// ═══════════════════════════════════════════════════════════════

//  ESP32-C6 智能火灾报警系统 v2.0

//  作者: iCEasy IoT Lab

//  功能: 火焰检测 + 气体检测 + 分级报警

//  硬件: ESP32-C6 + 火焰传感器 + MQ-2 + 有源蜂鸣器

// ═══════════════════════════════════════════════════════════════

// ─────────────────────────────────────────────────────────────

// 引脚定义

// ─────────────────────────────────────────────────────────────

#define FLAME\_SENSOR\_PIN  2   // 火焰传感器数字输出 (LOW=检测到火焰)

#define MQ2\_ANALOG\_PIN    3   // MQ-2模拟输出 (0-4095)

#define MQ2\_DIGITAL\_PIN   4   // MQ-2数字输出 (LOW=超阈值)

#define BUZZER\_PIN        5   // 有源蜂鸣器控制引脚

#define LED\_FIRE          6   // 红色LED（火灾警报）

#define LED\_GAS           7   // 黄色LED（气体预警）

// ─────────────────────────────────────────────────────────────

// 系统参数配置

// ─────────────────────────────────────────────────────────────

#define GAS\_THRESHOLD     1500  // MQ-2气体浓度阈值 (0-4095)

#define PREHEATING\_TIME   30000 // MQ-2预热时间（30秒）

#define SCAN\_INTERVAL     200   // 传感器扫描间隔（200ms）

#define STATUS\_PRINT\_INTERVAL 5000 // 状态打印间隔（5秒）

// ─────────────────────────────────────────────────────────────

// 报警级别枚举

// ─────────────────────────────────────────────────────────────

enum AlarmLevel {

  SAFE,         // 🟢 安全状态

  GAS\_WARNING,  // 🟡 气体预警（检测到烟雾/可燃气体）

  FIRE\_ALARM    // 🔴 火灾警报（检测到明火）

};

// ─────────────────────────────────────────────────────────────

// 全局变量

// ─────────────────────────────────────────────────────────────

AlarmLevel currentLevel = SAFE;       // 当前报警级别

unsigned long lastPrintTime = 0;      // 上次打印时间

unsigned long systemStartTime = 0;    // 系统启动时间

unsigned int alarmCount = 0;          // 报警次数统计

// ═══════════════════════════════════════════════════════════════

// 初始化函数

// ═══════════════════════════════════════════════════════════════

void setup() {

  // 初始化串口（用于调试）

  Serial.begin(115200);

  delay(1000);

  // 打印启动信息

  printWelcome();

  // 配置GPIO引脚

  initPins();

  // MQ-2传感器预热

  preheatingMQ2();

  // 系统自检

  selfTest();

  // 记录启动时间

  systemStartTime = millis();

  Serial.println("━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━");

  Serial.println("🛡️  系统监控已启动...\n");

}

// ═══════════════════════════════════════════════════════════════

// 主循环

// ═══════════════════════════════════════════════════════════════

void loop() {

  // ─────────────────────────────────────────────────────────

  // 1. 读取传感器数据

  // ─────────────────────────────────────────────────────────

  bool flameDetected = (digitalRead(FLAME\_SENSOR\_PIN) == LOW);  // LOW=检测到火焰

  int gasValue = analogRead(MQ2\_ANALOG\_PIN);                    // 气体浓度值 (0-4095)

  bool gasAlert = (digitalRead(MQ2\_DIGITAL\_PIN) == LOW);        // LOW=超阈值

  // ─────────────────────────────────────────────────────────

  // 2. 判断报警级别（优先级：火焰 > 气体）

  // ─────────────────────────────────────────────────────────

  AlarmLevel previousLevel = currentLevel;

  if (flameDetected) {

    // 🔴 最高优先级：检测到明火

    currentLevel = FIRE\_ALARM;

  } else if (gasAlert || gasValue > GAS\_THRESHOLD) {

    // 🟡 次优先级：检测到气体/烟雾

    currentLevel = GAS\_WARNING;

  } else {

    // 🟢 安全状态

    currentLevel = SAFE;

  }

  // ─────────────────────────────────────────────────────────

  // 3. 状态改变时打印详细信息并计数

  // ─────────────────────────────────────────────────────────

  if (currentLevel != previousLevel) {

    if (currentLevel != SAFE) {

      alarmCount++;  // 报警次数+1

    }

    printDetailedStatus(flameDetected, gasValue, gasAlert);

  }

  // ─────────────────────────────────────────────────────────

  // 4. 执行对应的报警动作

  // ─────────────────────────────────────────────────────────

  switch(currentLevel) {

    case FIRE\_ALARM:

      executeFireAlarm();

      break;

    case GAS\_WARNING:

      executeGasWarning();

      break;

    case SAFE:

      executeAllClear();

      break;

  }

  // ─────────────────────────────────────────────────────────

  // 5. 定期打印监控数据

  // ─────────────────────────────────────────────────────────

  if (millis() - lastPrintTime > STATUS\_PRINT\_INTERVAL) {

    lastPrintTime = millis();

    printMonitoringData(flameDetected, gasValue);

  }

  // 短暂延迟

  delay(SCAN\_INTERVAL);

}

// ═══════════════════════════════════════════════════════════════

// 初始化引脚

// ═══════════════════════════════════════════════════════════════

void initPins() {

  Serial.println("📌 配置GPIO引脚...");

  // 输入引脚

  pinMode(FLAME\_SENSOR\_PIN, INPUT);

  pinMode(MQ2\_DIGITAL\_PIN, INPUT);

  // MQ2\_ANALOG\_PIN 不需要pinMode（自动配置为模拟输入）

  // 输出引脚

  pinMode(BUZZER\_PIN, OUTPUT);

  pinMode(LED\_FIRE, OUTPUT);

  pinMode(LED\_GAS, OUTPUT);

  // 初始状态：全部关闭

  digitalWrite(BUZZER\_PIN, HIGH);  // 低电平触发蜂鸣器，HIGH=关闭

  digitalWrite(LED\_FIRE, LOW);

  digitalWrite(LED\_GAS, LOW);

  Serial.printf("   火焰传感器: GPIO%d\n", FLAME\_SENSOR\_PIN);

  Serial.printf("   MQ-2模拟:   GPIO%d (ADC)\n", MQ2\_ANALOG\_PIN);

  Serial.printf("   MQ-2数字:   GPIO%d\n", MQ2\_DIGITAL\_PIN);

  Serial.printf("   蜂鸣器:     GPIO%d\n", BUZZER\_PIN);

  Serial.printf("   红色LED:    GPIO%d (火灾)\n", LED\_FIRE);

  Serial.printf("   黄色LED:    GPIO%d (预警)\n\n", LED\_GAS);

}

// ═══════════════════════════════════════════════════════════════

// MQ-2传感器预热

// ═══════════════════════════════════════════════════════════════

void preheatingMQ2() {

  Serial.println("🔥 MQ-2传感器预热中...");

  Serial.println("   (建议预热30秒以确保读数准确)");

  Serial.println("   提示：预热期间黄色LED会闪烁\n");

  int totalSeconds = PREHEATING\_TIME / 1000;

  for(int i = totalSeconds; i > 0; i--) {

    Serial.printf("   ⏳ 倒计时: %2d 秒 | 当前MQ-2值: %4d\r",

                  i, analogRead(MQ2\_ANALOG\_PIN));

    // 黄色LED闪烁提示预热中

    digitalWrite(LED\_GAS, i % 2);

    delay(1000);

  }

  digitalWrite(LED\_GAS, LOW);

  Serial.println("\n✅ 预热完成！");

  // 显示预热后的基准值

  int baselineValue = 0;

  for(int i = 0; i < 10; i++) {

    baselineValue += analogRead(MQ2\_ANALOG\_PIN);

    delay(50);

  }

  baselineValue /= 10;

  Serial.printf("   📊 MQ-2基准值: %d (阈值: %d)\n\n", baselineValue, GAS\_THRESHOLD);

}

// ═══════════════════════════════════════════════════════════════

// 系统自检

// ═══════════════════════════════════════════════════════════════

void selfTest() {

  Serial.println("🔧 系统自检中...\n");

  // ─── 测试LED ───

  Serial.println("   [1/3] 测试LED指示灯...");

  Serial.print("         红色LED: ");

  digitalWrite(LED\_FIRE, HIGH);

  delay(300);

  Serial.print("✓ ");

  digitalWrite(LED\_FIRE, LOW);

  Serial.print("黄色LED: ");

  digitalWrite(LED\_GAS, HIGH);

  delay(300);

  Serial.println("✓");

  digitalWrite(LED\_GAS, LOW);

  delay(200);

  // ─── 测试蜂鸣器 ───

  Serial.println("   [2/3] 测试有源蜂鸣器...");

  Serial.print("         蜂鸣器: ");

  digitalWrite(BUZZER\_PIN, LOW);   // 低电平触发

  delay(200);

  digitalWrite(BUZZER\_PIN, HIGH);  // 高电平关闭

  Serial.println("✓ (应该听到一声\"滴\")\n");

  delay(300);

  // ─── 读取传感器初始值 ───

  Serial.println("   [3/3] 读取传感器初始状态...");

  bool flameStatus = digitalRead(FLAME\_SENSOR\_PIN);

  int gasValue = analogRead(MQ2\_ANALOG\_PIN);

  bool gasDigital = digitalRead(MQ2\_DIGITAL\_PIN);

  Serial.printf("         火焰传感器: %s\n",

                flameStatus == HIGH ? "✓ 无火焰" : "⚠️ 检测到火焰!");

  Serial.printf("         MQ-2模拟值: %d\n", gasValue);

  Serial.printf("         MQ-2数字值: %s\n",

                gasDigital == HIGH ? "✓ 正常" : "⚠️ 超阈值!");

  Serial.println("\n✅ 自检完成！所有组件正常\n");

  delay(1000);

}

// ═══════════════════════════════════════════════════════════════

// 🔴 火灾警报动作

// ═══════════════════════════════════════════════════════════════

void executeFireAlarm() {

  // 红色和黄色LED全部点亮

  digitalWrite(LED\_FIRE, HIGH);

  digitalWrite(LED\_GAS, HIGH);

  // 急促报警声（有源蜂鸣器：快速开关）

  // 模式：滴滴滴-停-滴滴滴-停

  for(int i = 0; i < 3; i++) {

    digitalWrite(BUZZER\_PIN, LOW);   // 低电平触发=响

    delay(100);

    digitalWrite(BUZZER\_PIN, HIGH);  // 高电平=停

    delay(100);

  }

  delay(300);  // 短暂停顿

}

// ═══════════════════════════════════════════════════════════════

// 🟡 气体预警动作

// ═══════════════════════════════════════════════════════════════

void executeGasWarning() {

  // 仅黄色LED点亮

  digitalWrite(LED\_FIRE, LOW);

  digitalWrite(LED\_GAS, HIGH);

  // 缓慢报警声（有源蜂鸣器：长响-长停）

  // 模式：滴~~~-停-滴~~~-停

  digitalWrite(BUZZER\_PIN, LOW);   // 低电平触发=响

  delay(200);

  digitalWrite(BUZZER\_PIN, HIGH);  // 高电平=停

  delay(800);  // 较长停顿

}

// ═══════════════════════════════════════════════════════════════

// 🟢 安全状态

// ═══════════════════════════════════════════════════════════════

void executeAllClear() {

  // 全部关闭

  digitalWrite(LED\_FIRE, LOW);

  digitalWrite(LED\_GAS, LOW);

  digitalWrite(BUZZER\_PIN, HIGH);  // 低电平触发蜂鸣器，HIGH=关闭

}

// ═══════════════════════════════════════════════════════════════

// 打印欢迎信息

// ═══════════════════════════════════════════════════════════════

void printWelcome() {

  Serial.println("\n");

  Serial.println("╔════════════════════════════════════════════════╗");

  Serial.println("║                                                ║");

  Serial.println("║    🚨 ESP32-C6 智能火灾报警系统 v2.0 🚨       ║");

  Serial.println("║                                                ║");

  Serial.println("║    功能：火焰检测 + 气体检测 + 分级报警        ║");

  Serial.println("║    作者：iCEasy IoT Lab                        ║");

  Serial.println("║    日期：2025-10-15                            ║");

  Serial.println("║                                                ║");

  Serial.println("╚════════════════════════════════════════════════╝");

  Serial.println();

}

// ═══════════════════════════════════════════════════════════════

// 打印详细状态（状态改变时）

// ═══════════════════════════════════════════════════════════════

void printDetailedStatus(bool flame, int gas, bool gasAlert) {

  Serial.println("\n╔════════════════════════════════════════════════╗");

  Serial.println("║            ⚠️  警报状态变更  ⚠️                ║");

  Serial.println("╚════════════════════════════════════════════════╝");

  // 时间信息

  unsigned long runTime = (millis() - systemStartTime) / 1000;

  Serial.printf("⏰ 运行时间: %lu 秒 | 报警次数: %u\n", runTime, alarmCount);

  Serial.printf("📅 当前时间: %lu ms\n", millis());

  Serial.println("━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━");

  // 传感器状态

  Serial.println("📊 传感器状态:");

  Serial.printf("   🔥 火焰传感器: %s\n",

                flame ? "🔴 检测到火焰!" : "✅ 正常");

  Serial.printf("   💨 MQ-2气体值: %d (阈值:%d) %s\n",

                gas, GAS\_THRESHOLD,

                gas > GAS\_THRESHOLD ? "🟡 超阈值!" : "✅");

  Serial.printf("   💨 MQ-2数字输出: %s\n",

                gasAlert ? "🟡 报警!" : "✅ 正常");

  Serial.println("━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━");

  // 当前级别和动作

  Serial.println("🚦 系统状态:");

  switch(currentLevel) {

    case FIRE\_ALARM:

      Serial.println("   级别: 🔴🔴🔴 【火灾警报】 🔴🔴🔴");

      Serial.println("   动作:");

      Serial.println("      → 红色LED常亮");

      Serial.println("      → 黄色LED常亮");

      Serial.println("      → 蜂鸣器急促报警 (滴滴滴-停-滴滴滴)");

      Serial.println("   建议:");

      Serial.println("      ⚠️  立即疏散人员！");

      Serial.println("      ⚠️  拨打火警电话119！");

      Serial.println("      ⚠️  使用灭火器灭火（小火情况下）！");

      break;

    case GAS\_WARNING:

      Serial.println("   级别: 🟡🟡 【气体预警】 🟡🟡");

      Serial.println("   动作:");

      Serial.println("      → 黄色LED常亮");

      Serial.println("      → 蜂鸣器缓慢报警 (滴~停~滴~停)");

      Serial.println("   可能原因:");

      Serial.println("      • 检测到烟雾（可能是早期火灾）");

      Serial.println("      • 检测到可燃气体（煤气/天然气泄漏）");

      Serial.println("   建议:");

      Serial.println("      ⚠️  立即检查周围环境");

      Serial.println("      ⚠️  打开门窗通风");

      Serial.println("      ⚠️  检查燃气阀门是否关闭");

      break;

    case SAFE:

      Serial.println("   级别: 🟢 【安全】");

      Serial.println("   动作:");

      Serial.println("      → 所有LED熄灭");

      Serial.println("      → 蜂鸣器静音");

      Serial.println("      → 继续监控中...");

      break;

  }

  Serial.println("╚════════════════════════════════════════════════╝\n");

}

// ═══════════════════════════════════════════════════════════════

// 打印监控数据（定期）

// ═══════════════════════════════════════════════════════════════

void printMonitoringData(bool flame, int gas) {

  Serial.printf("📊 [监控] 状态:%s | 火焰:%s | 气体:%4d | 运行:%lu秒 | 报警:%u次\n",

                getLevelIcon(currentLevel),

                flame ? "🔴检测" : "⚪无  ",

                gas,

                (millis() - systemStartTime) / 1000,

                alarmCount);

}

// ═══════════════════════════════════════════════════════════════

// 获取级别图标

// ═══════════════════════════════════════════════════════════════

const char\* getLevelIcon(AlarmLevel level) {

  switch(level) {

    case FIRE\_ALARM:   return "🔴火灾";

    case GAS\_WARNING:  return "🟡预警";

    case SAFE:         return "🟢安全";

    default:           return "❓未知";

  }

}