```
/*
  Hydrogen Electrolyser Controller — LCD + Safety + Latching
  ----------------------------------------------------------

  Purpose
  -------
  Control the electrolyser relay and display key measurements on a 20x4
I2C LCD.
  Enforce safety limits with latching faults for
pressure/voltage/current.

  Behaviour (priority order)
  --------------------------
  1) Pressure > 6.0 bar  → System Fault (latched) → Relay OFF
  2) Switch OFF          → "Press Switch"         → Relay OFF
(auto-recover)
  3) Water NOT OK        → "Tank Full"            → Relay OFF
(auto-recover)
  4) Otherwise           → RUN                    → Relay ON, LCD shows
Vin/Iin/VElec/Ielec

  Latching rules
  --------------
  • Pressure, Vin, VElec, Iin, Ielec over limits → latch fault. Clear
by toggling the switch OFF.
  • Switch OFF / Water NOT OK are not latched; system auto-recovers as
soon as condition clears.

  Display rules
  -------------
  • RUN: show only Vin, Iin, VElec, Ielec (one per line, 2 decimals).
  • Any latched or live safety fault: "System Fault".
  • Switch OFF: "Press Switch".
  • Water NOT OK: "Tank Full".

  Limits (edit as required)
  -------------------------
  • PTank  ≤ 6.0 bar       (P_MAX_ALLOWED_KPA = 600)
  • Vin    ≤ 12.0 V
  • VElec  ≤ 8.0 V
  • Iin    ≤ 5.0 A
  • Ielec  ≤ 5.0 A
```

```
  Notes
  -----
  • Relay is active-LOW (LOW = ON). It starts OFF at boot.
  • The ACS712 current conversion uses (5.0/1023.0) intentionally, as
the sensor is ratiometric
    to the 5 V analog range; using the tracked Vref here can introduce
error if the sensor and
    ADC reference differ. Voltage channels use the measured Vref.
  • Pressure mapping follows your proven formula: pressure_Pa = (V -
0.48) * 320000.

  I/O summary
  -----------
  • relayPin   (D2)   : Active-LOW relay drive
  • switchPin (D3)   : Rocker switch (HIGH = ON)
  • levelPin   (D4)   : NC float to GND, read with INPUT_PULLUP (HIGH =
water OK)
  • pPin       (A1)   : DFROBOT pressure sensor analog output
  • A3 (Vin), A2 (Iin), A4 (VElec), A5 (Ielec)

  Timing
  ------
  • Serial log: 500 ms
  • LCD refresh: 2000 ms
  • Vref tracking: 1000 ms with exponential smoothing
*/

#include <SoftwareWire.h>               // Bit-banged I2C bus
#include <LiquidCrystal_SoftI2C.h>      // 20x4 LCD over SoftwareWire

// I2C LCD (PCF8574)
#define SDA_PIN 8
#define SCL_PIN 9
SoftwareWire swWire(SDA_PIN, SCL_PIN);
LiquidCrystal_I2C lcd(0x27, 20, 4, &swWire);

// Pins
const int relayPin  = 2;   // Active-LOW relay output
const int switchPin = 3;   // Rocker switch (HIGH = ON)
const int levelPin  = 4;   // NC float to GND (use INPUT_PULLUP); HIGH
= water OK
const int pPin      = A1;  // Pressure sensor analog output
```

```
// Periodic tasks
const unsigned long serialInterval = 500;   // ms
const unsigned long lcdInterval    = 2000;  // ms
unsigned long lastSerialTime = 0;
unsigned long lastLcdUpdate  = 0;

// Sensor calibration / scaling
const float sensitivity  = 0.185;  // ACS712 sensitivity (V/A) — adjust
per sensor variant
const float dividerIn    = 3.0;    // Vin divider ratio
const float dividerElec  = 3.0;    // VElec divider ratio
int   rawOffsetIn   = 0;           // Zero-current ADC offset (Iin)
int   rawOffsetElec = 0;           // Zero-current ADC offset (Ielec)

// ADC reference tracking (Vref)
float Vref = 5.00;                       // Running estimate of Vcc in
volts
const unsigned long vrefInterval = 1000; // ms
unsigned long lastVrefCal = 0;           // ms
const float vrefAlpha = 0.10f;           // Exponential smoothing
factor (0..1)

// Pressure filtering and limits
float pTank_kPa = 0.0f;                  // Filtered tank pressure
(kPa)
const float pAlpha = 0.15f;              // IIR smoothing for pressure
(0..1)
const float P_MAX_ALLOWED_KPA = 600.0f;  // 6.0 bar cutoff

// Electrical safety limits
const float VIN_MAX_V   = 12.0f;
const float VELEC_MAX_V = 8.0f;
const float IIN_MAX_A   = 5.0f;
const float IELEC_MAX_A = 5.0f;

// Fault latching
enum FaultCode : uint8_t {
  FAULT_NONE = 0,
  FAULT_PRESSURE,
  FAULT_VIN_HIGH,
  FAULT_VELEC_HIGH,
  FAULT_IIN_HIGH,
  FAULT_IELEC_HIGH
```

```
};
bool      faultLatched   = false;
FaultCode latchedReason  = FAULT_NONE;

// Measure Vcc using the internal 1.1 V band-gap (AVR only).
// Returns millivolts. On non-AVR targets, returns 5000 mV.
long readVcc_mV() {
  #if defined(__AVR__)
    ADMUX = _BV(REFS0) | 0b1110;      // Vref = Vcc, MUX=1110 (1.1V
bandgap)
    delay(2);
    ADCSRA |= _BV(ADSC);              // Start conversion
    while (ADCSRA & _BV(ADSC));       // Wait for completion
    uint16_t adc = ADC;
    return (1125300L) / adc;          // = 1.1 * 1023 * 1000 / adc
  #else
    return 5000;
  #endif
}

void setup() {
  // Digital I/O
  pinMode(switchPin, INPUT);          // External pull-ups/downs define
state
  pinMode(levelPin, INPUT_PULLUP);    // NC float to GND → HIGH = water
present
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);       // Ensure relay is OFF at startup
(active-LOW)

  // Analog inputs
  analogReference(DEFAULT);
  pinMode(A3, INPUT);  // Vin
  pinMode(A2, INPUT);  // Iin
  pinMode(A4, INPUT);  // VElec
  pinMode(A5, INPUT);  // Ielec
  pinMode(pPin, INPUT);// Pressure

  // I/O init
  Serial.begin(9600);
  lcd.begin();
  lcd.backlight();
```

```
  // Initial Vref calibration (average several band-gap readings)
  {
    const int N = 10;
    long sum_mV = 0;
    for (int i = 0; i < N; i++) {
      sum_mV += readVcc_mV();
      delay(5);
    }
    Vref = (sum_mV / (float)N) / 1000.0f;
  }

  // Zero-current calibration for ACS712 channels (Iin / Ielec)
  lcd.clear();
  lcd.print("Calibrating...");
  long sumIn = 0, sumElec = 0;
  for (int i = 0; i < 2000; i++) {
    sumIn   += analogRead(A2);
    sumElec += analogRead(A5);
    delayMicroseconds(50);
  }
  rawOffsetIn   = sumIn   / 2000;
  rawOffsetElec = sumElec / 2000;

  // Serial header (CSV-like)
  Serial.print("Offset Iin: ");   Serial.println(rawOffsetIn);
  Serial.print("Offset Ielec: "); Serial.println(rawOffsetElec);
  Serial.print("Vref init: ");    Serial.print(Vref, 3);
Serial.println(" V");
  Serial.println("Time, Switch, Water, PTank(bar), Vin, Iin, VElec,
Ielec, Vref, State");

  // Force immediate first LCD update
  lastLcdUpdate = millis() - lcdInterval;
}

void loop() {
  const unsigned long now = millis();

  // 1) Track Vref (smoothed) for accurate voltage conversion
  if (now - lastVrefCal >= vrefInterval) {
    lastVrefCal = now;
    const float vNow = readVcc_mV() / 1000.0f;
    Vref = (1.0f - vrefAlpha) * Vref + vrefAlpha * vNow;
```

```
  }

  // 2) Read discrete inputs
  const bool switchOn     = (digitalRead(switchPin) == HIGH); // ON
when HIGH
  const bool waterPresent = (digitalRead(levelPin) == HIGH);  // HIGH =
water OK

  // 3) Read and scale analog channels
  // Vin (uses tracked Vref and divider)
  analogRead(A3); delayMicroseconds(50);
  const int   rawV  = analogRead(A3);
  const float Vin   = (rawV * Vref / 1023.0f) * dividerIn;

  // Iin (uses 5.0 reference by design; see note in header)
  analogRead(A2); delayMicroseconds(50);
  const int   rawI  = analogRead(A2);
  const float Iin   = ((rawI - rawOffsetIn) * (5.0f / 1023.0f)) /
sensitivity;

  // VElec (uses tracked Vref and divider)
  analogRead(A4); delayMicroseconds(50);
  const int   rawE  = analogRead(A4);
  const float VElec = (rawE * Vref / 1023.0f) * dividerElec;

  // Ielec (uses 5.0 reference by design)
  analogRead(A5); delayMicroseconds(50);
  const int   rawEc = analogRead(A5);
  const float Ielec = ((rawEc - rawOffsetElec) * (5.0f / 1023.0f)) /
sensitivity;

  // Pressure (volts → Pa → kPa; then IIR smoothing)
  analogRead(pPin); delayMicroseconds(50);
  const int   rawP  = analogRead(pPin);
  const float vP    = (rawP * Vref / 1023.0f);
  float pressure_Pa = (vP - 0.48f) * 320000.0f;   // Clamp below zero
  if (pressure_Pa < 0.0f) pressure_Pa = 0.0f;
  const float p_kPa = pressure_Pa / 1000.0f;

  if (pTank_kPa == 0.0f) {
    pTank_kPa = p_kPa;
  } else {
    pTank_kPa = (1.0f - pAlpha) * pTank_kPa + pAlpha * p_kPa;
```

```
  }
  const float pTank_bar = pTank_kPa / 100.0f;

  // 4) Fault evaluation (Pressure > Electrical > Switch > Water)
  //    Latch clearing: when user toggles the switch OFF.
  if (faultLatched && !switchOn) {
    faultLatched  = false;
    latchedReason = FAULT_NONE;
  }

  const bool overPressure = (pTank_kPa > P_MAX_ALLOWED_KPA);
  const bool vinHigh      = (Vin    > VIN_MAX_V   + 1e-6f);
  const bool velecHigh    = (VElec  > VELEC_MAX_V + 1e-6f);
  const bool iinHigh      = (Iin    > IIN_MAX_A   + 1e-6f);
  const bool iecHigh      = (Ielec  > IELEC_MAX_A + 1e-6f);

  // Pressure has highest priority and latches immediately
  if (overPressure) {
    faultLatched  = true;
    latchedReason = FAULT_PRESSURE;
  }

  // If no pressure fault latched, check electrical limits (also latch)
  if (!faultLatched) {
    if      (vinHigh)    { faultLatched = true; latchedReason =
FAULT_VIN_HIGH;   }
    else if (velecHigh)  { faultLatched = true; latchedReason =
FAULT_VELEC_HIGH; }
    else if (iinHigh)    { faultLatched = true; latchedReason =
FAULT_IIN_HIGH;   }
    else if (iecHigh)    { faultLatched = true; latchedReason =
FAULT_IELEC_HIGH; }
  }

  // Determine state for relay and LCD
  enum { STATE_FAULT, STATE_PRESS, STATE_TANK, STATE_RUN } state =
STATE_RUN;
  if      (faultLatched)   state = STATE_FAULT;
  else if (!switchOn)      state = STATE_PRESS;   // Ask user to turn
switch ON
  else if (!waterPresent)  state = STATE_TANK;    // Water NOT OK (tank
full)
  else                     state = STATE_RUN;
```

```
  // Relay control (active-LOW: LOW = ON)
  const bool relayOn = (state == STATE_RUN);
  digitalWrite(relayPin, relayOn ? LOW : HIGH);

  // 5) Serial log (CSV-like)
  if (now - lastSerialTime >= serialInterval) {
    lastSerialTime = now;

    Serial.print(now / 1000.0f, 2); Serial.print(", ");
    Serial.print(switchOn     ? "SW=ON"     : "SW=OFF");
Serial.print(", ");
    Serial.print(waterPresent ? "Water=YES" : "Water=NO");
Serial.print(", ");
    Serial.print(pTank_bar, 2 );
Serial.print(", ");
    Serial.print(Vin, 2);     Serial.print(", ");
    Serial.print(Iin, 2);     Serial.print(", ");
    Serial.print(VElec, 2);   Serial.print(", ");
    Serial.print(Ielec, 2);   Serial.print(", ");
    Serial.print(Vref, 3);    Serial.print(", ");

    switch (state) {
      case STATE_FAULT:
        Serial.print("FAULT:");
        switch (latchedReason) {
          case FAULT_PRESSURE:   Serial.print("P>6bar");   break;
          case FAULT_VIN_HIGH:   Serial.print("Vin>12V");  break;
          case FAULT_VELEC_HIGH: Serial.print("VElec>8V"); break;
          case FAULT_IIN_HIGH:   Serial.print("Iin>5A");   break;
          case FAULT_IELEC_HIGH: Serial.print("Ielec>5A"); break;
          default:               Serial.print("Unknown");   break;
        }
        break;
      case STATE_PRESS: Serial.print("PRESS_SWITCH"); break;
      case STATE_TANK:  Serial.print("TANK_FULL");    break;
      case STATE_RUN:   Serial.print("RUN");          break;
    }
    Serial.println();
  }

  // 6) LCD update
  if (now - lastLcdUpdate >= lcdInterval) {
```

```
    lastLcdUpdate = now;
    lcd.clear();

    if (state == STATE_RUN) {
      // Only the four requested values
      lcd.setCursor(0, 0); lcd.print("Vin: ");   lcd.print(Vin,   2);
lcd.print("V");
      lcd.setCursor(0, 1); lcd.print("Iin: ");   lcd.print(Iin,   2);
lcd.print("A");
      lcd.setCursor(0, 2); lcd.print("VElec: "); lcd.print(VElec, 2);
lcd.print("V");
      lcd.setCursor(0, 3); lcd.print("Ielec: "); lcd.print(Ielec, 2);
lcd.print("A");
    } else if (state == STATE_PRESS) {
      lcd.setCursor(0, 1); lcd.print("Press Switch");
    } else if (state == STATE_TANK) {
      lcd.setCursor(0, 1); lcd.print("Tank Full");
    } else { // STATE_FAULT
      lcd.setCursor(0, 1); lcd.print("System Fault");
    }
  }

  // Small delay to avoid a fully tight loop
  delay(10);
}
```