Portoscope

ESD Project

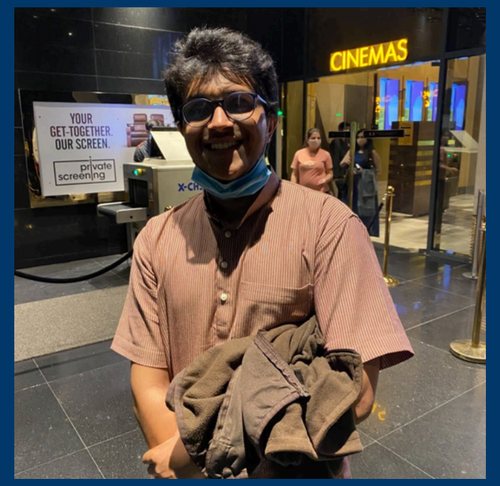
Abhishek Sethi,2023030
Madhav Maheshwari (2023304)
Adit Goel (2023036)
Arjun Chetan Pandya (2023120)

The students

Abhishek Sethi,2023030

Madhav Maheshwari (2023304)

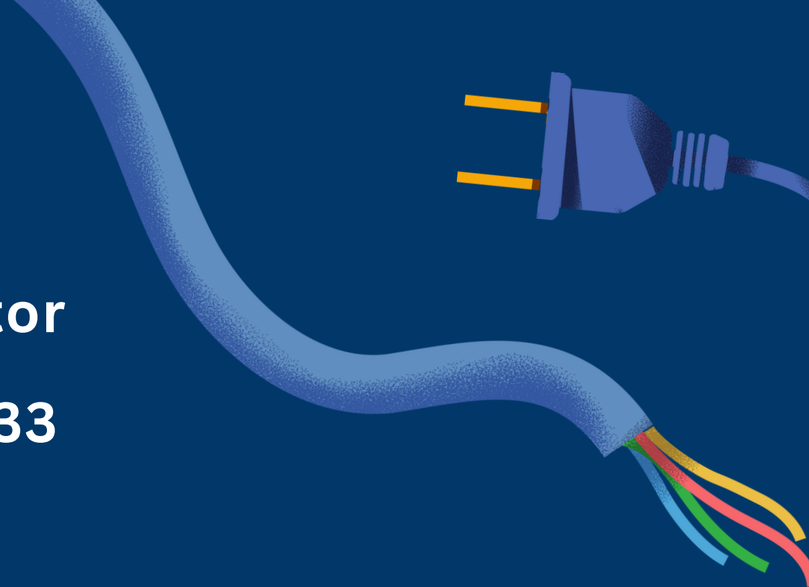
Adit Goel (2023036)



Arjun Chetan Pandya (2023120)

What is Portoscope?

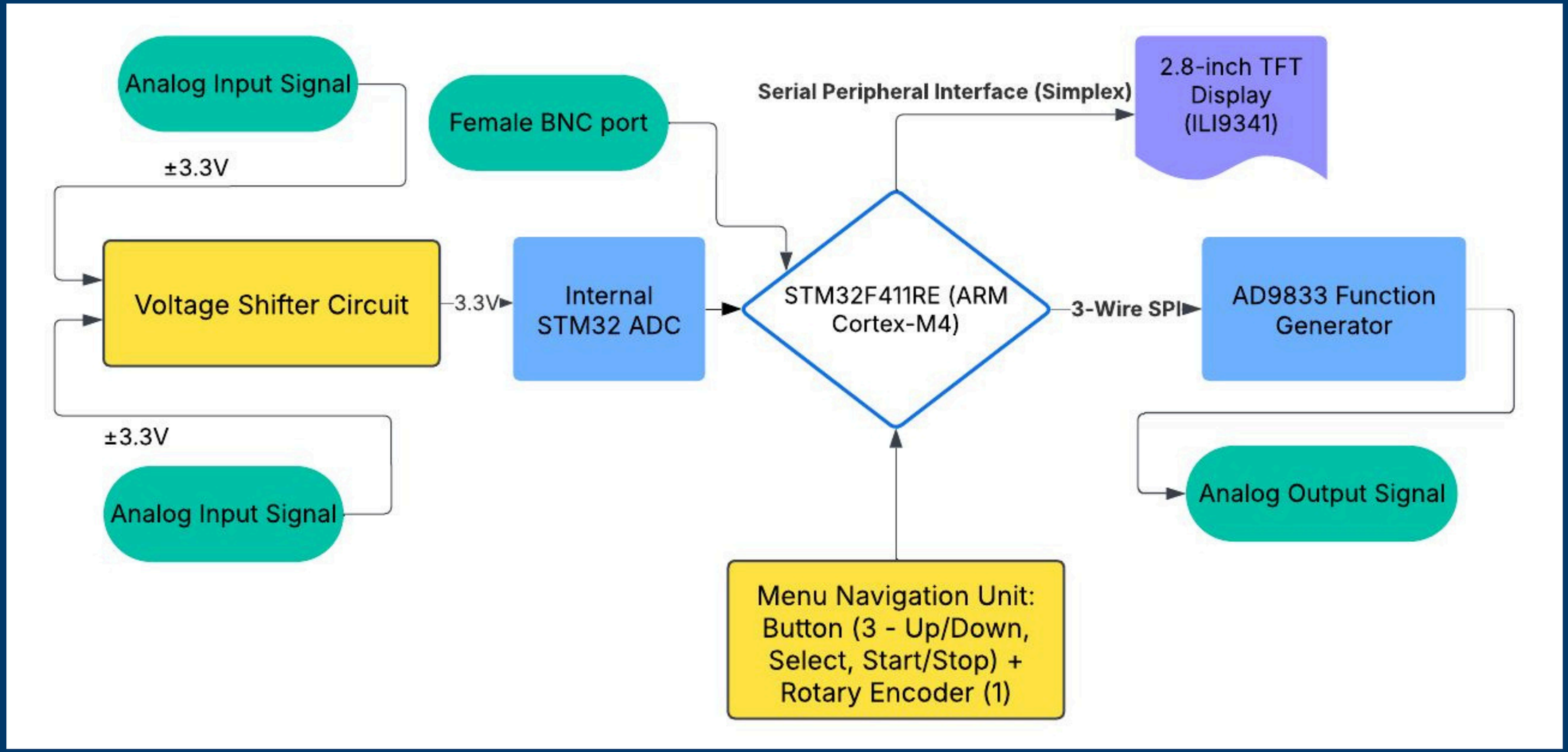
- A handheld dual-channel oscilloscope and signal generator
- Built using STM32F411RE, ILI9341 TFT Display, and AD9833
- Intended for educational, debugging, and lab use



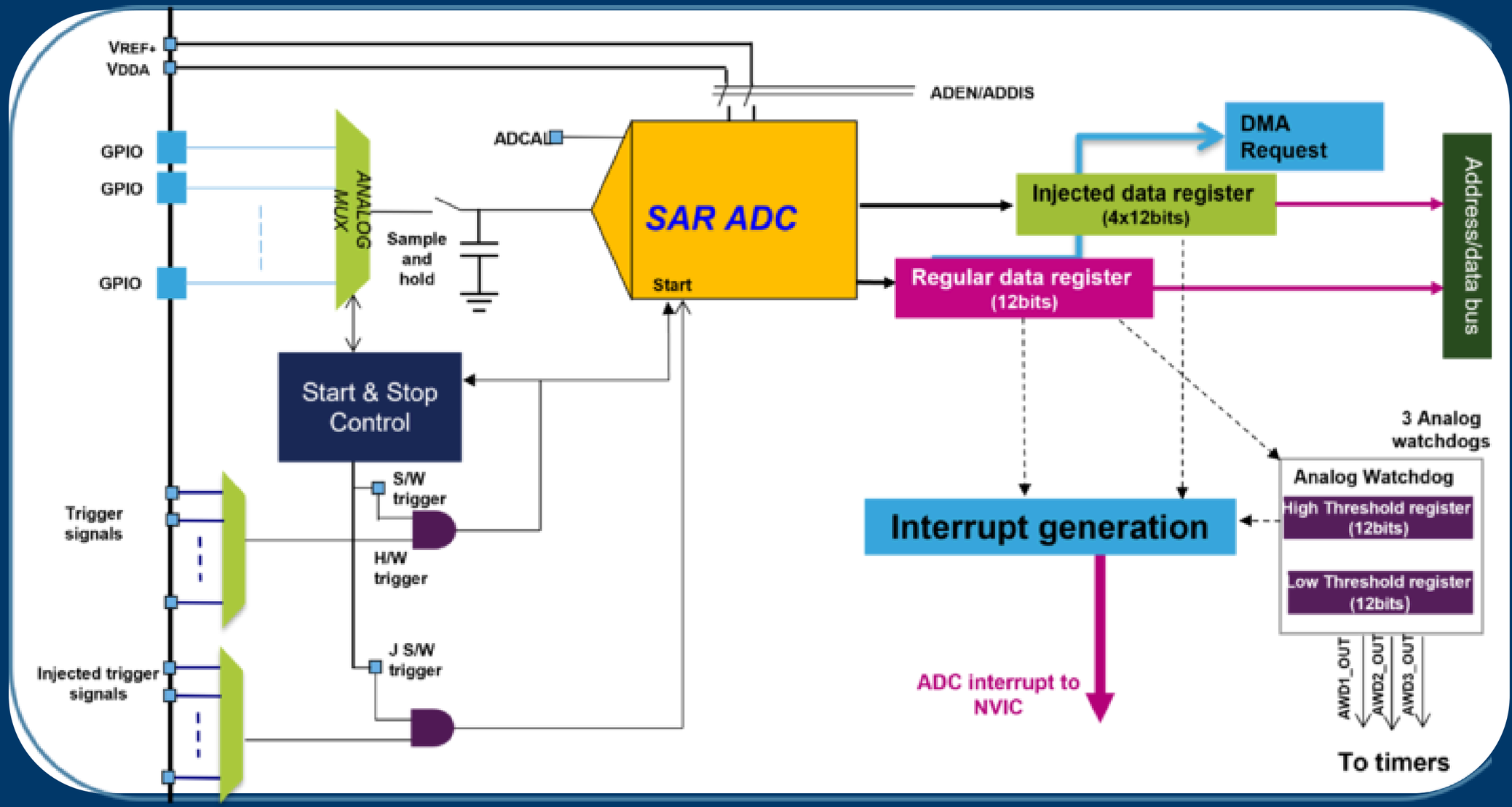
Why We Built It?

- Lab oscilloscopes are bulky and expensive
- Need for portable, low-cost testing tool
- Opportunity to integrate signal generation and visualization in one compact unit

Portoscope Genesis



STM32 ADC DIAGRAM FLOW



Operating Voltage & Frequency Ranges

1. STM32 ADC (Analog-to-Digital Converter)

- Voltage Range: 0V to Vref (Typically 3.3V or 5V, depending on the STM32 model)

2. AD9833 Waveform Generator

- Voltage Range: 2.3V to 5.5V
- UI Frequency Range: 0.1 Hz to 100 Hz (limited by software interface for precision)
- Hardware Frequency Limit: Up to 12.5 MHz (handled by AD9833)

3. 3.2-Inch TFT Screen Module Interface

- Measured Refresh Frequency: ~500 Hz waveform redraw (optimized for smooth UI)
- Flicker Avoidance: Enabled via high-speed SPI writes and partial refresh logic
- Typical Display Refresh: 60–75 Hz (screen controller spec)

Operating Voltage & Frequency Ranges

4. Power Supply & Regulation

- Input Voltage: 5 V (USB or regulated input)
- Regulation: Stepped down to 3.3 V for STM32

5. User Input Controls

- Push Buttons: Use active-low logic (connected to ground when pressed)
- Step-down Controls: Rotary encoder steps frequency down/up in fine increments

6. Communication Interfaces

- AD9833 SPI: Full-Duplex 3-wire serial interface (MOSI, MCLK, FSYNC)
- Screen UI Mode: Operates in Simple Duplex (toggle between oscilloscope and function generator screens)

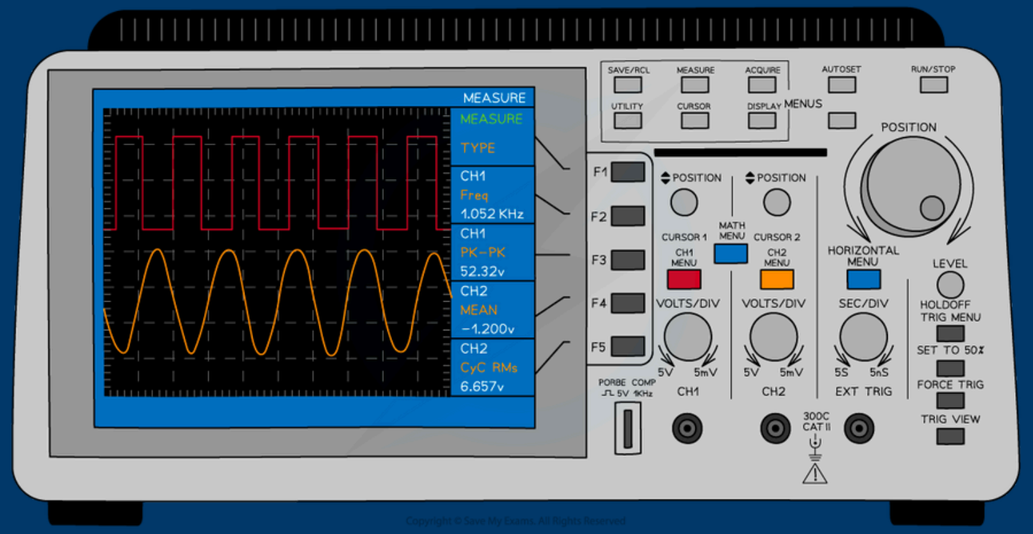
Functional Overview

Functional Overview

- **Inputs analog signals → Digitized by STM32 ADC → Displayed in real time**
- **Simultaneously outputs generated signals via AD9833**
- **User-controlled via encoder and buttons**

Core Components

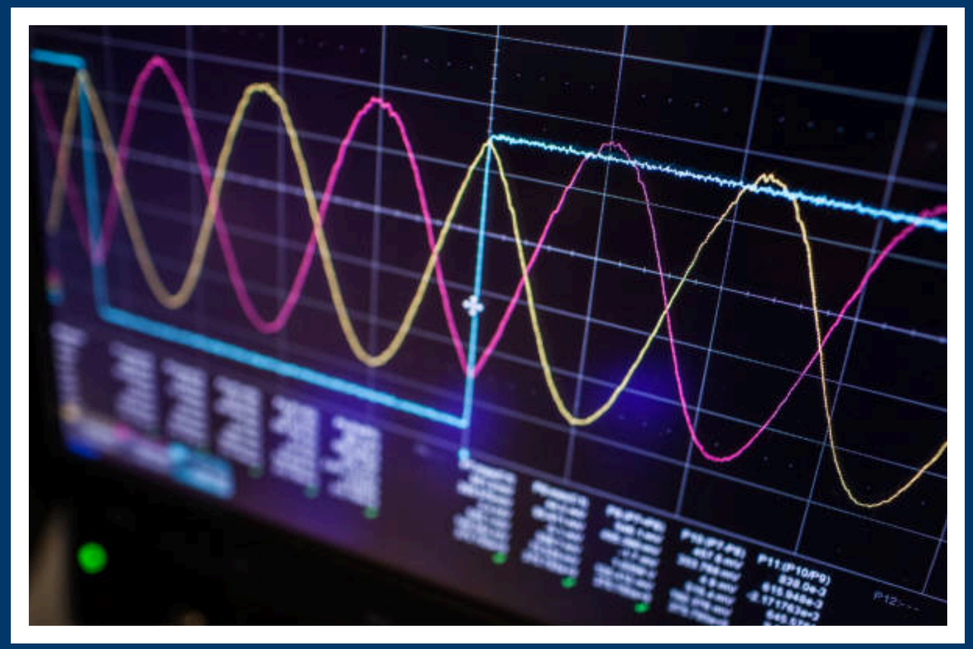
- **STM32F411RE (MCU with ADC)**
- **3.2" TFT Display (SPI)**
- **AD9833 Signal Generator Module**
- **Buttons + Rotary Encoder**



Key Features

Oscilloscope Mode

- Dual-channel analog inputs
- Adjustable time base, scroll speed
- Real-time waveform plotting



Signal Generator Mode

- Outputs sine, square, triangle waves
- Frequency control from menu

User Interface

- Menu-driven UI
- Rotary encoder + buttons
- Toggle display modes, grid, etc.

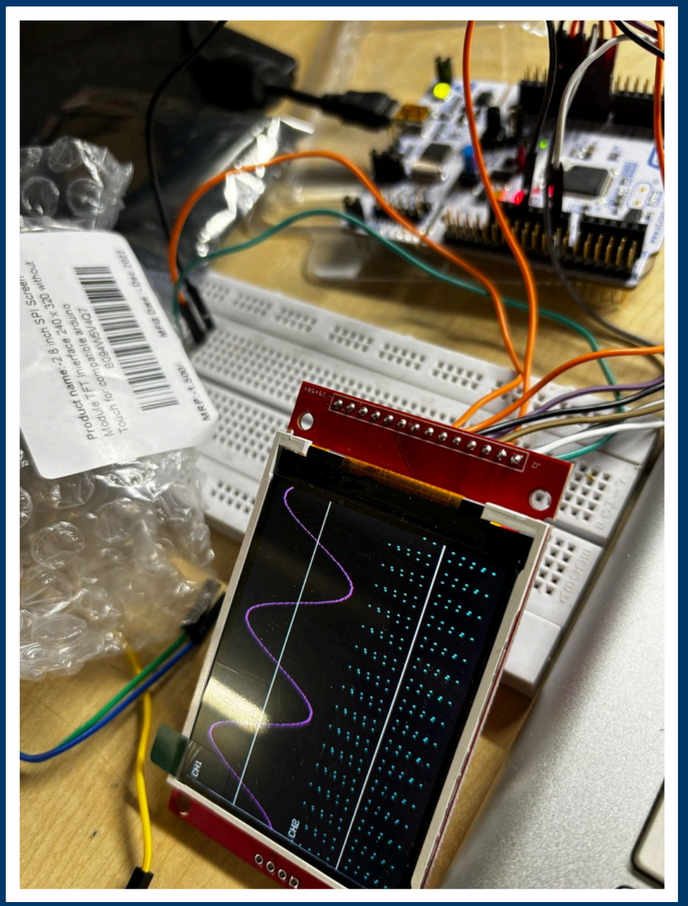
Display & User Interface

Display Specifications

- 3.2" SPI TFT (ILI9341)
- Resolution: 240 × 320
- Landscape mode

UI Features

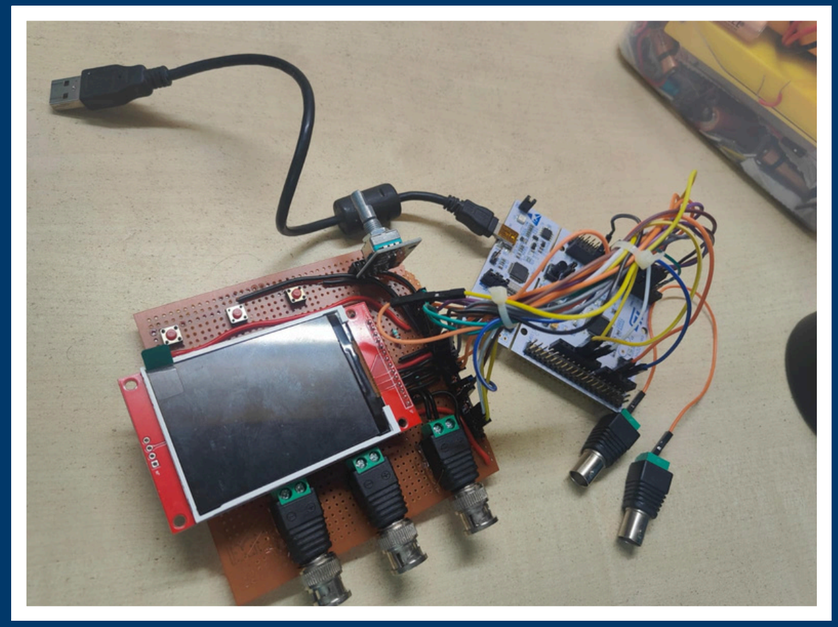
- Welcome splash screen: “PORTOSCOPE”
- Graph grid toggle (on/off)
- Axis labels and centered 0V line
- Real-time waveform scrolling
- Menu navigation with encoder/buttons



Oscilloscope Functionality

Input Configuration

- **Channel 1 → Analog pin A1**
- **Channel 2 → Analog pin A2**
- **Voltage range: ±3.3V**



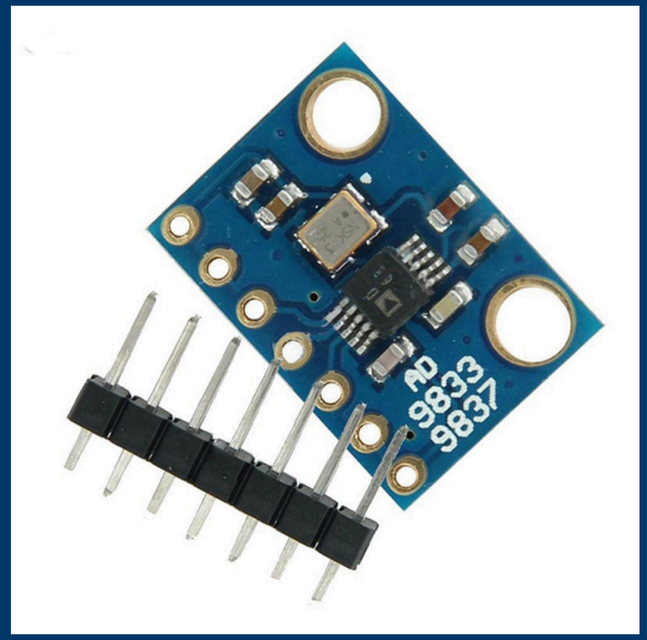
Sampling

- **Uses STM32 ADC (12-bit)**
- **Buffered to match screen refresh**
- **Max signal frequency: ~500 Hz**

Signal Generator (AD9833)

Waveform Output

- **Sine , Triangle , Square**



AD9833 Features

- **Frequency: 0.1 Hz to 100 Hz (software limited)**
- **Output: 0.6V peak-to-peak (amplifiable)**
- **Controlled via UI menu**
- **3.3V–5.5V supply**
- **3-wire SPI communication**

Controls & Interaction

Rotary Encoder

- **Adjust scroll speed (time scale)**
- **Change settings**
- **Menu navigation**

Buttons

- **UP/DOWN: Scroll through menu**
- **SELECT: Confirm / Enter**
- **START/STOP: Toggle oscilloscope operation**

Softwares Used

Arduino IDE

- Utilized to develop the core logic for the user interface, buffer management, and the input/output control of the Portoscope.
- Key libraries used:
- AD9833 Function Generator Library
- Adafruit GFX Library
- SPI Communication Library

Code Snippets

```
enum MenuItem {
  MODE_SINGLE,
  GRID_TOGGLE,
  START_SCOPE,
  FUNCTION_GENERATOR,
  MENU_ITEM_COUNT
};

int selectedItem = 0;
bool isDualChannel = true;
bool showGrid = true;

float phase = 0;
float speed = 0.2;
float freq = 0.05;

int prevX1[SCREEN_WIDTH];
int prevY1[SCREEN_WIDTH];
int prevX2[SCREEN_WIDTH];
int prevY2[SCREEN_WIDTH];

enum SignalType { SINUSOIDAL, TRIANGLE, SQUARE };
SignalType currentSignal = SINUSOIDAL;
float funcGenFreq = 1.0;
float funcGenAmp = 50.0;
int funcGenSelected = 0;
```

```
void drawGrid() {
  uint16_t gridColor = ILI9341_DARKGREY;
  int hSpacing = 20;
  int vSpacing = 20;

  for (int x = hSpacing; x < SCREEN_WIDTH; x += hSpacing) {
    for (int y = 0; y < SCREEN_HEIGHT; y += 4) {
      tft.drawPixel(x, y, gridColor);
    }
  }
  for (int y = vSpacing; y < SCREEN_HEIGHT; y += vSpacing) {
    for (int x = 0; x < SCREEN_WIDTH; x += 4) {
      tft.drawPixel(x, y, gridColor);
    }
  }
}
```

```
void setupButtons() {
  pinMode(BTN_UP, INPUT_PULLUP);
  pinMode(BTN_DOWN, INPUT_PULLUP);
  pinMode(BTN_SELECT, INPUT_PULLUP);
}

bool wasPressed(int pin) {
  static unsigned long lastDebounce = 0;  // Store the last debounce time globally
  static int lastButtonState = HIGH;      // Last state of the button (HIGH means unpressed)

  int buttonState = digitalRead(pin);

  // Check if button is pressed and debounce time has passed
  if (buttonState == LOW && lastButtonState == HIGH && millis() - lastDebounce > 40) {
    lastDebounce = millis();
    lastButtonState = LOW;
    return true;
  }
  else if (buttonState == HIGH) {
    lastButtonState = HIGH;  // Reset state if the button is released
  }

  return false;
}
```

```
void sampleChannels() {
  if(!oscilloscopeRunning)
  {
    return;
  }
  ch1Samples[sampleIndex] = analogRead(CH1_PIN);
  ch2Samples[sampleIndex] = analogRead(CH2_PIN);


  float voltage1 = ((ch1Samples[sampleIndex] - 2048.0) / 2048.0) * 3.3;
  float voltage2 = ((ch2Samples[sampleIndex] - 2048.0) / 2048.0) * 3.3;

  if(voltage1<minADC1){
    minADC1= voltage1;
  }
 if(voltage1>maxADC1)
 {
  maxADC1=voltage1;
 }


  if(voltage2<minADC2){
    minADC2= voltage2;
  }
 if(voltage2>maxADC2)
 {
  maxADC2=voltage2;
 }


  sampleIndex = (sampleIndex + 1) % SAMPLE_BUFFER_SIZE; // wrap around

  // delayMicroseconds(10);
  delay(counter);
}
```

Code Snippets

```
void showLogo() {
  tft.fillScreen(ILI9341_BLACK);
  for (int i = 0; i < 5; i++) {
    tft.drawRect(10 + i, 10 + i, SCREEN_WIDTH - 20 - 2 * i, SCREEN_HEIGHT - 20 - 2 * i, ILI9341_CYAN);
  }

  tft.setTextColor(ILI9341_MAGENTA);
  tft.setTextSize(3);
  tft.setCursor(70, 90);
  tft.print("PORTOSCOPE");

  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.setCursor(100, 150);
  tft.print("Welcome!");

  delay(2000);
}
```

```
void clearWaveBuffers() {
  for (int x = 0; x < SCREEN_WIDTH; x++) {
    prevX1[x] = prevY1[x] = -1;
    prevX2[x] = prevY2[x] = -1;
  }
}
int mapADCToY(int adcVal, int centerY) {
  float voltage = ((adcVal - 2048.0) / 2048.0) * 3.3;  // Map to -3.3V to +3.3V

  return constrain(centerY - (voltage / 3.3) * AMPLITUDE+5, 0, SCREEN_HEIGHT - 1);
}
```

```
void drawMenuItem(int i, const String& text, bool selected) {
  int x = 20;
  int y = 30 + i * 45;
  int w = SCREEN_WIDTH - 40;
  int h = 40;

  uint16_t bgColor = selected ? ILI9341_YELLOW : ILI9341_DARKGREY;
  uint16_t textColor = selected ? ILI9341_BLACK : ILI9341_WHITE;

  tft.fillRoundRect(x, y, w, h, 5, bgColor);
  tft.drawRoundRect(x, y, w, h, 5, ILI9341_WHITE);

  tft.setTextColor(textColor);
  tft.setTextSize(2);
  tft.setCursor(x + 10, y + 10);
  tft.print(text);
}

void drawFuncGenMenu() {
  tft.fillScreen(ILI9341_BLACK);
  String signalTypes[] = {"Sinusoidal", "Triangle", "Square"};
  String funcOptions[] = {
    "Type: " + String(signalTypes[currentSignal]),
    "Freq: " + String(funcGenFreq, 1) + "Hz"

  };

  for (int i = 0; i < 2; i++) {
    drawMenuItem(i, funcOptions[i], i == funcGenSelected);
  }
}
```

Code Snippets

```
    drawMenu();
  }
  if (wasPressed(BTN_SELECT)) {
    Serial.println("Button select");
    if (selectedItem == MODE_SINGLE) isDualChannel = !isDualChannel;
    else if (selectedItem == GRID_TOGGLE) showGrid = !showGrid;
    else if (selectedItem == START_SCOPE) {
      inScopeMode = true;
      firstTimeInScope = true;
    }
    else if (selectedItem == FUNCTION_GENERATOR) {
      inFuncGenMenu = true;
      funcGenSelected = 0;

      return;
    }
    drawMenu();
  }
}
else {
  if (firstTimeInScope) {
    tft.fillScreen(ILI9341_BLACK);
    drawAxes();
    clearWaveBuffers();
    firstTimeInScope = false;
  }

  drawScope();


  if (wasPressed(BTN_UP)) {
    inScopeMode = false;
    delay(10);
    drawMenu();
  }
}
```

```
void generateSignals() {
  static int rampVal = 0;
  static float sinPhase = 0;

  int sinVal = 127 + 127 * sin(sinPhase);
  analogWrite(SIN_OUT, sinVal);
  sinPhase += 0.05;
  if (sinPhase >= TWO_PI) sinPhase -= TWO_PI;

  analogWrite(RAMP_OUT, rampVal);
  rampVal += 2;
  if (rampVal > 255) rampVal = 0;
}

void setup() {
  tft.begin();
  tft.setRotation(1);
  setupButtons();
  showLogo();
  drawMenu();

  pinMode(SIN_OUT, OUTPUT);
  pinMode(RAMP_OUT, OUTPUT);
  AD.begin();
  Serial.begin(9600);
  analogReadResolution(12);


  pinMode(clkPin, INPUT);
  pinMode(dtPin, INPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);


  lastDtState = digitalRead(dtPin);

  // Generate a lower frequency sine wave (50 Hz)
  funcGenFreq = 100;
  AD.setFrequency(MD_AD9833::CHAN_0,100);  // Try
  AD.setMode(MD_AD9833::MODE_TRIANGLE); // SINE, S
}
```

```
  }
  if (wasPressed(BTN_SELECT)) {
    if (funcGenSelected == 0) {
      currentSignal = static_cast<SignalType>((currentSignal + 1) % 3);
      //  implemented setting the signal
      if (currentSignal == SINUSOIDAL) AD.setMode(MD_AD9833::MODE_SINE);
      else if (currentSignal == TRIANGLE) AD.setMode(MD_AD9833::MODE_TRIANGLE);
      else if (currentSignal == SQUARE) AD.setMode(MD_AD9833::MODE_SQUARE1);
    }
    else if (funcGenSelected == 1){

     funcGenFreq = (funcGenFreq >= 100.0) ? 0.0 : funcGenFreq + 10;
     AD.setFrequency(MD_AD9833::CHAN_0, funcGenFreq);  // Try 10, 20, or 50 Hz

    }
    else if(funcGenSelected == 2)
    {
      funcGenAmp = (funcGenAmp >= 100.0) ? 10.0 : funcGenAmp + 10.0;
    }
    drawFuncGenMenu();
  }

}
else if (!inScopeMode) {
  minADC1 = 3.3;
  minADC2 = 3.3;
  maxADC1 = -3.3;
  maxADC2= -3.3;

  if (wasPressed(BTN_UP)) {
    Serial.println("Button_up");
    selectedItem = (selectedItem - 1 + MENU_ITEM_COUNT) % MENU_ITEM_COUNT;
    drawMenu();
  }
  if (wasPressed(BTN_DOWN)) {
    Serial.println("Button down");
    selectedItem = (selectedItem + 1) % MENU_ITEM_COUNT;
```

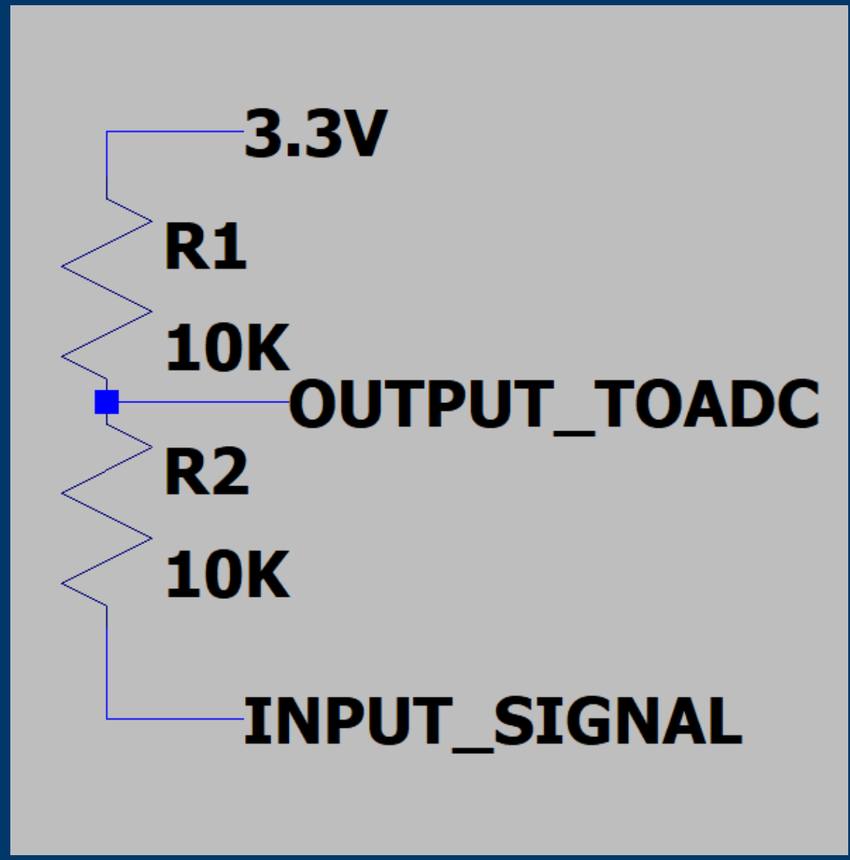
```
oid loop() {

   int dtState = digitalRead(dtPin);
int clkState = digitalRead(clkPin);

if (dtState != lastDtState) {
  if (dtState == LOW) {
    if (clkState == HIGH) {
      if (counter > 0) counter--;  // lower bound
    } else {
      if (counter < 20) counter++; // upper bound
    }
    Serial.print("Delay: ");
    Serial.println(counter);
  }
}

lastDtState = dtState;

sampleChannels();

if (inFuncGenMenu)
{
  minADC1 = 3.3;
  minADC2 = 3.3;
  maxADC1 = -3.3;
  maxADC2= -3.3;
  if (wasPressed(BTN_UP)) {
    inFuncGenMenu = false;
    delay(10);
    drawMenu();


  }
  if (wasPressed(BTN_DOWN)) {
    funcGenSelected = (funcGenSelected - 1 + 2) % 2;
    drawFuncGenMenu();

  }
  if (wasPressed(BTN_SELECT)) {
```

Challenges Faced : The -ve Voltage Issue

Internal ADC

Challenge :The internal Analog to Digital Converters(ADC) offered by most microcontrollers offers only an **single ended ADC** which means that it cannot measure voltages less than zero, which would be a problem for our oscilloscope

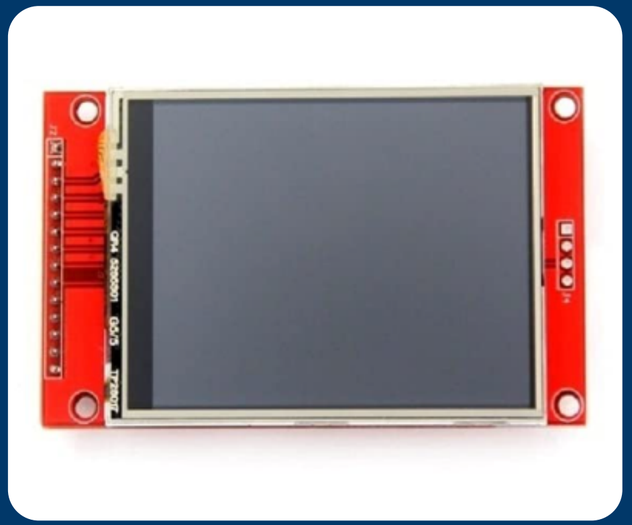
Solution: An external circuit had to be implemented and connected in between the input channel and the STM32 ADC , the circuit consists of a simple **voltage divider** and maps the voltage range of [-3.3V,3.3V] to [0,3.3V] which can be converted back into the original input before plotting



Challenges Faced : Refresh Rate Problems

Display Refresh Rate

Challenge :During the intial phase of the project we followed a simple approach in the code, the STM32, in a single method, would measure the signal at an instant and plot it at the same time, so everytime a sample was taken it was plotted and then the subsequent sample was taken.The issue this posed that the sampling process speed was limited by the refresh rate of the display.

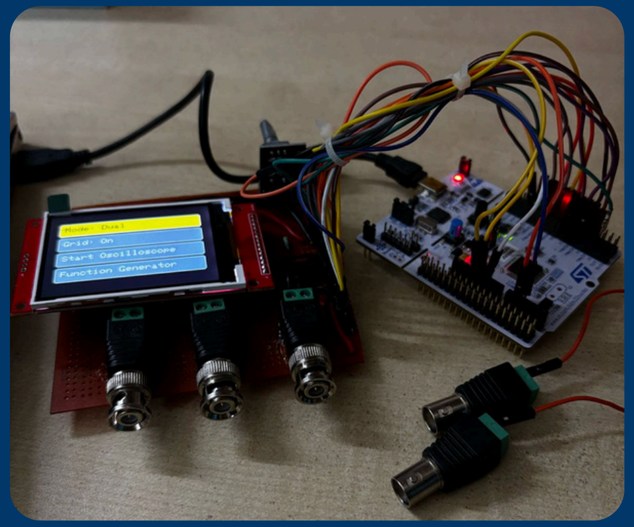


Solution: Splitting the sampling and plotting processes helped us overcome this challenge.In the final project the sampling is done continously and the captured data points are filled continously into a cyclic buffer.This allows the display to plot the data from the buffer whenever possible without interfering with signal sampling.

Challenges Faced : User Interfaces

Deciding Upon A User Interface

Challenge :Many different combinations of user interfaces were considered, such as touchscreens along with rotary encoders, the display along with buttons without the touchscreen and simply a rotary encoder and the display.



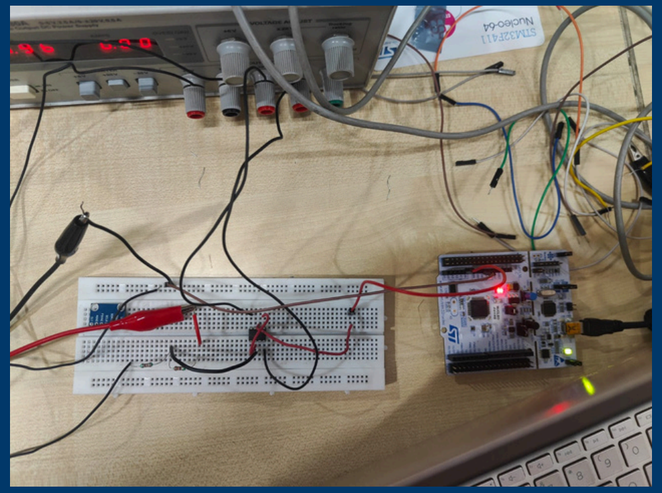
Solution:

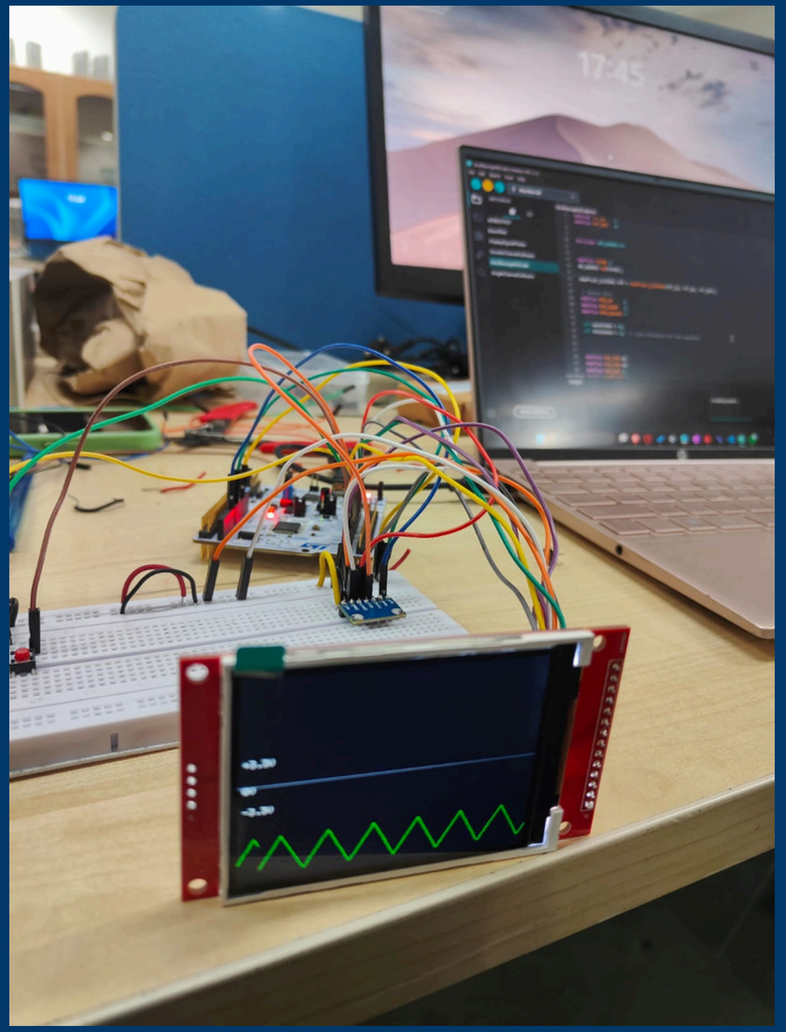
Our team eventually decided to use the display without a touchscreen and a combination of both buttons and rotary encoder as the user interface for our device.This was done keeping in mind the lab environments in which we believed that buttons and rotary encoders would be more reliable for configuring the oscilloscope

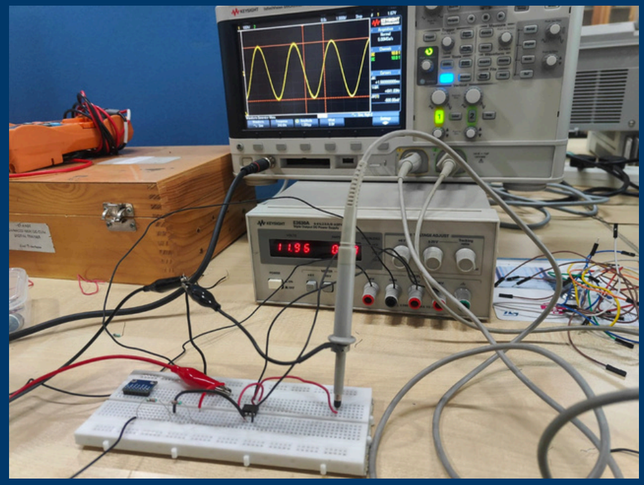
Work Logs

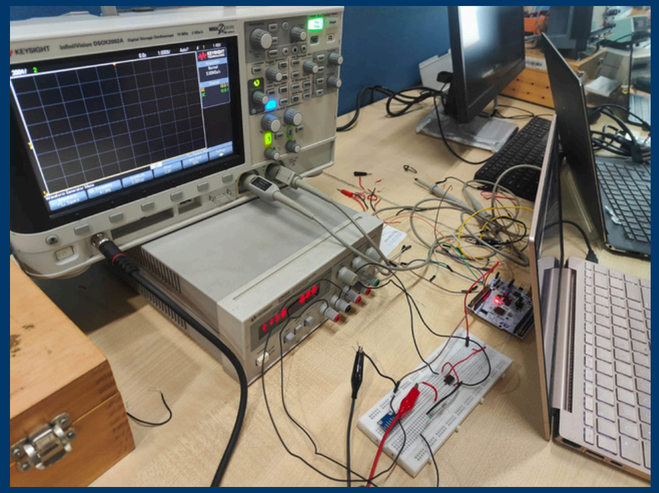
- Regularly attended all discussion sessions and consistently updated sir on our progress—both during online and offline meetings.
- Sent three detailed email updates to sir, each with a presentation summarizing that week’s progress.
- Each presentation highlighted the challenges faced, solutions implemented, hardware and software used, and included relevant images.

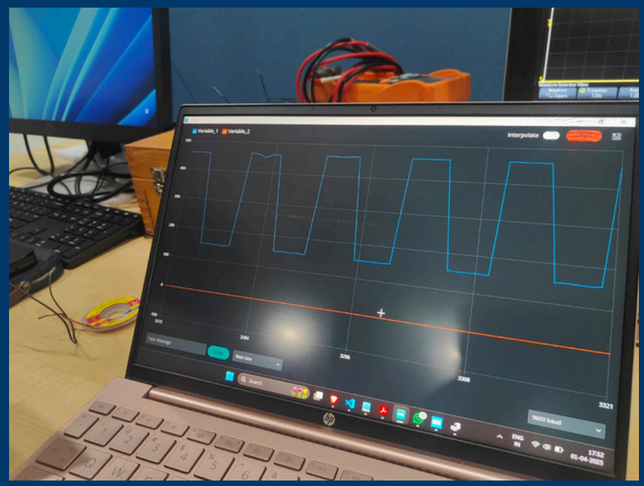
Bringing PORTOSCOPE to Life



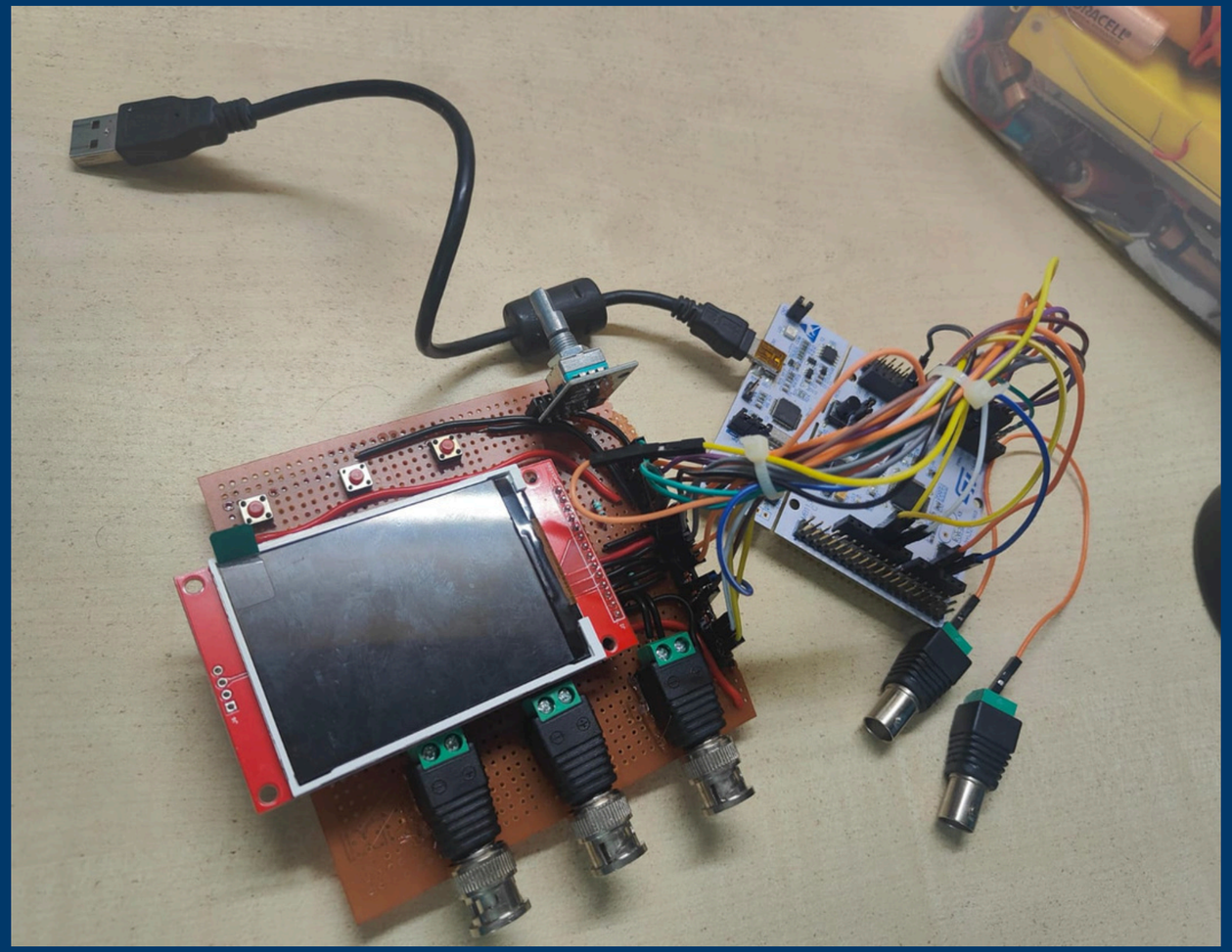








Final Result of our Work



THANK

YOU