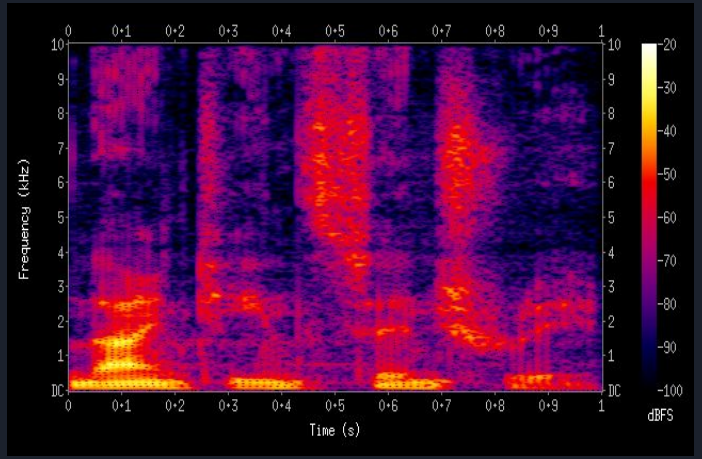
Team 7
Etch-A-Tune

Joshua Consenz, Jack McMahon, Cullen Sharp, Edward Tsoi

Background



A creative way of producing audio waveforms. Experimental Musical Artists and Sound Engineers are always trying to find new and cool sounds.

Etch-A-Tune spectrogram editing and production right into your pocket

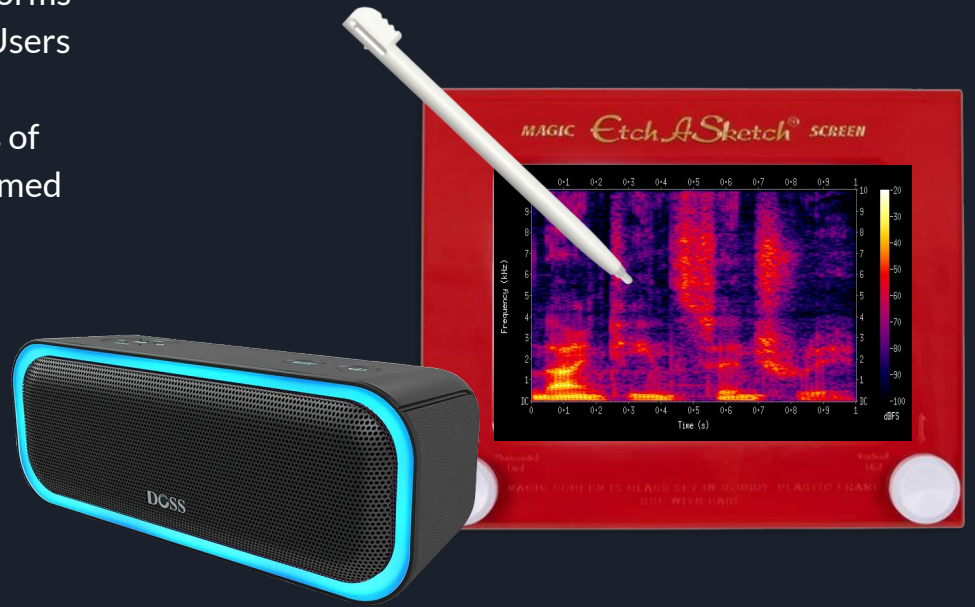
It makes generating sound simple, easy, and accessible to people of all ages and skill levels.

Closest alternatives would be a full analog synth, expensive and complex audio software, or an arbitrary waveform generator. Or something that combines all 3.

Essentially, it’s a cool toy to annoy everyone around you with

Our Approach

Concept of operations: Our project is a handheld embedded system that transforms hand-drawn spectrograms into sound. Users interact with a touchscreen interface to sketch time–frequency representations of audio, then those drawings are transformed into sound.



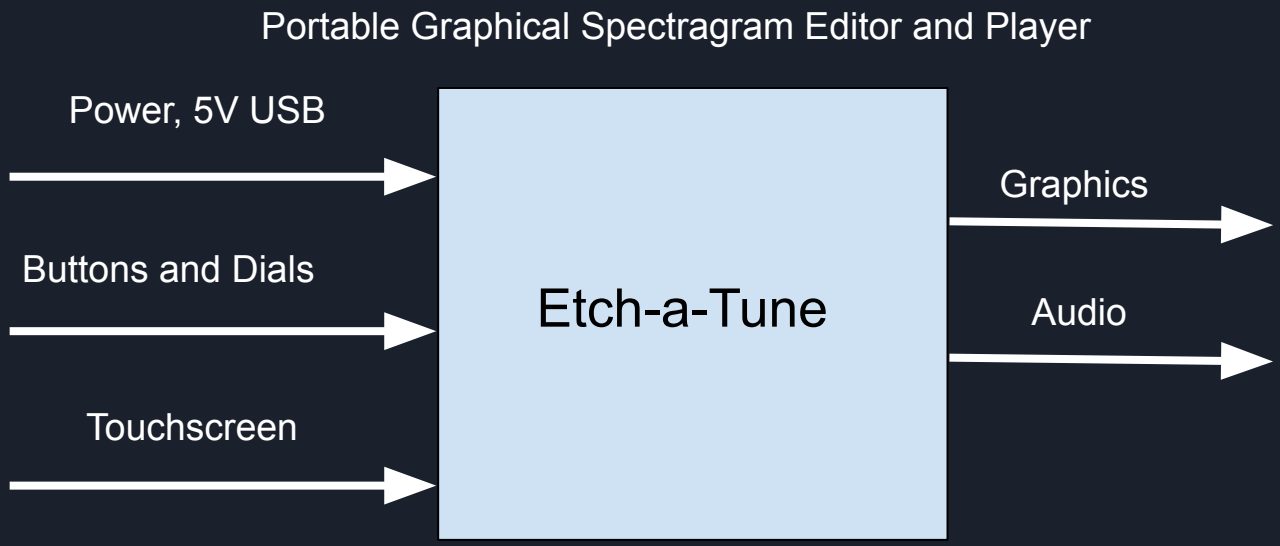
Project Specifications

1. Must
   a. Device can be recharged
   b. Device can take touch screen input
   c. Device has buttons
   d. Device uses a touch screen to convert a drawing into an audio output
   e. Device is a handheld form factor
   f. Device is portable
   g. Device is able to last one hour of use on battery
   h. Device is able to produce audio signals
   i. Device is able to switch between recording and playing mode
2. Should
   a. Device allows a spectrogram to be the drawn and processed
   b. Device is smaller than 220cm^3
   c. Device weighs less than 230g
   d. Device has a volume control knob
   e. Device has a frequency scaling knob
3. May
   a. Device is comfortable to use
   b. Device has a graphical user interface
   c. Device produces CD quality audio

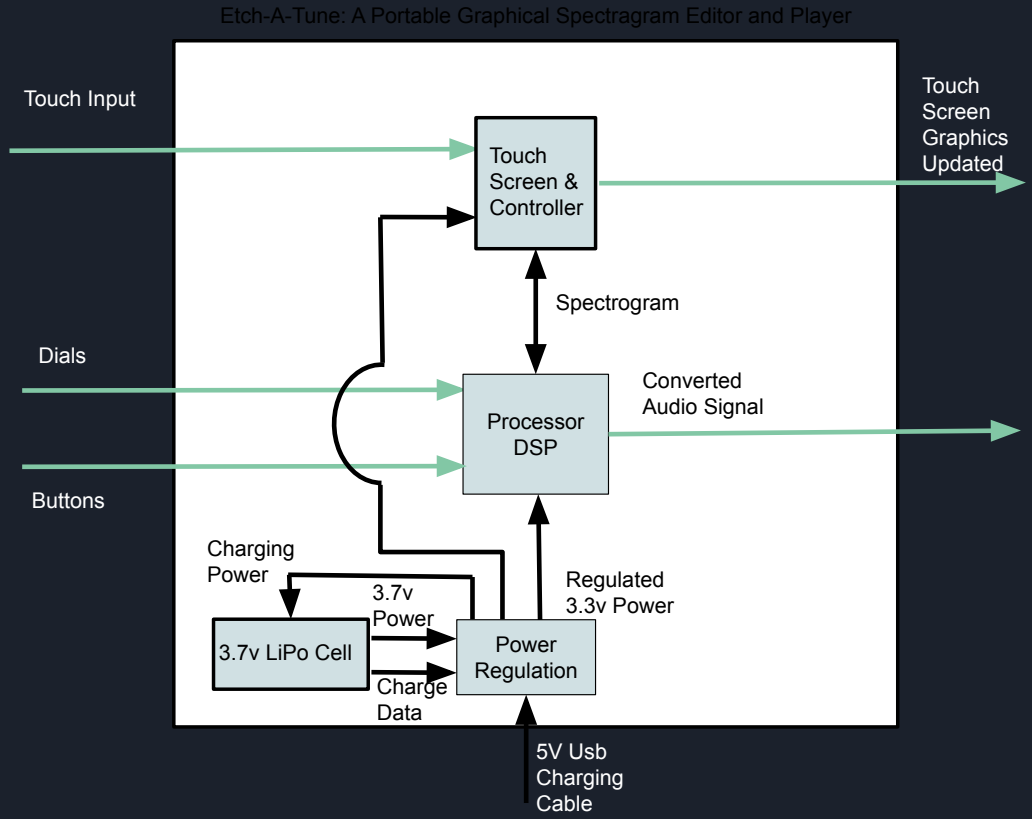
Design Overview

- Hardware
- Software
- Enclosure

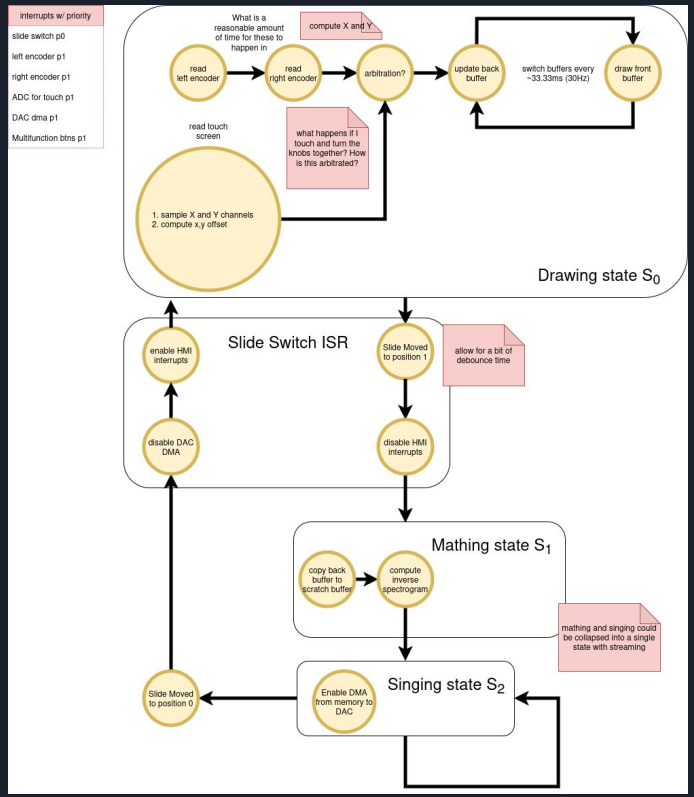
L0 Block Diagram



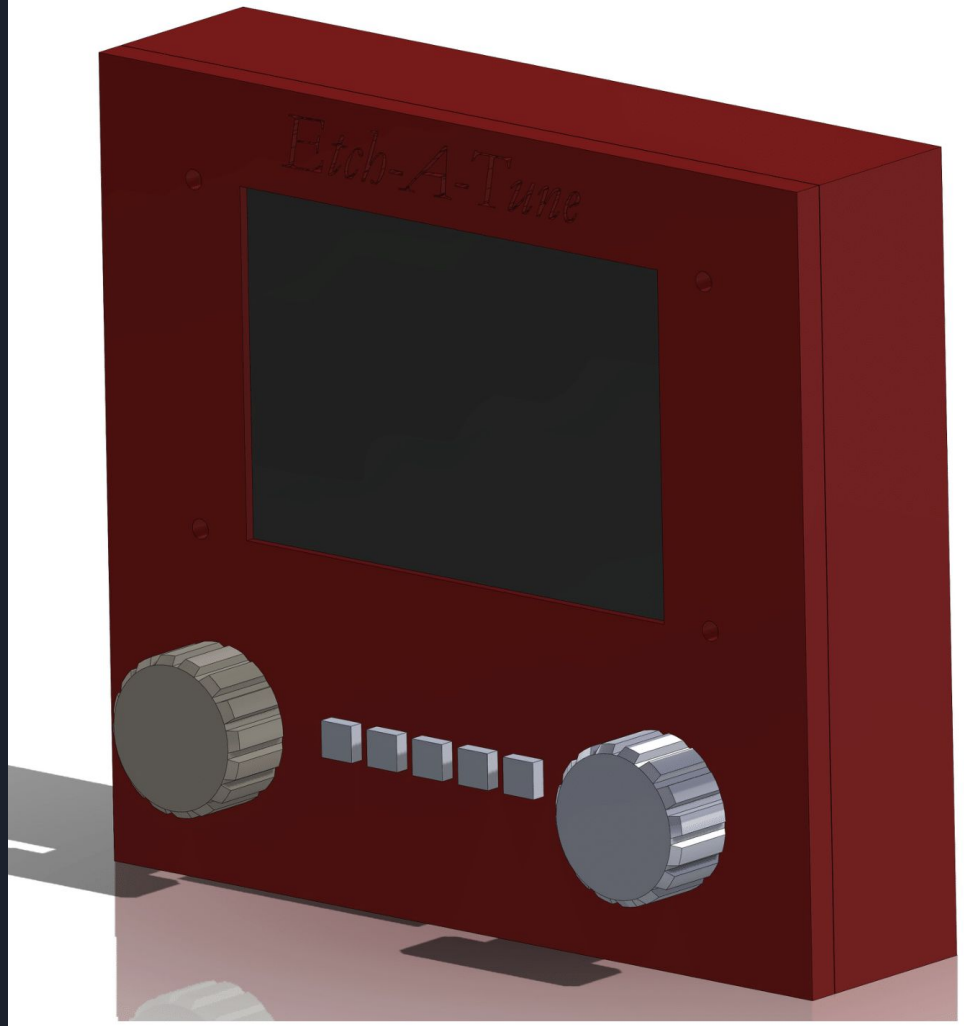
L1 Block Diagram



Rev 0 Firmware architecture



3D Modeling



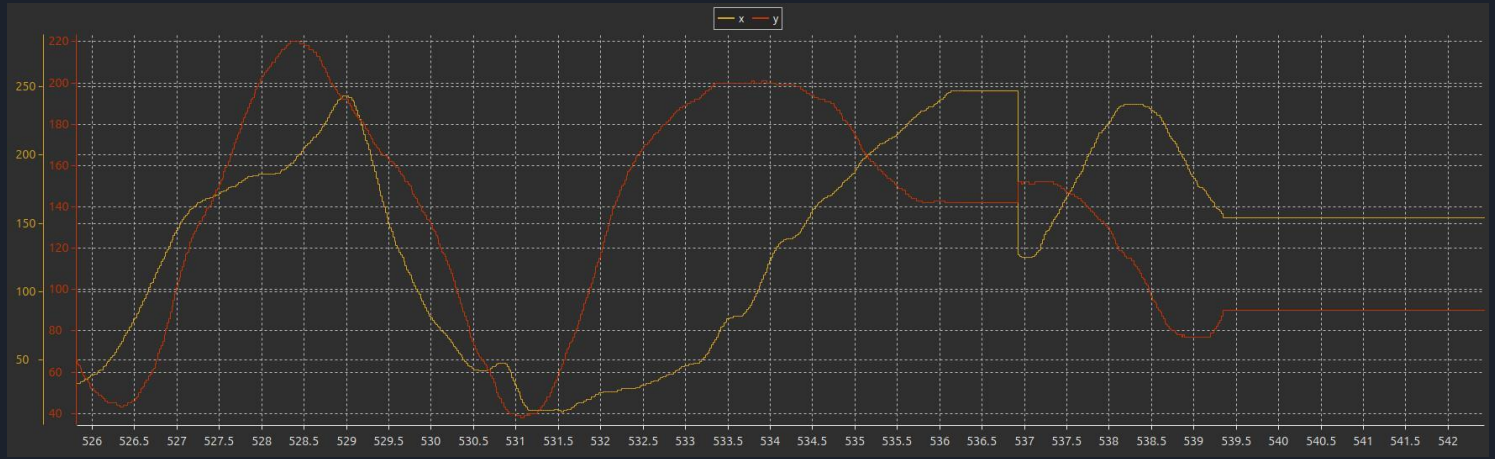
Implementation

- Software
- Hardware

Software

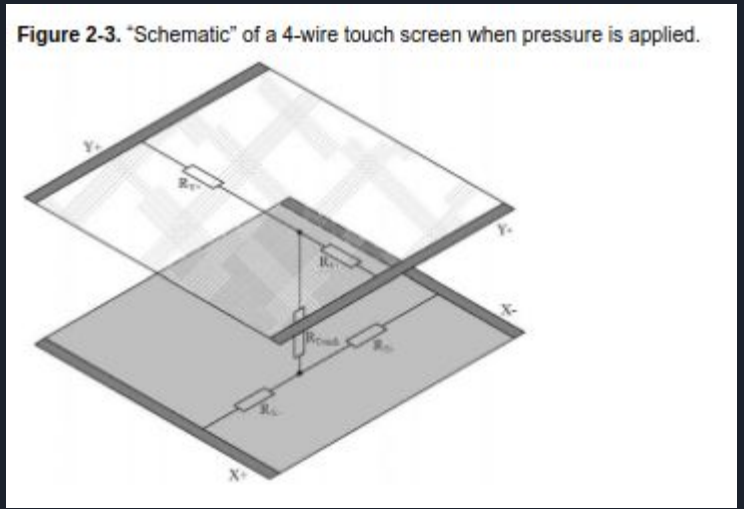
Software Tools Used

- STM32CubeIDE
- STM32CubeMX
- JLink SEGGER Software
- FreeRTOS



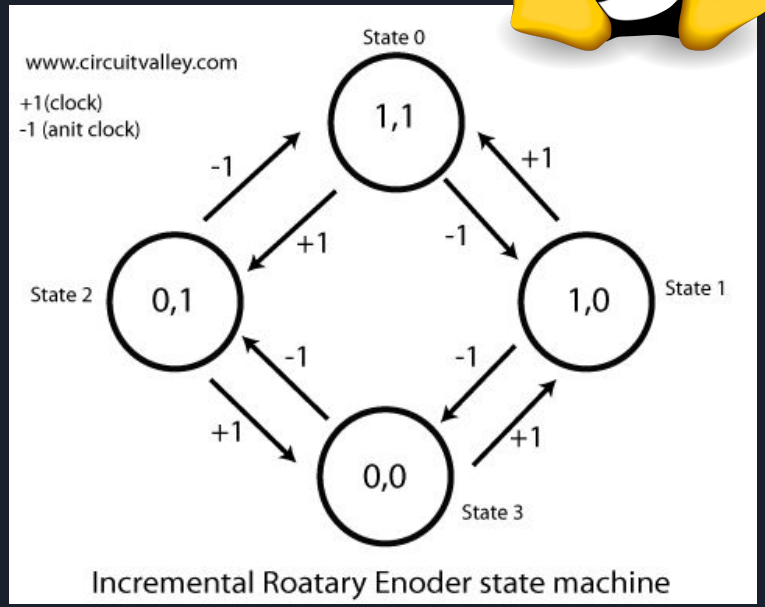
Touch and Backlight Screen Drivers

- Touch screen is a generic resistive touch screen driver that is adapted from the Adafruit library
- This is an analog component, and we had to deal with a log of noise from the LCD
  - Median filter
  - Settling delay



Rotary encoder driver

- Utilizes aspects of object oriented programming
- Adapted from a linux kernel driver
- Utilizes a statemachine



Multithreading

DSP

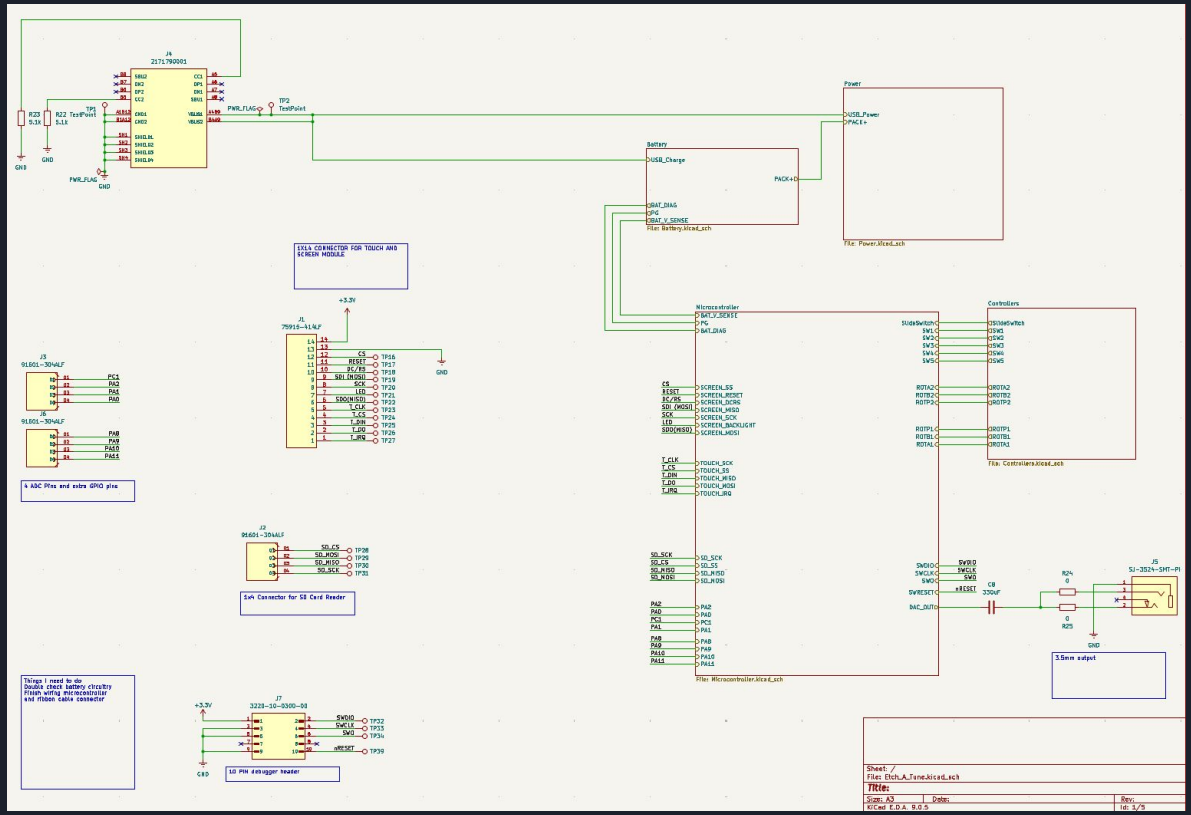
- We used the CMSIS DSP Library
- We didn’t get to this part.

Hardware

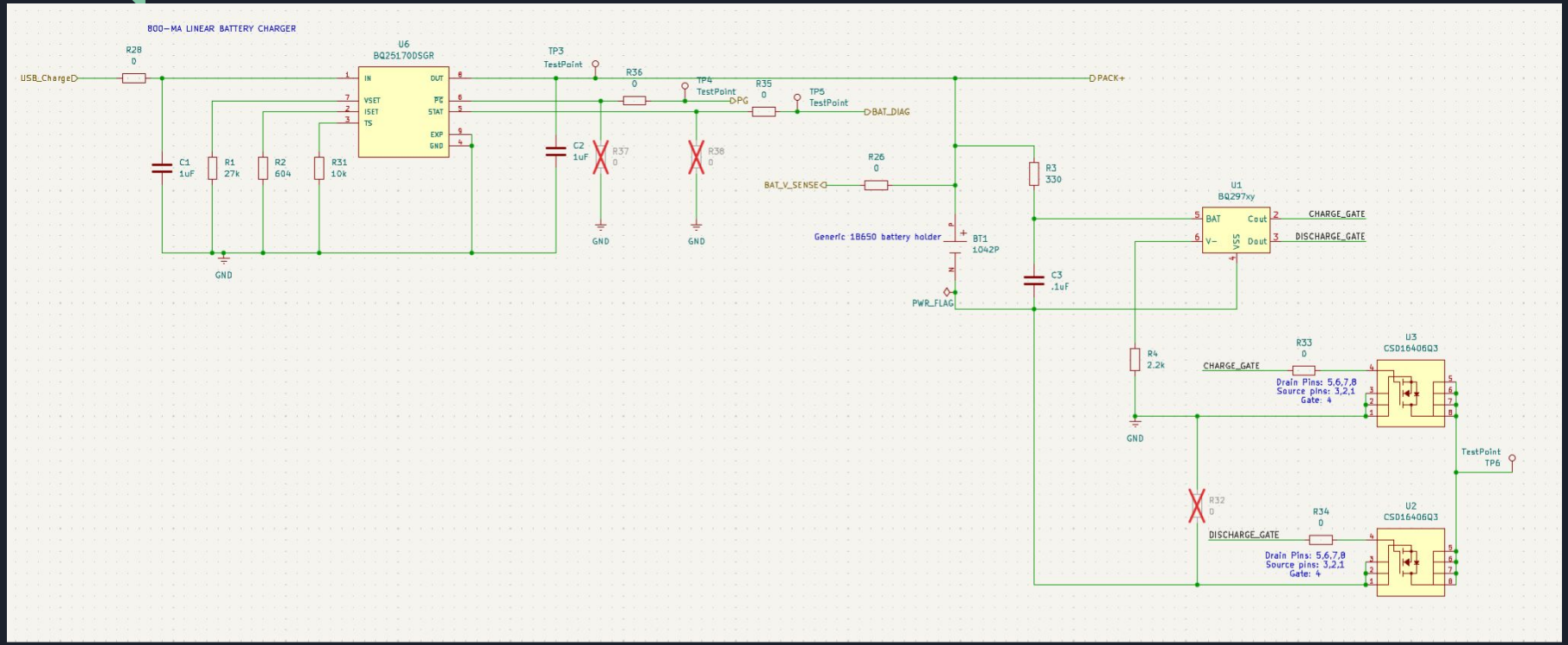
Hardware Tools used

- Altium
- Kicad
- Saturn PCB Design Toolkit
- LTSpice
- Samacsys Library Loader
- Oscilloscope
- Digital Multimeters
- DC Power Supplies
- J-Link Segger
- SolidWorks

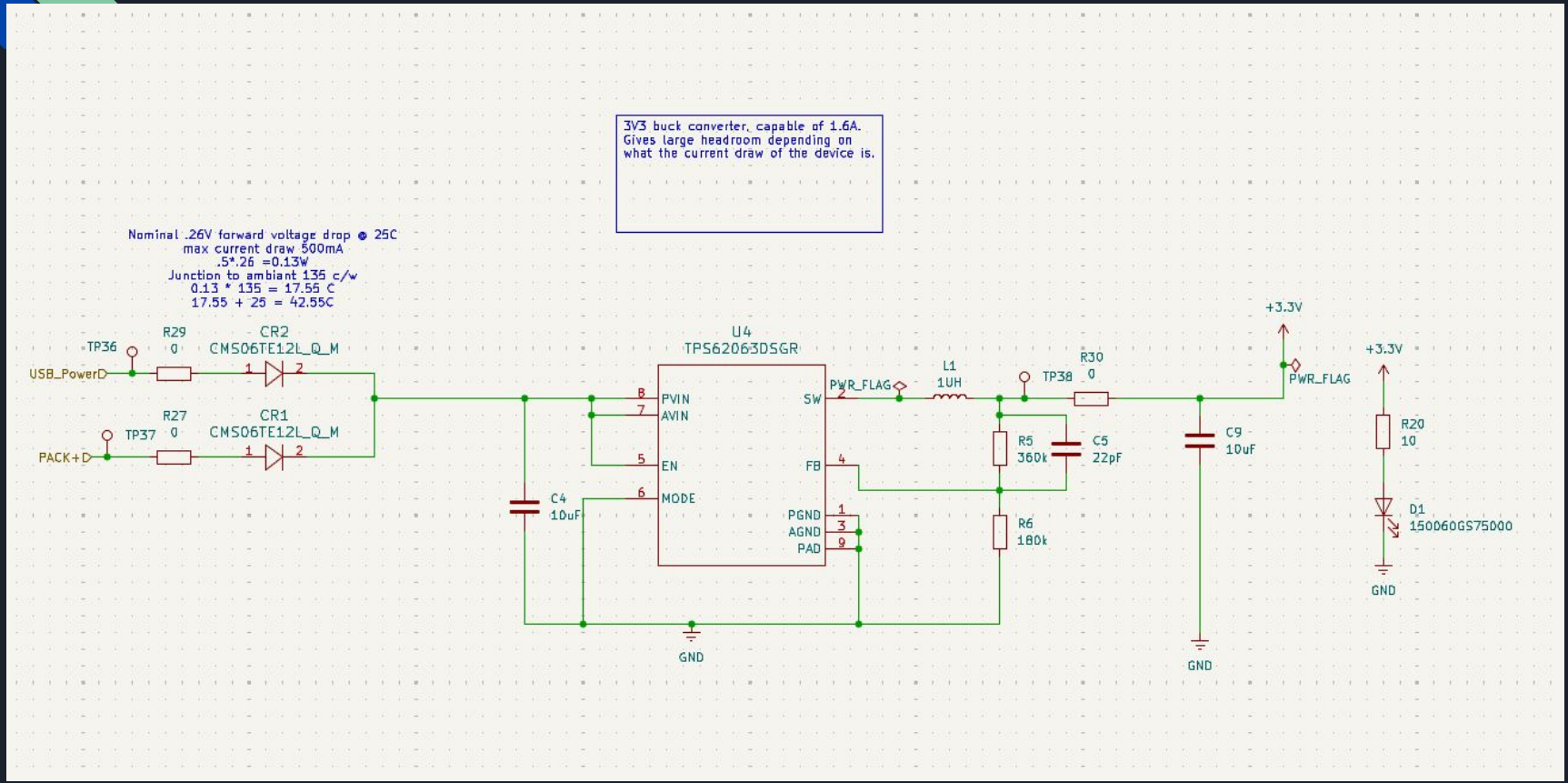
Hierarchy-view Schematic



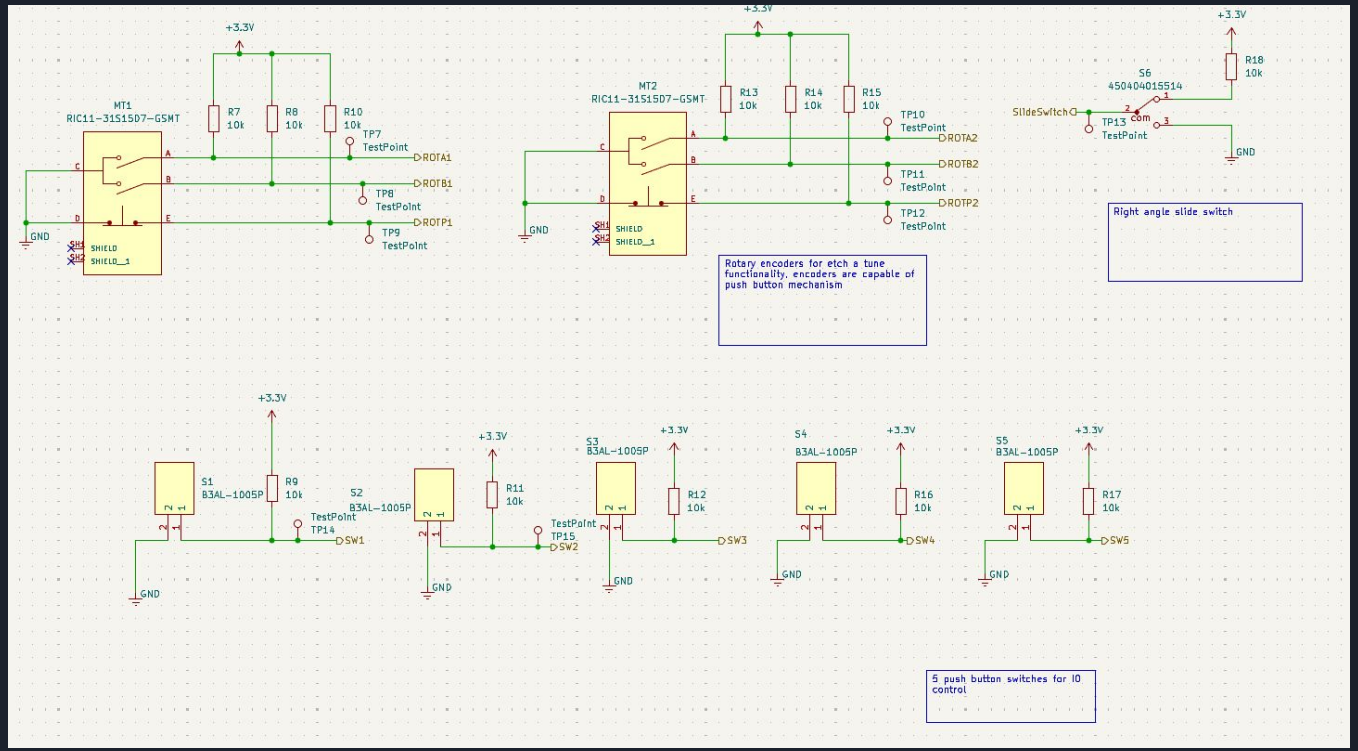
Battery Charger



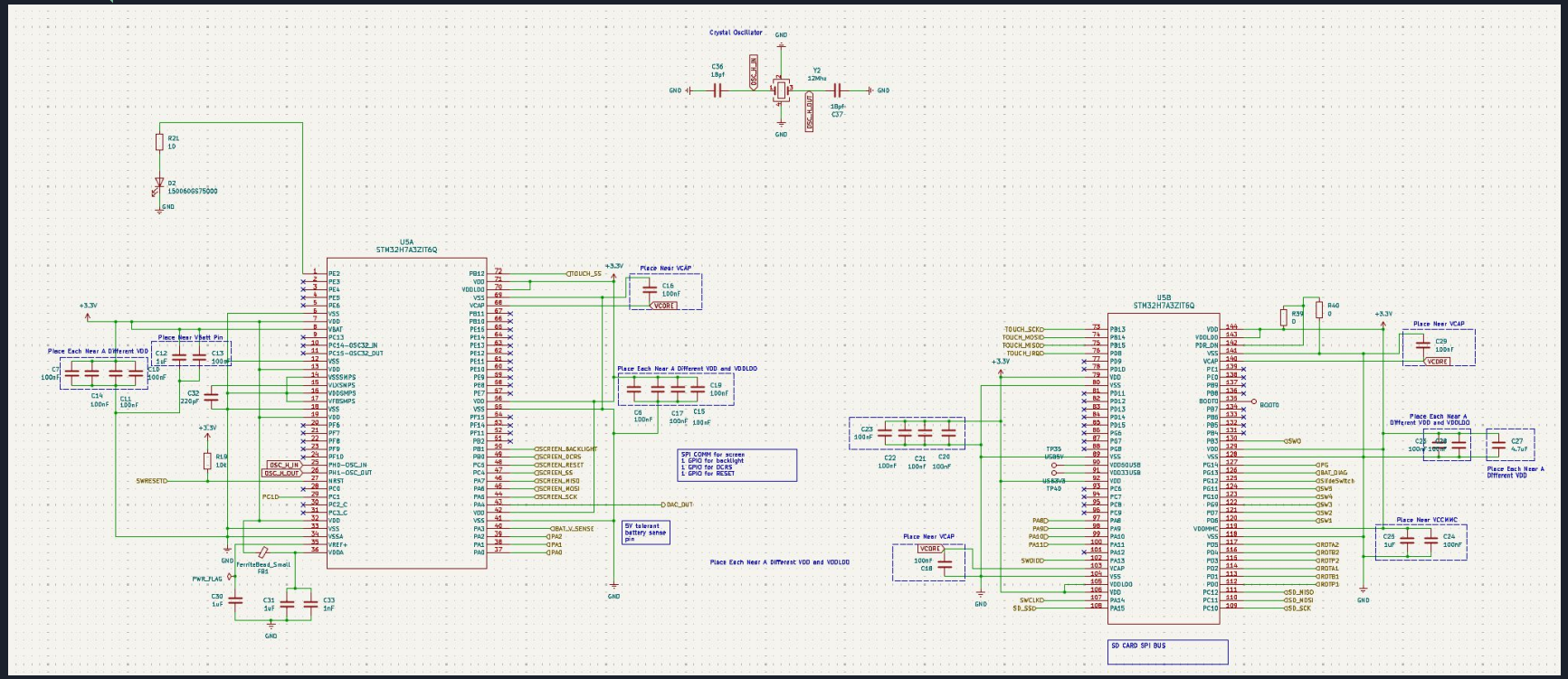
3.3 Volt Switching Mode Power Supply



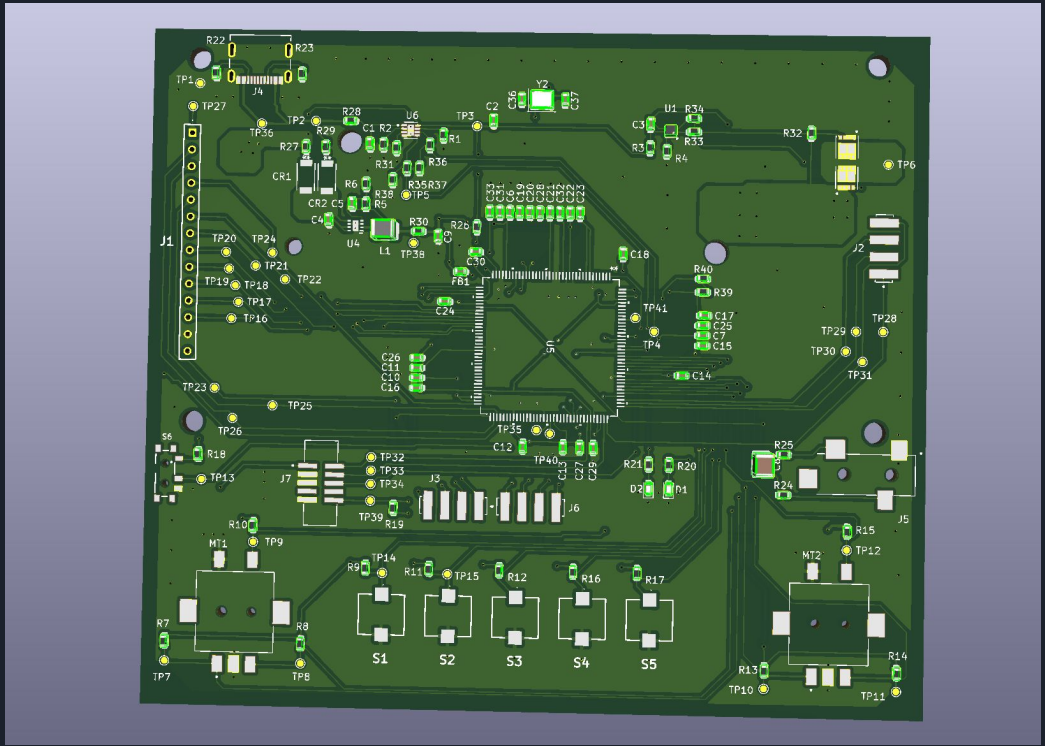
IO (Buttons and Rotary Encoders)



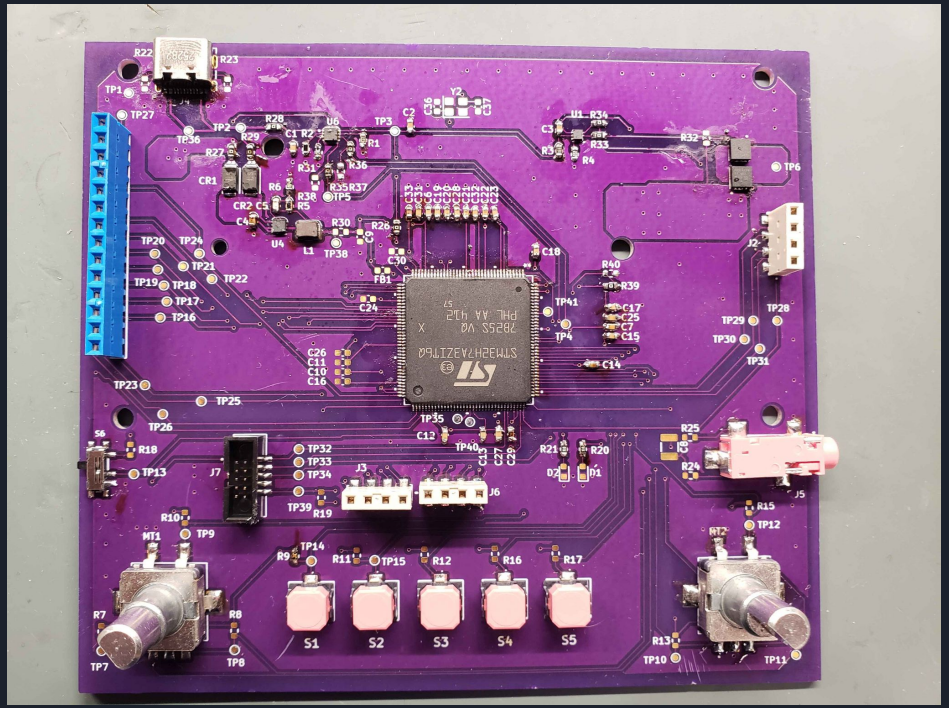
Microcontroller



Final PCB CAD



Built Board!



Total BOM Cost Per Unit

ece411-t07-etch-a-tune

Order Item	Quantity	Item Cost	Total
PCBs	3	$91.85	$91.85
Super Swift Service			
Medium Run			
2 oz copper, 0.8mm thickness			
After Dark (Black Substrate + Clear Mask)			

Service Coupon Apply +

Remove from Cart | Board Preview | Sub Total: $91.85

Software IP used

- RobertoBenjami “stm32_hal_graphics_display_drivers” Github repo
  - Built off of the Adafruit stm32 display code
- STMicro Example Code
- Adafruit touch screen driver
- Rotary encoder driver from the Linux Kernel

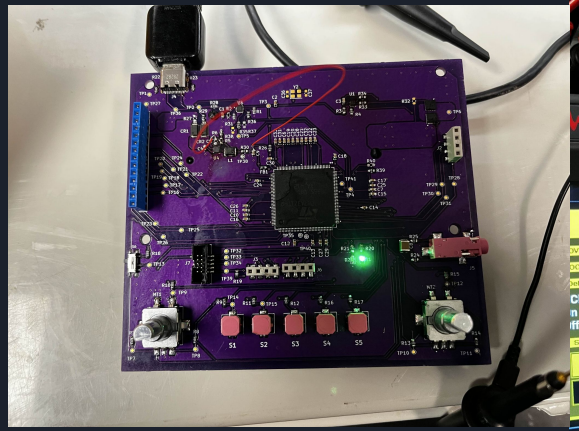
Hardware IP used

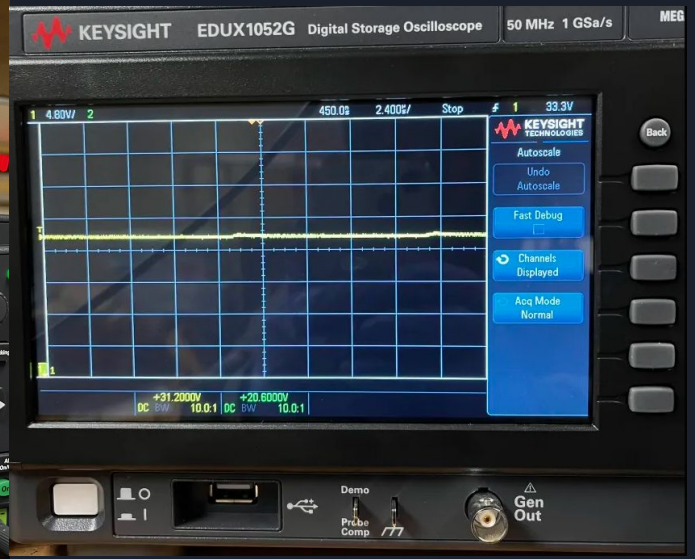
- Texas Instruments Datasheets
- STMicro Datasheets

Testing

- Microcontroller Communication with Host Computer
- Battery Management System
- USB Power Passthrough
- 3.3V Rail
- Touch Screen

3.3 USB/Battery Power





Connecting to the STM32H7A3 PCBA

```
Connecting to target via SWD
ConfigTargetSettings() start
ConfigTargetSettings() end - Took 36us
InitTarget() start
SWD selected. Executing JTAG -> SWD switching sequence.
DAP initialized successfully.
InitTarget() end - Took 10.7ms
Found SW-DP with ID 0x6BA02477
DPIDR: 0x6BA02477
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[3]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x84770001, ADDR: 0x00000000)
AP[1]: AHB-AP (IDR: 0x84770001, ADDR: 0x01000000)
AP[2]: APB-AP (IDR: 0x54770002, ADDR: 0x02000000)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FE000
CPUID register: 0x411FC271. Implementer code: 0x41 (ARM)
Cache: L1 I/D-cache present
Found Cortex-M7 r1p1, Little endian.
FPUnit: 8 code (BP) slots and 0 literal slots
CoreSight components:
ROMTbl[0] @ E00FE000
[0][0]: E00FF000 CID B105100D PID 000BB4C7 ROM Table
ROMTbl[1] @ E00FF000
[1][0]: E000E000 CID B105E00D PID 000BB00C SCS-M7
[1][1]: E0001000 CID B105E00D PID 000BB002 DWT
[1][2]: E0002000 CID B105E00D PID 000BB00E FPB-M7
[1][3]: E0000000 CID B105E00D PID 000BB001 ITM
[0][1]: E0041000 CID B105900D PID 001BB975 ETM-M7
[0][2]: E0043000 CID B105900D PID 004BB906 CTI
I-Cache L1: 16 KB, 256 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 16 KB, 128 Sets, 32 Bytes/Line, 4-Way
Memory zones:
  Zone: "Default" Description: Default access mode
Cortex-M7 identified.
J-Link>
```

```
Read register 'xpsr' (4 bytes) from hardware: 0x00000001
Read 4 bytes @ address 0x080025B4 (Data = 0xD038F8DF)
Read 2 bytes @ address 0x080025B4 (Data = 0xF8DF)
Read 2 bytes @ address 0x080025B6 (Data = 0xD038)
Reading register 'msp' = 0x24003F98
Reading register 'psp' = 0x00000000
Reading 64 bytes @ address 0x08001980
Read 2 bytes @ address 0x08001990 (Data = 0xF44F)
Setting breakpoint @ address 0x08001990, Kind = 2, Type = THUMB, BPHandle = 0x0001
Starting target CPU...
ERROR: Could not start CPU core. (ErrorCode: -1)
ERROR: Cannot read register 15 (R15) while CPU is running
Reading common registers: ERROR: Cannot read register 0 (R0) while CPU is running
Read register 'r0' (4 bytes) from hardware: 0xEFBEADDE
ERROR: Cannot read register 1 (R1) while CPU is running
Read register 'r1' (4 bytes) from hardware: 0xEFBEADDE
ERROR: Cannot read register 2 (R2) while CPU is running
Read register 'r2' (4 bytes) from hardware: 0xEFBEADDE
ERROR: Cannot read register 3 (R3) while CPU is running
```

Conclusion

What worked? How well? What didn't work? Why?
Include data and graphics/charts/plots where appropriate.

What Worked Well!

- Power rails!
- Buttons and Dials!
- USB Passthrough!
- Building the PCB and Soldering the board!
- Battery Management System!
- Screen integration with PCBA

What went Terribly Wrong!

- Integration - programming chip
- PCBs (Just like everybody else! We should have just ordered from JLCPCB. TOUGH LUCK LISTENING TO ANDREW!)
- Not enough test points for the size of our chip
- Meeting MVP requirements and deadlines
- Failure to work as a proper team
- Parallelism (the lack thereof)
- Storming, very storm, throughout the whole term

Contributions

- Joshua Consenz
  - 3D Modeling
  - Soldering
  - Integration
  - Documentation
  - Deliverables
- Jack McMahon
  - Assisted with 3D CAD Designs
  - Getting V0.1 Software Example working
  - Assisted with Schematic (CPU Power Plane)
  - Assisted With Assembly and Debug
  - Final BOM Part selection and purchasing
  - Writing deliverables
- Edward Tsoi
  - Hardware Component Selection
  - Hardware Architecture
  - Power Budgeting
  - Thermal Budgeting
  - Battery Management System
  - Schematic
  - PCB ECAD
  - Soldering
  - LTSpice Simulations
  - Kicad Component Footprint/Library
- Cullen Sharp
  - Touchscreen interface
  - V0.1 Touch Driver Addresses
  - V1.1 Demo Code

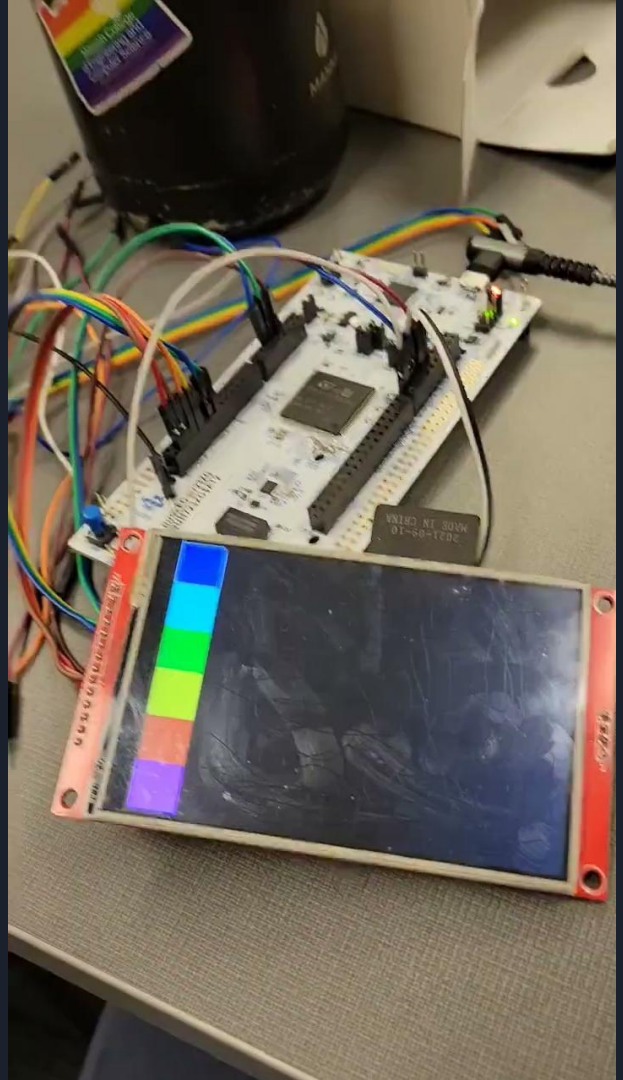
Team Lessons Learned

- Adjust your expectations for a 2 credit class
- Beware feature creep
- Properly use GitHub

Individual Lessons Learned

- Joshua Consenz
  - SMD Soldering
  - Hardware Troubleshooting
  - Communicating in a project
- Jack McMahon
  - Don't take 3 Master's Courses with Capstone
  - Keep communication styles in mind when forming a team
  - Learned about KiCAD
  - Learned more about STM32
- Edward Tsoi
  - Choose hardware that everyone is familiar with so everyone can participate (build and debug PCBA)
  - Remove SMPS with LDO, change power rails to just USB and remove the battery
  - Help out with the software
  - Pick larger parts
- Cullen Sharp
  - It takes longer than you expect to get up and running with a new chip family
  - Learned a lot about C and programming stm32 chips


SONY



Collaboration Site

https://github.com/CullenSharp/Etch-A-Tune