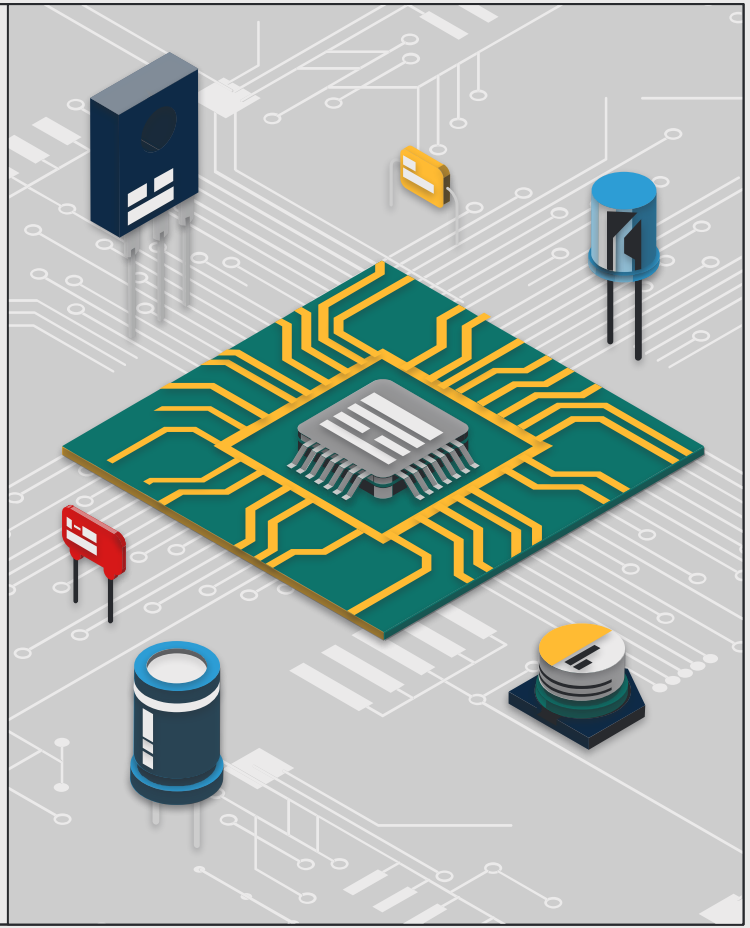
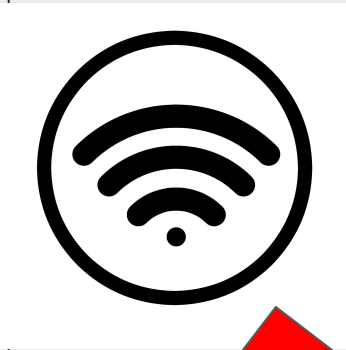
# SDR: Local Oscillator & User Interface
# Final Design



**Team C4**

Hyeonji J, Laura M, Zara O

# Imagine a World Without Wireless Communication...



No WIFI

No Cellular Networks

Just big.. Bulky analog radios..

# Our Motivation!

Our goal was to design Subsystem C that allows the radio to be functional and precise while remaining accessible for anyone- **even my grandmother!**

***"If she can use it without confusion, we know we've done our job right!"***

# Not Ready to recommend.

- Oversight in PCB
- Not completely functional

# Table of Contents

01

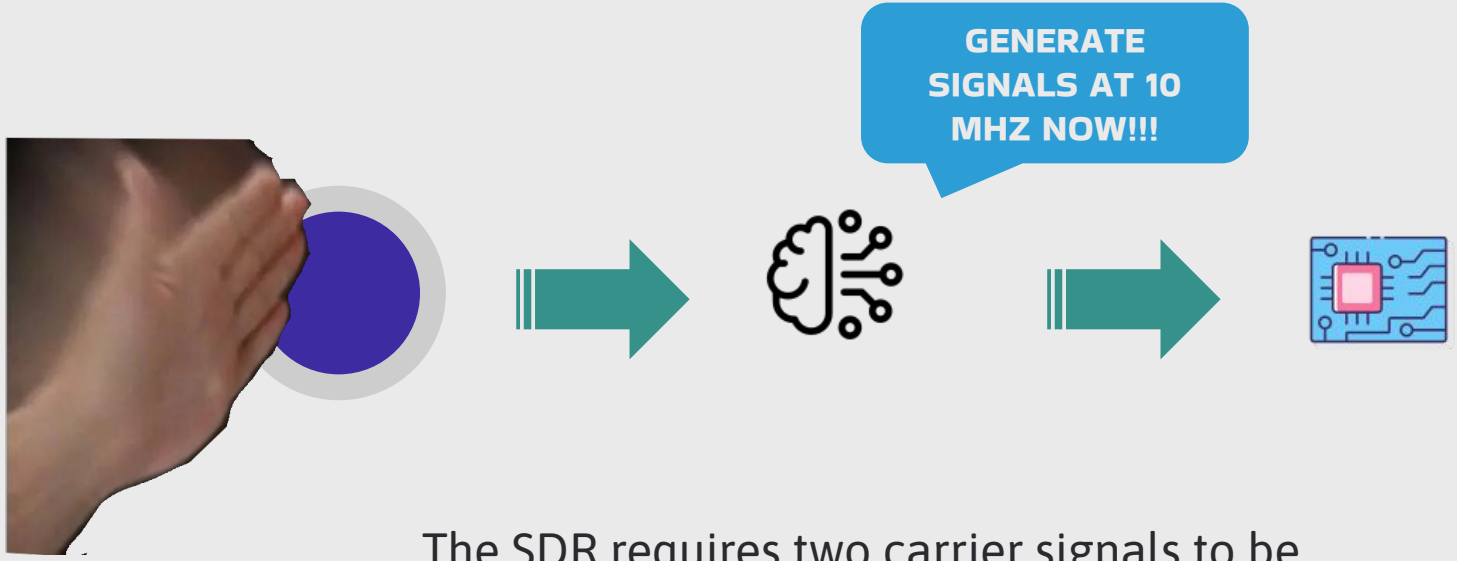
# Introduction

# SDR?

Uses digital signal processing techniques to implement radio.

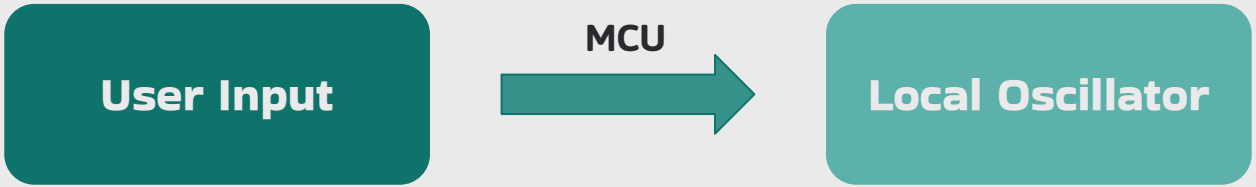
- **Smaller Size**
- **Less Circuitry**
- **Tailored to Use Case**

# The Big Idea of Subsystem C




The SDR requires two carrier signals to be generated from user input.

# What Subsystem C Needs to do




- Two LO signals 90 degrees phase shifted, 3.3 Vpp, 8 - 16 MHz
- Active low TXEN, transmit or receive enable
- Include specified headers to interconnect with mainboard
- Allow input manually or via USB

The information above is referenced directly from the ICD [1].

02

# High Level Design

# Block Diagram of Our Design

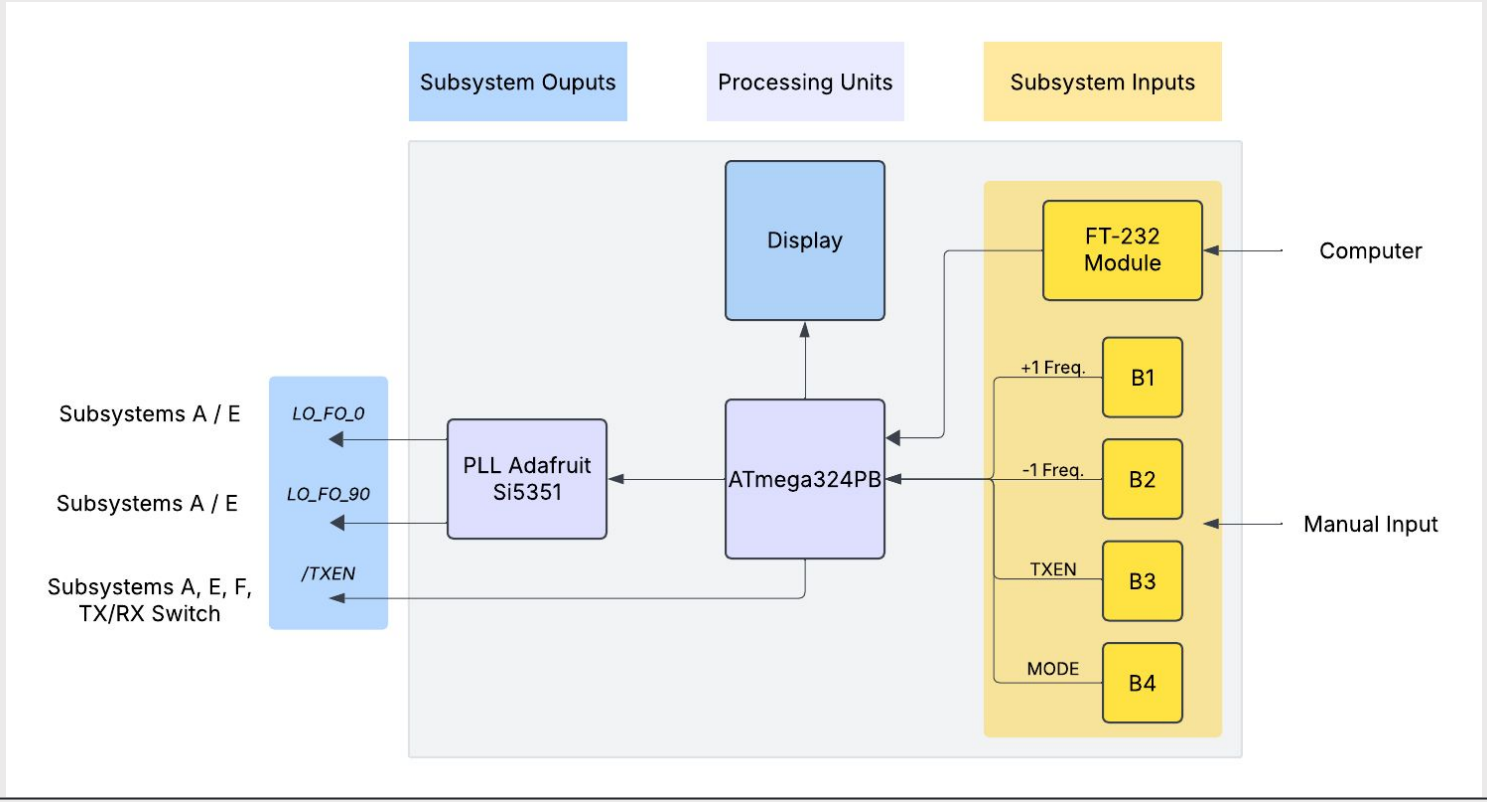
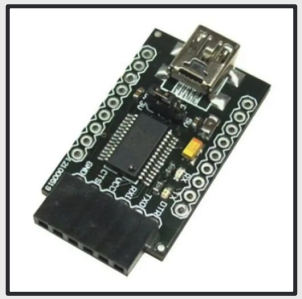



Figure 1. Full design block diagram.

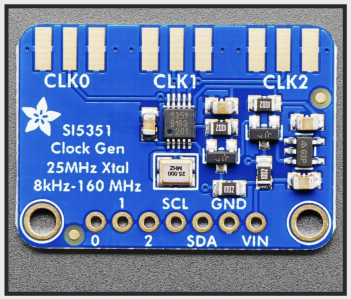
# Design Principles & UI

Selection was based on weighing **tradeoffs**...

Make a system that meets ICD.



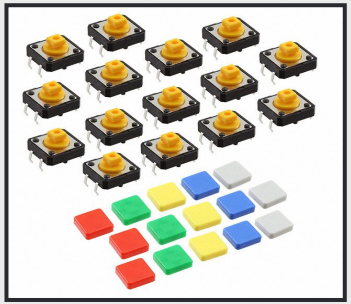
DFROBOT FTDI Module [5].




Adafruit Si5351a PLL [2].

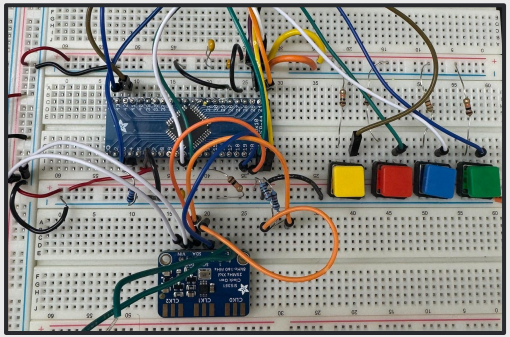
Make a system that is **easy** and **intuitive** **to use**.

Adafruit ST7789 IPS Display [3].



Adafruit 1010 buttons [4].

# How our Design Works



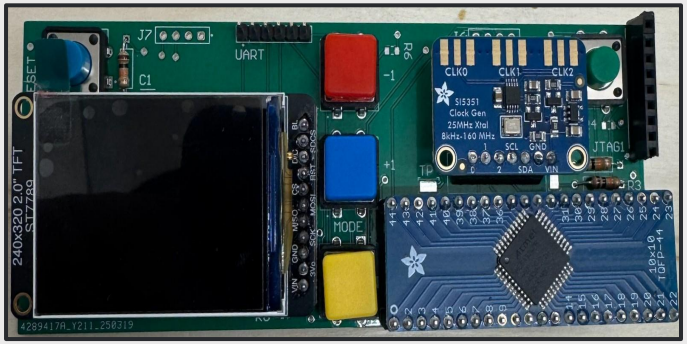



Figure 2. Design prototype on breadboard (above). Assembled design on PCB (below).

- Buttons for changing frequency.
- Full colour display to provide information to the user.
- PLL to generate the signals.
- All the required headers to ensure ICD requirements are met.

03

# Our Results

- PCB Assembly
- Testing & Debugging

# Our Final UI Design

Our Goal..

**EASY & INTUITIVE FOR ANYONE**

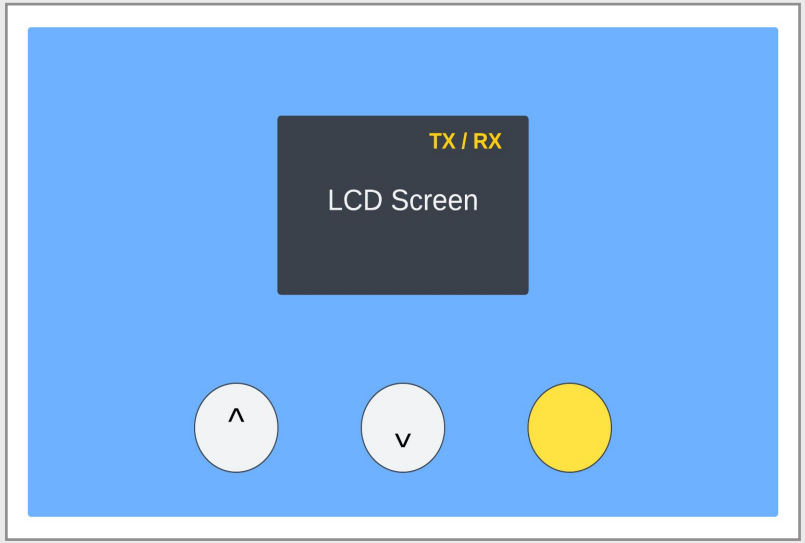



Figure 3. Image of our UI design.

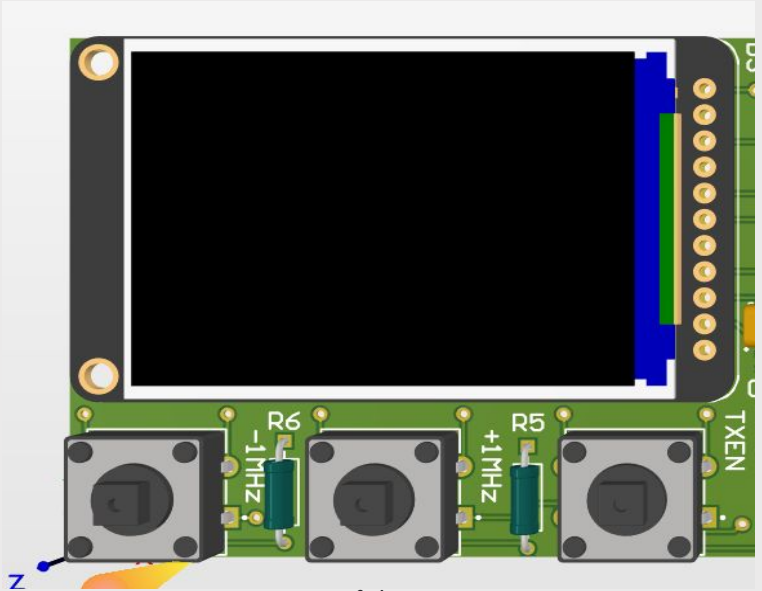



Figure 4. Our 3D view of the UI

# PCB Assembly

Assembling our PCB involved soldering our through hole and surface mount components

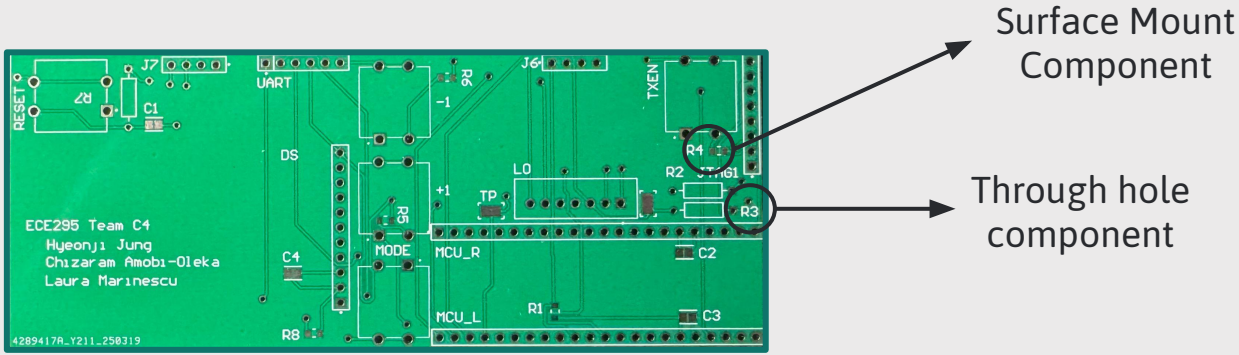
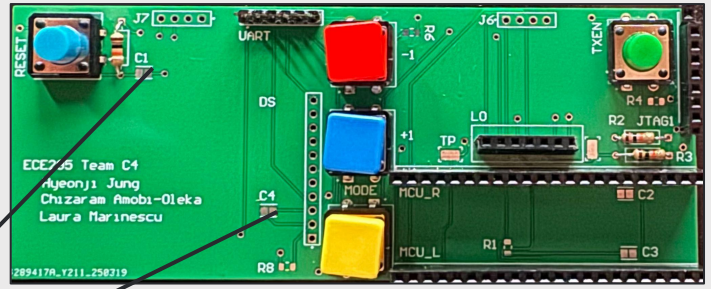



Figure 4. Image of our PCB.

# The Soldering mistake we didn't catch...

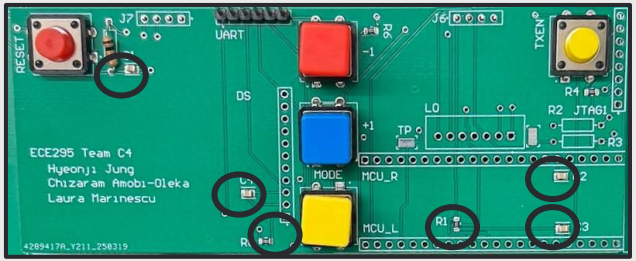
Our first oversight was soldering the **through hole** components *before* the **surface mount** components.




**Could not put in Oven!**

No surface mounts soldered

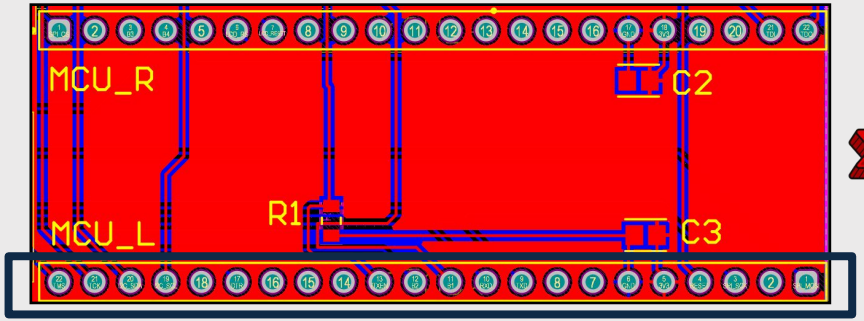
# Our Solution : Redo the Soldering



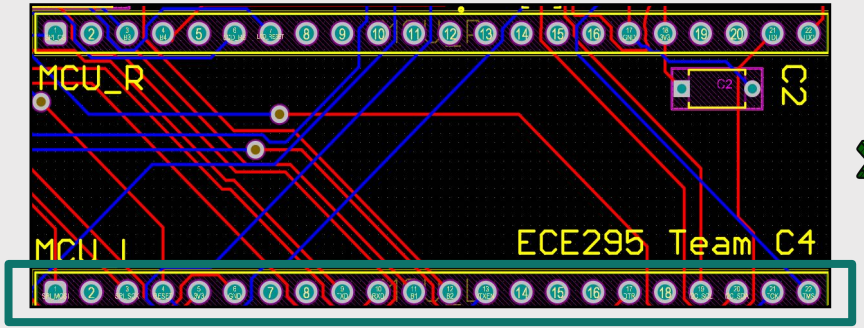

**Can put in Oven!**

The surface mounts were soldered first in the new PCB

# The mistakes we didn't catch...

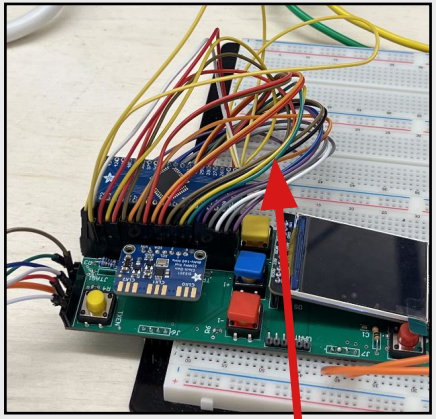


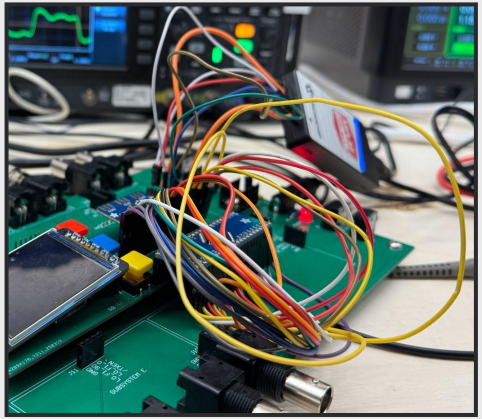

MCU_L in the wrong orientation (from 22-to-1)




MCU_L in the right orientation (from 1-to-22)

# Our Solution to the problem





We used **jumper wires** to connect the MCU to the PCB in the correct orientation

# Testing & Debugging

1. **Verifying the output frequencies and the phase**

Our output:

- Square Wave
- 90 degree Phase

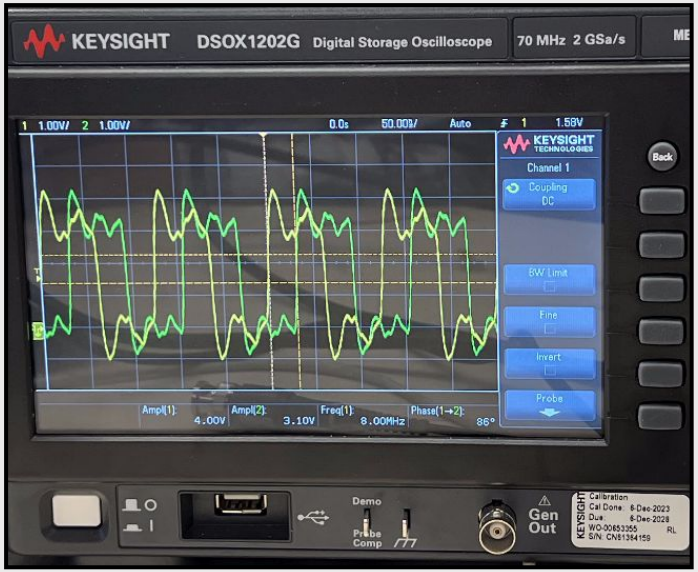
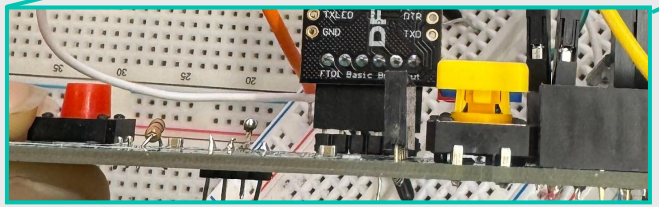



Figure 5. Output waveform of our subsystem.

# Testing & Debugging

## 2. Button Functionality

- Buttons worked on breadboard, but not on PCB.



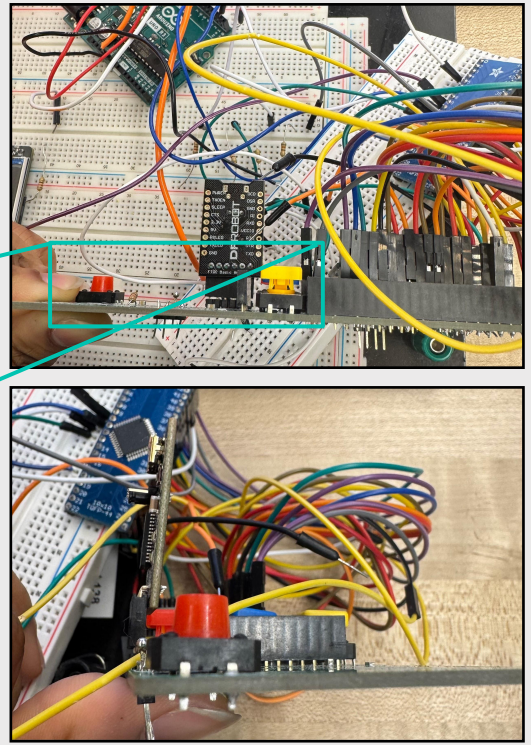



Figure 6. Image of our buttons on the PCB.

# Testing & Debugging

## 3. Verifying TXEN Signal

- The TXEN signal didn't toggle correctly when the TXEN button was pressed
- TXEN is by default on, but the radio should start with the signal off as it should be in RX mode

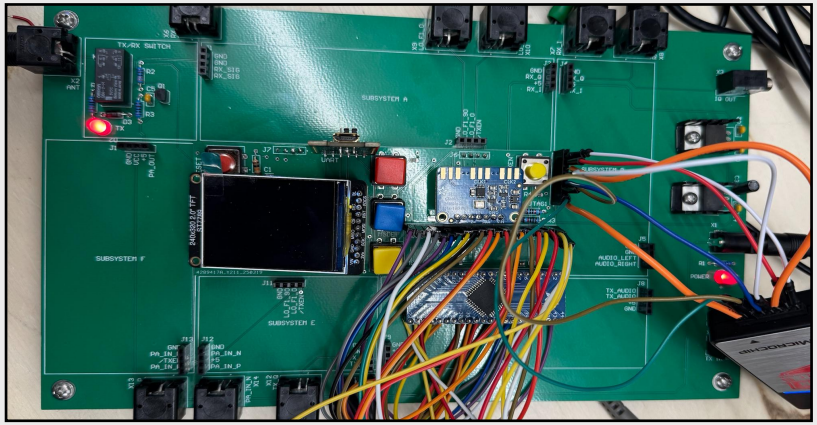


Figure 7. Image of our PCB inserted onto the mainboard. TX and power LEDS are on.

# Testing & Debugging

## 4. Verifying UART Functionality

- Serial Connection was established
- Incorrect strings were transmitted from our subsystem

```
PS C:\Users\zaraa\Downloads> python .\sub-c-cat.py
FA COMMAND
  CAT response:  . Expected: FA014074000;
  Result: FAIL
TX COMMAND
  CAT response:  . Expected: TX1;
  Result: FAIL
  CAT response:  . Expected: TX0;
  Result: FAIL
AI COMMAND
  CAT response:  . Expected: AI1;
  Result: FAIL
  CAT response:  . Expected: AI0;
  Result: FAIL
ID QUERY
  CAT response:  . Expected: ID0650;
  Result: FAIL
MD QUERY
  CAT response:  . Expected: MD0C;
  Result: FAIL
SH QUERY
  CAT response:  . Expected: SH0000;
  Result: FAIL
NA QUERY
  CAT response:  . Expected: NA00;
  Result: FAIL
IF QUERY
  CAT response:  . Expected: IF001014074000+000000C00000;
  Result: FAIL
ST COMMAND
  CAT response:  . Expected: ST1;
  Result: FAIL
  CAT response:  . Expected: ST0;
  Result: FAIL
```

Figure 8. Our design's response to UART testing.

# Testing & Debugging

## 5. Testing the Display

- Our display only worked with Arduino due to platform-specific drivers
- Power conflicts between the Arduino and the MCU

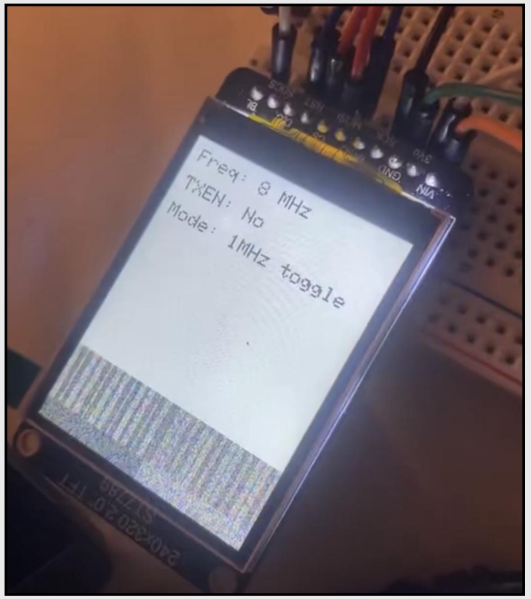
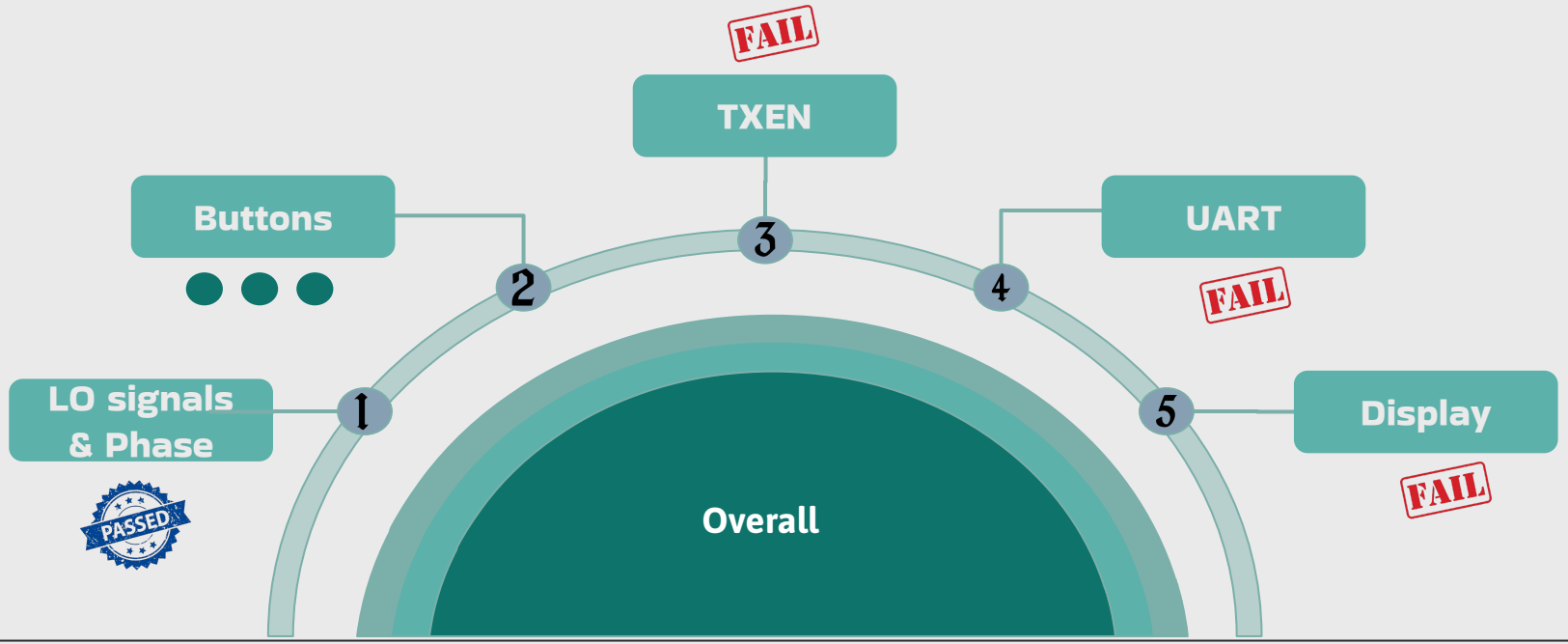



Figure 9. Display functioning with Arduino.

# How did our PCB  fair?



04

# Why did this Happen?

## And how we will fix it...

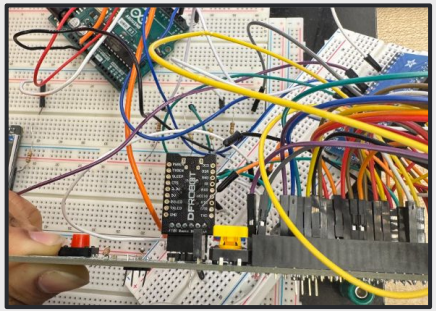
# Button Instability

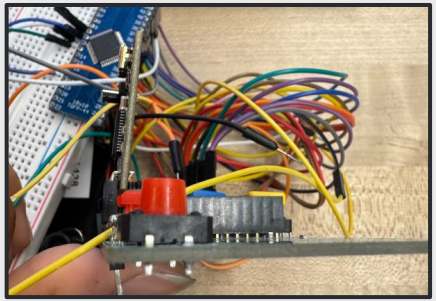
**Buttons worked on breadboard but failed after PCB assembly**

**Some buttons stopped responding after repeated use.**

**Likely due to poor contact from improper soldering or uneven mounting**

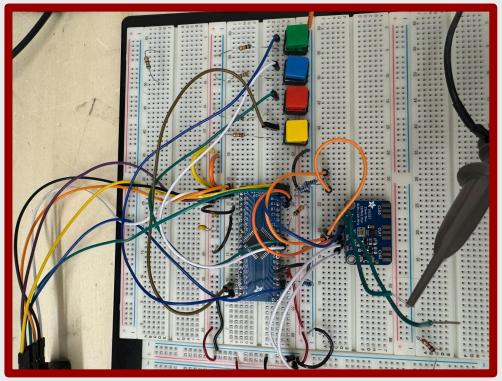
**RESET button (different type) works fine implies hardware issue**





# TXEN Toggling

**Problem traced to code — issue existed even on breadboard**



**After pressing the button, the TXEN signal toggled repeatedly before stabilizing**

**Likely a debounce or logic error in the enable signal handling.**

# How we tried to fix it...

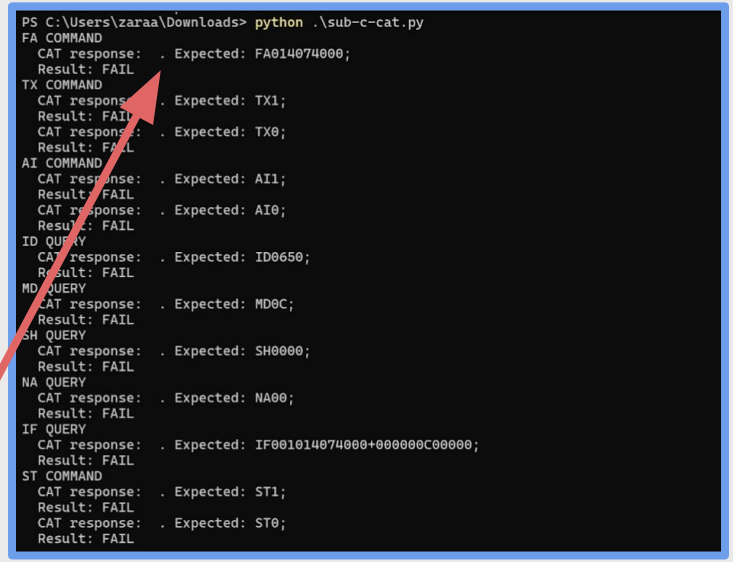
**We attempted to fix this by changing how we enable the output pin in our MCU code**

```
// Button-driven manual controls
if (b1_value)
    decFreq();
if (b2_value)
    incFreq();
if (!b3_value)
    toggle_txen(); // ADDED ! CONDITION.
if (b4_value)
    change_step_size();
```

```
/* Toggle TX/RX state via TXEN pin */
void toggle_txen(void)
{
    txen ^= 1;
    if (txen)
    {
        PORTD |= (1 << PD4); // TX enabled
    }
    else
    {
        PORTD &= ~(1 << PD4); // TX disabled
    }
}
```

# UART Incorrect Response

- System entered computer-controlled mode without a computer
- Fixed issue and established serial connection
- Python script ran, but responses were blank
- **RX** light flashed on command receive
- **TX** light flashed on sending blank response



# How we tried to fix it...

Tried fixing issue by switching UART to interrupt-based

Caused repeated interrupts → radio stuck in computer-controlled mode

Spent hours debugging, but issue remains unresolved

```
PS C:\Users\zaraa\Downloads> python .\sub-c-cat.py
FA COMMAND
  CAT response:  . Expected: FA014074000;
  Result: FAIL
TX COMMAND
  CAT response:  . Expected: TX1;
  Result: FAIL
  CAT response:  . Expected: TX0;
  Result: FAIL
AI COMMAND
  CAT response:  . Expected: AI1;
  Result: FAIL
  CAT response:  . Expected: AI0;
  Result: FAIL
ID QUERY
  CAT response:  . Expected: ID0650;
  Result: FAIL
MD QUERY
  CAT response:  . Expected: MD0C;
  Result: FAIL
SH QUERY
  CAT response:  . Expected: SH0000;
  Result: FAIL
NA QUERY
  CAT response:  . Expected: NA00;
  Result: FAIL
IF QUERY
  CAT response:  . Expected: IF00101407400O+000000C00000;
  Result: FAIL
ST COMMAND
  CAT response:  . Expected: ST1;
  Result: FAIL
  CAT response:  . Expected: ST0;
  Result: FAIL
```

# Display Complexity

4311 display by Adafruit

We chose a more complex display

## Therefore . . .

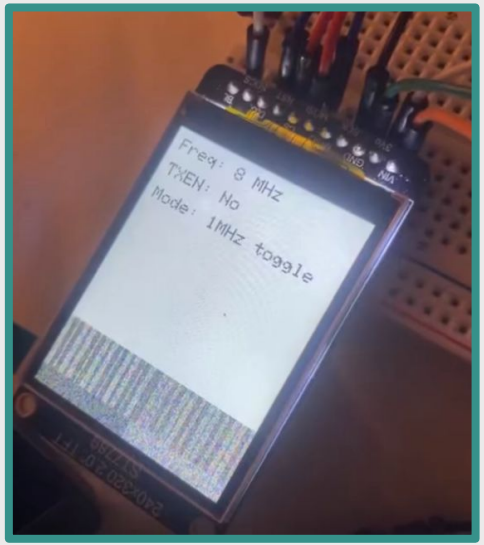
We had to write our own drivers in order for our microcontroller to properly write to the display.

# How we tried to get the display to work...

**Drive the display using external Arduino MCU**

**Attempted serial connection between Arduino and MCU**

**Sent string with frequency and TXEN state**



# Back to recommendations:

Therefore... ALTHOUGH we **do not recommend** our current state of final design...

**With future implementations ...**

We would recommend it!

As we put key focus on...

**EASE OF USE**

**ACCURACY**

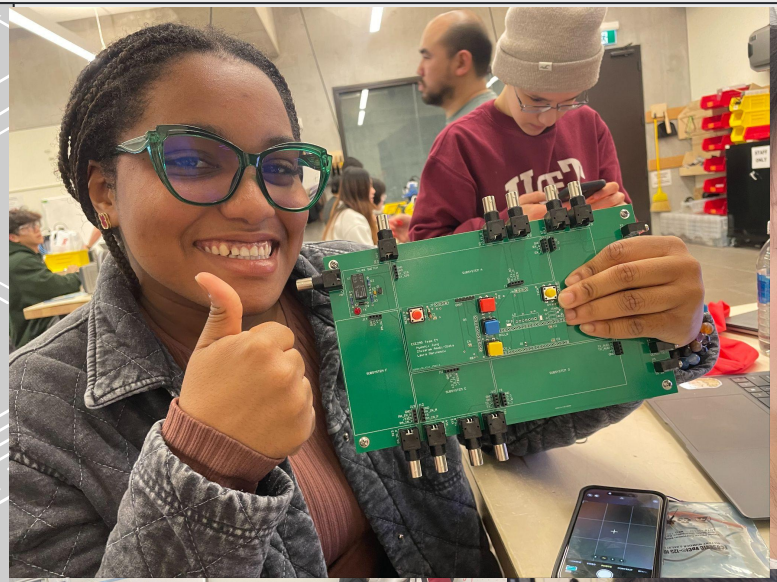
# What's Next?

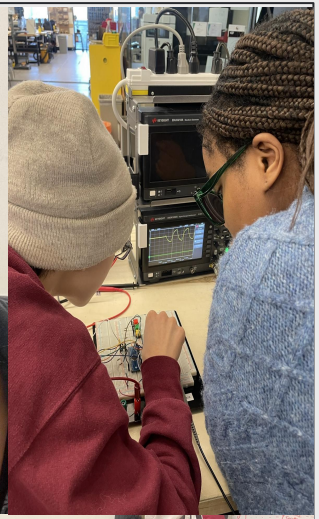
We propose the following three future implementations...

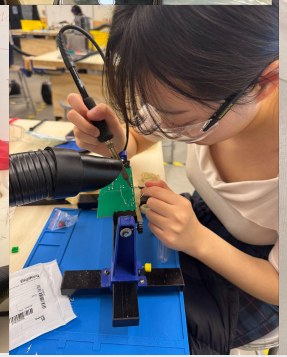
1. Correcting MCU orientation

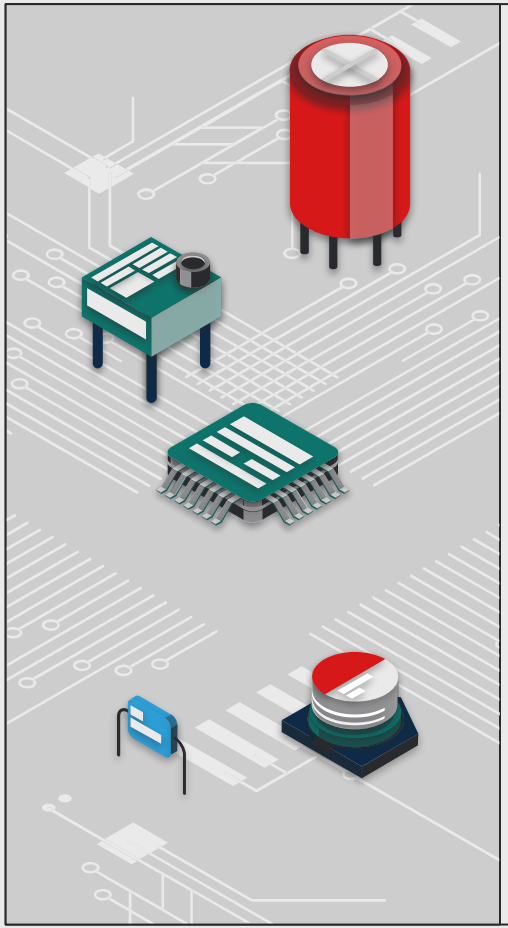
2. Updating Button Components

3. Simplifying Display Module

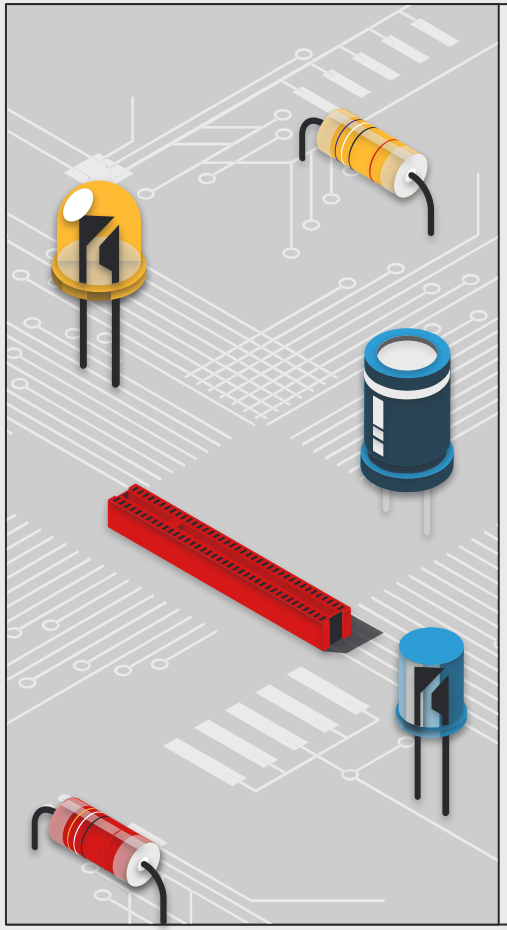






# Thank you for so much for listening!



# Q&A

# References


[1] S. V. Hum, "Flexible Radio Transceiver (FLRTRX) Interface Control Document," ECE295: Hardware Design and Communication, v4.0. [Online]. Available: https://www.ab4oj.com/sdr/sdr_handbook.pdf. [Accessed: Jan. 27, 2025].

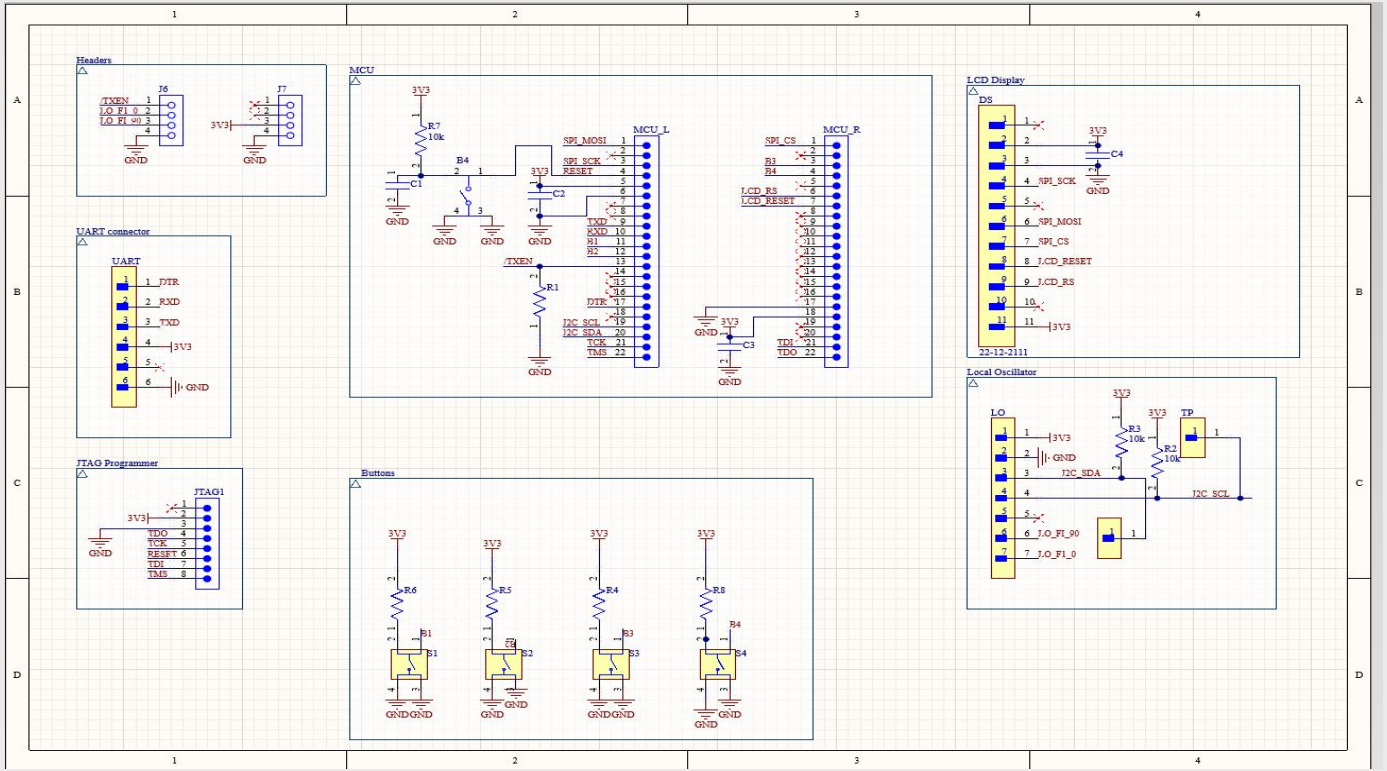
[2] [P] Adafruit Industries, “Adafruit Si5351A Clock Generator Breakout Board - 8KHz to 160MHz,” *Adafruit.com*, 2023. https://www.adafruit.com/product/2045 [Accessed: Jan 31, 2025].

[3] Adafruit Industries LLC, "4311," *Digi-Key*, [Online]. Available: https://www.digikey.ca/en/products/detail/adafruit-industries-llc/4311/10313914. [Accessed: 7-Feb-2025].

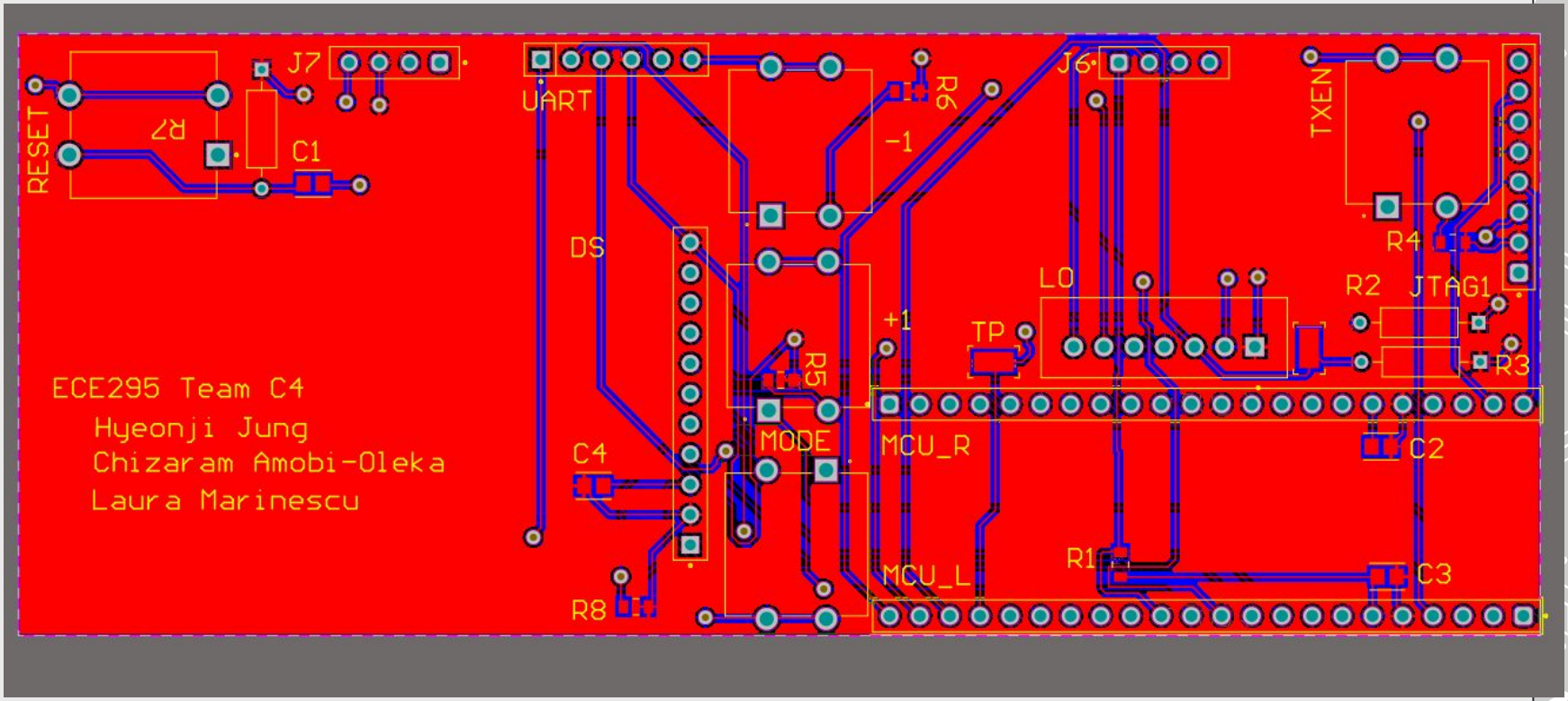
[4] Adafruit Industries LLC, "1010," *Digi-Key*, [Online]. Available: https://www.digikey.ca/en/products/detail/adafruit-industries-llc/1010/7244937. [Accessed: 7-Feb-2025].

[5] DFROBOT, "FTDI FT232RL," *Fermion*, [Online]. Available: https://www.dfrobot.com/product-147.html. [Accessed: 16-Mar-2025].

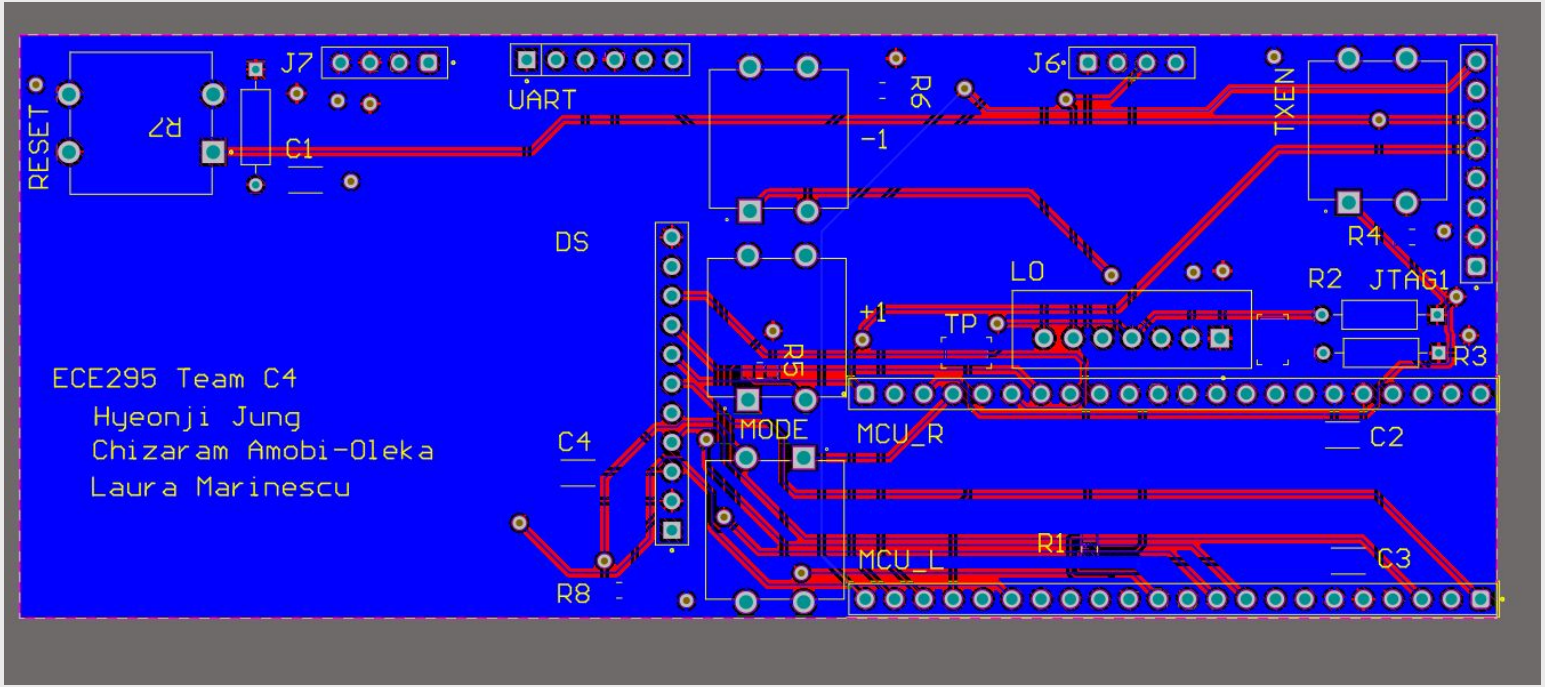
# Appendix A. Full Design schematic



# Appendix B. Top Layer of PCB



# Appendix C. Bottom Layer of PCB



# Appendix D. PLL Code

Code:
https://drive.google.com/file/d/1UUOpbFbok4EbWRMaIdT1hzmln-BDpvUP/view?usp=sharing

```
/* Configure Si5351 Frequency */
void configure_si5351(uint32_t frequency)
{
    uint32_t divider = PLL_FREQUENCY / frequency;
    uint32_t frac_num = PLL_FREQUENCY % frequency;
        uint32_t frac_den = frequency;

    setup_PLL(SI5351_PLL_A, PLL_FREQUENCY / SI5351_XTAL_FREQ, 0, 1);
    if(frac_num == 0){
    setup_clock(SI5351_PLL_A, SI5351_PORT0, divider, 0, 1);
    setup_clock(SI5351_PLL_A, SI5351_PORT1, divider, 0, 1); // ADDED
LINE FOR PORT1
    }else{
    setup_clock(SI5351_PLL_A, SI5351_PORT0, divider, frac_num,
frac_den);
    setup_clock(SI5351_PLL_A, SI5351_PORT1, divider, frac_num,
frac_den); // ADDED LINE FOR PORT1
    }

    set_phase(90);
    enable_clocks(true);
}

/* Increase Frequency */
void incFreq(void)
{
    if (output_frequency + step_size <= 16000000UL)
    {
        output_frequency += step_size;
        configure_si5351(output_frequency);
    }
}

/* Decrease Frequency */
void decFreq(void)
{
    if (output_frequency - step_size >=
8000000UL)
    {
        output_frequency -= step_size;
        configure_si5351(output_frequency);
    }
}

/* Change Frequency Step Size */
void change_step_size(void)
{
    precision_mode = (precision_mode + 1) %
4;
    switch (precision_mode)
    {
    case 0:
        step_size = 1000000UL;
        break; // 1 MHz
    case 1:
        step_size = 100000UL;
        break; // 100 kHz
    case 2:
        step_size = 10000UL;
        break; // 10 kHz
    case 3:
        step_size = 1000UL;
        break; // 1 kHz
    }
}
```