```
/*
  WORKBENCH DEVICE MONITOR
  Hardware:
    D1 Mini ESP32 WiFi + Bluetooth Development Board
    3 x ADS1115 16-bit 4-channel ADC
    10 x SCT-013 100A/1V split-core CT
    0.96" 128x64 I2C OLED

  Libraries:
    Adafruit ADS1X15
    Adafruit GFX
    Adafruit SSD1306

  IMPORTANT:
    This is a current-monitoring/device-state project, not a
    certified energy meter. Calibrate each CT channel against a
    trusted reference before relying on the readings.

    The SCT-013-100 is specified as 100A/1V. The nominal conversion
    is 100 A per 1 V RMS of CT output, but the actual installation
    must be calibrated.

    The CT must surround ONE current-carrying conductor only.
    Never work on energized mains wiring.
*/

#include <Wire.h>
#include <Adafruit_ADS1X15.h>
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <WebServer.h>
#include <math.h>

// ---------------- I2C ----------------
#define I2C_SDA 21
#define I2C_SCL 22

#define ADS1_ADDRESS 0x48
#define ADS2_ADDRESS 0x49
#define ADS3_ADDRESS 0x4A
#define OLED_ADDRESS 0x3C

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1

Adafruit_ADS1115 ads1;
Adafruit_ADS1115 ads2;
Adafruit_ADS1115 ads3;

Adafruit_SSD1306 display(
  SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET
);

WebServer server(80);

// ---------------- Wi-Fi ----------------
```

```
// Change these before uploading.
const char* WIFI_SSID = "YOUR_WIFI_NAME";
const char* WIFI_PASSWORD = "YOUR_WIFI_PASSWORD";

// ---------- Project ----------
const uint8_t NUM_CHANNELS = 10;

const char* deviceNames[NUM_CHANNELS] = {
  "Bench PSU",
  "Soldering Station",
  "Monitor",
  "USB Charger",
  "Dev Board PSU",
  "Oscilloscope",
  "Function Generator",
  "3D Printer",
  "Network Equipment",
  "Laptop Charger"
};

// Starting thresholds only. Calibrate for the actual devices.
float offThreshold[NUM_CHANNELS] = {
  0.03, 0.04, 0.03, 0.02, 0.02,
  0.04, 0.03, 0.04, 0.05, 0.03
};

float onThreshold[NUM_CHANNELS] = {
  0.15, 0.20, 0.08, 0.05, 0.06,
  0.15, 0.10, 0.30, 0.12, 0.10
```

```
};

// SCT-013-100 nominal ratio: 100A / 1V RMS.
// Change each channel after calibration.
float ampsPerVolt[NUM_CHANNELS] = {
  100.0, 100.0, 100.0, 100.0, 100.0,
  100.0, 100.0, 100.0, 100.0, 100.0
};

// Additional correction factor; normally 1.000.
float voltageCalibration[NUM_CHANNELS] = {
  1.000, 1.000, 1.000, 1.000, 1.000,
  1.000, 1.000, 1.000, 1.000, 1.000
};

// Approximate midpoint produced by a 10k/10k 3.3V bias divider.
// Measure the real bias voltage during calibration if possible.
const float ADC_BIAS_VOLTAGE = 1.65;

const uint16_t RMS_SAMPLES = 600;

float currentRMS[NUM_CHANNELS] = {0};

enum DeviceState {
  DEVICE_OFF,
  DEVICE_STANDBY,
  DEVICE_ON
};
```

```
DeviceState deviceState[NUM_CHANNELS] = {
  DEVICE_OFF, DEVICE_OFF, DEVICE_OFF, DEVICE_OFF, DEVICE_OFF,
  DEVICE_OFF, DEVICE_OFF, DEVICE_OFF, DEVICE_OFF, DEVICE_OFF
};

// ---------------- ADC helpers ----------------
Adafruit_ADS1115* getADC(uint8_t channel) {
  if (channel < 4) return &ads1;
  if (channel < 8) return &ads2;
  return &ads3;
}

uint8_t getADCInput(uint8_t channel) {
  return channel % 4;
}

int16_t readChannelRaw(uint8_t channel) {
  Adafruit_ADS1115* adc = getADC(channel);
  return adc->readADC_SingleEnded(getADCInput(channel));
}

float rawToVoltage(int16_t raw) {
  // GAIN_ONE = +/-4.096V, giving 0.125mV/bit.
  return (float)raw * 0.000125f;
}

// ---------------- RMS measurement ----------------
float measureCurrentRMS(uint8_t channel) {
  double sumSquares = 0.0;
```

```
  for (uint16_t n = 0; n < RMS_SAMPLES; n++) {
    int16_t raw = readChannelRaw(channel);
    float voltage = rawToVoltage(raw);

    // Remove the DC bias so only the AC CT waveform remains.
    float acVoltage = voltage - ADC_BIAS_VOLTAGE;

    sumSquares += (double)acVoltage * (double)acVoltage;
  }

  float ctRMSVoltage = sqrt(sumSquares / RMS_SAMPLES);

  ctRMSVoltage *= voltageCalibration[channel];

  float amps = ctRMSVoltage * ampsPerVolt[channel];

  if (!isfinite(amps) || amps < 0.0f) amps = 0.0f;

  return amps;
}

// ---------------- Device state ----------------
DeviceState determineState(uint8_t channel, float amps) {
  if (amps < offThreshold[channel]) return DEVICE_OFF;
  if (amps >= onThreshold[channel]) return DEVICE_ON;
  return DEVICE_STANDBY;
}
```

```
const char* stateText(DeviceState state) {
  switch (state) {
    case DEVICE_OFF: return "OFF";
    case DEVICE_STANDBY: return "STANDBY";
    case DEVICE_ON: return "ON";
  }
  return "UNKNOWN";
}

// ---------------- OLED ----------------
void showStartupScreen() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  display.setCursor(0, 0);
  display.println("Workbench Monitor");

  display.setCursor(0, 18);
  display.println("10 Current Channels");

  display.setCursor(0, 36);
  display.println("SCT-013 100A/1V");

  display.setCursor(0, 54);
  display.println("Starting...");

  display.display();
  delay(1500);
```

```
}

void displaySummaryPage() {
  uint8_t offCount = 0;
  uint8_t standbyCount = 0;
  uint8_t onCount = 0;

  for (uint8_t i = 0; i < NUM_CHANNELS; i++) {
    if (deviceState[i] == DEVICE_OFF) offCount++;
    else if (deviceState[i] == DEVICE_STANDBY) standbyCount++;
    else if (deviceState[i] == DEVICE_ON) onCount++;
  }

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  display.setCursor(0, 0);
  display.println("Workbench Status");

  display.setCursor(0, 18);
  display.print("OFF:     ");
  display.println(offCount);

  display.setCursor(0, 31);
  display.print("STANDBY: ");
  display.println(standbyCount);

  display.setCursor(0, 44);
```

```
  display.print("ON:      ");
  display.println(onCount);

  display.display();
}

void displayChannelPage(uint8_t firstChannel) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  for (uint8_t row = 0; row < 5; row++) {
    uint8_t ch = firstChannel + row;
    if (ch >= NUM_CHANNELS) break;

    uint8_t y = row * 12;

    display.setCursor(0, y);
    display.print(ch + 1);
    display.print(":");
    display.print(deviceNames[ch]);

    display.setCursor(91, y);
    display.print(currentRMS[ch], 2);
    display.print("A");
  }

  display.display();
}
```

```
// ---------------- Web page ----------------
String makeWebPage() {
  String html;
  html.reserve(7000);

  html += "<!DOCTYPE html><html><head>";
  html += "<meta name='viewport' content='width=device-width,initial-scale=1'>";
  html += "<meta http-equiv='refresh' content='5'>";
  html += "<title>Workbench Device Monitor</title>";
  html += "<style>";
  html += "body{font-family:Arial;margin:20px;background:#f5f5f5;color:#111}";
  html += "table{border-collapse:collapse;width:100%;background:#fff}";
  html += "th,td{border:1px solid #ccc;padding:8px;text-align:left}";
  html += "th{background:#eee}";
  html += "</style></head><body>";

  html += "<h1>Workbench Device Monitor</h1>";
  html += "<p>Current measurements from 10 SCT-013-100 channels.</p>";
  html += "<table>";
  html += "<tr><th>CH</th><th>Device</th><th>Current</th>";
  html += "<th>State</th><th>OFF threshold</th><th>ON threshold</th></tr>";

  for (uint8_t i = 0; i < NUM_CHANNELS; i++) {
    html += "<tr><td>" + String(i + 1) + "</td>";
    html += "<td>" + String(deviceNames[i]) + "</td>";
    html += "<td>" + String(currentRMS[i], 3) + " A</td>";
    html += "<td>" + String(stateText(deviceState[i])) + "</td>";
    html += "<td>" + String(offThreshold[i], 3) + " A</td>";
```

```
    html += "<td>" + String(onThreshold[i], 3) + " A</td></tr>";
  }

  html += "</table>";
  html += "<h2>Safety</h2>";
  html += "<p>This monitor is not a certified energy meter. ";
  html += "Do not work on energized mains wiring. ";
  html += "The CT must surround only one current-carrying conductor.</p>";
  html += "</body></html>";

  return html;
}

void handleRoot() {
  server.send(200, "text/html", makeWebPage());
}

void handleNotFound() {
  server.send(404, "text/plain", "Not found");
}

// ---------------- Wi-Fi ----------------
void setupWiFi() {
  if (strcmp(WIFI_SSID, "YOUR_WIFI_NAME") == 0) {
    Serial.println("Wi-Fi credentials not configured.");
    return;
  }

  WiFi.mode(WIFI_STA);
```

```
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.print("Connecting to Wi-Fi");

  unsigned long startTime = millis();

  while (WiFi.status() != WL_CONNECTED &&
         millis() - startTime < 15000) {
    delay(500);
    Serial.print(".");
  }

  Serial.println();

  if (WiFi.status() == WL_CONNECTED) {
    Serial.print("Wi-Fi connected. IP: ");
    Serial.println(WiFi.localIP());

    server.on("/", handleRoot);
    server.onNotFound(handleNotFound);
    server.begin();

    Serial.println("Web server started.");
  } else {
    Serial.println("Wi-Fi connection failed.");
  }
}

// ---------- Initialization ----------
```

```
void setupADCs() {
  if (!ads1.begin(ADS1_ADDRESS)) {
    Serial.println("ERROR: ADS1115 #1 not found.");
    while (true) delay(1000);
  }

  if (!ads2.begin(ADS2_ADDRESS)) {
    Serial.println("ERROR: ADS1115 #2 not found.");
    while (true) delay(1000);
  }

  if (!ads3.begin(ADS3_ADDRESS)) {
    Serial.println("ERROR: ADS1115 #3 not found.");
    while (true) delay(1000);
  }

  // +/-4.096V input range.
  ads1.setGain(GAIN_ONE);
  ads2.setGain(GAIN_ONE);
  ads3.setGain(GAIN_ONE);

  // Maximum ADS1115 data rate.
  ads1.setDataRate(RATE_ADS1115_860SPS);
  ads2.setDataRate(RATE_ADS1115_860SPS);
  ads3.setDataRate(RATE_ADS1115_860SPS);

  Serial.println("All ADS1115 devices initialized.");
}
```

```
void setupOLED() {
  if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDRESS)) {
    Serial.println("ERROR: OLED not found.");
    while (true) delay(1000);
  }

  showStartupScreen();
}

void printMeasurements() {
  Serial.println();
  Serial.println("================================================");
  Serial.println("WORKBENCH DEVICE MONITOR");
  Serial.println("================================================");

  for (uint8_t i = 0; i < NUM_CHANNELS; i++) {
    Serial.print("CH");
    Serial.print(i + 1);
    Serial.print(" | ");
    Serial.print(deviceNames[i]);
    Serial.print(" | ");
    Serial.print(currentRMS[i], 3);
    Serial.print(" A | ");
    Serial.println(stateText(deviceState[i]));
  }

  Serial.println("================================================");
}
```

```
// ---------------- Arduino setup ----------------
void setup() {
  Serial.begin(115200);
  delay(1000);

  Serial.println();
  Serial.println("Starting Workbench Device Monitor...");

  Wire.begin(I2C_SDA, I2C_SCL);

  setupADCs();
  setupOLED();
  setupWiFi();

  Serial.println("System ready.");
}

// ---------------- Main loop ----------------
void loop() {

  // Read all ten CT channels.
  for (uint8_t i = 0; i < NUM_CHANNELS; i++) {
    currentRMS[i] = measureCurrentRMS(i);
    deviceState[i] = determineState(i, currentRMS[i]);
  }

  printMeasurements();

  displaySummaryPage();
```

```
  delay(2500);

  displayChannelPage(0);
  delay(2500);

  displayChannelPage(5);
  delay(2500);

  if (WiFi.status() == WL_CONNECTED) {
    server.handleClient();
  }
}
```