ECE 299

Group 8 Final Project Report

Daryon Ambasse – V01034226

Rowyn Davis - V01038276

Aug 2, 2025

Executive Summary

This report outlines the ECE 299 final design project completed by students Daryon Ambasse and Rowyn Davis from May to August 2025. The course introduces students to the engineering design process through a term-long hands-on project, emphasizing planning, prototyping, testing, and documentation.

The project objective was to develop a functional clock radio system using a Raspberry Pi Pico W, programmed in Micropython. All time and radio data had to be displayed using dual SPI OLED screens, and a 2-layer custom PCB was required. The final product included a digital clock, alarm system, FM tuner, volume control, and world time zone viewer.

Key hardware components:

- Dual 128×64 OLEDs acting as one 256×64 display via SPI.
- FM radio receiver module (I^2C-controlled).
- Alarm and snooze system with persistent flash storage.
- Amplifier and speaker system with volume potentiometer.
- 4 tactile buttons for user interaction across 5 modes.
- Custom 2-layer PCB with 0.6 mm traces for higher current paths.
- SolidWorks-designed 3D-printed enclosure with all I/O on one edge.

Software features included five interactive modes: clock display, radio interface, alarm setting, system info (12/24hr toggle), and time zone switching. The system used edge-detection for button presses, persistent JSON-based settings storage, and clean display updates in real time.

Throughout development, students learned soldering, PCB design, safe tool use, audio system optimization, and team-based project execution. Challenges such as broken displays, pin damage, and enclosure misfits were resolved through iterative testing and repair.

The project was delivered on time, under a $63.26 budget, and met all technical and functional requirements. It demonstrates practical application of embedded systems design, teamwork, and engineering professionalism aligned with EGBC standards.

Table of Contents

List of Tables

List of Figures

Project Goals

The project goals that were agreed upon include:

1. Implement battery power using AA batteries to power the operation of the clock-radio system.
2. Enable user-adjustable clock and alarm settings, including the option to display alternate time zones.
3. Incorporate a snooze feature.
4. Provide intuitive FM radio channel selection through accessible user input.
5. Clear audio playback from a compact onboard speaker.
6. Display all relevant system information—including time, alarm status, FM channel, snooze, and alternate time zones—across two OLED screens using SPI communication.
7. Have an animal themed aesthetic by integrating a stuffed animal as the exterior.

8. Develop a structural “skeleton enclosure” using SolidWorks to support internal components and maintain structural stability.
9. Design a compact (<30 cm^2) 2-layer PCB in KiCad that integrates both the Raspberry Pi Pico W and the audio amplifier circuit.
10. Utilize a maximum of four buttons and one potentiometer to manage all user interface settings and volume control.

List of constraints

The constraints that were set out by the students and Prof. Ilamparithi Thirumarai Chelvan are as follows [2]:

1. Design and assemble the final clock/radio by July 29, 2025.
2. Only 1 Raspberry Pi Pico W is to be used as the microcontroller for all handling. Must be programmed with Micropython.
3. The clock displays radio and clock information. Communicates with the Raspberry Pi through SPI protocol.
4. The radio communicates with the Raspberry Pi through I2C protocol.
5. Design and manufacture an animal themed enclosure.
6. Team members must be from the same lab section.
7. Submit the final report by August 2, 2025.
8. This project will cost the students money.

Project Expectations

The expectations set out for this project [2]:

1. **Project Expectations**:
    a. Design and order a 2-layered PCB with a maximum footprint of 30cm^2.
    b. Design and assemble an enclosure using SolidWorks.
    c. Solder all the components to the ordered PCB.
    d. Present the clock/radio in an animal themed/enclosure.
    e. Display the finished project on July 29, 2025.
2. **Clock/Alarm Expectations**:
    a. Be able to set clock/alarm time through user interface, either hardware (i.e., buttons) or software (i.e., a website).
    b. Display time in 12- or 24- hour format.
    c. Have the capability of displaying different time zones through hardware or software user interface.
    d. Be able enable/disable the alarm through user interface.
    e. Be able to set a snooze button for the alarm.
3. **Radio Expectations**:
    a. Be able to tune to local FM radio channels through a user interface.

b. Play audio output through a speaker.
c. Be able to control the volume through a user interface.
d. Display the radio channel information to the user.

Design Choices

1. **Project Design Choices**:
    a. **2-layer PCB**: The team chose to use a 2-layer PCB in KiCad with the dimensions of 5.45cm x 5.49cm (29.92cm²). Using a 2-layer design allowed us to solder components on both sides of the board, effectively doubling the usable surface area. This improved the design flexibility and helped support components being placed closer together. Using a 4-layered board would increase the advantages of the 2 layered board by allowing more dense traces to save more room, however, according to Texas Instruments “PCB Design Guidelines For Reduced EMI”, 2-layer boards can achieve 95% of the effectiveness of 4-layer boards so long as [3];
        - Ground is routed underneath power.
        - Grid power and ground but be careful to not create unneeded common impedance connections.
        - Length-to-width ration should not exceed 3:1 for any traces between IC and ground voltage.
        - Under the microcomputer (i.e., the Raspberry Pi Pico W), build a solid ground plane to limit the noise. (this couldn’t be done without compromising the <30cm² constraint).
        - Route returns for direct connections to the processor I/O directly under the signal trace.
    b. **SolidWorks Enclosure Design**: The enclosure was designed in SolidWorks to fit the electrical components (Raspberry Pi, PCB, Batteries… etc.) as well as mount the buttons and OLED screens. When designing the enclosure following the 3D-printing design rules set by HYDRA RESEARCH [4]. These include but are not limited to:
        - Not having horizontal bridging no longer than 10mm without vertical supports,
        - R > 4mm for base corner rounding,
    c. **Component Soldering**: All through-hole components are soldered by hand using a temperature-controlled soldering iron, following the Kester Soldering process [5]:
        - Apply heat to through-hole & component pin.
        - Apply solder to through-hole & component pin.
        - Remove solder first after a small drop has melted into the connection and then remove heat last.
    d. **Present the clock/radio in an animal themed/enclosure**: According the set ECE299 project constraints [2], the clock/radio enclosure needs to be in an animal shape.
    e. **Display the finished project on July 29, 2025**: According the set ECE299 project constraints [2], presentation of the finished clock/radio to the professor and lab TA will

be conducted during the last week of labs (July 29, 2025, for Rowyn Davis and Daryon Ambasse).

2. **Clock/Alarm Design Choices**:
    a. **Be able to set clock/alarm time through user interface**: Using the maximum allowed number of buttons, the team created an intuitive UI that allowed users to change the time for either the clock or alarm by having using the following buttons as specified:
        - Button 1: Button 1 allowed for the change of "modes". These modes included set time and set alarm. In each mode, the use of the following buttons changed.
        - Button 2: Button 2 allowed for the change of either hour or minutes in the Set Time mode or enable/disable alarm function in the Set Alarm mode.
        - Button 3: Button 3 added +1 to either the hour or minute in the Set Time mode and +1 hour in the Set Alarm mode.
        - Button 4: Button 4 subtracted -1 from either the hour of minute in the Set Time mode and +1 minute to the Set Alarm mode.

    b. **Display time in 12- or 24- hour format**: By cycling to "mode 3" titled "Info" the user can activate button 4 to switch between 12 to 24-hour formatting. The process works in the following way:
        - Program reads the current set time and applies a modulo factor of twelve to the current hour.
        - Check if the remainder is zero; if so, replace it with 12 (i.e., midnight or noon is displayed as "12" instead of "0").
        - Determine whether to display "AM" or "PM" by checking if the original hour is less than 12 (AM) or 12 or greater (PM).
        - Display the formatted time with an "AM" or "PM" suffix if in 12-hour mode, or without suffix if in 24-hour mode.
        - Save the user's display format preference (12hr or 24hr) in settings. Json so the clock remembers it after reset or power loss.
        - The setting toggles every time the select button is pressed while in mode 3, updating both the display and the saved file.

    c. **Displaying different time zones:** By cycling to "mode 5", titled "Time Zone", the user can activate button 4 (select) to cycle through every global time zone from UTC−12 to UTC+14 (including half-hour and 45-minute offsets). The process works in the following way:
        - A predefined list of tuples stores all 39 time zones in the format ("UTC±X", offset_in_minutes).
        - When the user presses the select button in mode 5, a counter (tz_index) is incremented and wrapped around when it exceeds the list length.
        - The current local time (set manually in time_set mode) is retrieved and converted to a UNIX timestamp.

- The program adds the selected offset (in minutes × 60 seconds) to the timestamp to simulate that time zone.
- The updated timestamp is passed to local_time () to format it into a readable time structure.
- The formatted time (respecting 12/24-hour display setting) is shown on the dual OLED screen along with the label (e.g., “UTC+5:30”).
- The selected time zone does not change the actual internal clock — it is purely for viewing other world clocks.
- When the device resets, it starts at the default time zone (usually UTC+0), unless extended to save the last used tz_index.

d. **Be able to enable/disable the alarm through user interface:** When the alarm is set using “mode 2” and activated, it will go off at the designated time and can be disabled or snoozed by user input. The process works in the following way:

- The user enters mode 2 and uses buttons 1 and 2 to set the desired alarm hour and minute values.
- Button 4 toggles the alarm between ON and OFF states. A visual confirmation is shown on the screen (e.g., “Alarm On”).
- The alarm settings are saved to a persistent file (settings. Json) so they remain after power loss or reset.
- In the main loop, the current time is compared with the stored alarm time once per second.
- When the time matches and the alarm is ON, alarm_active is set to True, and a visual notification (flashing display and message) begins.
- If the user presses button 4 (select) while alarm_active is True and the system is in **mode 0 (default time mode)**, the alarm is deactivated for that session (i.e., alarm_active = False).
- Alternatively, a snooze function may delay the alarm by 5–10 minutes (if implemented with button 3, for example).
- The alarm will not trigger again until the time is set forward again, or the device is restarted (depending on snooze implementation).

e. Be able to set a snooze button for the alarm: While the alarm was on, users could press button 2 to start the snooze duration. How long the snooze lasted for could be set by the user by going to the Snooze mode (using button 1) and selecting anywhere from 0-30 mins.

3. Radio Design Choices:

a. Be able to tune to local FM radio channels through a user interface: Modifying the provided code from Exp. 3 [6], the team managed to connect and select specific channels.

- Using buttons 3 & 4, users could go up (+0.2MHz) or down (-0.2MHz) in frequency.

- The radio frequency would be displayed on modes "Change Radio" and "Clock".

b. Play audio output through a speaker: Using the audio amplifier circuit in combination with RDA5807M module, LM386-4 audio amplifier, and I2C protocol, the received FM radio input was outputted to the 8Ω, 0.5W speaker.

c. Be able to control the volume through a user interface: Using a 0-200Ω potentiometer to control the input voltage from the Raspberry Pi to the audio amp circuit. This gives direct control from the user to the radio.

d. Display the radio channel information to the user: Using the code from Exp. 3 [7], users were able to display and play the wanted FM-radio (i.e., 101.9) to the OLED screen.

Project Planning

The original project planning milestone designated team members roles and designations split between hardware and software, table 2 "Roles and Responsibilities" [2] shows the initial expectations of all members. these initial conditions laid the groundwork for assigning tasks based on team members skills and responsibilities. As per the project milestone report [2], team members set deliverable dates to complete the above tasks including testing and correcting the user experience. The following table demonstrates an accurate model of the tasks completed by team members and the effective date of delivery.

Team member 1: Daryon Ambasse		
Task	Testing model	Date of delivery
Wire and test the FM receiver with a single button input.	Upload the provided radio initializer file from "Lab 3" [6] adding button capabilities to change the frequency by a factor of $\pm$0.2MHz. Then updating the model to change volume on each button press.	June 12, 2025
Design and Manufacture PCB	Design the PCB to include: • Radio and Amplifier circuit with external speaker connection to the mounted 2 Position PCB Terminal Block. • External connection ports for dual-OLED screens. • External 5-volt power connections for proper distribution to V-BUS and V-SYS on the Raspberry Pi Pico W. • External connection points for four micro-pushbuttons. • Proper M3 sized holes for mounting on the enclosure. Upon completion of the design, fully test it on a breadboard to determine points of weakness or shorting in the circuit.	July 2, 2025
Build LM386 Amplifier circuit using KiCAD.	Design the amplifier circuit in KiCAD then verify using a breadboard. Confirm audio quality in ELW	July 25, 2025

	building to record how the signal is affected in a Faraday Cage, extend antennae accordingly to capture the frequency 101.9.	
Provide initialization file to accommodate for an extended OLED screen using two smaller screens.	Use a daisy-chain method for shared MOSI and MISO connections to the screens with separate Chip select pins, creating an extended display. Test the extended display by placing text files in both screens and observing the effect of trying to display one image spanning both screens.	July 25, 2025
Soldering PCB components	Solder all through-hole components onto the PCB using a lead-free solder wire.	July 18, 2025

Table 1 - Team Member 1, tasks and deliverables

Team member 2: Rowyn Davis		
Task	Testing model	Date of delivery
Design and 3D print the Enclosure	Design the enclosure to accomplish: • Design the top lid to mount dual-OLED screens, 4 buttons, and a potentiometer for user interface. • Design the bottom enclosure to hold and manage the custom designed PCB, Raspberry Pi Pico W, and 2 AA batteries. • Have enough structural stability to stand up on its own without any supports. • Add a hole for wires to connect from the mounted UI components and the PCB/Raspberry Pi. Upon 3D printing the enclosure, the corners and screw holes were carved and sanded. The UI components were added, and the full project was assembled and tested.	July 18, 2025,
Wire and Test User inputs	Mount and test button input (appendix D).	July 26, 2025
Soldering UI components	Soldered all needed connections (i.e., buttons, potentiometer…) using lead free solder wire.	July 28, 2025
Installation & Final Testing	Assemble and wire all connections for final presentation. Use testing files (appendixes D, E, and F) to inspect all functionalities.	July 29, 2025

Table 2 - Team Member 2, tasks and deliverables

Circuit Design & Schematic

The following circuit integrates available hardware for a working clock-radio around a central Raspberry Pi Pico W microcontroller to implement a multi functional system. The schematic consists of the following subsystems:

- **Dual display OLED system**
    - Two SSD1306 OLED displays (128 x 64 each) connected via SPI.
    - Sharing MOSI and SCK lines.
        - MOSI = GPIO-19.
        - SCK = GPIO-18.
    - Individually using separate Chip Select (CS) pins.
        - Left display CS = GPIO-17.
        - Right display CS = GPIO-05.
    - Shared DC (GPIO-20) and RES (GPIO-21) lines for command data control and hardware reset.
    - Finished product has the two screens interact as one extended display with a size of 256 x 64.

- **User input using buttons and a potentiometer**
    - Four push buttons connected to GPIO pins 0 to 3 completing the following tasks:
        - Mode cycling.
        - Increment selection by +1.
        - Decrement selection by -1.
        - Activate/Select.
    - Buttons are pulled to low by default and high when pressed.
    - Rotary potentiometer [7] bridges the radio module and amplifier circuit to control volume by varying the voltage through the circuit.

- **FM radio module**
    - Connected via **I2C bus**:
        - SDA → GPIO 26.
        - SCL → GPIO 27.
    - Controlled with a custom Radio class (via Micropython).
    - Allows tuning, scanning, and volume control through software interface.

4. **Power management through external 5-volt connection or via USB**
    - Internal regulator on Pico supplies power to displays and buttons at a steady 3.3-volts.

All the above features were taken into consideration while designing the PCB, ensuring appropriate GPIO pin assignments, power distribution, and component placement to support dual SPI displays, I²C radio communication, button inputs, and future expandability. Signal traces were routed with attention to minimizing interference between high-frequency SPI lines and analog components like the potentiometer. Additional considerations included mounting hole placement for enclosure integration, silkscreen labels for easier debugging, and decoupling capacitors near ICs to ensure stable operation.

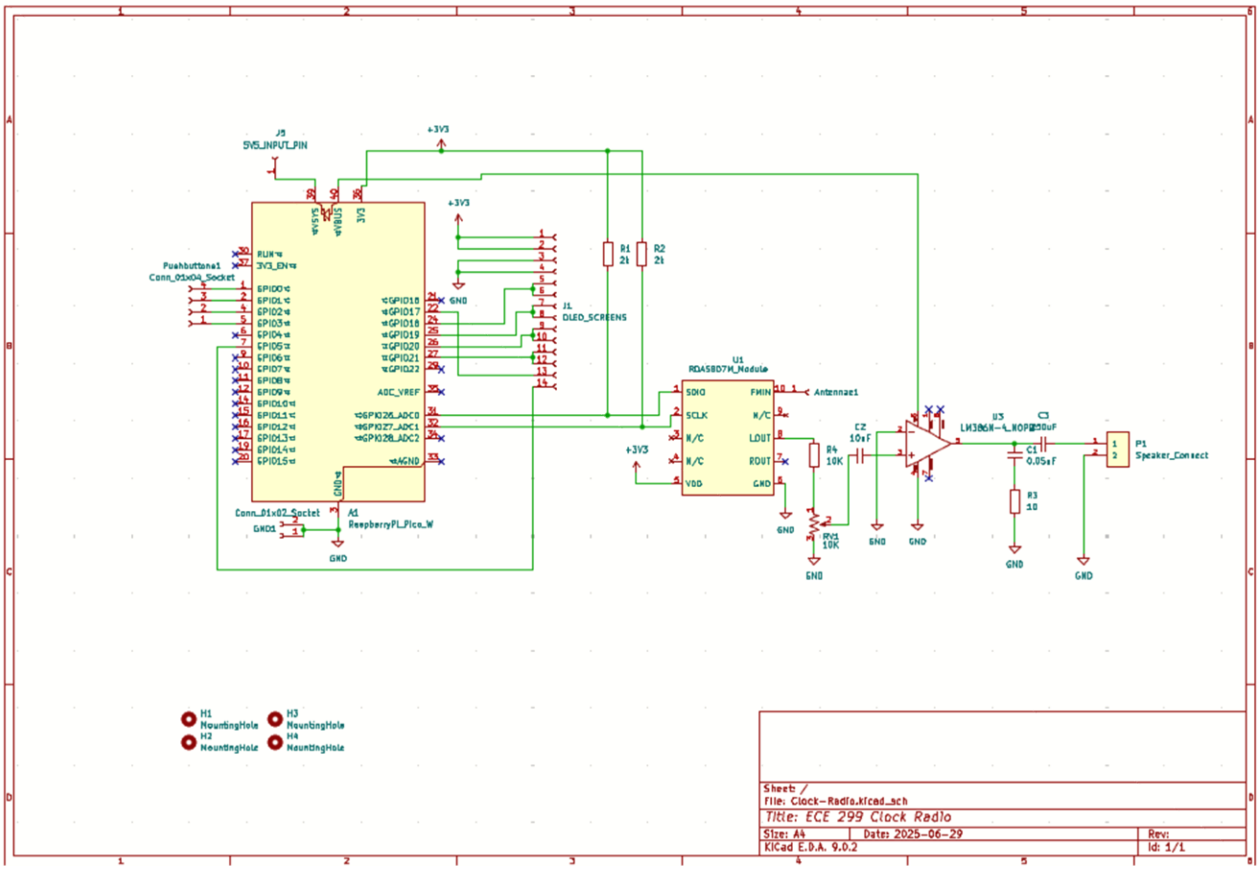

Figure 1 - PCB schematic

Bill of Materials & Cost Analysis

Label in Schematic/Drawing	Component Description	Part Number	Unit Quantity	Cost per Unit	Source of Cost Information
Raspberry Pi Pico W			1	Free	ELW B320
GND_PIN	Ground test point on PCB	N/A	1	Free	JLC PCB
0.05uF	50 nano-farad capacitors.	399-9794-ND	1	Free	ELW B320
10uF	10 micro-farad capacitors.	493-13534-3-ND	1	Free	ELW B320
250uF	470 micro-farad capacitors.	493-1826-ND	1	Free	ELW B320
Conn_01x02_Socket	Socket headers for external ground connections.	2057-PH1-07-UA-ND	1	Free	ELW B320
Mounting Hole	Mounting holes for M3 screws.	N/A	4	Free	JLC PCB
OLED_SCREENS	1.3” OLED Display SPI	1528-1512-ND	2	$4.00	ELW B320
5V5_INPUT_PIN	5-volt input test pad for external power supply.	N/A	1	Free	JLC PCB
Speaker Connect	2 Position PCB panel mount	2057-EBAA-02-C-ND	1	$1.50	ELW B320
Conn_01x04_Socket	Pin header connections for buttons.	2057-PH1-07-UA-ND	1		ELW B320
2k	2000-ohm resistor	13-MFR-25FTF52-2KTB-ND	2	Free	ELW B320
10	10-ohm resistor	2019-CF1/4CT52R100JCT-ND	1	Free	ELW B320
10K	10000-ohm resistor	2019-CFS1/4CT52R103JCT-ND	1	Free	ELW B320
10K	10000-ohm potentiometer for user interface.	PTV09A-4020U-B103-ND	1	$1.00	ELW B320
RDA5807M_Module	Through hole PCB mounted radio module.	RRD-102V2.0	1	$2.00	ELW B320
LM386N-4_NOPB	Through hole mounted audio amplifier unit.	LM386N-4/NOPB	1	$1.50	ELW B320

Conn_01x14_Socket	Pin-header connections for externally connecting OLED screens.	2057-PH1-07-UA-ND	1	$0.25	ELW B320
Pushbutton Microswitch PCB	Microswitch pushbuttons for user interface.	732-7004-2-ND	4	$0.25	ELW B320
Speaker	External Speaker rated 2W/4Ω	102-3546-ND	1	$4.00	ELW B320
Monkey Stuffed Animal	Enclosure Exterior	N/A	1	$13.41	Walmart
M2 Screw		Rm2X8MM 2701	10	$0.58	Digikey
M3 Screw		Rm3X6MM 2701	8	$0.48	Digikey
M6 Screw		770-270	5	$0.99	Digikey
Conn_01x_05_Socket	5 position socket headers	2057-PH1-07-UA-ND	2	$0.50	ELW B320
Enclosure Manufacturing	Internal SolidWorks design for the monkey stuffed animal	N/A	1	$15.00	UVIC Digital Scholarship Commons

Table 3 – Bill of Materials for entire project

PCB Design

The custom PCB was designed to meet specific functional and physical constraints critical to the success of the clock radio project. The total board area was kept under 30 cm^2 to ensure compactness and compatibility with the planned enclosure. Four mounting holes, one at each corner, were included to facilitate secure installation. A key design consideration was preserving the clearance zone around the Pico W's internal antenna; no traces or copper pours were routed in this region to prevent interference with wireless communication. To accommodate higher current demands, 0.6 mm wide traces were used for both the 5V power lines and the connections to the speaker output. Finally, to reduce wiring complexity and improve assembly and debugging, all external connectors were positioned along one edge of the board. This deliberate layout minimizes internal wire clutter and supports clean integration within the final device enclosure.

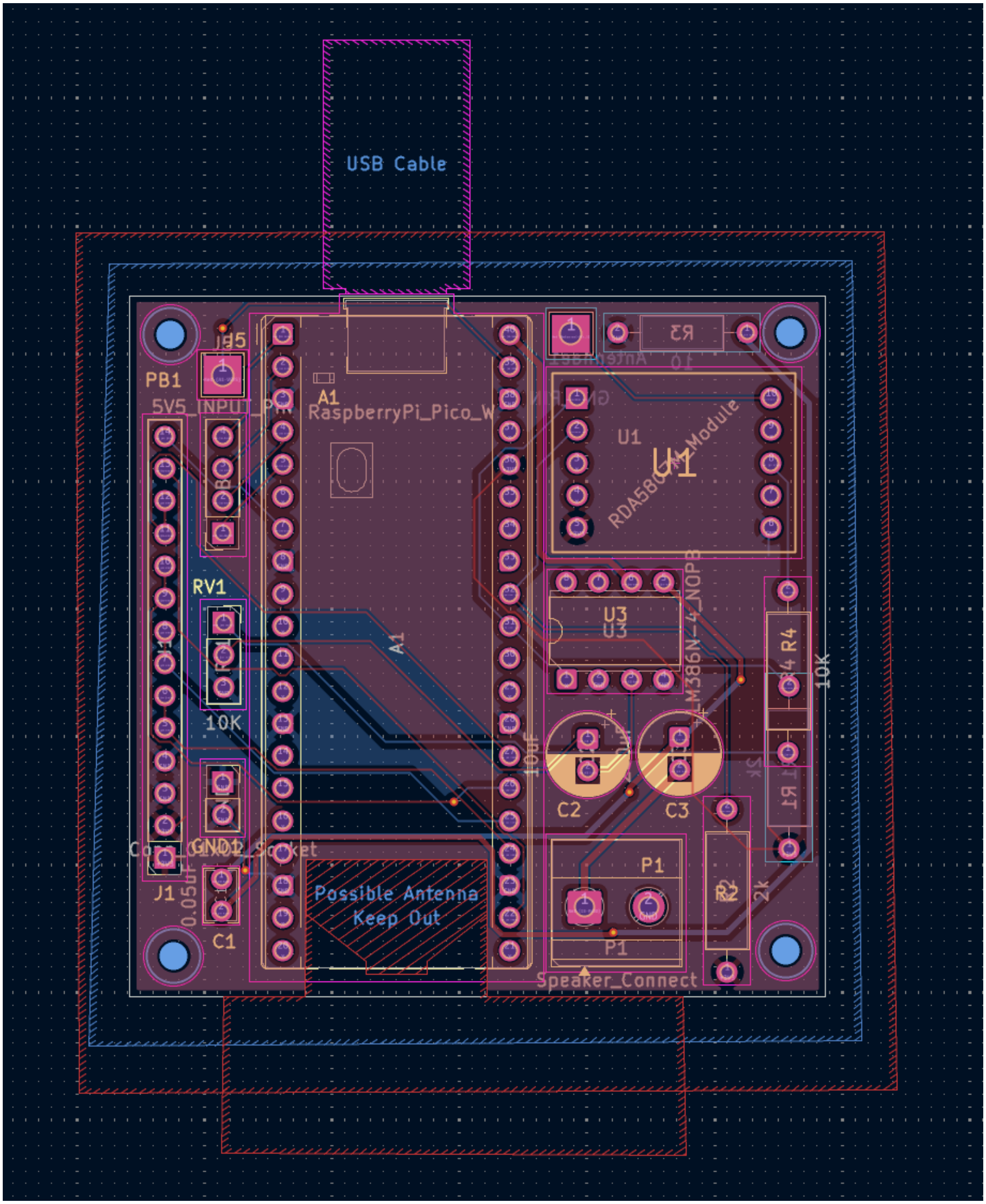

Figure 2 - PCB editor layout in KiCAD

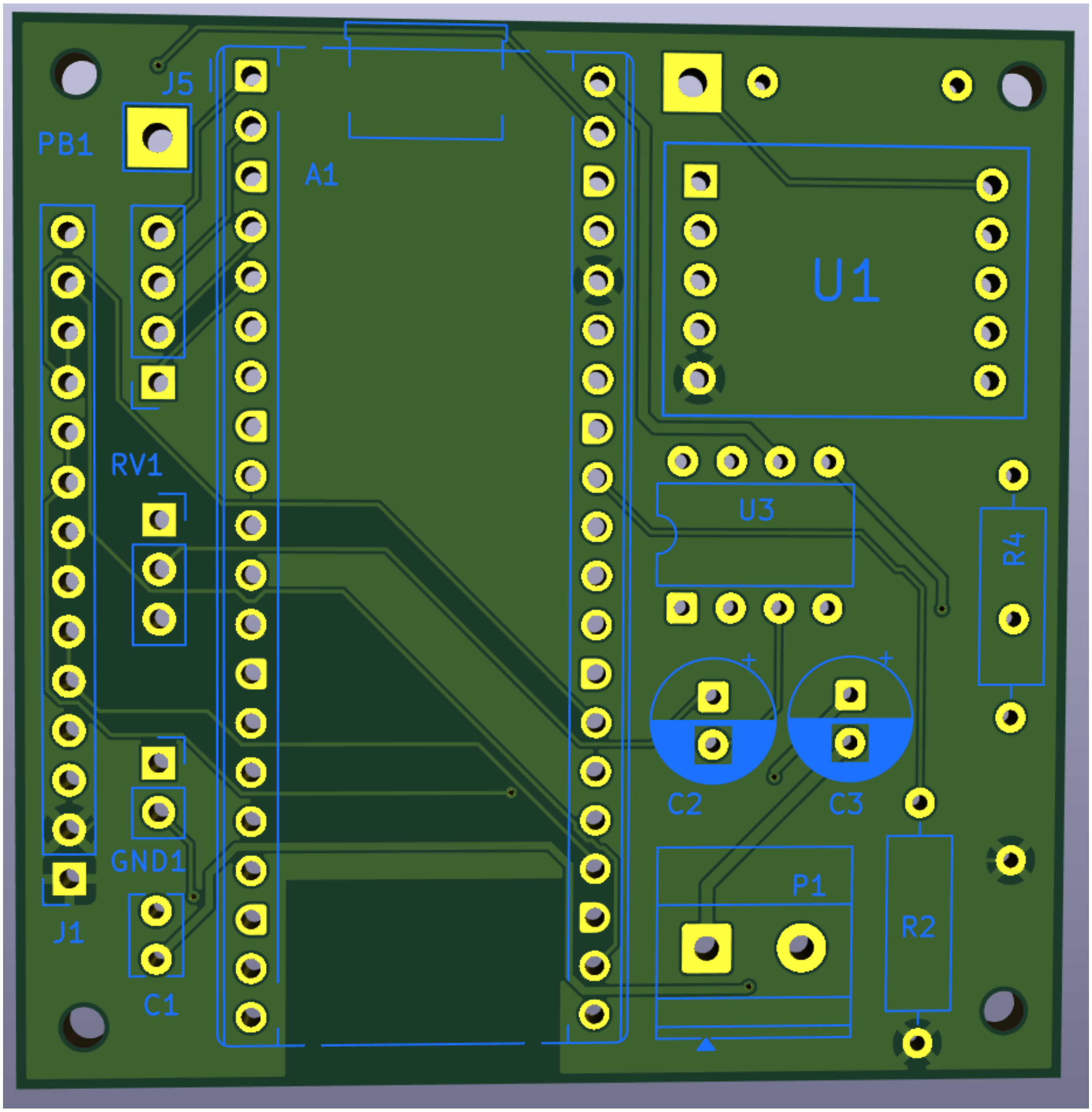

Figure 3 - 3D PCB front view in KiCAD

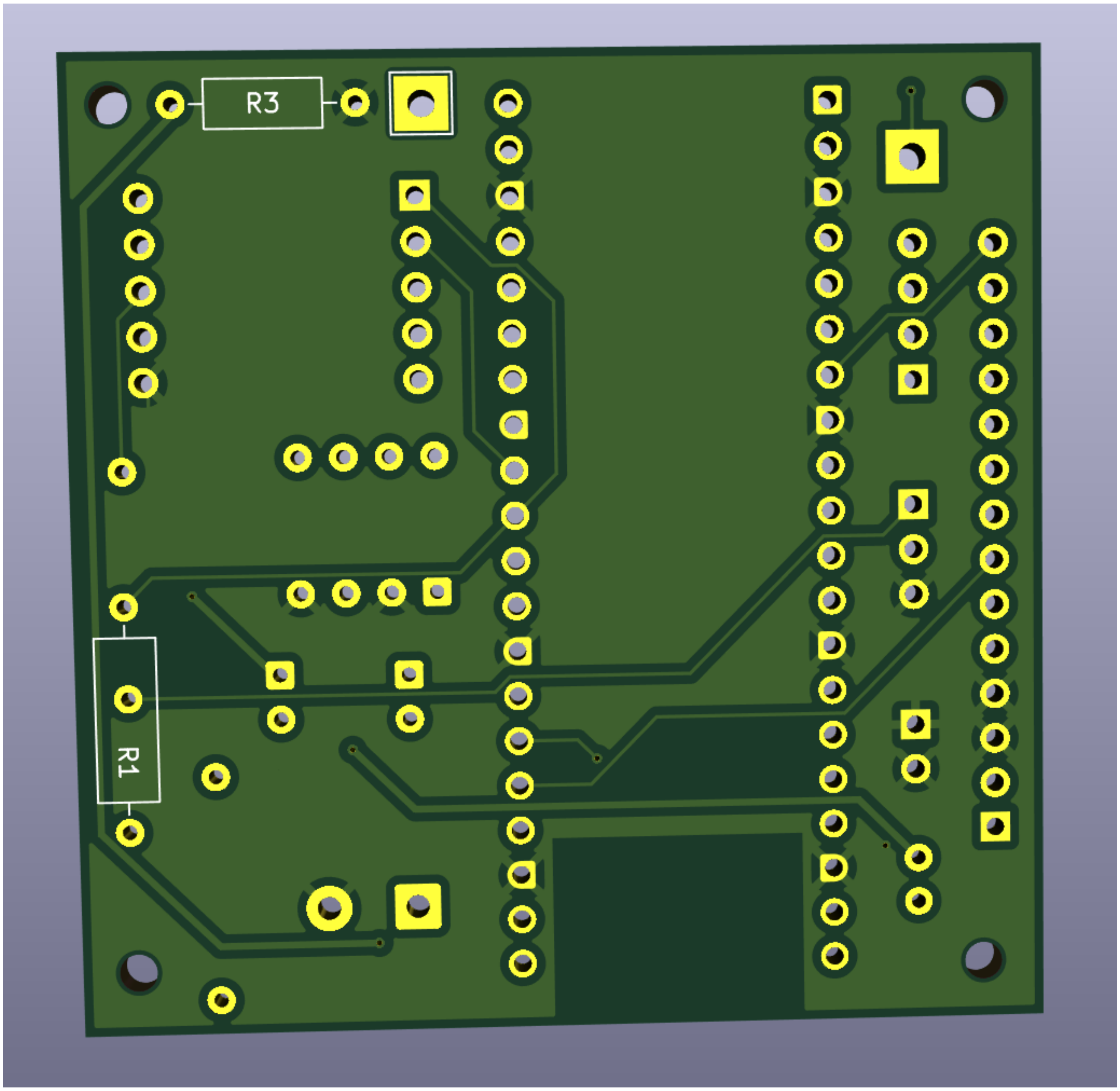

Figure 4 - 3D PCB back view in KiCAD

Enclosure Design

The 3D-printed enclosure was designed to be the internal "skeleton" structure of the clock/radio. To keep the design choice of using a stuffed animal as the outer appearance, the internal enclosure needed to achieve the following:

- Be compact enough to be hidden inside the stuffed animal.
- Have purposeful built mounts for the 2 OLED screens and buttons.
- Hold the custom designed PCB and batteries in an easily reachable position.
- Have the bottom enclosure be heavy enough to act as a weight to keep the whole structure upright.

The lid was the best place to mount the screens and buttons. Special consideration was given to ensure that the neck was narrow but still strong enough to easily hold the weight of the screens. The button mount was designed to have 2 vertical supports (following the HYDRA RESEARCH guidelines [3]). Eight M3 screws were used to hold the OLED screens and four M6 screws to hold the lid to the bottom enclosure.

The bottom enclosure was designed with the intention of holding all the electrical components. Since the PCB had 2 layers, the team created 4 mounting points that could accommodate M2 screws to keep the board in place. They also added room for the batteries to be connected, leaving enough wires to be connected safely inside. A hole was placed in one of the walls for the connection wires from the PCB and Raspberry Pi to be connected to the external components (OLED screens, buttons, and speaker). Since the bottom was used as a counterweight, extra material was used to ensure that the whole clock/radio wouldn't tip over.

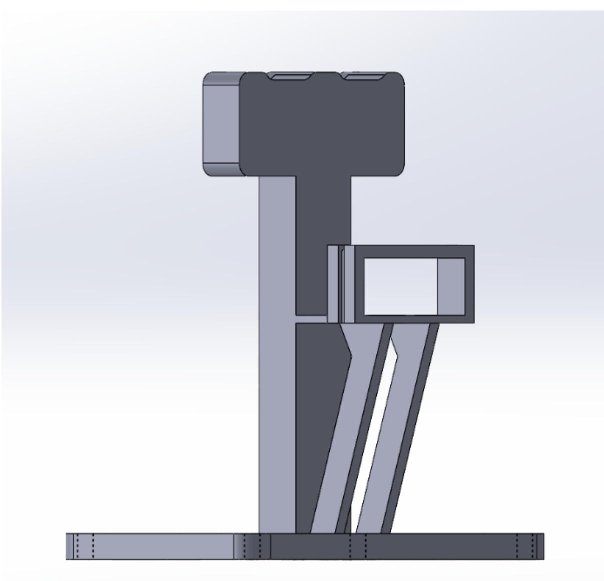

Figure 5 - 3D Lid Back View in SolidWorks

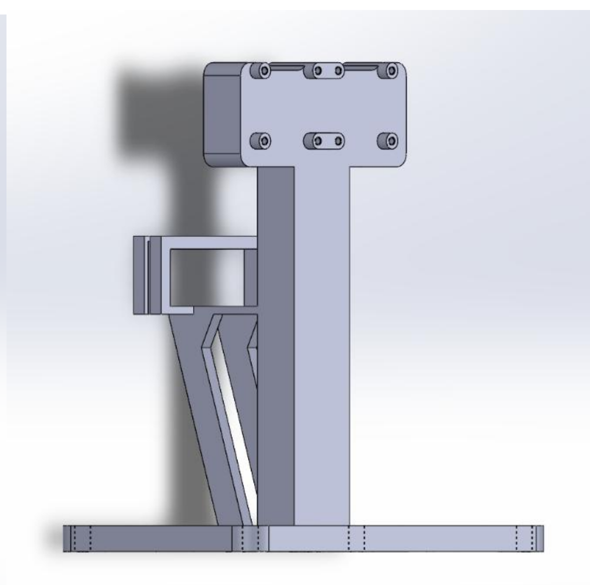

Figure 6 - 3D Lid Front View in SolidWorks

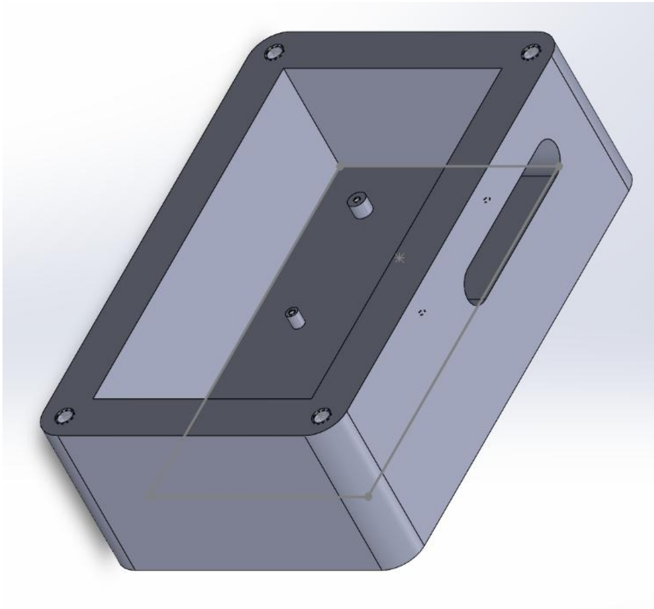

Figure 7 - 3D Bottom Back View in SolidWorks

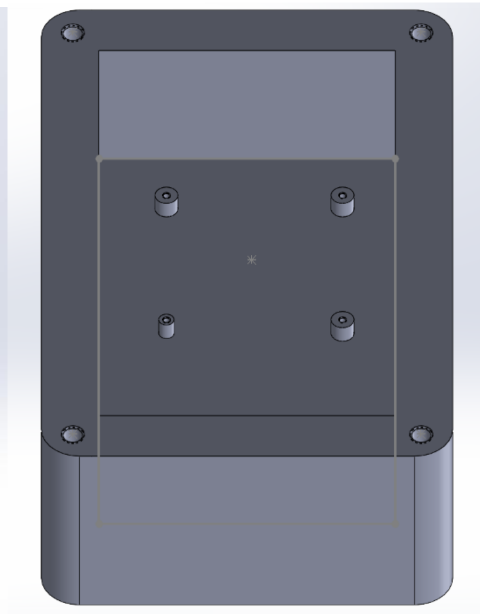

Figure 8 - 3D Bottom Front View in SolidWorks

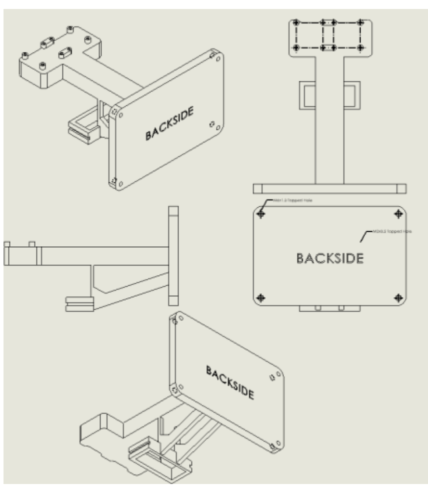

Figure 9 – Lid Drawing in SolidWorks

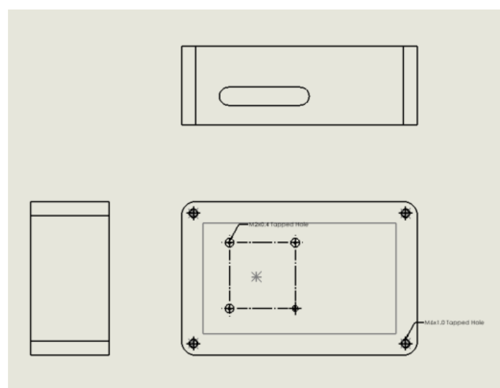

Figure 10 – Bottom Enclosure Drawing in SolidWorks

Testing & Validation

Test Procedure 1 – Button Test (Appendix D):

Throughout the assembly of the clock/radio, there were multiple times where the user interface wouldn't respond to button inputs. To troubleshoot this, the team developed a button test file that would see whether the problem came from the buttons/physical wiring or from the main.py file itself. The code itself worked by whenever a button is pressed when the code is running, the terminal on the laptop will print out which button was pressed. This helped troubleshoot if the buttons could be detected and if not, then the problem lied in the hardware/physical wiring.

Test Steps:

1. Load the Button Test code onto the Raspberry Pi Pico W using a micro-USB cable.
2. Run code and push wanted button(s).
3. Check terminal on laptop and see if the pressed button(s) are detected. The two possible outcomes are:
    a. The pressed button(s) is detected: The terminal prints out "Button X pressed". This means that the buttons are detected by the Raspberry Pi and that the problem lies somewhere in the clock/radio code.
    b. The pressed button(s) is NOT detected: The terminal will not print out anything. This means that the problem lies in the hardware and could be a loose connection or a soldered point breaking.

```
Button test running. Press any button...
Button 1 (GPIO {0}) pressed
Button 2 (GPIO {3}) pressed
Button 3 (GPIO {6}) pressed
Button 4 (GPIO {7}) pressed
Button 1 (GPIO {0}) pressed
Button 2 (GPIO {3}) pressed
Button 3 (GPIO {6}) pressed
Button 4 (GPIO {7}) pressed
Button 1 (GPIO {0}) pressed
Button 1 (GPIO {0}) pressed
Button 2 (GPIO {3}) pressed
Button 2 (GPIO {3}) pressed
Button 3 (GPIO {6}) pressed
Button 3 (GPIO {6}) pressed
Button 4 (GPIO {7}) pressed
Button 4 (GPIO {7}) pressed

>>>
```

Figure 11 – Terminal Output from button test in Appendix D

The results above proved that the system was able to read button inputs from the user and was used to determine a pin issue from the board on GPIO-1 and GPIO-2 since they would automatically read high and wouldn't change. This issue was solved by re-routing buttons three and four to GPIO-6 and GPIO-7.

Test Procedure 2 – Radio Test (Appendix F):

Since the audio circuit is crucial to the working of the Clock/Radio, developing a radio test was important to determine the functionality of the whole project. Using the code from Exp. 3 [6] (which we know works), again using deductive reasoning to decide whether the problem lies in the circuit (if the code still does not work), or if the problem lies in our code.

Test Steps:

1. Load the radio test code onto the Raspberry Pi.
2. Connect to any local channel (i.e., 101.9 or 98.5 FM).
    a. If no audio plays, then the fault lies in the hardware. The team can then pinpoint where the signal stops using a multimeter.
    b. If audio does play, then the fault lies in the radio portion of the code and can be fixed.

20250728_152111.mp
4

Figure 12 – MP4 file of the speaker output using the radio test file in Appendix F (channel frequency 98.5)

From the above figure it can be observed that the audio being produced by the radio module is clear and works as intended by the uploaded code. Afterwards all that is required would be to integrate with the dual OLED code and buttons to give the user complete control over the functionality of the clock-radio. Due to formatting if the video is inaccessible through the above "Figure 12" then please select the following link: Online Test Link.

Test Procedure 3 – Dual OLED Screen Test (Appendix E):

Due to the way the team wanted to present information to the user, it was feasible to utilize two screens as one extended display rather than a single larger screen. After developing the initialization file, which can be found in Appendix C, it was necessary to test how information would display to the user. The team used the following logic to determine the necessary changes to be made before running our main project file (Appendix A):

Test Steps:

1. Determine the range where text would appear on either screen one or two.
2. Run some random text onto each screen to determine the pixel boundary between screens.
3. Try and run text across both screens to find any breaking point/places where text would cut out and be lost.

a. If text can be seen across both screens, then upload the main file and test the refresh rate of the clock along with button inputs.
b. If text is lost or cannot be displayed then update the initializer file, or a connection issue may be present between the shared MOSI and MISO pins on the Raspberry Pi.

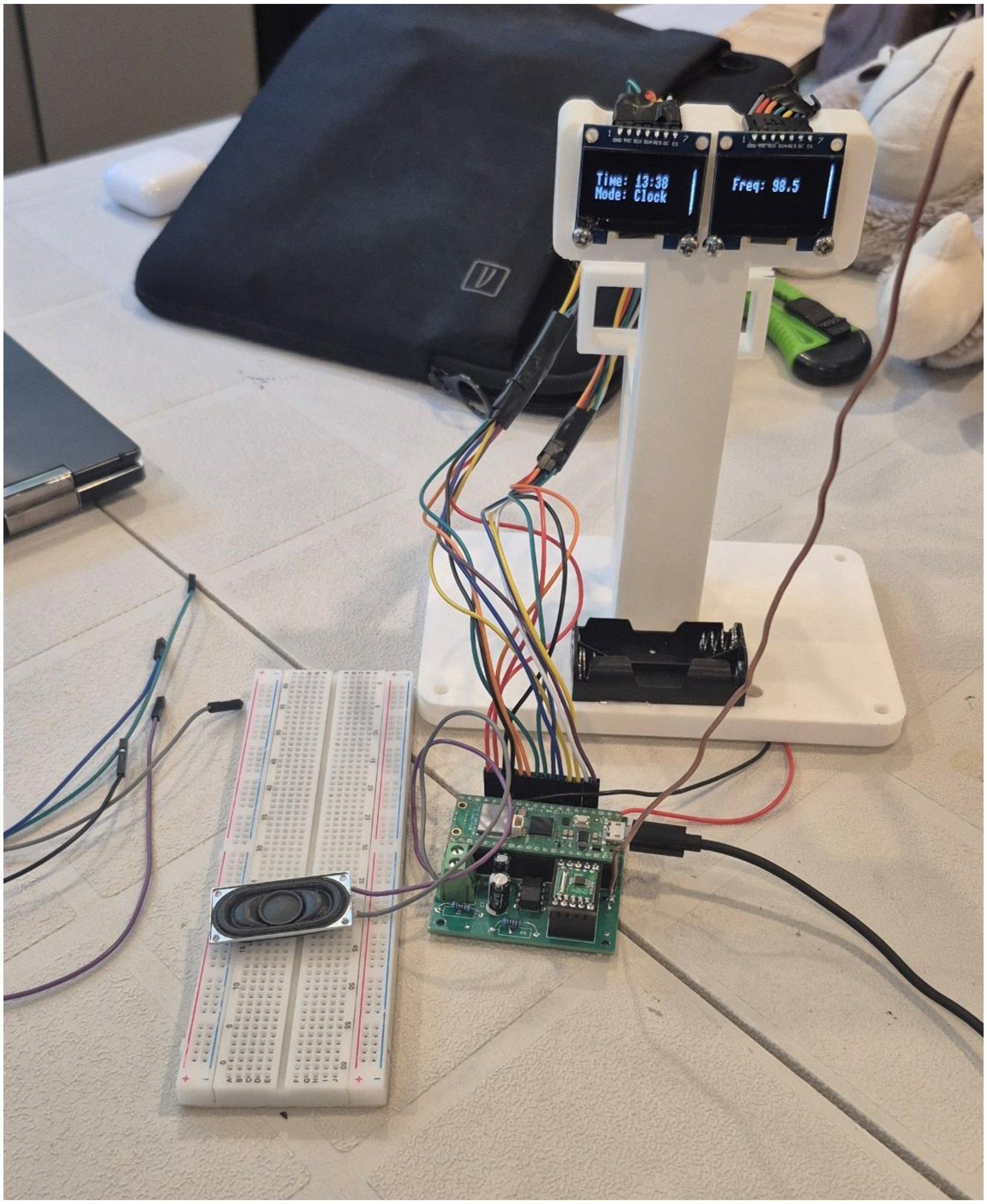

Figure 13 – Dual OLED test display as per Appendix E

As per the figure above it can be seen that the screen is displaying the Micropython code in Appendix E. This proves that both screens are interfacing with each other to create one larger extended display since the only difference in the texts being displayed are their positioning. This allowed the team to fully integrate display features into the main file knowing that the refreshed content would be properly updated and displayed without a loss of data.

Code of Ethics

List the relevant code of ethics of EGBC and **describe how you have followed the relevant codes in your project**

The relevant list of ethics pertaining to this project include the following from the registered EGBC "Code of Ethics" [8]:

1. **Hold paramount the safety, health, and welfare of the public, including the protection of the environment and the promotion of health and safety in the workplace.**

    The team prioritized safe electrical design practices, ensuring that all power lines and components were selected within safe operating limits. Proper PCB trace widths were used for higher current paths, and all connections were insulated and tested to minimize the risk of electrical hazards during operation.

2. **Practice only in those fields where training and ability make the registrant professionally competent.**

    The project was conducted strictly within the domain of Electrical and Computer Engineering, leveraging knowledge gained through coursework, specifically ECE 299: Electrical Design, which provided us with the relevant training in circuit design, PCB layout, and embedded systems development.

3. P**rovide accurate information in respect of qualifications and experience**

    All project documentation, including the Bill of Materials (BOM), schematic diagrams, and 3D models of both the PCB and enclosure, were created and presented with transparency and accuracy. Each figure was appropriately labelled and referenced to reflect our actual design and implementation.

4. **Undertake work and documentation with due diligence and in accordance with any guidance developed to standardize professional documentation for the applicable profession.**

    The team applied a methodical and detail-oriented approach throughout the design, testing, and reporting phases of the project. Schematics and PCB layouts followed industry standards for clarity and reproducibility, and design files were reviewed and verified by group members before submission.

5. **Conduct themselves with fairness, courtesy, and good faith towards clients, colleagues, and others, give credit where it is due and accept, as well as give, honest and fair professional comment.**

> Collaboration was central to our project. All team members contributed equally, and credit was given accordingly. Feedback was exchanged respectfully during design reviews, and any constructive criticism was received in the spirit of continuous improvement.

Conclusions & Recommendations

Over the course of this project, the team gained valuable hands-on experience and developed critical engineering competencies through iterative design, testing, and problem-solving. Key technical skills acquired include proper soldering techniques for both surface-mount and through-hole components, machine-level programming using Micropython, and the successful implementation of an extended OLED display using custom SPI initialization routines. The team also became proficient in managing embedded audio output through speaker optimization, wire splicing for reliable power and signal delivery, and safe operation of power tools for enclosure fabrication. Additionally, the team strengthened their ability to manage time and distribute workload effectively within a high stress environment, which was crucial for synchronizing hardware and software integration across subsystems.

Despite the teams' achievements, they encountered several challenges that tested their ability to adapt and troubleshoot under pressure. Critical issues included physical hardware failures such as screen fractures and battery ruptures, enclosure misalignment due to dimensional inaccuracies, radio interference issues likely stemming from poor shielding or grounding, code integration errors preventing successful compilation, and instances of short or disconnected wires affecting power and signal continuity. Furthermore, the team observed hardware degradation such as broken microcontroller pins due to repeated handling during prototyping.

To improve future iterations of the design, the team has compiled the following actionable recommendations based on their observations:

- Connect the potentiometer to an additional GPIO pin to record and display live volume levels on the OLED interface.
- Replace the traditional push-button system with a rotary encoder to reduce overall hardware complexity and improve user interface design.
- Begin debugging procedures earlier in the development timeline to identify electrical and software inconsistencies before integration.
- Conduct comprehensive tests on all subsystems independently, including multi-I2C initialization and display handling.
- Invest more time in familiarizing ourselves with advanced features of Micropython and embedded libraries relevant to real-time system control.
- Schedule regular technical consultations with lab technicians to receive timely feedback on circuit design and implementation practices.

- Ensure all custom enclosure manufacturing is completed by a vetted supplier to avoid delays and improve dimensional precision for mounting PCBs and displays.
- Perform rigorous power supply testing under different load conditions to prevent battery leakage, overheating, or explosion.

Through the successes and setbacks of this project, the students gained a deeper understanding of embedded systems development and the importance of iterative design practices. These lessons will be directly applicable to future engineering projects, particularly those involving complex hardware-software integration, user interfaces, and embedded control systems.

Bibliography

[1] "ECE 299 Outline." Accessed: Jul. 30, 2025. [Online]. Available: https://heat.csc.uvic.ca/coview/course/2021051/ECE299

[2] "Project expectations and marking scheme - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3301080/View

[3] Texas Instruments, "PCB Design Guidelines for Reduced EMI," SZZA009. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.ti.com/lit/an/szza009/szza009.pdf

[4] "Design Rules & Best Practices for FFF 3D Printing," Hydra Research. Accessed: Jul. 30, 2025. [Online]. Available: https://www.hydraresearch3d.com/design-rules

[5] M. Kaminsky, "Hand Soldering Short Course." Accessed: Jul. 25, 2025. [Online]. Available: www.kester.com

[6] "ECE299_Lab_Manual - updated on June 12 - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3338370/View

[7] "ptv09-potentiometer-datasheet - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3336401/View

[8] "Code of Ethics." Accessed: Jul. 30, 2025. [Online]. Available: https://www.egbc.ca/complaints-discipline/code-of-ethics/code-of-ethics

[9] "MicroPython - Raspberry Pi Documentation." Accessed: Jul. 30, 2025. [Online]. Available: https://www.raspberrypi.com/documentation/microcontrollers/micropython.html

[10] "Pico-W-datasheet - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3312758/View

[11] "Connecting-to-the-internet-with-pico-w - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3312759/View

[12] "audio-amplifier-lm386-datasheet - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3314237/View

[13] "Introduction to SPI Interface | Analog Devices." Accessed: Jul. 30, 2025. [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html

[14] "ECE Kicad Libraries - Summer 2025 ECE 299 A01 (30279)." Accessed: Jul. 30, 2025. [Online]. Available: https://bright.uvic.ca/d2l/le/content/410580/viewContent/3342363/View

Appendices

Appendix A: Main project code for clock-radio functionality


```python
# Import necessary modules
from machine import Pin, SPI, I2C
from display import SSD1306_DualSPI
from radio import Radio
import time
import ujson
from utime import localtime, mktime

# Setup SPI for the dual OLED display
spi = SPI(0, sck=Pin(18), mosi=Pin(19))
dc = Pin(20)
res = Pin(21)
cs1 = Pin(17)
cs2 = Pin(5)

# Initialize the dual-screen OLED display
display = SSD1306_DualSPI(256, 64, spi, dc, res, cs1, cs2)

# Initialize user interface buttons
btn_mode = Pin(0, Pin.IN, Pin.PULL_UP)
btn_select = Pin(3, Pin.IN, Pin.PULL_UP)
btn_up = Pin(6, Pin.IN, Pin.PULL_UP)
btn_down = Pin(7, Pin.IN, Pin.PULL_UP)

# Track button states for edge detection
last_button_state_btn_mode = 1
last_button_state_btn_select = 1
last_button_state_btn_up = 1
last_button_state_btn_down = 1

# UI and system state variables
mode = 0
edit_hour = True
flash_state = True
radio_info_toggle = False

# Default alarm and display settings
alarm_time = [6, 30]
alarm_active = False
snooze_minutes = 0
show_24hr = True
alarm_triggered = False
snooze_until = None

# Use a simulated clock that increments every second
sim_time = list(time.localtime())
sim_last_tick = time.ticks_ms()

# Initialize FM radio once at 98.5 MHz
fm = Radio(101.9, 2, False)
#i2c_radio = I2C(1, scl=Pin(27), sda=Pin(26))
# Alternate time zones (simulated)
alternate_timezones = [
    ("UTC-12", -12.0),
    ("UTC-11", -11.0),
    ("UTC-10", -10.0),
    ("UTC-09.5", -9.5),
    ("UTC-09", -9.0),
    ("UTC-08", -8.0),
    ("UTC-07", -7.0),
    ("UTC-06", -6.0),
    ("UTC-05", -5.0),
    ("UTC-04.5", -4.5),
    ("UTC-04", -4.0),
```

```python
    ("UTC-03.5", -3.5),
    ("UTC-03", -3.0),
    ("UTC-02", -2.0),
    ("UTC-01", -1.0),
    ("UTC+0", 0.0),
    ("UTC+01", 1.0),
    ("UTC+02", 2.0),
    ("UTC+03", 3.0),
    ("UTC+03.5", 3.5),
    ("UTC+04", 4.0),
    ("UTC+04.5", 4.5),
    ("UTC+05", 5.0),
    ("UTC+05.5", 5.5),
    ("UTC+05.75", 5.75),
    ("UTC+06", 6.0),
    ("UTC+06.5", 6.5),
    ("UTC+07", 7.0),
    ("UTC+08", 8.0),
    ("UTC+08.75", 8.75),
    ("UTC+09", 9.0),
    ("UTC+09.5", 9.5),
    ("UTC+10:00", 10.0),
    ("UTC+10.5", 10.5),
    ("UTC+11", 11.0),
    ("UTC+12", 12.0),
    ("UTC+12.75", 12.75),
    ("UTC+13", 13.0),
    ("UTC+14", 14.0)
]


selected_timezone_index = 0

# Path to settings file
SETTINGS_FILE = "settings.json"

# Attempt to load saved settings
try:
    with open(SETTINGS_FILE, "r") as f:
        data = ujson.load(f)
        alarm_time = data.get("alarm_time", alarm_time)
        alarm_active = data.get("alarm_active", alarm_active)
        snooze_minutes = data.get("snooze_minutes", snooze_minutes)
        show_24hr = data.get("show_24hr", show_24hr)
except:
    pass

# Save settings to flash storage
def save_settings():
    try:
        with open(SETTINGS_FILE, "w") as f:
            ujson.dump({
                "alarm_time": alarm_time,
                "alarm_active": alarm_active,
                "snooze_minutes": snooze_minutes,
                "show_24hr": show_24hr
            }, f)
    except:
        pass

# Handle button press edge detection
def button_pressed(button, last_state):
    current_state = button.value()
    if last_state == 1 and current_state == 0:
        time.sleep(0.05)
```

```python
        if button.value() == 0:
            return True, 0
    return False, current_state

# Draw the main clock view
def draw_clock():
    hour = sim_time[3]
    minute = sim_time[4]
    display.text("Freq: {:.1f}".format(fm.Frequency), 140, 10)
    if not show_24hr:
        suffix = "AM" if hour < 12 else "PM"
        hour = hour % 12 or 12
        display.text("Time: {:02d}:{:02d} {}".format(hour, minute, suffix), 10, 10)
    else:
        display.text("Time: {:02d}:{:02d}".format(hour, minute), 10, 10)
    if alarm_active:
        display.text("Alarm: {:02d}:{:02d}".format(*alarm_time), 140, 20)
    display.text("Mode: Clock", 10, 50)

# Draw the alarm time setting view
def draw_alarm_set():
    display.text("Set Alarm:", 10, 5)
    display.text("Hour: {:02d}".format(alarm_time[0]), 140, 10)
    display.text("Min : {:02d}".format(alarm_time[1]), 140, 50)
    display.text("Mode: Alarm Set", 10, 50)

# Draw the FM radio interface
def draw_radio():
    display.text("FM Radio", 10, 10)
    display.text("Freq: {:.1f}".format(fm.Frequency), 140, 10)
    if alarm_active:
        display.text("A", 240, 0)
    if radio_info_toggle:
        display.text("Retro", 180, 20)
    display.text("Mode: Radio", 10, 50)

# Draw the info/settings view
def draw_info():
    display.text("Alarm: {:02d}:{:02d}".format(*alarm_time), 140, 10)
    display.text("Snooze: +{}min".format(snooze_minutes), 140, 50)
    display.text("Mode: Info", 10, 50)

# Draw the manual time change interface
def draw_time_change():
    display.text("New Time: {:02d}:{:02d}".format(sim_time[3], sim_time[4]), 140, 10)
    display.text("Edit: Hour" if edit_hour else "Edit: Minute", 140, 25)
    display.text("Mode: Time Edit", 10, 50)

# Flashing display effect for alarm trigger
def draw_alarm_trigger():
    global flash_state
    flash_state = not flash_state
    if flash_state:
        display.text("!!! WAKE", 20, 15)
        display.text("UP !!!", 140, 15)

# Time shifting for alternate timezones
def get_shifted_time(base_time, offset_hours):
    shifted = base_time.copy()
    total_minutes = shifted[3] * 60 + shifted[4] + int(offset_hours * 60)
    total_minutes %= 1440
    shifted[3] = total_minutes // 60
    shifted[4] = total_minutes % 60
    return shifted

```

```python
# Alternate timezone screen
def draw_alternate_timezones():
    display.text("Alt Timezones:", 10, 5)
    visible = alternate_timezones[selected_timezone_index:selected_timezone_index+2]
    y_offset = 10
    for label, offset in visible:
        shifted = get_shifted_time(sim_time, offset)
        display.text(f"{label}: {shifted[3]:02}:{shifted[4]:02}", 130, y_offset)
        y_offset += 15
    display.text("Mode: TZ View", 10, 50)

# Main loop
while True:
    now_tick = time.ticks_ms()
    if time.ticks_diff(now_tick, sim_last_tick) >= 1000:
        sim_last_tick = now_tick
        sim_time[5] += 1
        if sim_time[5] >= 60:
            sim_time[5] = 0
            sim_time[4] += 1
        if sim_time[4] >= 60:
            sim_time[4] = 0
            sim_time[3] += 1
        if sim_time[3] >= 24:
            sim_time[3] = 0

    current_hour = sim_time[3]
    current_minute = sim_time[4]
    now_minutes = current_hour * 60 + current_minute
    alarm_minutes = alarm_time[0] * 60 + alarm_time[1]
    snooze_minutes_val = snooze_until[0] * 60 + snooze_until[1] if snooze_until else None

    if alarm_active:
        if snooze_until and now_minutes >= snooze_minutes_val:
            snooze_until = None
        if snooze_until is None and now_minutes == alarm_minutes:
            alarm_triggered = True
        elif alarm_triggered and now_minutes != alarm_minutes:
            alarm_triggered = False

    # Check all button presses at top of loop
    pressed_mode, last_button_state_btn_mode = button_pressed(btn_mode, last_button_state_btn_mode)
    pressed_select, last_button_state_btn_select = button_pressed(btn_select,
last_button_state_btn_select)
    pressed_up, last_button_state_btn_up = button_pressed(btn_up, last_button_state_btn_up)
    pressed_down, last_button_state_btn_down = button_pressed(btn_down, last_button_state_btn_down)

    display.fill(0)

    if alarm_triggered:
        draw_alarm_trigger()
        if pressed_select:
            alarm_triggered = False
            if snooze_minutes > 0:
                snooze_until = [current_hour, (current_minute + snooze_minutes) % 60]
                if snooze_until[1] < current_minute:
                    snooze_until[0] = (snooze_until[0] + 1) % 24

    else:
        if pressed_mode:
            mode = (mode + 1) % 6

        if mode == 0:
            draw_clock()
            if pressed_select and alarm_triggered:
```

```
                alarm_triggered = False

        elif mode == 1:
            draw_radio()
            if pressed_select:
                radio_info_toggle = not radio_info_toggle

            if pressed_up:
                fm.Mute = True
                fm.ProgramRadio()
                new_freq = fm.Frequency + 0.2
                if new_freq > 108:
                    new_freq = 88.1
                fm.SetFrequency(new_freq)
                time.sleep(0.1)
                fm.Mute = False
                fm.ProgramRadio()

            if pressed_down:
                fm.Mute = True
                fm.ProgramRadio()
                new_freq = fm.Frequency - 0.2
                if new_freq < 88:
                    new_freq = 107.9
                fm.SetFrequency(new_freq)
                time.sleep(0.1)
                fm.Mute = False
                fm.ProgramRadio()

        elif mode == 2:
            draw_alarm_set()
            if pressed_select:
                alarm_active = not alarm_active
                save_settings()

            if pressed_up:
                alarm_time[0] = (alarm_time[0] + 1) % 24
                save_settings()

            if pressed_down:
                alarm_time[1] = (alarm_time[1] + 1) % 60
                save_settings()

        elif mode == 3:
            draw_info()
            if pressed_select:
                snooze_minutes += 5
                if snooze_minutes > 30:
                    snooze_minutes = 0
                save_settings()

            if pressed_up:
                show_24hr = False
                save_settings()

            if pressed_down:
                show_24hr = True
                save_settings()

        elif mode == 4:
            draw_time_change()
            if pressed_select:
                edit_hour = not edit_hour

            if pressed_up or pressed_down:
```

```
                if edit_hour:
                    if pressed_up:
                        sim_time[3] = (sim_time[3] + 1) % 24
                    if pressed_down:
                        sim_time[3] = (sim_time[3] - 1) % 24
                else:
                    if pressed_up:
                        sim_time[4] = (sim_time[4] + 1) % 60
                    if pressed_down:
                        sim_time[4] = (sim_time[4] - 1) % 60

        if mode == 5:
            draw_alternate_timezones()
            if pressed_select:
                selected_timezone_index += 1
    if selected_timezone_index >= len(alternate_timezones):
        selected_timezone_index = 0

    display.show()
    time.sleep(0.05)
```

Appendix B: Radio initialization

7/30/25, 2:40 PM radio.py

```python

from machine import Pin, I2C

class Radio:
    def __init__(self, freq, vol, mute):
        self.Volume = 2
        self.Frequency = 88
        self.Mute = False

        self.SetVolume(vol)
        self.SetFrequency(freq)
        self.SetMute(mute)

        self.i2c_sda = Pin(26)
        self.i2c_scl = Pin(27)
        self.i2c_device = 1
        self.i2c_device_address = 0x10
        self.Settings = bytearray(8)
        self.radio_i2c = I2C(self.i2c_device, scl=self.i2c_scl, sda=self.i2c_sda, freq=200000)
        self.ProgramRadio()

    def SetVolume(self, v):
        try:
            v = int(v)
            if 0 <= v < 16:
                self.Volume = v
                return True
        except:
            pass
        return False

    def SetFrequency(self, f):
        try:
            f = float(f)
            if 88.0 <= f <= 108.0:
                self.Frequency = f
                return True
        except:
            pass
        return False

    def SetMute(self, m):
        try:
            self.Mute = bool(int(m))
            return True
        except:
            return False

    def ComputeChannelSetting(self, f):
        f = int(f * 10) - 870
        return bytearray([(f >> 2) & 0xFF, ((f & 0x03) << 6) & 0xC0])

    def UpdateSettings(self):
        self.Settings[0] = 0x80 if self.Mute else 0xC0
        self.Settings[1] = 0x09 | 0x04
        self.Settings[2:4] = self.ComputeChannelSetting(self.Frequency)
        self.Settings[3] |= 0x10
        self.Settings[4] = 0x04
        self.Settings[5] = 0x00
        self.Settings[6] = 0x84
        self.Settings[7] = 0x80 + self.Volume

    def ProgramRadio(self):
        self.UpdateSettings()
```

file:///C:/Users/daryo/AppData/Roaming/Thonny/temp/thonny__lxhwz3r.html 1/2

7/30/25, 2:40 PM radio.py

```python
        self.radio_i2c.writeto(self.i2c_device_address, self.Settings)

```

Appendix C: Dual OLED Screen Initialization


```python

from micropython import const
import framebuf
import time

# Register definitions
SET_CONTRAST = const(0x81)
SET_ENTIRE_ON = const(0xA4)
SET_NORM_INV = const(0xA6)
SET_DISP = const(0xAE)
SET_MEM_ADDR = const(0x20)
SET_COL_ADDR = const(0x21)
SET_PAGE_ADDR = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP = const(0xA0)
SET_MUX_RATIO = const(0xA8)
SET_IREF_SELECT = const(0xAD)
SET_COM_OUT_DIR = const(0xC0)
SET_DISP_OFFSET = const(0xD3)
SET_COM_PIN_CFG = const(0xDA)
SET_DISP_CLK_DIV = const(0xD5)
SET_PRECHARGE = const(0xD9)
SET_VCOM_DESEL = const(0xDB)
SET_CHARGE_PUMP = const(0x8D)

class SSD1306_DualSPI(framebuf.FrameBuffer):
    def __init__(self, width, height, spi, dc, res, cs1, cs2, external_vcc=False):
        self.width = width
        self.height = height
        self.external_vcc = external_vcc
        self.pages = self.height // 8
        self.buffer = bytearray(self.pages * self.width)
        super().__init__(self.buffer, self.width, self.height, framebuf.MONO_VLSB)

        self.spi = spi
        self.dc = dc
        self.res = res
        self.cs1 = cs1
        self.cs2 = cs2
        self.rate = 10 * 1024 * 1024

        for pin in (dc, res, cs1, cs2):
            pin.init(pin.OUT, value=0)

        self.reset()
        self.init_display()

    def reset(self):
        self.res(1)
        time.sleep_ms(1)
        self.res(0)
        time.sleep_ms(10)
        self.res(1)

    def write_cmd(self, cmd, cs):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        cs(1)
        self.dc(0)
        cs(0)
        self.spi.write(bytearray([cmd]))
        cs(1)

    def write_data(self, buf, cs):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
```

```
        cs(1)
        self.dc(1)
        cs(0)
        self.spi.write(buf)
        cs(1)

    def init_display(self):
        for cs in (self.cs1, self.cs2):
            for cmd in (
                SET_DISP,
                SET_MEM_ADDR, 0x00,
                SET_DISP_START_LINE,
                SET_SEG_REMAP | 0x01,
                SET_MUX_RATIO, self.height - 1,
                SET_COM_OUT_DIR | 0x08,
                SET_DISP_OFFSET, 0x00,
                SET_COM_PIN_CFG, 0x02 if self.width > 2 * self.height else 0x12,
                SET_DISP_CLK_DIV, 0x80,
                SET_PRECHARGE, 0x22 if self.external_vcc else 0xF1,
                SET_VCOM_DESEL, 0x30,
                SET_CONTRAST, 0xFF,
                SET_ENTIRE_ON,
                SET_NORM_INV,
                SET_IREF_SELECT, 0x30,
                SET_CHARGE_PUMP, 0x10 if self.external_vcc else 0x14,
                SET_DISP | 0x01,
            ):
                self.write_cmd(cmd, cs)
        self.fill(0)
        self.show()

    def show(self):
        for page in range(0, self.pages):
            left_start = page * 256
            right_start = left_start + 128

            self.write_cmd(0xB0 | page, self.cs1)
            self.write_cmd(0x00, self.cs1)
            self.write_cmd(0x10, self.cs1)
            self.write_data(self.buffer[left_start:left_start + 128], self.cs1)

            self.write_cmd(0xB0 | page, self.cs2)
            self.write_cmd(0x00, self.cs2)
            self.write_cmd(0x10, self.cs2)
            self.write_data(self.buffer[right_start:right_start + 128], self.cs2)

    def poweroff(self):
        for cs in (self.cs1, self.cs2):
            self.write_cmd(SET_DISP, cs)

    def poweron(self):
        for cs in (self.cs1, self.cs2):
            self.write_cmd(SET_DISP | 0x01, cs)

    def contrast(self, contrast):
        for cs in (self.cs1, self.cs2):
            self.write_cmd(SET_CONTRAST, cs)
            self.write_cmd(contrast, cs)

    def invert(self, invert):
        for cs in (self.cs1, self.cs2):
            self.write_cmd(SET_NORM_INV | (invert & 1), cs)

    def rotate(self, rotate):
        for cs in (self.cs1, self.cs2):
```



```
            self.write_cmd(SET_COM_OUT_DIR | ((rotate & 1) << 3), cs)
            self.write_cmd(SET_SEG_REMAP | (rotate & 1), cs)
```

Appendix D: Button Test

7/30/25, 2:45 PM button_test.py

```python
# button_test.py
from machine import Pin
import time

# Define GPIO pins for the buttons
button_pins = [0, 3, 6, 7]

# Create Pin objects with internal pull-ups
buttons = [Pin(pin, Pin.IN, Pin.PULL_UP) for pin in button_pins]
last_states = [1] * len(buttons)

print("Button test running. Press any button...")

while True:
    for i, btn in enumerate(buttons):
        state = btn.value()
        if state == 0 and last_states[i] == 1:  # falling edge = button pressed
            print(f"Button {i} (GPIO {button_pins[i]}) pressed")
        last_states[i] = state
    time.sleep(0.01)

```

Appendix E: Screen Test

7/30/25, 8:28 PM OLED__Initialize_Test.py

```python
from machine import Pin, SPI
from ssd1306_dual import SSD1306_DualSPI
from time

#Initialization

spi = SPI(0, sck=Pin(18), mosi=Pin(19))
dc = Pin(20)
res = Pin(21)
cs1 = Pin(17) #screen 1
cs2 = Pin(5)  #screen 2

display = SSD1306_DualSPI(256, 64, spi, dc, res, cs1, cs2)

#main

display.fill(0)
display.text("Time: "time.local_time, 0, 10)
display.text("Mode: Clock", 0, 30)
display.text("Freq: 98.5", 140, 10)

display.show()
```

Appendix F: Radio Test


```python
from machine import Pin, I2C
import time
import sys
import select

# Setup push button on GPIO 0 with pull-up resistor
button = Pin(0, Pin.IN, Pin.PULL_UP)
last_button_state = 1  # Start unpressed

# --- RADIO CLASS DEFINITION ---
class Radio:
    def __init__(self, NewFrequency, NewVolume, NewMute):
        self.Volume = 2
        self.Frequency = 88
        self.Mute = False

        self.SetVolume(NewVolume)
        self.SetFrequency(NewFrequency)
        self.SetMute(NewMute)

        self.i2c_sda = Pin(26)
        self.i2c_scl = Pin(27)

        self.i2c_device = 1
        self.i2c_device_address = 0x10
        self.Settings = bytearray(8)

        self.radio_i2c = I2C(self.i2c_device, scl=self.i2c_scl, sda=self.i2c_sda, freq=200000)
        self.ProgramRadio()

    def SetVolume(self, NewVolume):
        try:
            NewVolume = int(NewVolume)
        except:
            return False
        if not isinstance(NewVolume, int):
            return False
        if NewVolume < 0 or NewVolume >= 16:
            return False
        self.Volume = NewVolume
        return True

    def SetFrequency(self, NewFrequency):
        try:
            NewFrequency = float(NewFrequency)
        except:
            return False
        if not isinstance(NewFrequency, float):
            return False
        if NewFrequency < 88.0 or NewFrequency > 108.0:
            return False
        self.Frequency = NewFrequency
        return True

    def SetMute(self, NewMute):
        try:
            self.Mute = bool(int(NewMute))
        except:
            return False
        return True

    def ComputeChannelSetting(self, Frequency):
        Frequency = int(Frequency * 10) - 870
        ByteCode = bytearray(2)
```

```
        ByteCode[0] = (Frequency >> 2) & 0xFF
        ByteCode[1] = ((Frequency & 0x03) << 6) & 0xC0
        return ByteCode

    def UpdateSettings(self):
        self.Settings[0] = 0x80 if self.Mute else 0xC0
        self.Settings[1] = 0x09 | 0x04
        self.Settings[2:3] = self.ComputeChannelSetting(self.Frequency)
        self.Settings[3] = self.Settings[3] | 0x10
        self.Settings[4] = 0x04
        self.Settings[5] = 0x00
        self.Settings[6] = 0x84
        self.Settings[7] = 0x80 + self.Volume

    def ProgramRadio(self):
        self.UpdateSettings()
        self.radio_i2c.writeto(self.i2c_device_address, self.Settings)

    def GetSettings(self):
        self.RadioStatus = self.radio_i2c.readfrom(self.i2c_device_address, 256)
        MuteStatus = not ((self.RadioStatus[0xF0] & 0x40) != 0x00)
        VolumeStatus = self.RadioStatus[0xF7] & 0x0F
        FrequencyStatus = ((self.RadioStatus[0x00] & 0x03) << 8) | (self.RadioStatus[0x01] & 0xFF)
        FrequencyStatus = (FrequencyStatus * 0.1) + 87.0
        StereoStatus = (self.RadioStatus[0x00] & 0x04) != 0x00
        return MuteStatus, VolumeStatus, FrequencyStatus, StereoStatus


# --- MAIN PROGRAM STARTS HERE ---
fm_radio = Radio(101.9, 2, False)


def poll_button(fm_radio):
    global last_button_state
    current_state = button.value()
    if last_button_state == 1 and current_state == 0:
        print("Button Pressed: Increasing Volume")
        if fm_radio.Volume < 15:
            fm_radio.SetVolume(fm_radio.Volume + 1)
            fm_radio.ProgramRadio()
            print("Volume increased to", fm_radio.Volume)
        else:
            print("Volume already at max (15)")
        time.sleep(0.5)  # debounce
    last_button_state = current_state


# --- Main interactive loop ---
while True:
    print("\nECE 299 FM Radio Demo Menu")
    print("1 - change radio frequency")
    print("2 - change volume level")
    print("3 - mute audio")
    print("4 - read current settings")
    print("Enter menu number > ", end="")

    user_input = ""
    while True:
        poll_button(fm_radio)

        ready = select.select([sys.stdin], [], [], 0)
        if ready and sys.stdin in ready[0]:
            user_input = sys.stdin.readline().strip()
            break

```

7/30/25, 2:52 PM <untitled> *

```
        time.sleep(0.05)

    select_option = user_input

    # --- Handle Menu Selection ---
    if select_option == "1":
        Frequency = input("Enter frequency in MHz (e.g., 100.3) > ")
        if fm_radio.SetFrequency(Frequency):
            fm_radio.ProgramRadio()
        else:
            print("Invalid frequency")

    elif select_option == "2":
        Volume = input("Enter volume level (0 to 15) > ")
        if fm_radio.SetVolume(Volume):
            fm_radio.ProgramRadio()
        else:
            print("Invalid volume")

    elif select_option == "3":
        Mute = input("Enter mute (1 for Mute, 0 for Audio) > ")
        if fm_radio.SetMute(Mute):
            fm_radio.ProgramRadio()
        else:
            print("Invalid mute setting")

    elif select_option == "4":
        Settings = fm_radio.GetSettings()
        print("\nRadio Status")
        print("Mute:", "enabled" if Settings[0] else "disabled")
        print("Volume:", Settings[1])
        print("Frequency: %5.1f MHz" % Settings[2])
        print("Mode:", "stereo" if Settings[3] else "mono")
    else:
        print("Invalid option")

    time.sleep(0.05)
```

Checklist

[Final-report-checklist]()