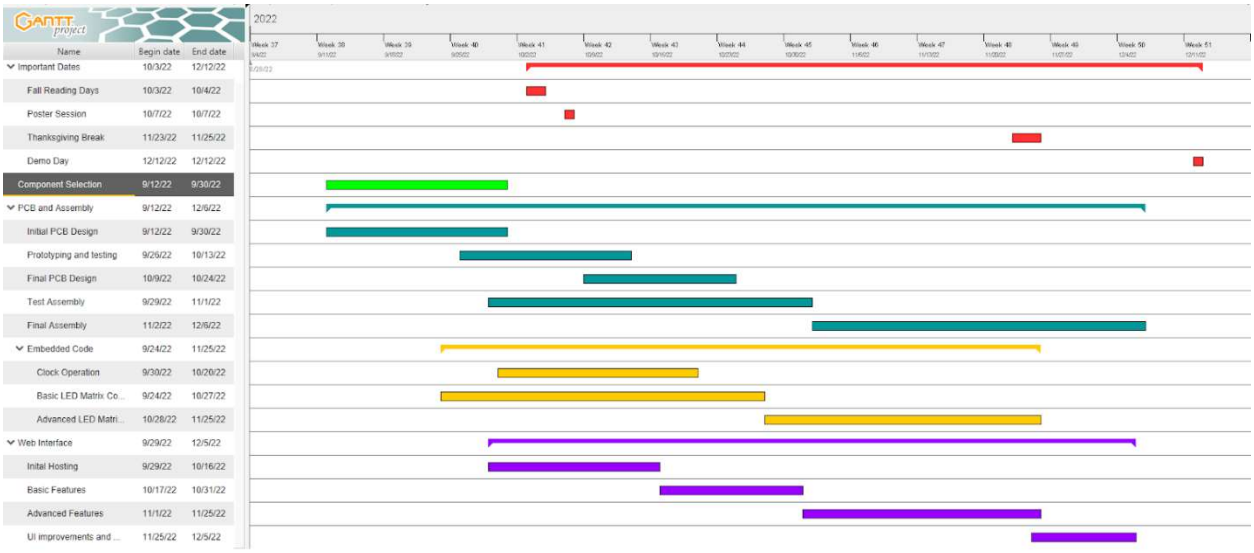


Figure 23: Initial Gantt Chart

Figure 24: Final Gantt Chart

Overall, the timeline for our project changed a medium amount over the course of the semester. The largest changes were to the order of a number of parallel tasks and the timing of tasks related to PCB design and assembly. PCB and assembly related tasks were pushed back across the board, primarily due to delays with PCB manufacturing and the need for major PCB revisions for the final design to accommodate a second Raspberry Pi Pico on the board. The delays in manufacturing pushed back initial PCB design and testing, while the major revisions pushed back final PCB design. As these tasks were rather serialized in regards to final assembly, this resulted in it also being pushed back. As for the order of parallel tasks, this is demonstrated

by the extension for the development of the clock operation, as it was found to be less essential for the testing of other components, and as such was delayed so that basic LED matrix control could be completed, as it was found to be more related to tasks regarding the Web Interface. Aside from additional time dedicated to the implementation of advanced features, the timeline for the development of the web interface remained largely the same. Component selection time was greatly extended due to issues that were discovered with originally selected components, forcing new components to be chosen.

Generally, the tasks in each category of our Gantt chart were well parallelized with the other categories. PCB and Assembly tasks, Embedded Code tasks, and Web Interface tasks were all able to be worked on separately without any major bottlenecks. Within each category, tasks were rather serial, but this was not found to be an issue as each team member was largely responsible for their own category, meaning instances of one team member waiting on another were minimized during the design and creation processes.

Division of Responsibilities

Each member of the team was primarily tasked with the completion of one of the 3 main categories found in the Gantt chart. Neil was tasked with PCB and Assembly, and led the design of the prototype and final PCBs, as well as final product assembly. He also led component selection, as the majority of components were selected based on the design of our PCB. Tahsin was tasked with Embedded Code, and was responsible for writing the LED matrix code and the 7-segment display code. He also assisted Neil with component selection, as the signal speed of the LED matrix dictated appropriate components for powering and buffering the matrix inputs. William was tasked with the Web Interface, and was responsible for creating the web interface and other front-end aspects. He worked with Tahsin to write the code that integrated the web interface with the LED matrix code. Neil's secondary roles involved managing orders and any woodworking required for assembly. Tahsin's secondary roles involved the design and printing of any 3D printed components. William's secondary roles included leading document writing and time management for the overall project.

Test Plan

As the original proposal for the event clock lacked a test plan, a new test plan was drafted during the process of creating the clock.

Our new test plan required dividing the system into a number of submodules, which would each be tested iteratively until deemed fully working. These submodules were as follows: PCB, LED matrix control, 7-segment control, and website hosting. Each submodule was designed to be testable independent of any other modules, allowing for the work to be effectively distributed, and to ensure that issues with one module did not cause bottlenecks with others. Once a module was found to be functional, it was combined with other modules in integration testing, with both modules being modified together should an issue arise with integration testing. This process would continue until each module was integrated and the full system was able to be tested.

The first completed submodule was LED matrix control, which was first tested displaying non-scrolling text using a limited set of characters. Tests were then made with the same limited set of characters on scrolling text. This was then tested with a full set of characters at multiple speeds. No issues were detected when testing the submodule on its own, so it was added to the pool of modules for integration testing.

The next completed submodule was the website hosting, which was first tested as a plaintext website with no interactive features hosted on the Pi Pico W. Interactive features were then added and tested for proper parsing by phew, the selected website framework. When found to be full working, it was added to the pool of modules for integration alongside LED matrix control and tested together. This revealed a number of issues with the website hosting module, as it was found that the processes that allow phew to operate are blocking, and thus the code for LED matrix control could not run. This limitation forced a partial redesign, as it was determined that there was not a reasonable alternative to phew. The partial redesign involved the introduction of a second Pi Pico for the handling of event, 7-segment, and LED matrix control, with connection to the server hosting Pi Pico W through UART. This constituted the creation of a UART communication submodule, which would need to be fully working before the LED matrix and website hosting modules could be tested together. It also triggered a minor redesign of board connections, and thus a new batch of testing.

The UART and 7-segment submodules were completed simultaneously, with the 7-segment module being tested for the display of all possible variations of characters that could be displayed on a 24-hour clock. The UART submodule was first tested with simple strings, then with the strings generated by the website submodule. Once confirmed to work with the website submodule, the LED matrix submodule was added to the chain of devices, with an event being created using the website submodule, sent over the UART submodule, and displayed on the LED matrix submodule. These tests were found to be successful, solving the previous issue and allowing for event loop features to be tested.

As it relied on all submodules other than the PCB, the design and testing of the event loop occurred near the end of the creation of the event clock. It was first tested using hardcoded events, which were parsed by the event loop and found to display properly on the LED matrix display. UART communication with the web server was then added, and tested to ensure behavior was the same as with hardcoded events. Finally, the 7-segment module was added to the loop, to ensure that it would be able to successfully update while the event loop handled UART and the LED matrix.

Concurrently with the testing of all other submodules and the event loop, testing of PCB submodule occurred. In testing the PCB to verify functionality the initial operation was to verify the placement of parts, and do a visual examination of any solder joints. Connectivity between active components as well as between the two Picos was then tested in order to ensure that all traces were properly constructed and connected to relevant portions of the board system. This was done using a traditional multimeter. This then shifted to verifying distribution of power, ensuring that the 5V supply was adequately supplying all relevant components. The final component tested was the passing of signals from the event loop Pico through the level shifter

components of the board. This revealed an issue with the initially purchased shifters, as they were not capable of updating fast enough for needed frequencies. This caused a switch to a second model of level shifter, which was tested and found to be functional at the frequencies needed.

Once fully working, the PCB submodule was integrated with the event loop set of submodules, constituting full system testing. The same tests performed on the event loop without the PCB were tested on the PCB and found to mostly work correctly. The non-working component was found to be the RTC, a component of the event loop. Once fixed, the behavior of the event loop on the PCB was the same as off of it, constituting a fully working design.

Final Results

Overall, the creation of the event clock was successful. The event clock is successfully able to create and display events at scheduled times, and these events are able to be repeated on a weekly basis. Time is able to be kept in the event of power loss, and can be set using the web interface. When no events are currently displaying, the next upcoming event will be displayed instead. When events cannot fit onto the display, the display is able to scroll do display the full event. Events do not flash to indicate when they are about to end, but an indicator is present, with this change having been made due to epilepsy concerns. Of the features described in the proposal, the only one unable to be achieved was the editing of events after creation. The criteria initially set for the event clock is as follows:

- Time is properly displayed on the primary clock
- Time is kept in the event of a disconnection from the network
- Events can be scheduled using the web interface, and display at their scheduled time on the event clock
- Events can be edited or removed after being created
- Events are able to be repeated at future times
- The display is able to scroll in the event that the full information cannot be displayed at a single time and for displaying multiple events.
- If enabled in the web interface, the LED Matrix display of the clock will flash when an event is about to end
- If enabled in the web interface, the server's RTC clock should be able to be set by the user
- In the absence of scheduled events the display panel will be capable of cycling through upcoming events.

Of these 9 features, 8 were fully implemented, with the only feature not fully implemented being 'Events can be edited or removed after being created,' as events are removed upon completion, but cannot be edited. In accordance with the grading rubric shown in Figure 25, the theoretical grade for the event clock as it currently exists should be an A.

Letter Grade	Criteria
A+	All features are implemented
A	7 or more features are implemented
B+	6 or more features are implemented
B	5 or more features are implemented
C	4 or more features are implemented
D	3 or fewer features are implemented

Figure 25: Proposal Grading Rubric

Costs

A breakdown of the total cost of our system, as well as the cost for a mass production order, is included in Appendix A. It includes part name, function, quantity, and unit and total cost. Based on components used in the final design, the total cost to assemble a working event clock was $216.58. This cost does not include the costs of labor, to include soldering, oversight of 3D printing, and assembly.

The cost to produce the system in quantity was estimated from known DigiKey and Mouser mass quantity prices (10,000 units). Using these mass quantities, the calculated cost of the Shared Space Event Clock was calculated to be approximately $174.89, almost $45 less than the single bespoke unit. Production cost can likely be further reduced by streamlining the system to eliminate the need for headers and instead opting for direct surface mount of the Pi Pico, Pico W, and RTC, and a shift to injection molding plastic rather than relying upon 3D printing and wood. Mass production potential is constrained by the Creative Commons License the 7 Segment Display is under, preventing commercialization in the current form. A move to a different, injection molded, 7 segment system would remove this constraint. There were no cost savings in ordering the Pico, RTC, or Matrix Displays in mass quantities, which have a significant impact on unit cost regardless of scale.

Future Work

The use of commodity parts and an open source, well supported microcontroller such as the Raspberry Pi Pico and use of 3D printed objects give us a lot of flexibility in expanding and improving our product design. One of the first lines of improvement for our project would be to create a more polished frame with lightweight design so that it can hang from a standard nail and take the place of a regular clock in a room. The 3D printed enclosure for our PCB could also be improved by creating a lower profile design with the change of header pins and connectors used in the current design to a slimmer design. With the website available to easily expose additional future features, we could add a toggle for 12-hour or 24-hour representation on the display and the clock. Additionally, the originally proposed feature of editing or removing existing events

could potentially be added. A possible change that we can consider would be to allow for selectable scrolling speeds. This is not currently possible with the implementation of MicroPython running on the Pi Pico running the lights, but converting the event loop codebase to C/C++, would improve the performance of our microcontroller and thus allow for faster scrolling speed. The shortcoming of MicroPython is that it is single threaded and is unable to run multi-threaded programs. With the change to C/C++ codebase, we could also add multi core programming that would allow the matrix display and seven segment display to run independently by using different cores on the microcontroller.

References

[1] "Space availability -," *UVA Library Calendar - UVA Library*. [Online]. Available: https://cal.lib.virginia.edu/allspaces. [Accessed: 22-Sep-2022].

[2] "Features - Glance Clock," *Glance Clock Shop*. [Online]. Available: https://glanceclock.com/pages/features. [Accessed: 22-Sep-2022].

[3] Raspberry Pi Foundation, "RP2040 Datasheet, A microcontroller by Raspberry Pi," Pi Pico W datasheet, Jan. 2021 [Revised Jun. 2022].

[4] Pimoroni, "Phew! the Pico (or Python) HTTP Endpoint Wrangler," Sep-2022. [Online]. Available: https://github.com/pimoroni/phew#basic-example.

[5] "DigiKey electronics – electronic components distributor," DigiKey Electronics –Electronic Components Distributor. https://www.digikey.com/.

[6] "Electronic Components Distributer – Mouser Electronics," Mouser Electronics, Inc. https://www.mouser.com/

[7] Thonny, "Thonny, Python IDE for beginners," 11-Sep-2022. [Online]. Available: https://thonny.org/.

[8] M. Otto, "Introduction to Bootstrap 4," Bootstrap 4 Documentation, 2019. https://getbootstrap.com/docs/4.0/getting-started/introduction/

[9] "Getting started with Fusion 360," Fusion 360 Help. [Online]. Available: https://help.autodesk.com/view/fusion360/ENU/.

[10] "Ultimaker Cura: Powerful, easy-to-use 3D printing software," Ultimaker. [Online]. Available: https://ultimaker.com/software/ultimaker-cura.

[11] "Documentation," KiCad. [Online]. Available: https://docs.kicad.org/.

[12] F. C. McMichael and C. Henderson, "Recycling batteries," in IEEE Spectrum, vol. 35, no. 2, pp. 35-42, Feb. 1998, doi: 10.1109/6.648673.

[13] Chatterjee, A., Abraham, J. Efficient management of e-wastes. *Int. J. Environ. Sci. Technol.* 14, 211–222 (2017). https://doi.org/10.1007/s13762-016-1072-6

[14] National Electrical Manufacturers Association, “NEMA Enclosure Types,” NEMA 250, [Revised Nov. 2021].

[15] Infineon Technologies, “Single-Chip IEEE 802.11 b/g/n MAC/Baseband/Radio with Integrated Bluetooth 5.2 Compliance,” Document No. 002-30348, [Revised Mar. 2021].

[16] B. P. Crow, I. Widjaja, J. G. Kim and P. T. Sakai, "IEEE 802.11 Wireless Local Area Networks," in IEEE Communications Magazine, vol. 35, no. 9, pp. 116-126, Sept. 1997, doi: 10.1109/35.620533.

[17] Federal Communications Commission, “Raspberry Pi Trading Ltd – PICOW,” Jun 2022

[18] MDN Contributors, “HTTP,” MDN Web Documents, Aug. 03, 2019. https://developer.mozilla.org/en-US/docs/Web/HTTP

[19] “I2C Info – I2C Bus, Interface and Protocol,” I2C Info – I2C Bus, Interface and Protocol. http://i2c.info/

[20] IPC International, “Generic Standard on Printed Board Design,” IPC-2221A, May 2003 [Revised Jun. 2010].

[21] WorldSemi, “WS2812B Intelligent control LED integrated light source,” WS2812B datasheet, May 2013.

[22] “USB standards: USB 1, USB 2, USB 3, USB 4 - capabilities & comparisons,” Electronics Notes. https://www.electronics-notes.com/articles/connectivity/usbuniversal-serial-bus/standards.php

[23] Display method and system for wireless intelligent multi-screen display, by C. Yu. (2017, Oct 27). Patent US20200257487A1 [Online]. Available: https://patents.google.com/patent/US20200257487A1/

[24] Synchronized display of screen content on networked devices, by M. Fryman. (2020, Apr 14). Patent US11029911B2 [Online]. Available: https://patents.google.com/patent/US11029911B2/

[25] Programmable lighting device and method and system for programming lighting device, by B. Lai. (2017, Jan 6). Patent US9854651B2 [Online]. Available: https://patents.google.com/patent/US9854651B2/

[26] Cikic, D. “Retro 7 Segment Clock,” Thingiverse, Jul. 22, 2018. https://www.thingiverse.com/thing:3014572

[27] “<input type="datetime-local">,” MDN Web Documents. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/datetime-local

[28] “time – time related functions,” MicroPython Documentation v1.19.1. https://docs.micropython.org/en/v1.19.1/library/time.html

Appendix

In this section you should include helpful information that does not fit into the above *categories* but will be helpful in understanding and assessing your work.

Appendix A, Costs

Item	Function	Unit Cost	Units Used	Total Cost	Unit Cost /10,000	Total Cost /10,000
865090640001	Blocking Capacitor	$0.18	8	$1.44	$0.10	$0.80
	RTC	$10	1	$10		$10
	Panel	$21	2	$42		$42
	LED Strip for 7 segment	$30		$30	$10	$10
0ZCJ0200FF2C	Matrix Fuse	$0.42	1	$0.42	$0.162	$0.162
1206L050YR	7 Segment Fuse	$0.49	1	$0.49	$0.19798	$0.19798
74LVC1T45W6-7	Level Shifters	$0.299	2	$0.598	$0.059	$0.118
SD0603S040S0R2	Diode	$0.39	2	$0.78	$0.08186	$0.16372
SSW-120-01-G-S-LL	20 Position Female Header	$5.71	4	$22.84	$3.30977	$16.54885
D01-9922046	20 Position Male Header	$4.66	4	$18.64	$2.83967	$11.35868
	Raspberry Pi Pico W	$5.00	1	$5.00	$5.00	$5.00
	Raspberry Pi Pico	$4.00	1	$4.00	$4.00	$4.00
PJ-080BH	DC Jack	$2.38	1	$2.38	$1.17136	$1.17136
1755749	3 Position Female Connector	$1.24	2	$2.48	$0.72200	$1.444
1784778	3 Position Male	$3.70	2	$7.40	$2.03500	$4.07

	Connector					
1755736	2 Position Female Connector	$0.77	2	$1.54	$0.44840	$0.8968
1757019	2 Position Male Connector	$2.51	2	$5.02	$1.38875	$2.7775
RS1-04-G	4 Position Female Header	$0.31	1	$0.31	$0.11878	$0.11878
SX1100-B	2 Pin shunts	$0.06	4	$0.24	$0.02878	$0.05756
	2 Pin Header		4	$0.50		$0.50
	3D Filament		1	$25		$25
	PCB		1	$33		$33
	2x4 x24 softwood		1	$2.50		$2.50
			Total:	$216.58		$174.89