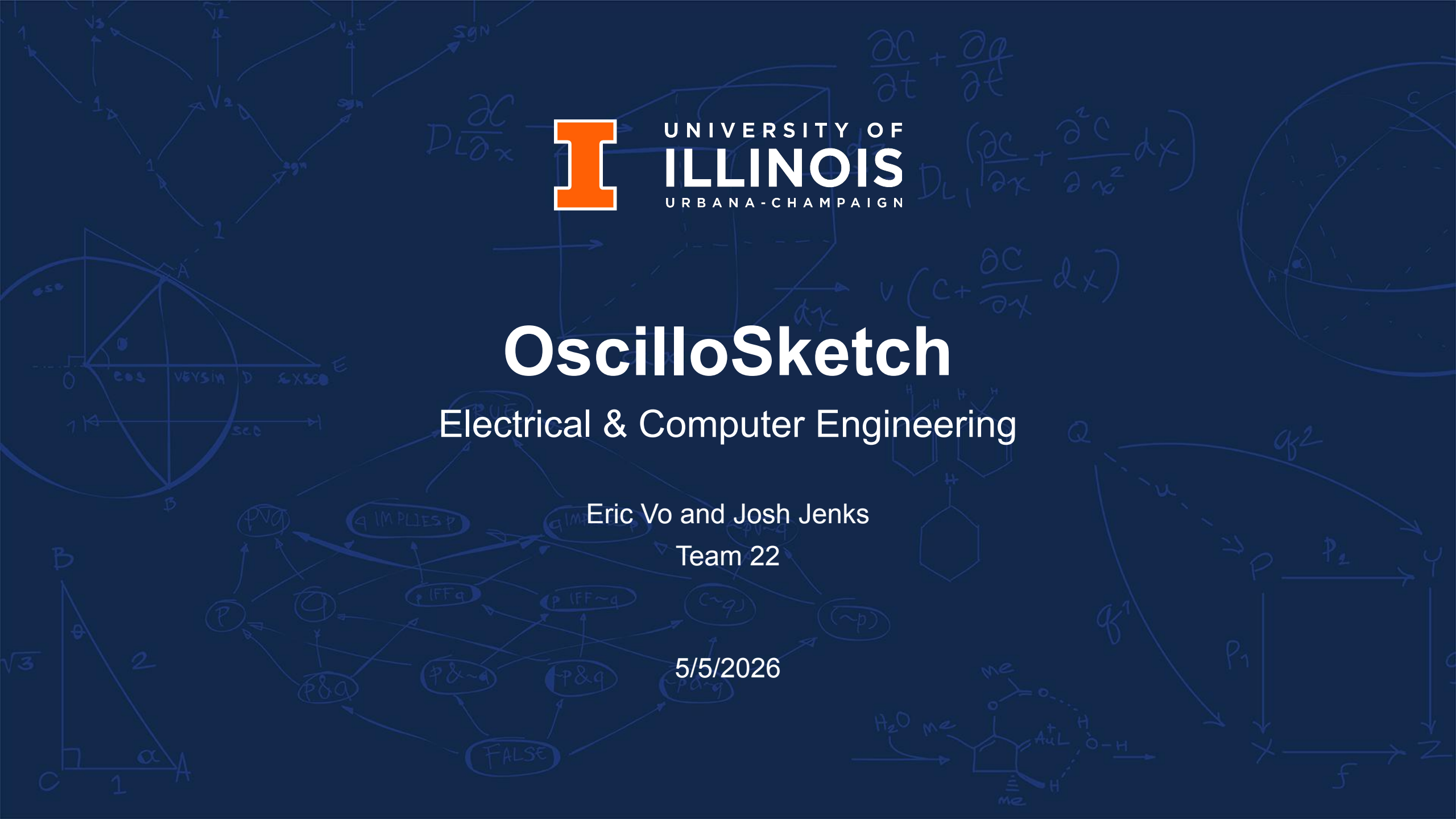


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

OscilloSketch

Electrical & Computer Engineering

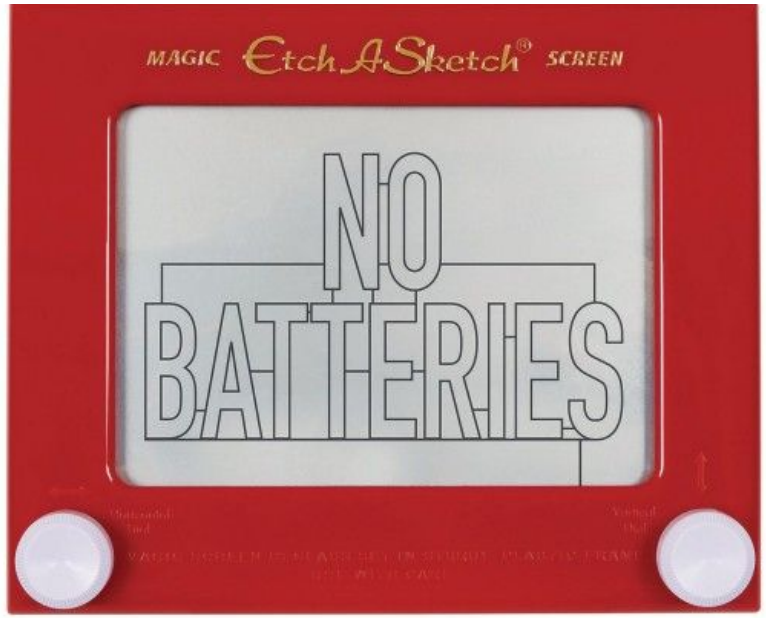
Eric Vo and Josh Jenks

Team 22

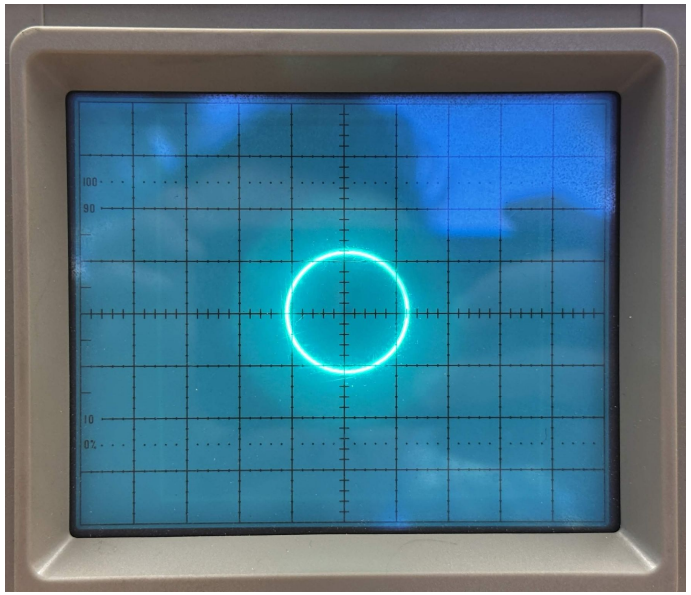
5/5/2026

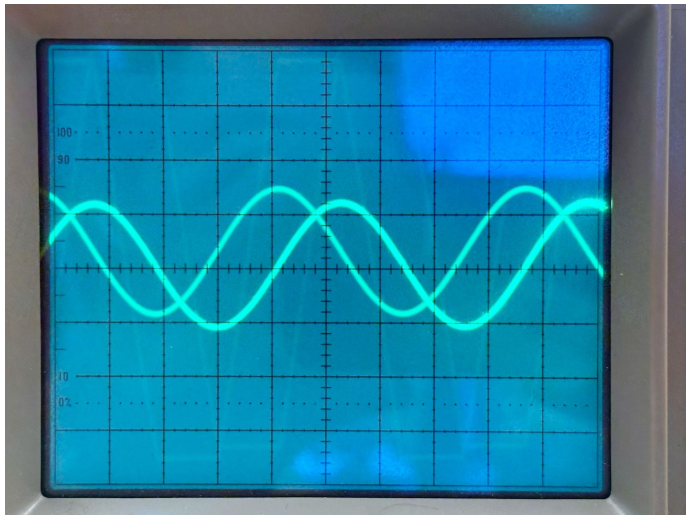
Introduction to OscilloSketch

- Portable embedded device for **oscilloscope XY mode**
- Inspired by the kids drawing toy **Etch-e-Sketch**
- Produces **synchronized analog X/Y outputs**
- User controls drawings with two encoders and four buttons
- Displays drawings, preset shapes, and audio visualization
- Uses **USB-C power** and protected analog outputs



Purpose



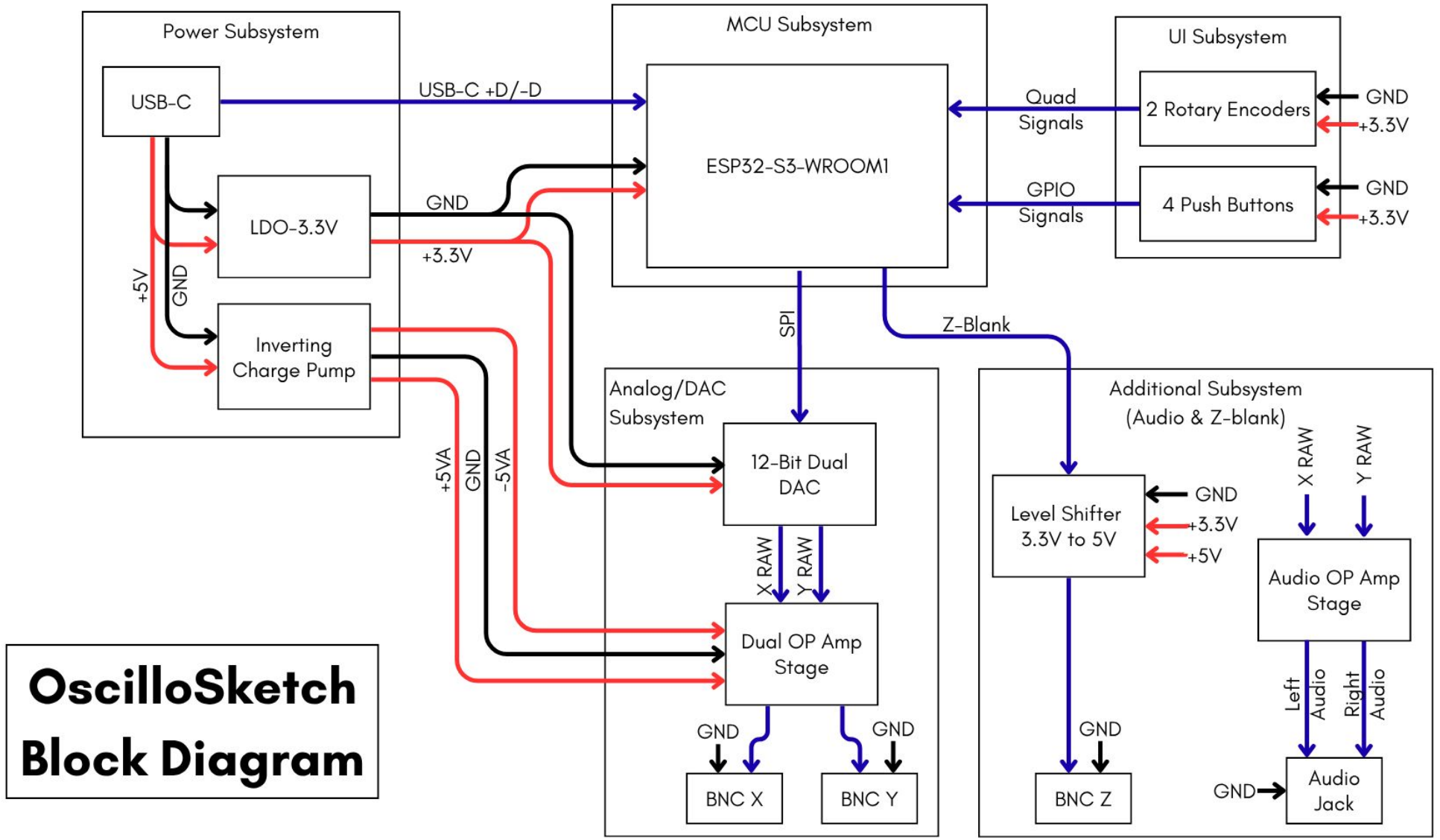


What problem do we solve?

OscilloSketch addresses the lack of an intuitive, interactive way to learn XY oscilloscope behavior and signal relationships.

- Oscilloscope XY mode is difficult for beginners to understand from static lecture examples alone
- Our system converts paired analog signals into live vector drawings, making X/Y signal interaction visually obvious
- Interactive modes let users explore how waveform shape, synchronization, and filtering affect the resulting display
- This makes signal behavior more engaging and easier to learn in a hands-on way

Hardware Design

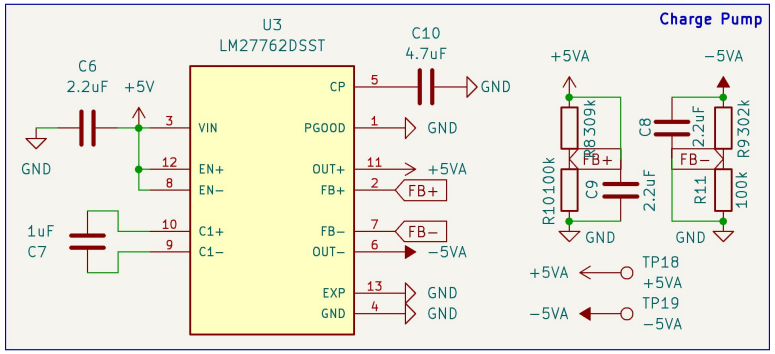


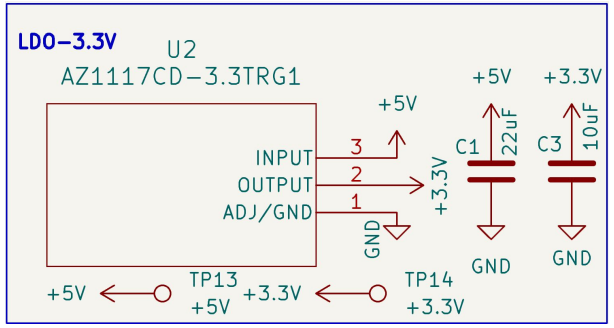
High Level Requirements

- **Analog Output Range and Resolution**
  - Generate two independent analog outputs at least 12-bit resolution and an output range of approximately ±5 V for oscilloscope
- **USB-C Power Operation**
  - Operate from a single 5 V USB-C input while drawing no more than 500 mA during normal operation
- **Real-Time User Control**
  - Utilizes two rotary encoders to control the X/Y cursor position with less than 20 ms input-to-output latency
- **Smooth XY Display Update**
  - Update the coordinated X/Y outputs at a rate of at least 10,000 points per second for smooth oscilloscope motion

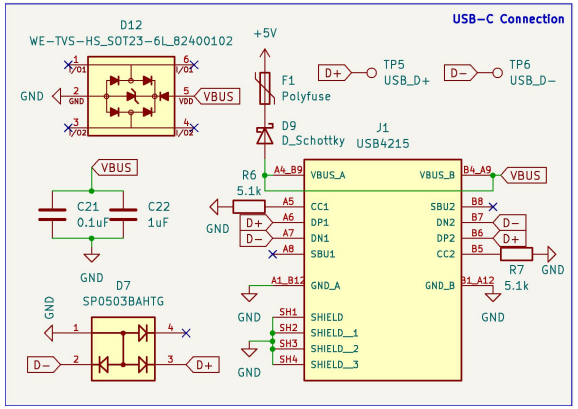
Power Subsystem

- **USB-C** provides the main 5 V input for power
- **AZ1117C LDO** converts 5 V to a regulated 3.3 V rail
- **LM27762 Charge Pump** generates +5 VA and −5 VA rails
- **Input protection** Polyfuse, Schottky diode, and TVS diodes



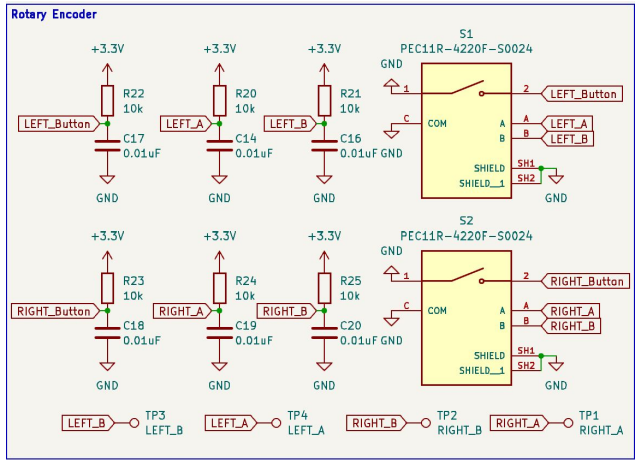


Criterias	Required	Actual
USB-C connector Voltage	4.75V to 5.25V	5.23V
Voltage Output from Charge pump	±5 ∓ 0.3V	+4.98 -4.87
Voltage Output from LDO	~3.3V	3.285V

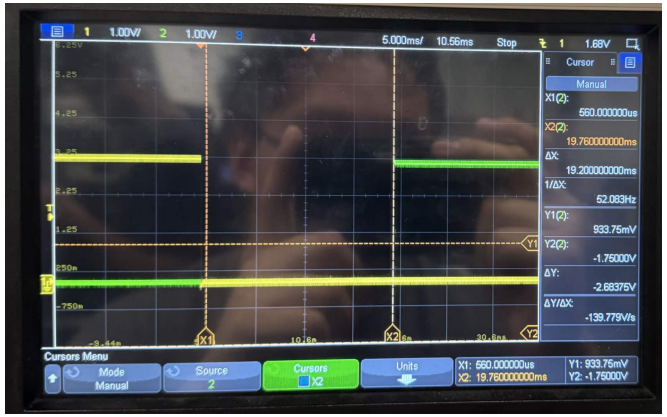


UI Subsystem

- **Two rotary encoders** control X and Y movement
  - Outputs quadrature A/B signals for CCW or CW
- **Four push buttons** provide functions up, down, right and left
- Buttons use **active-high inputs** with 10 kΩ pull-ups
- Encoder signals use pull-ups and **RC filtering** for debouncing

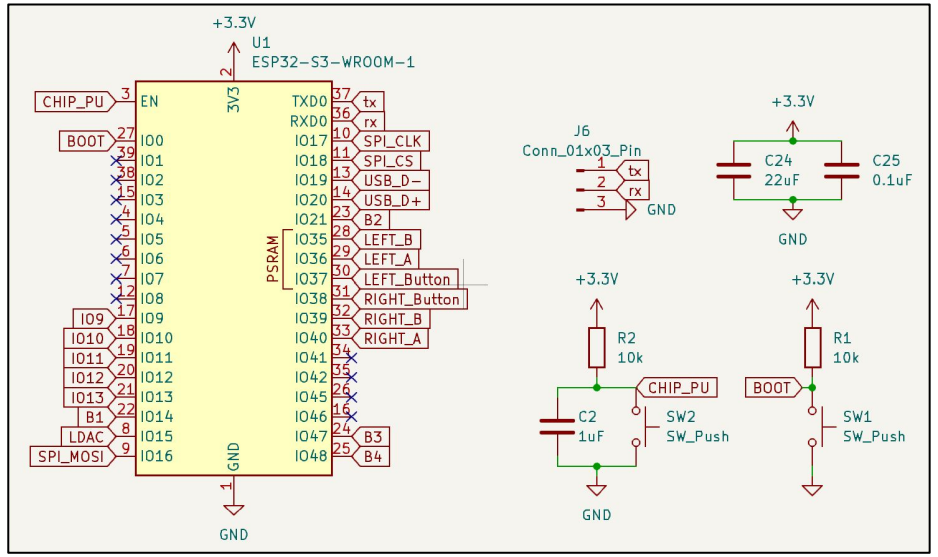


Criterias	Required	Actual
X encoder voltage change per 20	3.9 V ± 0.2 V	4.06 V
Y encoder voltage change per 20	3.9 V ± 0.2 V	3.86 V
Button response latency	≤ 20 ms	19.76 ms



MCU Subsystem

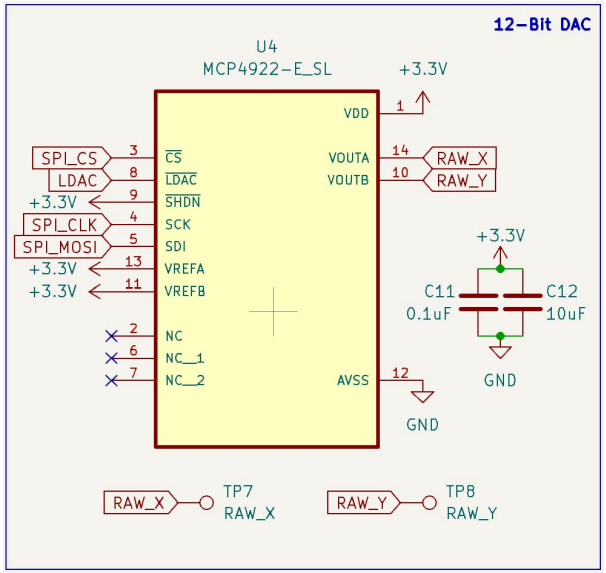
- **ESP32-S3-WROOM-1-N16** is MCU for this project
  - 16MB Flash memory, **Dual Core**, 240 MHz
  - **Native USB** for programming and serial debug
- **3.3 V rail** powers the MCU and digital logic
- Outputs SPI and LDAC to dual DAC

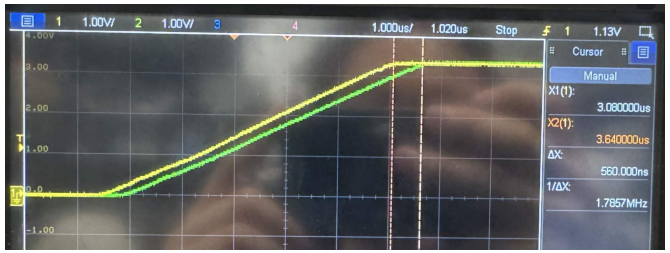


Criterias	Required	Actual
XY update rate	≥ 10,000	110 kHz
SPI speed for DAC updates	≥ 5 MHz	20 MHz
Firmware reliability	No freeze for 5 minutes	No freeze

DAC Subsystem

- **MCP4922** dual 12-bit DAC generates RAW_X and RAW_Y
  - Fast Settling Time of 4.5 µs -> 222 MHz
- SPI transfers X/Y data from the ESP32-S3
- **LDAC synchronizes** both DAC outputs
- **0–3.3 V** DAC outputs with **4096 steps** (12 bit)

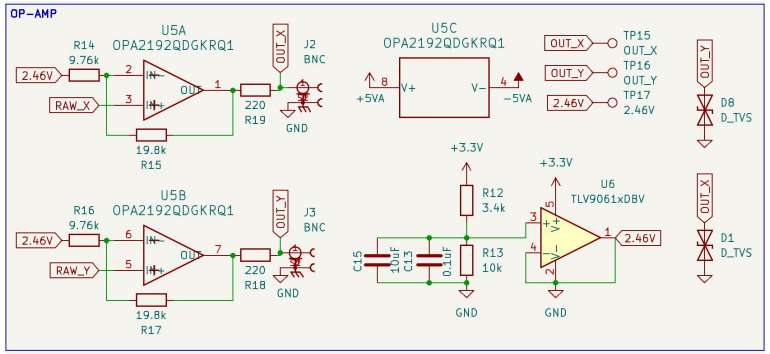




Criterias	Required	Actual
RAW_X output range	~0 V to 3.3 V	0.002 to 3.30 V
RAW_Y output range	~0 V to 3.3 V	0.001 to 3.29 V
X/Y update synchronization	Minimal delay	560 ns

Analog Subsystem

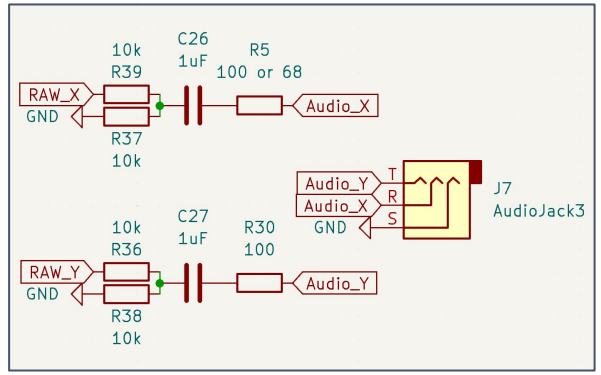
- Converts DAC outputs from **0–3.3 V into ±5 V** for XY mode
- **OPA2192 dual op-amp** to scale and shift DAC outputs
  - $Vout = (1 + \frac{Rf}{Rref}) \cdot Vin - (\frac{Rf}{Rref}) \cdot Vref = 3.0287 \cdot Vin - 4.996$
- Buffered **2.46 V reference** is required for the level-shift
- Series resistors and TVS diodes protect the outputs

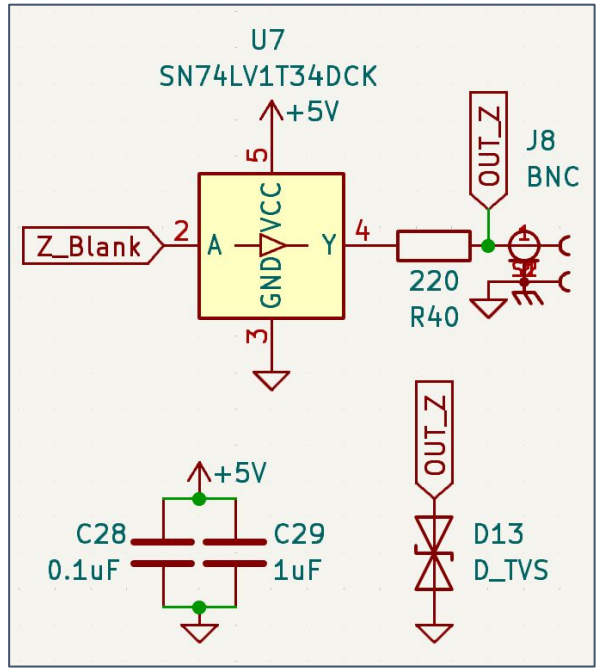


Criterias	Required	Actual
X & Y BNC center voltage	±0.10 V of 0	~0.0022 V
OP AMP output voltage swing	>+4.5V <-4.5V	−4.88 to +4.99V
Output step size	2.44 mV ±15%	2.41 mV/code

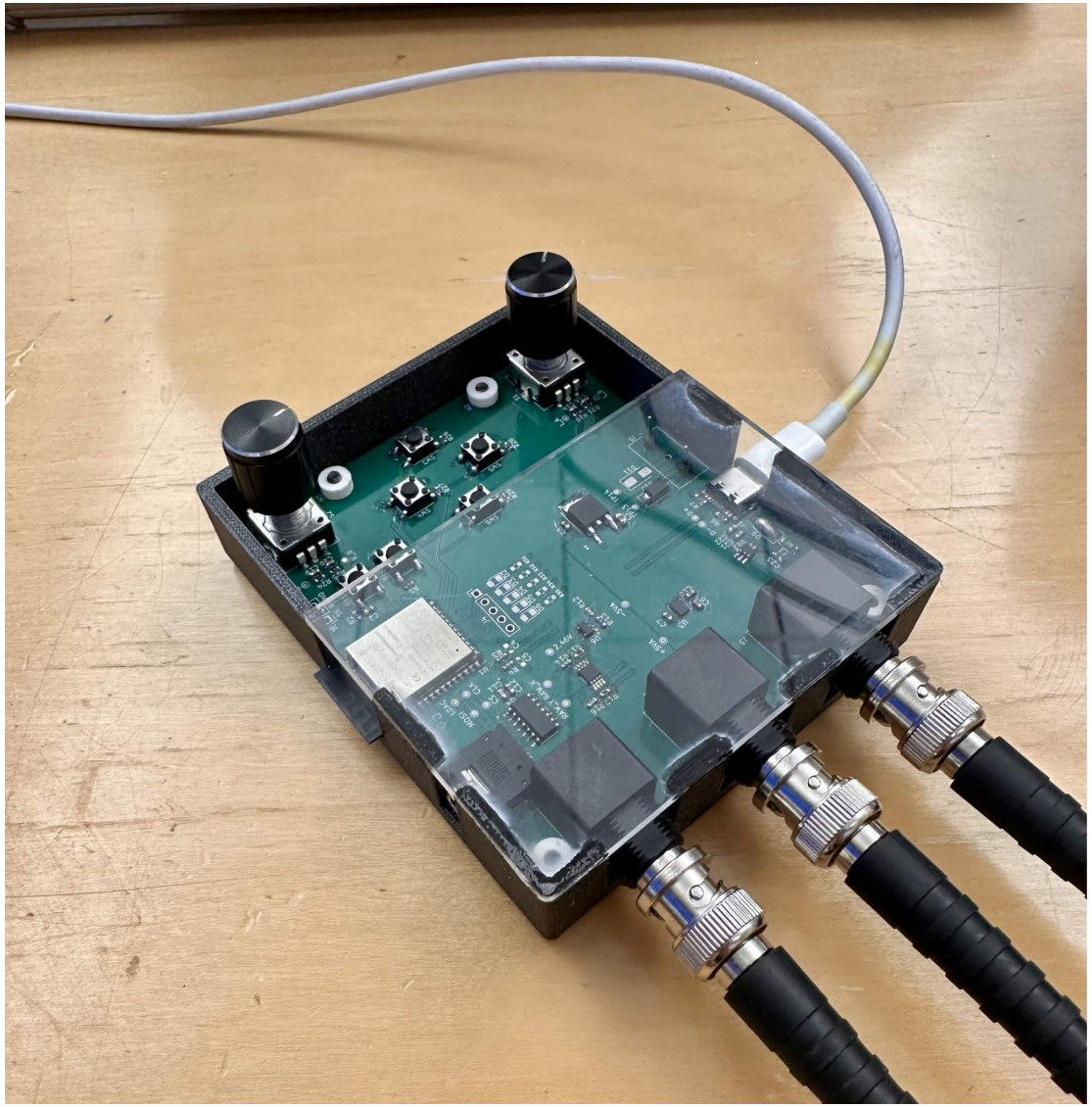
Additional Subsystem

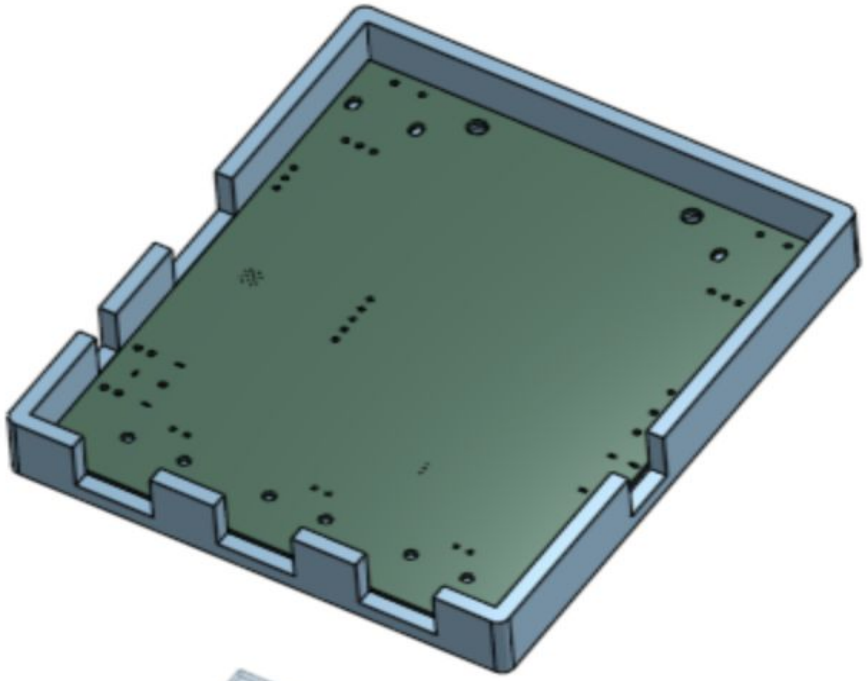
- **Audio System**
  - Provides an extra output path through the **audio jack**
  - Takes the generated X/Y signal path and processes it through the **audio op-amp stage**
- **Z-Blanking**
  - Provides an optional control signal for **beam blanking**
  - Can make the oscilloscope trace discontinuous
  - Utilizes the SN74LV1T34DCK **level shifter**

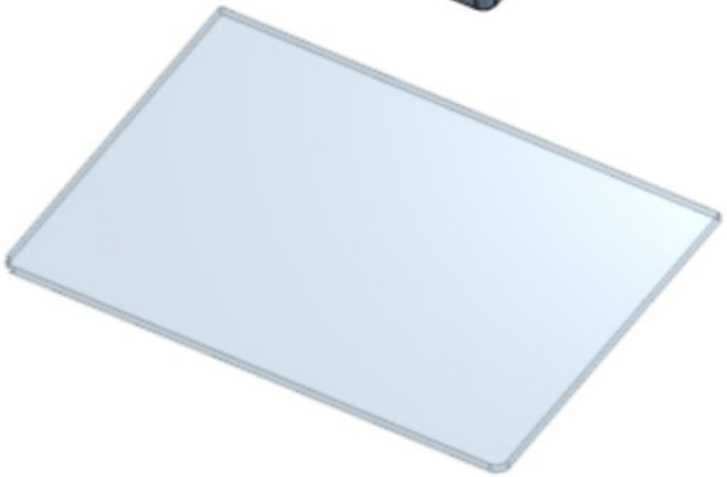




Mechanical Design







Overview

- Firmware reads user inputs and continuously updates synchronized X and Y DAC outputs for its respective mode
- Generated X/Y points are stored and replayed at high refresh rate so drawings remain visible on the oscilloscope

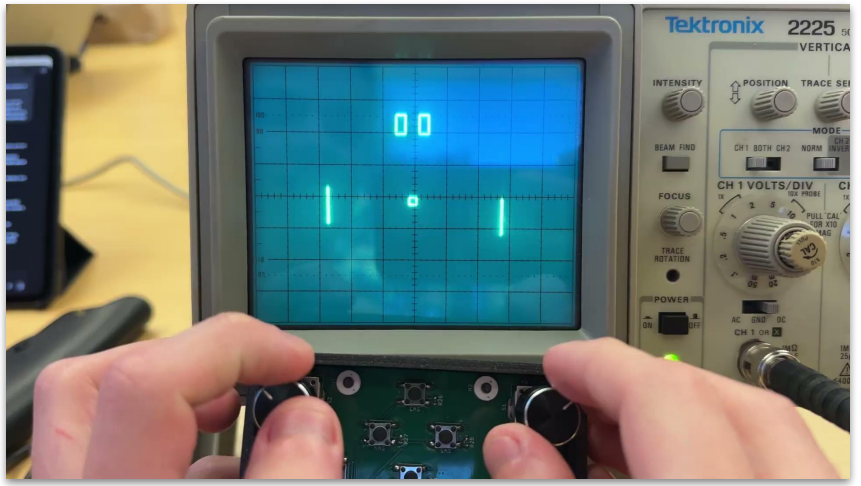
- **Interactive operating modes**
  1. ***Etch-a-Sketch mode:*** rotary encoders directly control X and Y position for freehand drawing
  2. ***Preset shape mode:*** firmware outputs predefined vector patterns to demonstrate coordinated signal generation
  3. ***Pong mode:*** real-time game logic drives moving paddles, ball motion, and on-screen interaction in XY mode
  4. ***Audio mode:*** Uploaded stereo audio is mapped to X/Y output, with adjustable low-pass and high-pass filtering to change the displayed image

```
static esp_timer_handle_t g_replayTimer = nullptr;
static TaskHandle_t g_replayTaskHandle = nullptr;

static void replayOnePoint() {
  const XYPoint p = drawingGetNextReplayPoint();
  dacWriteXY(p.x, p.y);
}

static void replayTimerCallback(void* arg) {
  (void)arg;
  replayOnePoint();
}

static void replayTask(void* arg) {
  (void)arg;
```



Implementation Challenges

- **Challenge:** each mode generated display data differently, but all needed to drive the same XY output path
- **Solution:** separated mode logic from low-level replay / DAC output
- **Mode layer:** Etch, Shape Demo, Pong, and Audio each focus only on generating their own coordinate data
- **Shared driver layer:** common drawing/replay code handles point storage, replay ordering, and synchronized DAC updates
- **Result:** reduced duplicated display code and kept output behavior consistent across all modes

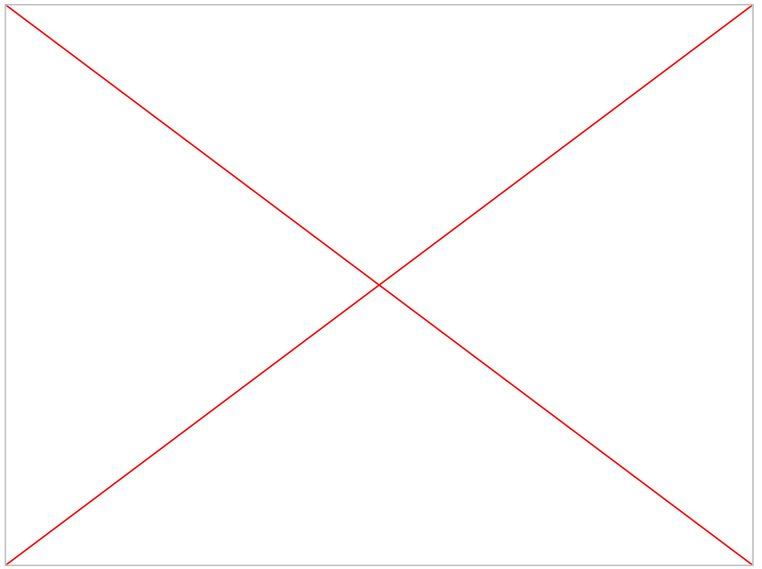
```
void dacWriteXY(uint16_t x, uint16_t y) {
  const uint16_t cmdX = buildDACWord(false, x); // DAC A = X
  const uint16_t cmdY = buildDACWord(true,  y); // DAC B = Y

  sendWord(cmdX);
  sendWord(cmdY);
  pulseLDAC();
}
```

```
// ======================================
// Etch path storage
// ======================================
static XYPoint g_path[MAX_PATH_POINTS];
static volatile size_t g_pathCount = 0;
static volatile size_t g_replayIndex = 0;
```

Implementation Challenges

- **Challenge:** oscilloscope drawings only remain visually solid if stored points are replayed fast enough
- **Initial approach:** software-timer replay was limited to about 20 kHz, causing visible retrace after longer drawings
- **Improvement:** switching to the ESP32 hardware timer increased replay to about 40 kHz
- **Final architecture:** split replay and UI/mode logic across the ESP32's two cores, enabling about 110 kHz refresh
- **Design impact:** high-speed replay became a core part of the firmware architecture, not just a parameter to tune



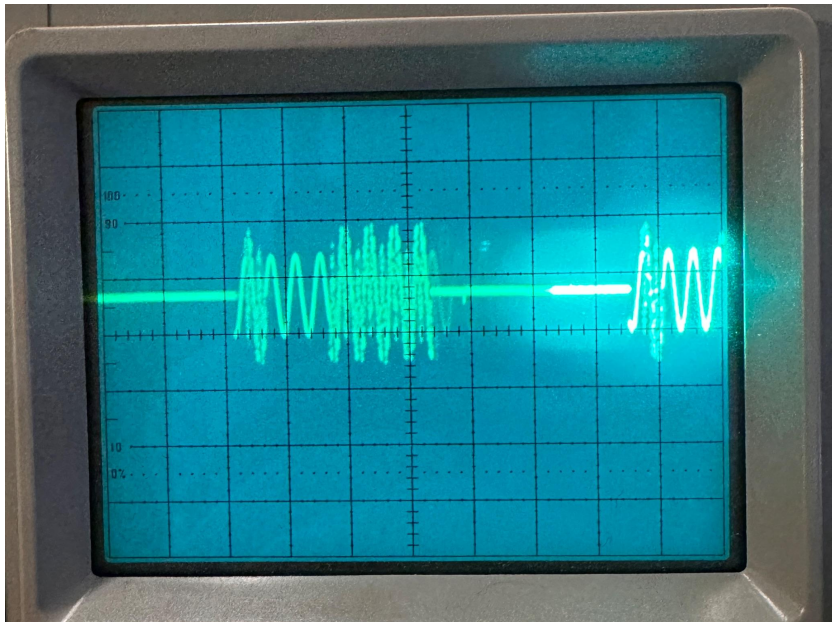
```
TaskHandle_t replayIdleHandle = xTaskGetIdleTaskHandleForCore(replayCore);
if (replayIdleHandle != nullptr) {
  esp_task_wdt_delete(replayIdleHandle);
}

const BaseType_t ok = xTaskCreatePinnedToCore(
    replayTask,
    "oscillo_replay",
    4096,
    nullptr,
    replayTaskPriority,
    &g_replayTaskHandle,
    replayCore);

(void)ok;
```

Implementation Challenges

- **Goal:** stream audio from a host computer and display it as real-time XY output
- **Challenge:** serial transport had to be both fast enough for continuous playback and reliable enough to avoid corrupted output
- **Observed issues:** underruns, packet loss/desynchronization, and start/stop playback behavior
- **Refinement:** added buffering, packet/session structure, and a more direct audio-to-DAC playback path
- **Result:** audio clarity improved significantly, but reliable continuous live playback remained one of the most difficult software problems in the project

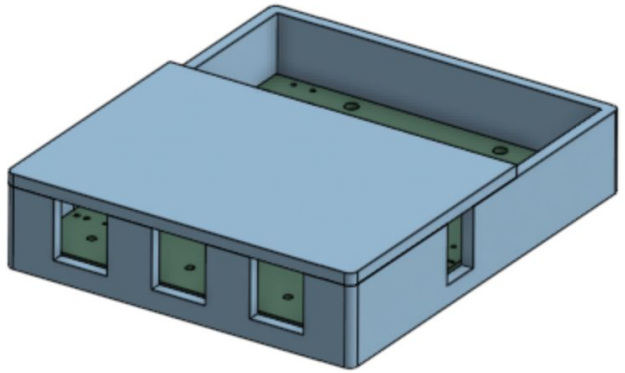


```
print(f"Loading: {audio_path}")
raw_pcm = load_and_convert_audio(audio_path)
total_frames = len(raw_pcm) // 4
duration_s = total_frames / AUDIO_SAMPLE_RATE
print(f"Converted to {AUDIO_SAMPLE_RATE} Hz stereo 16-bit PCM")
print(f"Frames: {total_frames}")
print(f"Duration: {duration_s:.2f} s")
print(f"Packets: {math.ceil(total_frames / AUDIO_PACKET_FRAMES)}
stream_loop(port, raw_pcm, loop_file=args.loop)
```

Future Works

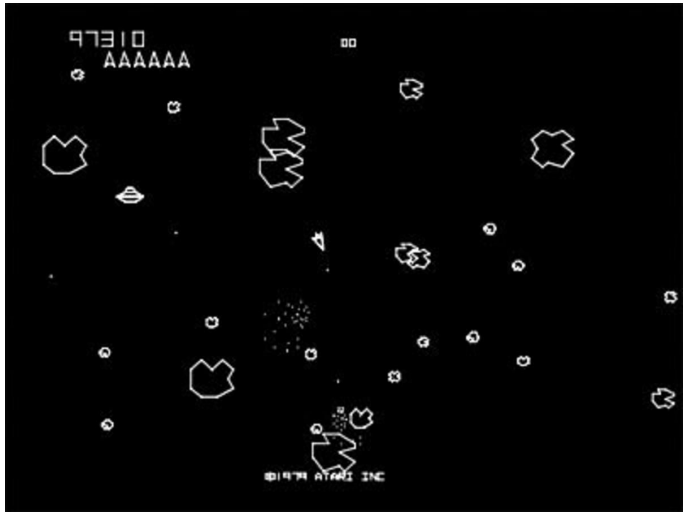
Improve Enclosure Design

- Redesign for better ergonomics and comfort
- Add rounded edges and improved grip
- Reposition buttons and encoders for easier control



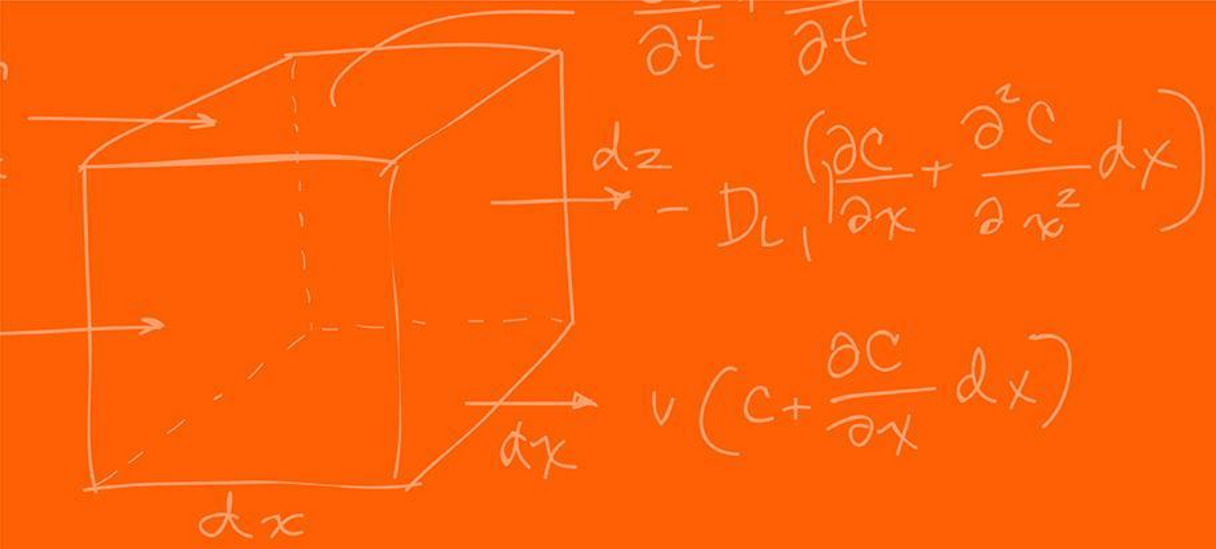
Add More Functionality

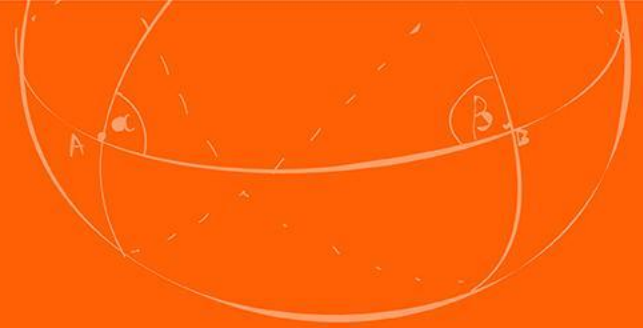
- Fourier series visualization
- More oscilloscope games, such as Asteroids or Snake
- Live music stream with low and high pass filtering
- Function generator modes for basic waveforms
- Musical note generation / piano mode

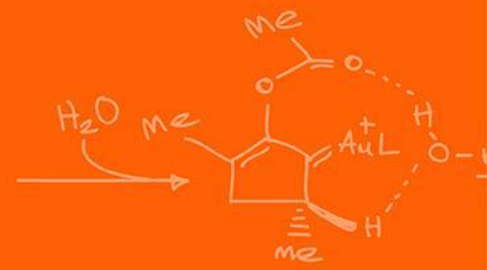


Thank You!

Contacts:
ericvo@illinois.edu
jajenks2@illinois.edu







The Grainger College of Engineering

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN