```
// Integration of Digital S-Meter Styles into the ESP32 DDR VFO

#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <si5351.h>
#include <Encoder.h>
#include <ArduinoFFT.h>
#include <EEPROM.h>

#define MAIN_OLED_ADDR 0x3D
#define METER_OLED_ADDR 0x3C
#define SCREEN_W 128
#define SCREEN_H 64
#define METER_W 128
#define METER_H 32

Adafruit_SSD1306 displayMain(SCREEN_W, SCREEN_H, &Wire, -1);
Adafruit_SSD1306 displayMeter(METER_W, METER_H, &Wire, -1);
Si5351 si5351;

#define SDA_PIN 21
#define SCL_PIN 22
#define ENC_A 32
#define ENC_B 33
#define ENC_BTN 25
```

```
#define PTT_DETECT_PIN 34
#define PTT_KEY_PIN 26
#define BEEP_PIN 27
#define AUDIO_IN_PIN 35  // For S-Meter/AGC
#define VOLT_IN_PIN 36   // For Voltmeter

Encoder encoder(ENC_A, ENC_B);

#define SAMPLES 256
#define SAMPLERATE 8000
double vReal[SAMPLES];
double vImag[SAMPLES];
ArduinoFFT<double> FFT = ArduinoFFT<double>(vReal,
vImag, SAMPLES, SAMPLERATE, true);

uint32_t baseFreq = 27000000UL;
int currentMode = 1;
long clarifier = 0;
int stepIndex = 3;
const long stepSizes[] = {1, 10, 100, 1000, 10000};
long modeOffsetHz[4] = {0, -1500, 1500, 0};
const char* modeNames[4] = {"AM", "USB", "LSB", "CW"};

unsigned long lastDisplay = 0;
const unsigned long DISPLAY_INTERVAL = 80;
unsigned long bootTime = 0;

long memories[10] = {0};

bool menuActive = false;
```

```
int menuState = 0; // 0: normal, 1: main menu, 2: mode sub, 3: save mem sub, 4: load mem sub, 5: meter sub, 6: meter mode sub
int menuSelection = 0;

int meterMode = 0; // 0: S-Meter, 1: Voltmeter, 2: SWR
int brightnessLevel = 4; // 0-6 (0=dim, 6=bright)
bool invertMeter = false;
float voltCalibration = 3.3 / 4095.0 * 2.0; // Default for 2:1 divider (13.8V max)
float voltOffset = 0.0;

void applySiFreq(uint32_t freq) {
  long finalHz = (long)freq + clarifier + modeOffsetHz[currentMode];
  if (finalHz < 1500000L) finalHz = 1500000L;
  if (finalHz > 55000000L) finalHz = 55000000L;
  si5351.set_freq((uint64_t)finalHz * 100ULL, SI5351_CLK0);
}

void loadSettings() {
  for (int i = 0; i < 10; i++) {
    EEPROM.get(i * sizeof(long), memories[i]);
  }
  EEPROM.get(40 * sizeof(int), meterMode);
  EEPROM.get(41 * sizeof(int), brightnessLevel);
  EEPROM.get(42 * sizeof(bool), invertMeter);
  EEPROM.get(43 * sizeof(float), voltCalibration);
  EEPROM.get(44 * sizeof(float), voltOffset);
  if (brightnessLevel < 0 || brightnessLevel > 6)
```

```
brightnessLevel = 4;
  if (meterMode < 0 || meterMode > 2) meterMode = 0;
  displayMeter.invertDisplay(invertMeter);

displayMeter.ssd1306_command(SSD1306_SETCONTRAST);
  displayMeter.ssd1306_command(map(brightnessLevel, 0,
6, 0, 255));
}

void saveSettings() {
  EEPROM.put(40 * sizeof(int), meterMode);
  EEPROM.put(41 * sizeof(int), brightnessLevel);
  EEPROM.put(42 * sizeof(bool), invertMeter);
  EEPROM.put(43 * sizeof(float), voltCalibration);
  EEPROM.put(44 * sizeof(float), voltOffset);
  EEPROM.commit();
}

void calibrateVoltage() {
  displayMain.clearDisplay();
  displayMain.setTextSize(1);
  displayMain.setCursor(0, 0);
  displayMain.println("Calibrate Volt:");
  displayMain.println("Set 12.0V, press btn");
  displayMain.display();
  while (digitalRead(ENC_BTN) == HIGH) delay(10);
  float adc12 = analogRead(VOLT_IN_PIN) * (3.3 / 4095.0);
  delay(500);

  displayMain.clearDisplay();
```

```
  displayMain.println("Set 13.8V, press btn");
  displayMain.display();
  while (digitalRead(ENC_BTN) == HIGH) delay(10);
  float adc138 = analogRead(VOLT_IN_PIN) * (3.3 / 4095.0);
  delay(500);

  voltCalibration = (13.8 - 12.0) / (adc138 - adc12);
  voltOffset = 12.0 - (adc12 * voltCalibration);
  saveSettings();
  displayMain.clearDisplay();
  displayMain.println("Calibrated! Reboot.");
  displayMain.display();
  delay(2000);
}

void drawSplashScreen() {
  displayMeter.clearDisplay();
  displayMeter.setTextSize(1);
  displayMeter.setCursor(0, 0);
  displayMeter.println("S-METER/SWR ");
  displayMeter.setCursor(0, 10);
  displayMeter.println("KEVIN NELEZEN");
  displayMeter.setCursor(0, 20);
  displayMeter.println("V 2.0");
  displayMeter.display();
}

void drawSMeter() {
  displayMeter.clearDisplay();
  int val = analogRead(AUDIO_IN_PIN);
```

```
  int level = map(val, 0, 4095, 0, 7); // 0-7 segments
  const char* labels[8] = {"S1", "S3", "S5", "S7",
"S9","+20","+40","+60"};
  displayMeter.setTextSize(1);
  displayMeter.setCursor(0, 0);
  displayMeter.print(digitalRead(PTT_DETECT_PIN) == LOW ?
"PWR" : "S"); // PWR for TX
  for (int i = 0; i < 8; i++) {
    int x = 4 + i * 15;
    displayMeter.drawLine(x, 8, x, 12, SSD1306_WHITE); //
Ticks
    displayMeter.setCursor(x - 3, 16);
    displayMeter.print(labels[i]);
    if (digitalRead(PTT_DETECT_PIN) == LOW) {
      // TX power scale 0-10, map to 8 segments
      int pwrLevel = map(val, 0, 4095, 0, 10);
      if (i < (pwrLevel * 8 / 10)) displayMeter.fillRect(x - 6, 0, 12,
8, SSD1306_WHITE);
    } else {
      if (i <= level) displayMeter.fillRect(x - 6, 0, 12, 8,
SSD1306_WHITE);
    }
  }
  displayMeter.display();
}

void drawVoltmeter() {
  displayMeter.clearDisplay();
  float rawVolt = analogRead(VOLT_IN_PIN) * (3.3 / 4095.0);
  float voltage = rawVolt * voltCalibration + voltOffset;
```

```
  char voltStr[6];
  dtostrf(voltage, 4, 1, voltStr);
  displayMeter.setTextSize(1);
  displayMeter.setCursor(0, 0);
  displayMeter.print("Voltmeter ");
  displayMeter.print(voltStr);
  // Centered bar like PDF
  int barLen = map(voltage, 0, 15.0, 0, METER_W - 16);
  displayMeter.fillRect(8, 20, barLen, 4, SSD1306_WHITE);
  displayMeter.drawLine(8, 24, METER_W - 8, 24,
SSD1306_WHITE); // Scale line
  displayMeter.display();
}

void drawSWR() {
  displayMeter.clearDisplay();
  // Placeholder; native radio SWR used, but digital tap
possible
  displayMeter.setTextSize(1);
  displayMeter.setCursor(0, 0);
  displayMeter.println("SWR: Use Radio");
  displayMeter.setCursor(0, 16);
  displayMeter.println("Meter (Native)");
  // Bar for visual
  int dummySWR = map(analogRead(AUDIO_IN_PIN), 0, 4095,
10, 40); // Dummy 1.0-4.0
  int barLen = map(dummySWR, 10, 40, 0, METER_W - 16);
  displayMeter.fillRect(8, 24, barLen, 4, SSD1306_WHITE);
  displayMeter.display();
}
```

```
void setup() {
  Serial.begin(115200);
  EEPROM.begin(512);
  loadSettings();
  Wire.begin(SDA_PIN, SCL_PIN);
  delay(50);
  if (!displayMain.begin(SSD1306_SWITCHCAPVCC,
MAIN_OLED_ADDR)) Serial.println("Main display init failed");
  if (!displayMeter.begin(SSD1306_SWITCHCAPVCC,
METER_OLED_ADDR)) Serial.println("Meter display init
failed");
  pinMode(ENC_BTN, INPUT_PULLUP);
  pinMode(PTT_DETECT_PIN, INPUT);
  pinMode(PTT_KEY_PIN, OUTPUT);
  pinMode(BEEP_PIN, OUTPUT);
  pinMode(VOLT_IN_PIN, INPUT);
  digitalWrite(PTT_KEY_PIN, HIGH); // Default RX
  if (!si5351.init(SI5351_CRYSTAL_LOAD_10PF,
25000000ULL, 0)) Serial.println("Si5351 init failed");
  applySiFreq(baseFreq);

  // Boot splash on meter
  drawSplashScreen();
  bootTime = millis();
  delay(2000); // Show splash 2s

  displayMain.setTextSize(1);
  displayMain.setTextColor(SSD1306_WHITE);
  displayMain.setCursor(0, 0);
```

```
  displayMain.println("ESP32 DDR VFO");
  displayMain.display();

  // Boot calibration check
  if (digitalRead(ENC_BTN) == LOW) {
    delay(500);
    calibrateVoltage();
  }
}

void loop() {
  static long lastPos = 0;
  long pos = encoder.read() / 4;
  if (pos != lastPos) {
    long d = pos - lastPos;
    lastPos = pos;
    if (!menuActive && millis() > 3000) { // Ignore encoder
during boot splash
      long step = stepSizes[stepIndex];
      long nf = (long)baseFreq + d * step;
      if (nf < 1500000L) nf = 1500000L;
      if (nf > 55000000L) nf = 55000000L;
      baseFreq = nf;
      applySiFreq(baseFreq);
    } else if (menuActive) {
      menuSelection += d;
      switch (menuState) {
        case 1: case 2: case 5: menuSelection =
constrain(menuSelection, 0, 4); break;
        case 3: case 4: menuSelection =
```

```
constrain(menuSelection, 0, 9); break;
        case 6: menuSelection = constrain(menuSelection, 0, 3);
break;
      }
    }
  }

  static unsigned long btnDown = 0;
  bool isPTT = digitalRead(PTT_DETECT_PIN) == LOW;
  if (digitalRead(ENC_BTN) == LOW) {
    if (btnDown == 0) btnDown = millis();
    if (millis() - btnDown > 1000) { // Long press for menu/
invert
      if (!menuActive && millis() > 3000) {
        menuActive = true;
        menuState = 1;
        menuSelection = 0;
        beepRoger(60);
      } else if (menuActive && menuState == 1 &&
menuSelection == 4) {
        menuActive = false;
        beepRoger(60);
      } else if (!menuActive) {
        invertMeter = !invertMeter;
        displayMeter.invertDisplay(invertMeter);
        saveSettings();
        beepRoger(100);
      }
      btnDown = millis() + 200;
    }
```

```
  } else {
    if (btnDown != 0 && millis() - btnDown < 1000) { // Short
press
      if (!menuActive && millis() > 3000) {
        // Cycle meter modes (PDF function)
        meterMode = (meterMode + 1) % 3;
        saveSettings();
        beepRoger(40);
      } else if (menuActive) {
        // Menu actions (as before)
        if (menuState == 1) {
          if (menuSelection == 0) menuState = 2; // Mode
          else if (menuSelection == 1) menuState = 3; // Save
Mem
          else if (menuSelection == 2) menuState = 4; // Load
Mem
          else if (menuSelection == 3) menuState = 5; // Meter
Settings
          else menuActive = false; // Exit
          menuSelection = 0;
        } else if (menuState == 2) {
          currentMode = menuSelection;
          menuActive = false;
          applySiFreq(baseFreq);
        } else if (menuState == 3) {
          EEPROM.put(menuSelection * sizeof(long), baseFreq);
          menuActive = false;
        } else if (menuState == 4) {
          long memFreq;
          EEPROM.get(menuSelection * sizeof(long), memFreq);
```

```
        if (memFreq > 0) baseFreq = memFreq;
        menuActive = false;
        applySiFreq(baseFreq);
      } else if (menuState == 5) {
        if (menuSelection == 0) {
          menuState = 6;
          menuSelection = meterMode;
        } else if (menuSelection == 1) {
          brightnessLevel = (brightnessLevel + 1) % 7; // PDF: 7
steps

displayMeter.ssd1306_command(SSD1306_SETCONTRAST);

displayMeter.ssd1306_command(map(brightnessLevel, 0, 6,
0, 255));
          saveSettings();
        } else if (menuSelection == 2) {
          invertMeter = !invertMeter;
          displayMeter.invertDisplay(invertMeter);
          saveSettings();
        } else if (menuSelection == 3) {
          calibrateVoltage();
        } else {
          menuActive = false;
        }
        menuSelection = 0;
      } else if (menuState == 6) {
        meterMode = menuSelection % 3; // Exclude Back
        menuState = 5;
      }
```

```
      }
    }
    btnDown = 0;
  }

  // PTT handling
  digitalWrite(PTT_KEY_PIN, isPTT ? LOW : HIGH);
  if (isPTT && !menuActive) {
    uint8_t bins[SAMPLES / 2];
    sampleFFT(bins);
    displayMain.clearDisplay();
    // Draw spectrum below frequency (shifted for space)
    for (int i = 0; i < SAMPLES / 2; i++) {
      int col = map(i, 0, SAMPLES / 2 - 1, 0, SCREEN_W - 1);
      int intensity = min(15, (int)(bins[i] / 8));
      for (int y = 0; y < intensity; y++) {
        displayMain.drawPixel(col, 40 + y, SSD1306_WHITE); // Bottom half
      }
    }
    displayMain.display();
  }

  if (millis() - lastDisplay > DISPLAY_INTERVAL) {
    displayMain.clearDisplay();
    if (menuActive) {
      displayMain.setTextSize(1);
      displayMain.setCursor(0, 0);
      // Menu rendering (as before, expanded for meter submenus)
```

```
    if (menuState == 1) {
      const char* opts[] = {"Mode", "Save Mem", "Load Mem",
"Meter", "Exit"};
      displayMain.println("Menu:");
      for (int i = 0; i < 5; i++) {
        displayMain.setCursor(0, 10 + i * 10);
        if (menuSelection == i) displayMain.print(">");
        displayMain.println(opts[i]);
      }
    } else if (menuState == 5) {
      const char* opts[] = {"Mode Sel", "Bright", "Invert", "Cal
Volt", "Back"};
      displayMain.println("Meter Menu:");
      for (int i = 0; i < 5; i++) {
        displayMain.setCursor(0, 10 + i * 10);
        if (menuSelection == i) displayMain.print(">");
        displayMain.println(opts[i]);
      }
    } else if (menuState == 6) {
      const char* opts[] = {"S-Meter", "Voltmeter", "SWR",
"Back"};
      displayMain.println("Meter Mode:");
      for (int i = 0; i < 4; i++) {
        displayMain.setCursor(0, 10 + i * 10);
        if (menuSelection == i) displayMain.print(">");
        displayMain.println(opts[i]);
      }
    } // Other menus unchanged...
    // (Abbreviated for brevity; full menu code from previous
version)
```

```
    } else {
      // Frequency display unchanged
      char freqMain[8];
      sprintf(freqMain, "%2lu.%03lu", baseFreq / 1000000UL,
(baseFreq / 1000UL) % 1000UL);
      displayMain.setTextSize(2);
      displayMain.setCursor(0, 0);
      displayMain.print(freqMain);
      char freqFine[4];
      sprintf(freqFine, "%03lu", baseFreq % 1000UL);
      displayMain.setTextSize(1);
      displayMain.setCursor(84, 8);
      displayMain.print(".");
      displayMain.print(freqFine);
      displayMain.setCursor(110, 8);
      displayMain.print("MHz");
      int underlineX[] = {90, 84, 78, 60, 48};
      int underlineW[] = {6, 6, 6, 12, 12};
      displayMain.drawLine(underlineX[stepIndex], 16,
underlineX[stepIndex] + underlineW[stepIndex], 16,
SSD1306_WHITE);
      displayMain.setTextSize(1);
      displayMain.setCursor(0, 24);
      displayMain.print(modeNames[currentMode]);
      displayMain.setCursor(100, 24);
      displayMain.print(isPTT ? "TX" : "RX");
    }
    displayMain.display();

    // Meter rendering per PDF
```

```
    if (millis() < 2000) {
      drawSplashScreen(); // Boot splash
    } else {
      switch (meterMode) {
        case 0: drawSMeter(); break;
        case 1: drawVoltmeter(); break;
        case 2: drawSWR(); break;
      }
    }

    lastDisplay = millis();
  }
}

void sampleFFT(uint8_t *outbins) {
  unsigned long us = micros();
  unsigned long T = 1000000UL / SAMPLERATE;
  for (int i = 0; i < SAMPLES; i++) {
    while (micros() - us < T);
    us += T;
    int v = analogRead(AUDIO_IN_PIN);
    vReal[i] = (double)v;
    vImag[i] = 0;
  }
  FFT.windowing(FFT_WIN_TYP_HAMMING, FFT_FORWARD);
  FFT.compute(FFT_FORWARD);
  FFT.complexToMagnitude();
  for (int i = 0; i < SAMPLES / 2; i++) outbins[i] =
(uint8_t)min(255, (int)(vReal[i] / 8.0));
}
```

```
void beepRoger(int ms) {
  tone(BEEP_PIN, 1000);
  delay(ms);
  noTone(BEEP_PIN);
}
```