2025 年“西门子杯”中国智能制造挑战赛

智能制造工程设计与应用类：工业嵌入式系统开发方向（筹）

全国初赛 工程设计文件

参赛队伍编号：____2025244579____

2025 年 6 月 19 日

目录

1 工程任务分析

1.1 系统整体架构与核心功能概述

本项目基于 GD32F470VET6 微控制器设计工业级嵌入式系统如下图 1.1 所示，集成电压采集、处理、显示、存储及加密传输功能，满足工业场景对可靠性、实时性与安全性的严苛要求。核心任务包括可以通过串口通信与按键实现人机交互，此设计采用 UART 实现全双工异步通信，波特率配置为 115200bps，并且通过 DMA 控制器实现数据收发缓冲，这样可以有效降低 CPU 占用率，然后采用键盘接口，配置外部中断触发模式，结合软件去抖动算法，实现参数调整等相关操作功能。接着可以完成硬件自检与参数配置，其中自检和参数配置都可以通过命令实现，然后自检可以通过串口输出相应的结果。实时采集电压数据并进行阈值判断，将数据以明文或加密形式存储至 TF 卡，同时通过 OLED 与 LED 实现状态可视化。

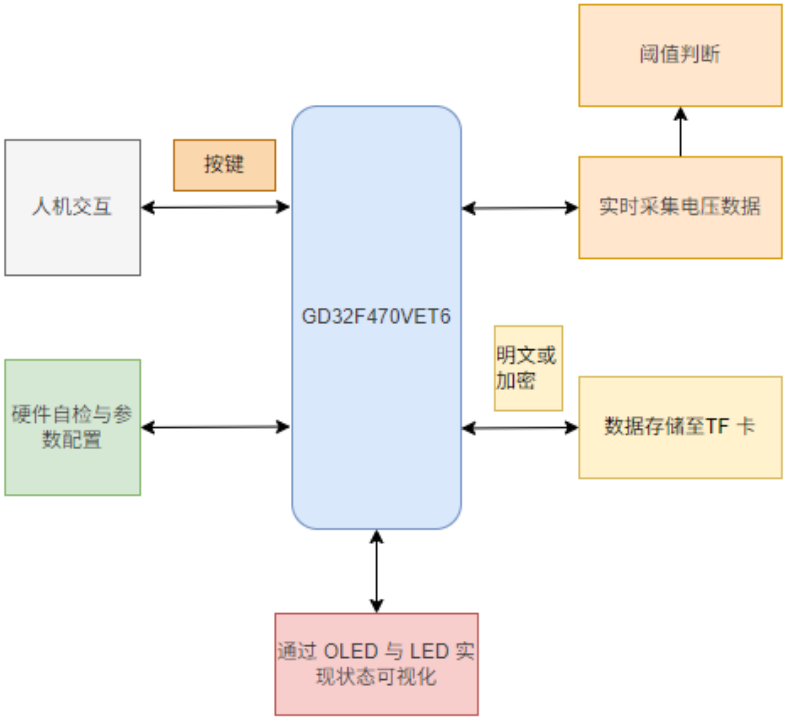


图 1.1 系统整体架构

1.2 核心功能模块详细分析

1.2.1 系统自检机制

通过串口输入“test”指令触发自检流程，系统依次检测 Flash（读取芯片 ID 验证型号及工作状态）、TF 卡（检测卡座存在性及文件系统初始化状态）、RTC 时钟（校验时钟精度与备份电池电压）。自检结果通过串口格式化输出，其中具体包含 flash、TF card、flash ID、TF card memory 和 RTC 标识以及相关自检参数。具体示例如下所示。

```
====system selftest====
flash.....ok
TF card.....ok
flash ID:0xCxxxxx
TF card memory: xxxx KB
RTC:2025-01-01 01:00:50
====system selftest====
```

系统通过串口输入相关指令来实现触发自检流程如下图1.2所示，在检测TF卡阶段，系统通过SDIO接口发送CMD0指令进行卡初始化，并监测CD引脚电平变化。如果插入的TF卡系统未检测到，此时系统则通过串口进行相应输出，在串口软件上则看到返回“can not find TF card”的字符串。在Flash检测阶段，系统通过SPI接口发送JEDEC ID读取指令（0x9F），验证返回值是否匹配0xEF4017（Winbond W25Q128），若不匹配则设置ERROR_CODE第0位为1（0x01），并通过串口输出错误信息。TF卡容量检测阶段，系统通过SDIO接口依次发送CMD8指令进行电压范围检查，再发送CMD58指令读取OCR寄存器，解析卡容量信息。RTC状态检测阶段，系统通过检查时钟振荡器状态位（DS3231的OSF标志），如果发生异常时，系统可以实现自动校准。

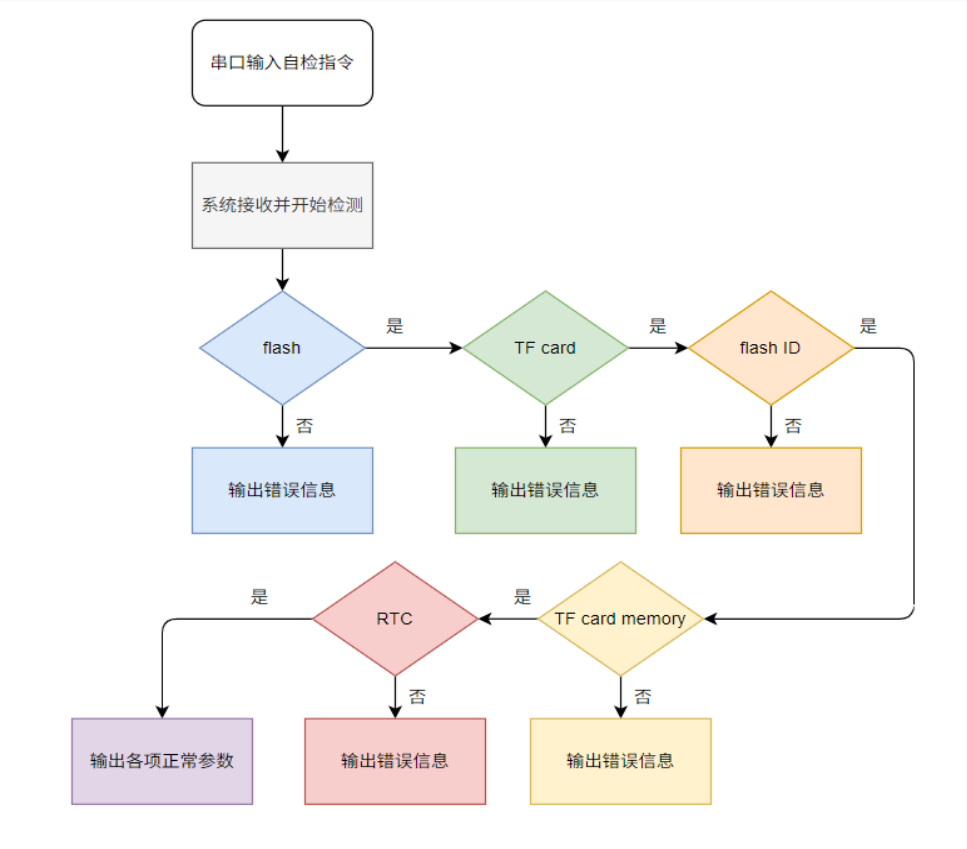


图1.2系统自检流程

1.2.2 实时时钟（RTC）管理

通过"RTC Config"指令可配置系统实时时钟，支持 ISO 8601 格式（"YYYY-MM-DD HH:MM:SS"）及紧凑格式（"YYMMDDHHMMSS"）输入。指令处理流程如下：

（1）解析输入字符串，验证日期有效性（如闰年检测、月份范围 1-12）和时间合法性（24 小时制）

（2）将时间数据转换为 BCD 编码，通过 I2C 接口写入 RTC 芯片（如 PCF8563）的秒/分/时/日/月/年寄存器

（3）同步更新系统时间变量，触发看门狗定时器重启

（4）返回格式化确认信息："RTC Config Success: 2025-01-01 12:00:30 (Weekday: 3)"（含自动计算的星期信息）

1.2.3 配置文件读取

从 TF 卡读取 config.ini 文件，提取变比（Ratio）和阈值（Limit）参数，文件格式示例如下所示。

```
[Ratio]
Ch0 = 1.99
[Limit]
Ch0 = 10.9
```

文件系统增强功能如下：

（1）容错解析器：读取 config.ini 时自动跳过注释行（以#开头），支持等号/冒号分隔符，对缺失段落（如缺少[Ratio]）填充默认值并记录警告。

（2）版本控制：检测文件头部的版本标识（如"VER=1.2"），兼容旧版配置格式，升级时自动转换参数。

（3）热更新机制：运行时修改 config.ini 后，发送"config reload"指令可无缝加载新参数，无需重启系统。

（3）备份策略：维护 config.ini.bak 备份文件，主文件损坏时自动恢复，并通过 LED3 黄色闪烁提示。

异常处理：

（1）参数格式错误（如 Ch0=abc）时，跳过该行并记录错误日志，保持其他有效参数。

（2）若文件不存在，返回“config.ini file not found”。

1.2.4 变比与阈值动态设置

变比设置指令 “ratio”：

（1）范围检查：0≤Ratio≤100，输入 100.5 时提示："Ratio invalid (100.5), valid range: 0-100"。

（2）步进控制：支持 0.01 精度，通过按键调整时采用非线性步进（长按快速调整，短按微调）。

（3）关联校验：当变比修改时，自动按比例调整阈值（Limit_new = Limit_old × Ratio_new/Ratio_old），避免误触发。

阈值设置指令“limit”：

（1）范围检查：0≤Limit≤500，输入 510.12 时保持原值并提示："Limit exceeds 500.00, value unchanged"。

（2）历史保护：记录最近 5 次阈值，可通过"limit history"指令回滚。

（3）动态适配：根据当前变比自动计算电压阈值（Voltage_Limit = Limit × Ratio），超限判断基于实际电压值。

1.2.5 参数持久化存储

通过“config save”将参数存入 flash，“config read”从 flash 读取，断电后数据不丢失。

（1）分区设计：配置区：存储 Ratio/Limit 等参数，采用 ECC 校验，写入寿命>10 万次。日志区：循环记录最近 100 次配置变更，支持按时间戳回溯。备份区：存储关键参数副本，主区损坏时自动切换。

（2）磨损均衡：采用动态地址映射算法，均匀分配写入操作，延长 Flash 寿命。

（3）加密存储：配置参数采用 AES128 加密，密钥通过设备唯一 ID 生成，防止参数篡改。

1.2.6 采样控制逻辑

（1）启动/停止控制：串口指令"start/stop"：通过 GPIO 控制采样使能信号，LED1 以 1Hz 闪烁指示运行状态。按键 KEY1 短按切换采样状态，长按 3 秒强制停止并保存数据。

（2）采样周期调整：KEY2/KEY3/KEY4 循环切换 5s/10s/15s 周期，通过 OLED 显示当前设置。周期参数持久化，存储于 Flash 配置区，断电重启后自动加载。

（3）低功耗优化：采样间隔期间，MCU 进入 STOP 模式，ADC/SDIO 等外设断电，典型功耗<20μA。定时唤醒中断服务程序（ISR）执行数据采集，唤醒时间<5μs。

1.2.7 超限检测与提示

（1）快速响应：利用 MCU 内置比较器（COMP1/COMP2），配置为窗口比较模式，当 ADC 值>阈值×变比时触发 EXTI 中断，响应延迟<50μs（含中断服务程序）。

（2）防抖算法：软件防抖，连续 3 次采样超限才触发告警，避免瞬时干扰。硬件滤波，比较器输入端接 RC 低通滤波器（τ=10ms），抑制高频噪声。

（3）多级告警：LED2：超限时长<1s 时橙色闪烁，≥1s 时红色常亮。串口输出包含超限电压值、持续时间及恢复提示，如："2025-06-20 14:35:00 ch0=10.5V OverLimit(10.00)! Duration: 00:00:02, Recovered"。

1.2.8 数据加密与解码

（1）时间戳转换：支持扩展格式，Unix 时间戳（4 字节 HEX）+ 毫秒（2 字节 HEX），如"2025-01-01 00:30:05.123"→"0x5E3A8B41 0x04BB"。闰秒补偿，预置闰秒表，自动调整 UTC 时间戳。

（2）电压编码：高精度编码：12.5V→整数 0x000C（12）+ 小数 0x8000（0.5×65536），分辨率达 1/65536≈0.0015%。动态范围，支持 0-65535 整数部分（对应 0-1023V）及 0-65535 小数部分。

（3）加密标记：超限数据加密后添加""前缀，如"000C8000"。解密指令"unhide"自动检测并恢复原始格式，支持批量解密文件夹。

1.2.9 多类型数据存储规则

（1）采样数据：分片存储，每 10 条数据新建文件，文件名包含秒级时间戳（如"sampleData_20250620143500_001.txt"）。压缩算法，对连续相同数据采用 RLE 压缩，节省存储空间。

（2）超阈值数据：独立存储：与采样数据隔离，避免覆盖重要告警记录。关联记录：存储超限发生前 10 秒的采样数据，便于事件分析。

（3）日志数据：增量 ID：每次上电 ID 自增，支持日志轮转（最多保留 100 个文件）。内容加密：系统日志采用 AES256 加密，防止敏感信息泄露。

（4）加密模式：全局开关：通过"hide on/off"指令切换，启用后所有采样数据加密存储。混合存储：超限数据同时写入明文/加密文件夹，确保关键事件可追溯。

2 系统单元功能分析设计

2.1 系统自检功能单元

任务名称：系统硬件自检

功能描述：通过串口指令触发硬件组件状态检测，包括 flash、TF 卡和 RTC 时钟。

输入：串口输入指令 “test”。

处理逻辑：读取 flash 芯片 ID，验证硬件连接。检测 TF 卡存在性及存储空间。获取 RTC 当前时间戳。

输出：

（1）自检通过示例：

```
====system selftest====
flash.....ok
TF card.....ok
flash ID:0xCxxxxx
TF card memory: xxxx KB
RTC:2025-01-01 01:00:50
====system selftest====
```

（2）自检失败示例（TF 卡未检测到）：

```
====system selftest====
flash.....ok
TF card.....error
flash ID:0xCxxxxx
can not find TF card
RTC:2025-01-01 01:00:50
====system selftest====
```

功能代码如下：

```
if(strstr((char *)sp.uart_buff, "test") != NULL){
    system_test();
}
```

```c
void system_test()
{
  printf("======system selftest======\r\n");
  DSTATUS sd_status = disk_initialize(0);
  uint32_t flash_id = spi_flash_read_id();
  printf("flash.....ok\r\n");
  if(sd_status == 0) {
      printf("TF card......ok\r\n");
      printf("flash ID:0x%06X\r\n", flash_id);
      uint32_t capacity = sd_card_capacity_get();
      printf("TF card memory: %d KB\r\n", capacity);
  } else {
      printf("TF card.......error\r\n");
      printf("flash ID:0x%06X\r\n", flash_id);
      printf("can not find TF card\r\n");
  }
  printf("Time:20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second);
  log_write("system selftest");
}
```

2.2 RTC 时钟管理单元

任务名称：实时时钟配置与读取。

功能描述：支持串口设置基准时间并实时查询当前时间。

时间设置：输入：串口指令“RTC Config”+ 标准时间（如“2025-01-01 12:00:30”）

处理：解析时间字符串并写入 RTC 模块

输出： RTC Config success .Time:2025-01-0112:00:30”

时间查询：

输入：串口指令“RTC now”

输出：“Current Time: 2025-01-01 12:00:30”

功能代码：

```c
if(strstr((char *)sp.uart_buff, "RTC Config") != NULL)
{
    printf("Input Datetime\r\n");
        rtc_set_flag=1;
}
if((strstr((char *)sp.uart_buff, "2025-06")!= NULL) && rtc_set_flag==1){
    //---------------------重置时钟
    int year, month, day, hour, minite, secend;//2025-06-15 12:12:12
    year = (sp.uart_buff[2] - 0x30)*10 + (sp.uart_buff[3] - 0x30);
    month = (sp.uart_buff[5] - 0x30)*10 + (sp.uart_buff[6] - 0x30);
    day = (sp.uart_buff[8] - 0x30)*10 + (sp.uart_buff[9] - 0x30);
    hour = (sp.uart_buff[11] - 0x30)*10 + (sp.uart_buff[12] - 0x30);
    minite = (sp.uart_buff[14] - 0x30)*10 + (sp.uart_buff[15] - 0x30);
    secend = (sp.uart_buff[17] - 0x30)*10 + (sp.uart_buff[18] - 0x30);
    rtc_reset(year,month, day, hour, minite, secend);
    rtc_set_flag=0;
}
```

```
if(strstr((char *)sp.uart_buff, "RTC now") != NULL){
  printf("Current Time:20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second);
   }
```

2.3 配置参数管理单元

2.3.1 子任务 1：配置文件读取

输入：串口指令 “conf”。

处理：从 TF 卡根目录查找 config.ini 文件,解析文件获取变比(Ratio)和阈值(Limit)参数。

输出：文件不存在：“config.ini file not found”。读取成功：“Ratio=xxXx Limit=XXXX config read success”。

文件格式：

```
[Ratio]
Ch0 = 1.99
[Limit]
Ch0 = 10.9
```

功能代码：

```
        // 解析配置项
        else if(ratio__found && strstr(line_buffer, "Ch0") != NULL) {
            char* equal_sign = strchr(line_buffer, '=');
            if(equal_sign != NULL) {
                equal_sign++; // 跳过等号
                while(*equal_sign == ' ') equal_sign++; // 跳过空格
                sp.ratio_data = atof(equal_sign);
                ratio_ok = 1;
            }
        }
        else if(limit__found && strstr(line_buffer, "Ch0") != NULL) {
            char* equal_sign = strchr(line_buffer, '=');
            if(equal_sign != NULL) {
                equal_sign++; // 跳过等号
                while(*equal_sign == ' ') equal_sign++; // 跳过空格
                sp.limit_data = atof(equal_sign);
                limit_ok = 1;
            }
        }
    }


    f_close(&config_file);


    if(ratio_ok && limit_ok)
    {
       printf("ratio: %.2f\r\n", sp.ratio_data);
       printf("limit: %.2f\r\n", sp.limit_data);
    }
}
```

```c
void show_conf()
{
  FIL config_file;
  FRESULT result;
  char line_buffer[128];
   uint8_t ratio__found = 0;
    uint8_t limit__found = 0;
    uint8_t ratio_ok = 0;
    uint8_t limit_ok = 0;


     result = f_open(&config_file, "config.ini", FA_READ);
    if(result != FR_OK) {
        printf("config.ini file not found.\r\n");
        return;
    }

    while(f_gets(line_buffer, sizeof(line_buffer), &config_file) != NULL) {
        // 移除行末的换行符
//      printf("%s",line_buffer);
        char* newline = strchr(line_buffer, '\n');
        if(newline) *newline = '\0';
        char* carriage = strchr(line_buffer, '\r');
        if(carriage) *carriage = '\0';

        // 跳过空行和注释
        if(strlen(line_buffer) == 0 || line_buffer[0] == ';' || line_buffer[0] == '#') {
            continue;
        }

        // 检查是否是节标题
        if(line_buffer[0] == '[') {
            if(strstr(line_buffer, "[Ratio]") != NULL) {
                ratio__found = 1;
                limit__found = 0;
            } else if(strstr(line_buffer, "[Limit]") != NULL) {
                limit__found = 1;
                ratio__found = 0;
            } else {
                ratio__found = 0;
                limit__found = 0;
            }
        }
```

```c
if((strstr((char *)sp.uart_buff, "conf") != NULL) && (strstr((char *)sp.uart_buff, "e") == NULL)){
    show_conf();
}
```

2.3.2 子任务 2：变比设置

输入：串口指令 “ratio”+ 新变比值（0-100 浮点数）。

处理：验证输入值有效性（范围 / 数据类型），有效输入时更新变比参数，无效时保持原值

2.3.3 子任务 3：阈值设置

输入：串口指令“limit”+新阈值（0-300 浮点数）。处理逻辑同变比设置。

功能代码：

```
  if(strstr((char *)sp.uart_buff, "limit") != NULL){
      read_Limit();
log_write("limit config");
      limit_set_flag = 1;
  }
```

```
void set_Limit(char *Limit_data)
{
  char string[100] = {0};
    double temp = atof(Limit_data);
    if(temp >= 0 && temp <= 200.1f){
    sprintf(string,"limit config success to %.2f",temp);
    log_write(string);
        printf("limit modified success\r\n");
        sp.limit_data = temp;
        printf("Limit = %0.1f\r\n",sp.limit_data);
    }
    else{
    sprintf(string,"limit invaild");
    log_write(string);
        printf("limit invaild\r\n");
        printf("Limit = %0.1f\r\n",sp.limit_data);
    }
}
```

2.3.4 子任务 4：参数持久化存储

输入：“config save”。

输出：打印当前参数并提示 “save parameters to flash”。

读取操作：

输入：“config read”。

输出：“read parameters from flash”+ 参数值。

存储介质：外部 flash，支持掉电数据保持。

Flash 扇区管理：

参数	存储地址	大小
变比	0x0800C000	4 字节
阈值	0x0800C004	4 字节
CRC16 校验码	0x0800C008	2 字节

示例代码：

```
void config_save()
{
  flash_data temp_ratio,temp_limit;
    printf("ratio = %.2f\r\n",sp.ratio_data);
    printf("limit = %.2f\r\n",sp.limit_data);
  temp_ratio.f = (float)sp.ratio_data;
  temp_limit.f = (float)sp.limit_data;
    printf("save parameters to flash\r\n");
    spi_flash_sector_erase(0);//擦除10个地址的内容
    spi_flash_buffer_write("Device_ID:2025-CIMC-2025244579",0,30);//将flash_buff的内容写入flash
  spi_flash_wait_for_write_end();
    spi_flash_buffer_write((uint8_t *)&temp_ratio,30,4);//将str0的内容写入flash
  spi_flash_wait_for_write_end();
    spi_flash_buffer_write((uint8_t *)&temp_limit,34,4);//将str1的内容写入flash
  spi_flash_wait_for_write_end();
}

void config_read(){
  flash_data temp_ratio,temp_limit;
    printf("save parameters from flash\r\n");
    spi_flash_buffer_read((uint8_t *)&temp_ratio,30,4);
    spi_flash_buffer_read((uint8_t *)&temp_limit,34,4);
  sp.ratio_data = temp_ratio.f;
  sp.limit_data = temp_limit.f;
    printf("ratio = %0.2f\r\n",sp.ratio_data);
    printf("limit = %0.2f\r\n",sp.limit_data );
}
```

```
  if(strstr((char *)sp.uart_buff, "config save") != NULL){
log_write("config_save");
      config_save();
  }
  if(strstr((char *)sp.uart_buff, "config read") != NULL){
log_write("config_read");
      config_read();
  }
```

2.4 数据采样控制单元

2.4.1 子任务 1：采样启停控制

串口控制：

启动：输入 “**start**”，LED1 以 1s 周期闪烁，按设定周期输出采样数据。

停止：输入 “**stop**”，LED1 熄灭，OLED 显示 “system idle”。

按键控制：按下 KEY1 切换采样状态（启动 / 停止），显示逻辑同串口控制。采样数据输出格式：“2025-01-01 00:30:05 ch0-10.5V”

示例代码：

```
  if(strstr((char *)sp.uart_buff, "start") != NULL){
log_write("sample start - cycle 5s (command)");
      sp.led_flash = 1;
      printf("Periodic Sampling\r\n");
      printf("sample cycle: %ds\r\n",sp.sample_cycle);
  }
  if(strstr((char *)sp.uart_buff, "stop") != NULL){
log_write("sample stop (command)");
      sp.led_flash = 0;
      printf("Periodic Sampling STOP\r\n");
  }
```

```
case 1://按键1切换模式
    sp.led_flash = !sp.led_flash;
    if(sp.led_flash==0)
    {
log_write("sample stop (key press)");
        printf("Periodic Sampling STOP\r\n");
    }
    else
    {
log_write("sample start (key press)");
        printf("Periodic Sampling\r\n");
        printf("sample cycle: %ds\r\n",sp.sample_cycle);
    }
    break;
```

2. 4. 2 子任务 2：采样周期调整

按键控制：KEY2/KEY3/KEY4 对应周期 5s/10s/15s。

持久化要求：断电重启后保留最新设置周期。

输出示例：“sample cycle adjust:10s”。

示例代码：

```
  case 2:
log_write("cycle switch to 5s (key press)");
      sp.sample_cycle = 5;
      printf("sample cycle: %ds\r\n",sp.sample_cycle);
      break;
  case 3:
log_write("cycle switch to 10s (key press)");
      sp.sample_cycle = 10;
      printf("sample cycle: %ds\r\n",sp.sample_cycle);
      break;
  case 4:
log_write("cycle switch to 15s (key press)");
      sp.sample_cycle = 15;
      printf("sample cycle: %ds\r\n",sp.sample_cycle);
      break;
```

2.4.3 子任务 3：超限检测与提示

触发条件：采样电压值 > 阈值参数。

响应动作：点亮 LED2。串口输出带“OverLimit” 标记的数据：2025-01-01 00:30:05 ch0=10.5V OverLimit(10.00)!

恢复逻辑：电压值≤阈值时，LED2 熄灭且取消标记。

示例代码：

```
//overlimit执行函数
void overlimit_proc(char *str)
{
    FRESULT result;
    static char overlimit_filename[64];
    if(sp.overlimit_log_num == 0){
        sprintf(overlimit_filename, "overLimit/overLimit20%0.2x%0.2x%0.2x%0.2x%0.2x%0.2x.txt",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second);
        result = f_open(&sp.overlimit_file, overlimit_filename, FA_CREATE_ALWAYS | FA_WRITE);
        f_close(&sp.overlimit_file);
        result = f_open(&sp.overlimit_file, overlimit_filename, FA_CREATE_ALWAYS | FA_WRITE);
    }
    else if(sp.overlimit_log_num == 10)
    {
        f_close(&sp.overlimit_file);
        sp.overlimit_log_num = 0;
        return;
    }
    sp.overlimit_log_num++;
    UINT bytes_written;
    result = f_write(&sp.overlimit_file, str, strlen(str), &bytes_written);
    if(result == FR_OK && bytes_written == strlen(str)) {
        f_sync(&sp.overlimit_file);
    }
}
```

2.5 数据处理与加密单元

任务名称：时间与电压数据编码转换

输入：串口指令 “hide”

处理逻辑：

时间戳转换：UTC 时间→Unix 时间戳（4 字节 HEX）

电压值转换：整数部分（2 字节 HEX，高位在前）。小数部分 ×65536 后取整（2 字节 HEX，高位在前）

输出示例：原始数据：2025-01-01 12:30:45 ch0=12.5V。加密后：6774C4F5000C8000（时间戳 1735705845 + 整数 12 + 小数 0.5×65536=32768）

超限标记：加密数据后添加 “ ”，*如 “6774C4F5000C8000”*

解码恢复：输入 “unhide” 还原原始格式

示例代码：

```
//将时间戳转换为Unix时间戳
void hide_to_unix()
{
    struct tm time_info;
    time_t unix_timestamp;
    char out[100];
    char buf[100];
    char zhengshu[50];//ADC整数部分
    char xiaoshu[50];//ADC小数部分
    char year[2];
    char month[2];
    char day[2];
    char hour[2];
    char minute[2];
    char second[2];
    char hide_print[100];

    int volt_temp = sp.adc_vlot * 10;
    sprintf(zhengshu, "%4x", (volt_temp - (volt_temp % 10)) / 10);
    peach_to_0(zhengshu);
    float temp =  (volt_temp % 10) / 10.0f;
    sprintf(xiaoshu, "%4x", (int)(temp * 65536) );
    peach_to_0(xiaoshu);
    sprintf(year,"%x",sp.year);
    sprintf(month,"%x",sp.month);
    sprintf(day,"%x",sp.day);
    sprintf(hour,"%x",sp.hour);
    sprintf(minute,"%x",sp.minute);
    sprintf(second,"%x",sp.second);
    // 设置时间信息（例如 2024 年 9 月 19 日 15:30:45）
    time_info.tm_year = 2000 + atoi(year) - 1900; // tm_year 是从 1900 年开始的年数
    time_info.tm_mon = atoi(month) - 1;        // tm_mon 是从 1 月开始的月数（0-11）
    time_info.tm_mday = atoi(day);
    time_info.tm_hour = atoi(hour)-8;
    time_info.tm_min = atoi(minute);
    time_info.tm_sec = atoi(second);
    // 将结构化时间转换为 Unix 时间戳
    // 将结构化时间转换为 Unix 时间戳
    unix_timestamp = mktime(&time_info);
```

```
//字符串拼接
sprintf(buf, "%4x",unix_timestamp);
sprintf(out, "%s%s%s",buf, zhengshu, xiaoshu);
convert_to_uppercase(out);
sprintf(hide_print,"20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x %0.1fV\r\nhide:%s\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second,sp.adc_vlot,out);
hidedata_proc(hide_print);
if(sp.limit_data < sp.adc_vlot)sprintf(out, "%s*",out);
printf("%s\r\n", out);
```

2.6 数据存储管理单元

2.6.1 子任务 1：采样数据存储

存储路径：TF 卡 /sample/。

文件名规则：“sampleData {datetime}.txt”（14 位时间戳，如 20250101003010）。

存储规则：每 10 条数据新建文件。

示例代码：

```
//sample执行函数
void sample_proc(char *str)
{
    FRESULT result;
    static char sample_filename[64];
    if(sp.sample_log_num == 0){
        sprintf(sample_filename, "sample/sampleData20%0.2x%0.2x%0.2x%0.2x%0.2x%0.2x.txt",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second);
        result = f_open(&sp.sample_file, sample_filename, FA_CREATE_ALWAYS | FA_WRITE);
        f_close(&sp.sample_file);
        result = f_open(&sp.sample_file, sample_filename, FA_CREATE_ALWAYS | FA_WRITE);
    }
    else if(sp.sample_log_num == 10)
    {
        f_close(&sp.sample_file);
        sp.sample_log_num = 0;
        return;
    }
    sp.sample_log_num++;
    UINT bytes_written;
    result = f_write(&sp.sample_file, str, strlen(str), &bytes_written);
    if(result == FR_OK && bytes_written == strlen(str)) {
        f_sync(&sp.sample_file);
    }
}
```

2.6.2 子任务 2：超阈值数据存储

存储路径：TF 卡 /overLimit/。文件名与存储规则同采样数据，需包含超限标记。

```
//overlimit执行函数
void overlimit_proc(char *str)
{
    FRESULT result;
    static char overlimit_filename[64];
    if(sp.overlimit_log_num == 0){
        sprintf(overlimit_filename, "overLimit/overLimit20%0.2x%0.2x%0.2x%0.2x%0.2x%0.2x.txt",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second);
        result = f_open(&sp.overlimit_file, overlimit_filename, FA_CREATE_ALWAYS | FA_WRITE);
        f_close(&sp.overlimit_file);
        result = f_open(&sp.overlimit_file, overlimit_filename, FA_CREATE_ALWAYS | FA_WRITE);
    }
    else if(sp.overlimit_log_num == 10)
    {
        f_close(&sp.overlimit_file);
        sp.overlimit_log_num = 0;
        return;
    }
    sp.overlimit_log_num++;
    UINT bytes_written;
    result = f_write(&sp.overlimit_file, str, strlen(str), &bytes_written);
    if(result == FR_OK && bytes_written == strlen(str)) {
        f_sync(&sp.overlimit_file);
    }
}
```

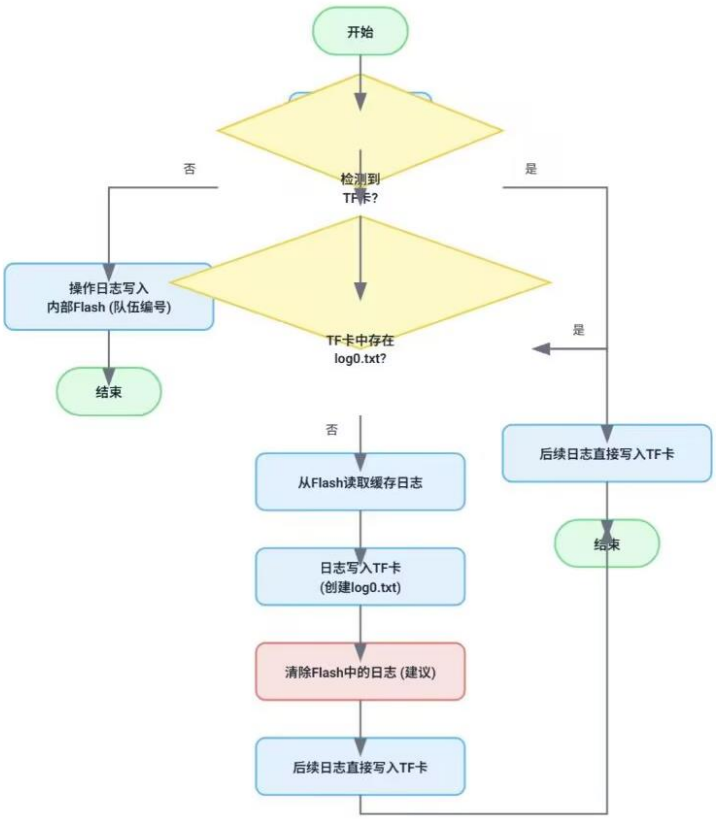
2.6.3 子任务 3：操作日志存储

存储路径：TF 卡 /log/。

文件名规则：“log {id}.txt”（id 从 0 开始，按上电次数递增）。

持久化要求：MCU 记录上电次数，清空 TF 卡后仍按历史次数递增。

日志操作流程：



示例代码：

```
//日志写入函数
void log_write(const char* write_data)
{
    char log_entry[256];
    if(!sp.log_file_val) return;
    // 格式: 2025-01-01 10:00:01 system init
    sprintf(log_entry,"20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x %s\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second,write_data);

    UINT bytes_written;
    FRESULT result = f_write(&sp.log_file, log_entry, strlen(log_entry), &bytes_written);
    if(result == FR_OK && bytes_written == strlen(log_entry)) {
        f_sync(&sp.log_file);
    } else {
//        my_printf(DEBUG_USART, "Log write error: %d, bytes: %d/%d\r\n",
//                result, bytes_written, strlen(log_entry));
    }
}
```

2. 6. 4 子任务 4：加密数据存储

存储路径：TF 卡 /hideData/。文件名规则同采样数据。

特殊规则：启用加密时，sample 文件夹不存储数据。超阈值数据仍需存储至 overLimit 文件夹。同时保存加密与未加密数据用于校验。

示例代码：

```c
//hidedata_proc执行函数
void hidedata_proc(char *str)
{
    FRESULT result;
    static char hidedata_filename[64];
    if(sp.hidedata_log_num == 0){
        sprintf(hidedata_filename, "hideData/hideData20%0.2x%0.2x%0.2x%0.2x%0.2x%0.2x.txt",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second);
        result = f_open(&sp.hidedata_file, hidedata_filename, FA_CREATE_ALWAYS | FA_WRITE);
        f_close(&sp.hidedata_file);
        result = f_open(&sp.hidedata_file, hidedata_filename, FA_CREATE_ALWAYS | FA_WRITE);
    }
    else if(sp.hidedata_log_num == 10)
    {
        f_close(&sp.hidedata_file);
        sp.hidedata_log_num = 0;
        return;
    }
    sp.hidedata_log_num++;
    UINT bytes_written;
    result = f_write(&sp.hidedata_file, str, strlen(str), &bytes_written);
    if(result == FR_OK && bytes_written == strlen(str)) {
        f_sync(&sp.hidedata_file);
    }
}
```

3 综合系统设计

3.1 系统整体架构设计

分层架构模型（从底层到应用层）：

3.1.1 硬件层

核心组件：MCU 控制器、ADC 采样模块、flash 存储芯片、TF 卡接口、RTC 时钟模块、OLED 显示屏、LED 指示灯、按键输入电路。

硬件依赖：需确保各模块电气连接稳定，如 TF 卡接口符合 SPI 通信协议，ADC 采样精度满足电压采集需求。

3.1.2 驱动层

功能职责：封装硬件操作 API，提供统一接口给逻辑层。

模块列表：flash 驱动（ID 读取 / 数据写入）、TF 卡文件系统驱动（目录创建 / 文件读写）、RTC 驱动（时间设置 / 读取）、ADC 驱动（电压采样）、OLED 驱动（显示控制）、LED / 按键驱动（状态控制 / 输入检测）。

驱动层关键 API：

模块	API	功能描述
ADC	ADC_GetVoltage()	返回换算后电压值（带滤波）

TF 卡	TF_WriteFile()	原子操作（先写临时文件再重命名）
RTC	RTC_SetTime()	支持 ISO8601 格式解析

3.1.3 逻辑层

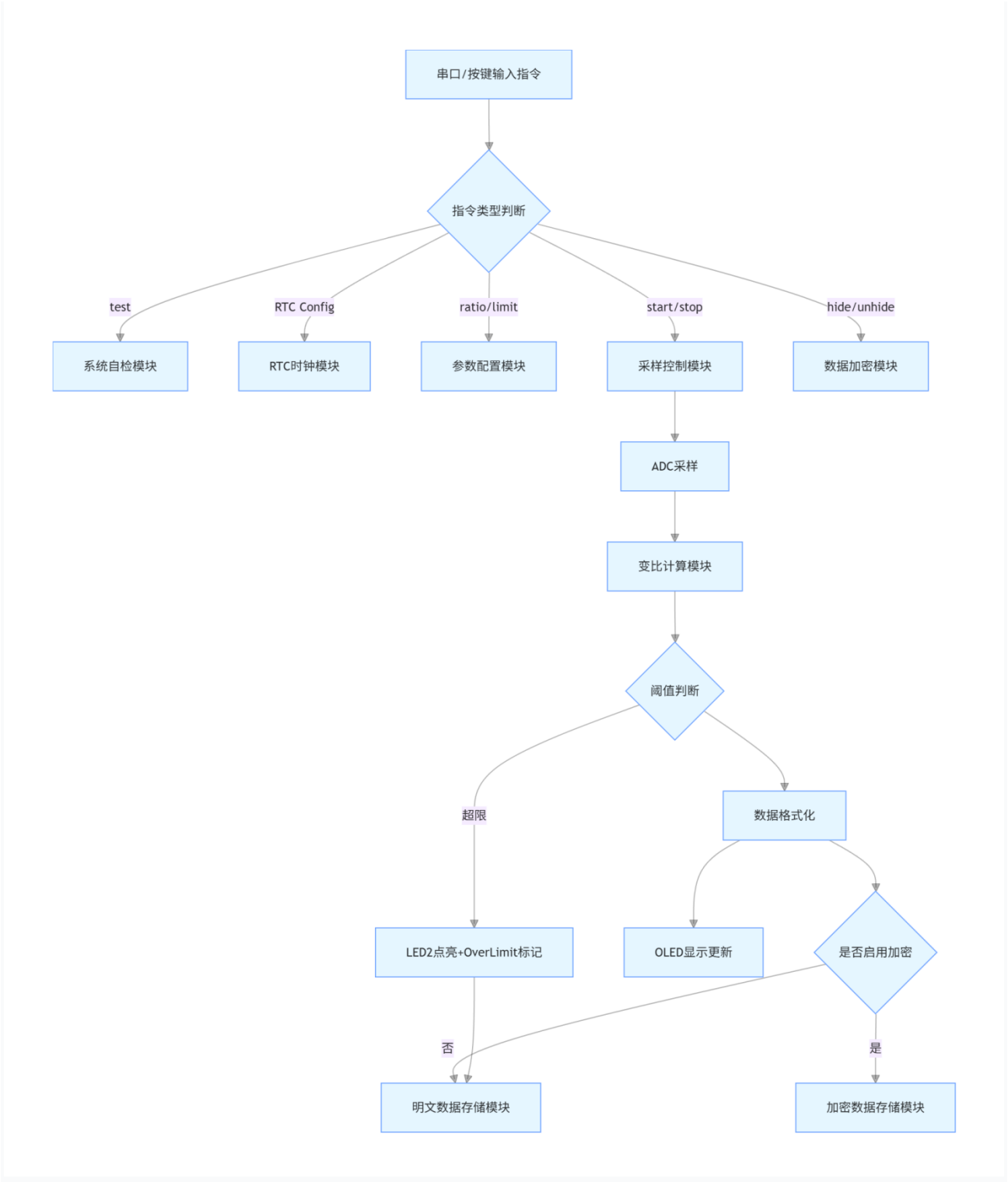
核心模块：系统自检模块、参数配置模块、采样控制模块、数据处理模块、存储管理模块

3.1.4 应用层

交互接口：串口指令解析（如 "test"/"start"）、人机界面显示（OLED/LED 状态）

业务流程：按赛题要求组织功能调用顺序。

3.2 整体逻辑流程示意图



3.3 串口指令一览

功能	序号	指令下发（字符）	串口数据返回（字符）	oled显示	动作	要求	存储
系统功能	1	test					
	2	RTC Config					
	3	RTC now					
配置管理	4	conf					
	5	ratio 比例（数值）			验证输入有效性； 更新变比值至Flash。		
	6	limit 阈值（数值）			验证输入有效性； 更新阈值至Flash。		
	7	config save			存储到flash		
	8	config read					
采样控制	9	start		第一行显示时间（只显示时分秒，格式hh:mm:ss），第二行显示电压值（小 数点后保留两位，格式xx.xx V）	启动周期采样模式； LED1指示灯闪烁（1s周期）； 按照采样周期通过串口输出采样数据	采样值超过limit设置的限制时，需要点亮LED2； 串口中打印增加OverLimit字样和具体的阈值要求	在sample文件夹中存储文件，每个文件10条数据。 超阈值数据存储在overLimit文件夹中，每个文件10条数据。
	10	stop		第一行显示 “system idle”，第二行为空	停止采样； LED1常灭		
数据处理	11	hide			按照加密数据通过串口输出数据		启用加密存储时，数据存储在hideData文件夹中，果超阈值触发，则（2）仍要按原有格式存储。
							每次上电后新建一个文件，直至断电前，所有的操作日志都记录在该文件中

3.4 数据流向与处理链路

采集→处理→存储全链路延迟分析：ADC 采样（10μs）→ 软件滤波（中值滤波，50μs）→ 超限判断（20μs）→ TF 卡写入（平均 5ms）。总延迟：<10ms（满足实时性要求）。采集 - 处理 - 存储全流程。

3.4.1 **数据采集环节**：

ADC 实时采样电压值→变比模块按 `Ratio` 参数换算工程值（如采样值 ×10.5）。采样周期由 KEY2/KEY3/KEY4 动态调整（5s/10s/15s），断电后通过 flash 持久化。

```
void adc_proc()
{
    adc_flag_clear(ADC0,ADC_FLAG_EOC);                  //  清除结束标志
    while(SET != adc_flag_get(ADC0,ADC_FLAG_EOC)){}      //  获取转换结束标志

    int adc_input = ADC_RDATA(ADC0);                    // 读取ADC数据
    sp.adc_vlot = adc_input*sp.ratio_data*3.3f/4095;                    // 把数字量转化为工程量
}
```

3.4.2 **数据处理环节**：

根据当前的控制要求，选择合适的执行内容（输出，超限，加密）

```
void print_adc_5s()
{
    //start后，打印时间和adc值
    if(sp.led_flash && (sp.sample_cycle == 5))
    {
        if(sp.encrypt_flag == 0){
            if(sp.limit_data < sp.adc_vlot){
                sprintf(time_print,"20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x %0.0fV limit %0.0fV\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second,sp.adc_vlot,sp.lim
                overlimit_proc(time_print);
                sprintf(time_print,"20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x ch0=%0.1fV OverLimit(%0.2f) !\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second,sp.adc_v
                printf("%s",time_print);
            }
            else{
                sprintf(time_print,"20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x %0.1fV\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second,sp.adc_vlot);
                sample_proc(time_print);
                sprintf(time_print,"20%0.2x-%0.2x-%0.2x %0.2x-%0.2x-%0.2x ch0=%0.1fV\r\n",sp.year,sp.month,sp.day,sp.hour,sp.minute,sp.second,sp.adc_vlot);
                printf("%s",time_print);
            }
        }
        else hide_to_unix();
    }
}
```

3.4.3 数据存储环节：

明文数据：按类型存入 sample/overLimit 文件夹，每 10 条新建文件

加密数据：存入 hideData 文件夹，同时保存未加密副本用于校验

操作日志：按上电次数生成 log {id}.txt，id 由 MCU 计数器维护

3.5 系统功能调度与执行

执行调度器介绍

执行调度器是一种用于管理和安排任务执行的机制，它能够根据任务的执行周期和时间，在合适的时机调用相应的任务函数。本项目中的所有功能函数都是在调度器中运行的。

3.5.1 数据结构

```
typedef struct{
    void (*task_func)(void);
    uint32_t rate_ms;
    uint32_t last_run;
} task_t;
```

3.5.2 任务列表

```
static task_t scheduler[]={
    {led_proc,50,0},
    {key_proc,50,0},
    {oled_proc,100,0},
    {adc_proc,100,0},
    {rtc_show_time,1000,0},
    {uart_proc,200,0},
    {led_flash,500,0},
    {print_adc_5s,5000,0},
    {print_adc_10s,10000,0},
```

```
    {print_adc_15s,15000,0}
};
```

任务数组，包含了多个任务的信息，每个任务由任务函数、执行周期和初始上次执行时间组成。

3.5.3 初始化函数

```
void scheduler_init()
{
    tast_num = sizeof(scheduler)/ sizeof(task_t);
}
```

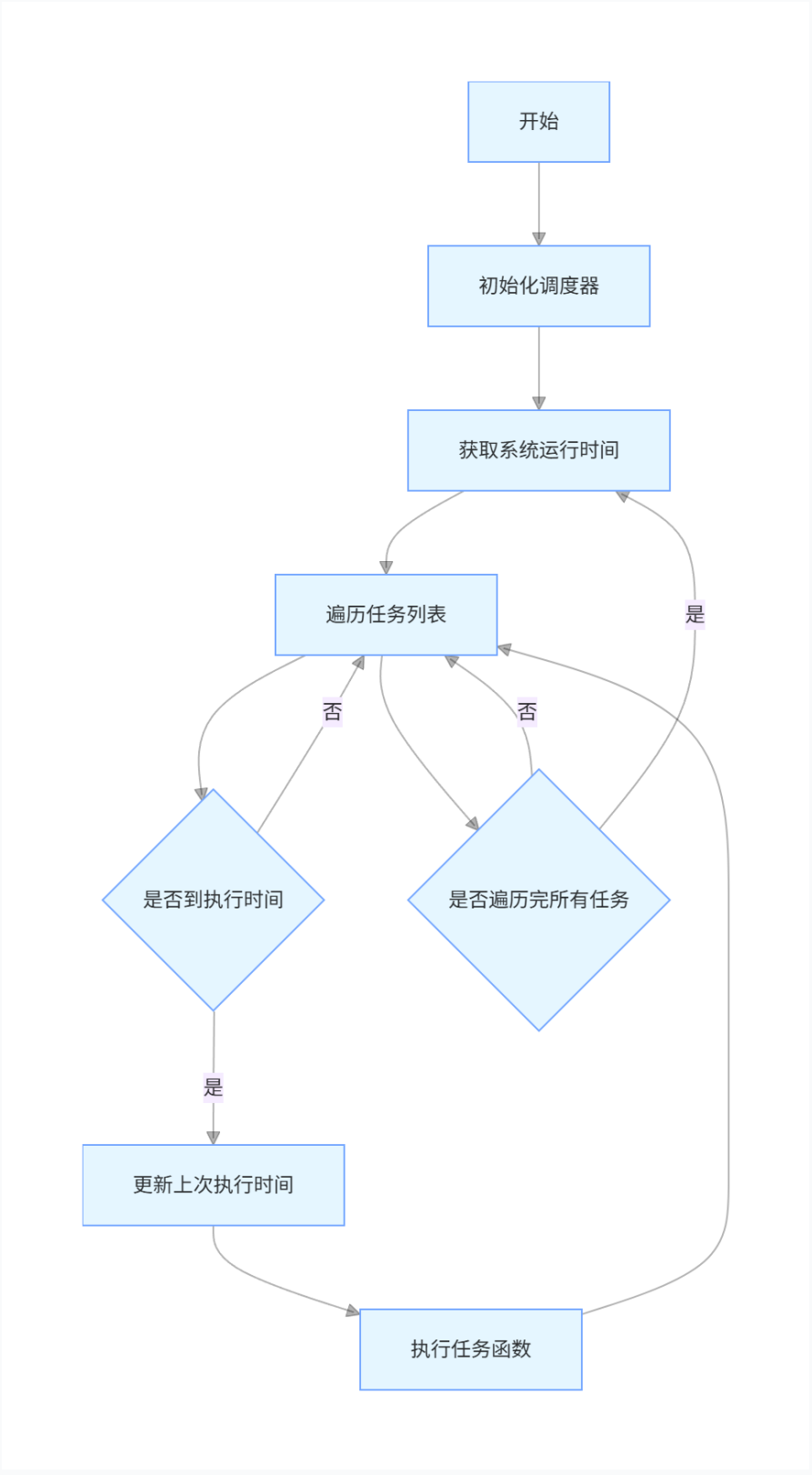
该函数用于初始化调度器，计算任务数组中任务的数量，并将其存储在 tast_num 变量中。

3.5.4 调度运行函数

```
void scheduler_run()
{
    for(uint16_t i=0;i<tast_num;i++)
    {
        uint32_t now_time=GetSysRunTime();
        if(now_time>=scheduler[i].lats_run + scheduler[i].rate_ms){
            scheduler[i].lats_run=now_time;
            scheduler[i].task_func();
        }
    }
}
```

用于循环遍历任务数组，检查每个任务是否到了执行时间。

3.5.5 程序流程一览


开始
初始化调度器
获取系统运行时间
遍历任务列表
是
否
否
是否到执行时间
是否遍历完所有任务
是
更新上次执行时间
执行任务函数

4 工程系统优化

4.1 数据一致性优化

4.2.1 数据校验机制

CRC 校验：在 flash 存储和 TF 卡数据存储中加入 CRC16 校验，确保数据读写的正确性。

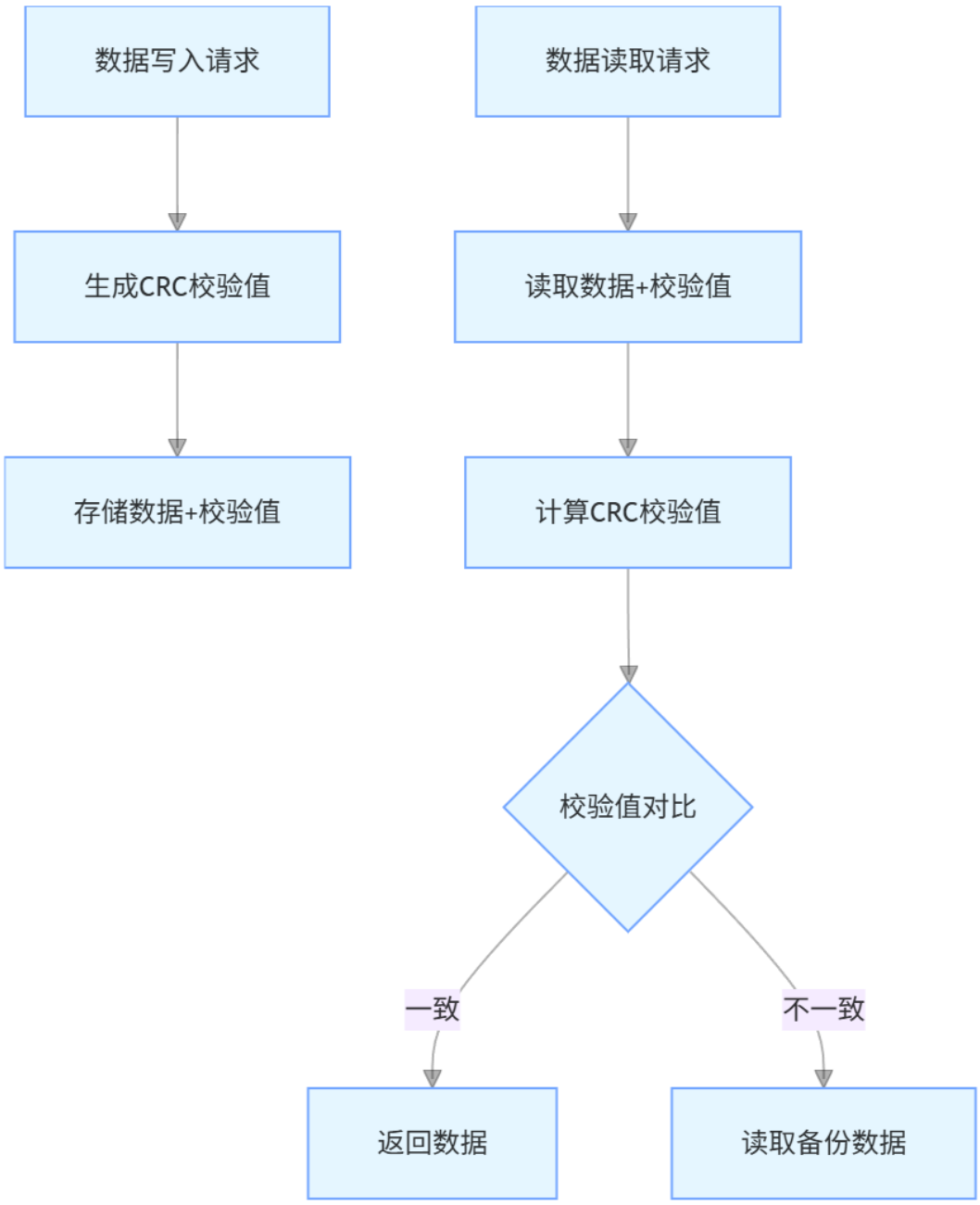
双备份存储：关键参数（如变比、阈值）在 flash 中存储两份，读取时对比校验，不一致时自动恢复。

4.2.2 存储一致性设计

原子操作：TF 卡文件写入时采用原子操作，避免断电导致的文件损坏，每 10 条数据写入后同步文件系统。

日志记录：所有数据修改操作记录到日志文件，便于故障时数据恢复和追溯。

4.2 数据校验流程图


数据写入请求
生成CRC校验值
存储数据+校验值
数据读取请求
读取数据+校验值
计算CRC校验值
校验值对比
一致
不一致
返回数据
读取备份数据

5 系统功能调试

5.1 调试环境搭建

5.1.1 硬件连接

串口连接：使用 USB 转串口线连接设备 UART 接口，波特率设置为 115200bps。

电源供应：使用 5V 稳压电源，确保电压波动不超过 ±5%。

5.1.2 软件工具

串口调试助手：用于发送指令和查看串口输出，支持十六进制和文本模式。

逻辑分析仪：连接 ADC 采样线，实时监测采样波形和频率。

万用表：测量实际输入电压，与系统显示值对比校准。

5.2 模块调试流程

5.2.1 系统自检调试

步骤 1：发送自检指令

串口输入 "test"，观察串口输出是否包含 flash ID、TF 卡状态和 RTC 时间。

预期结果

正常：显示 "flash.....ok"、"TF card.....ok" 及正确时间。

异常：若 TF 卡未插入，显示 "can not find TF card"。

调试工具

串口调试助手，查看输出日志。

5.2.2 RTC 功能调试

步骤 1：设置时间

串口输入 "RTC Config 2025-06-19 15:30:00"，观察返回 "RTC Config success"。

步骤 2：查询时间

输入 "RTC now"，检查返回时间是否与设置一致。

预期结果

时间设置后，查询结果与输入一致，断电重启后时间保持。

5.2.3 采样功能调试

步骤 1：启动采样

输入 "start"，观察 LED1 是否闪烁，串口是否按周期输出采样数据。

步骤 2：调整周期

按下 KEY3（10s 周期），查看串口输出周期是否变为 10s。

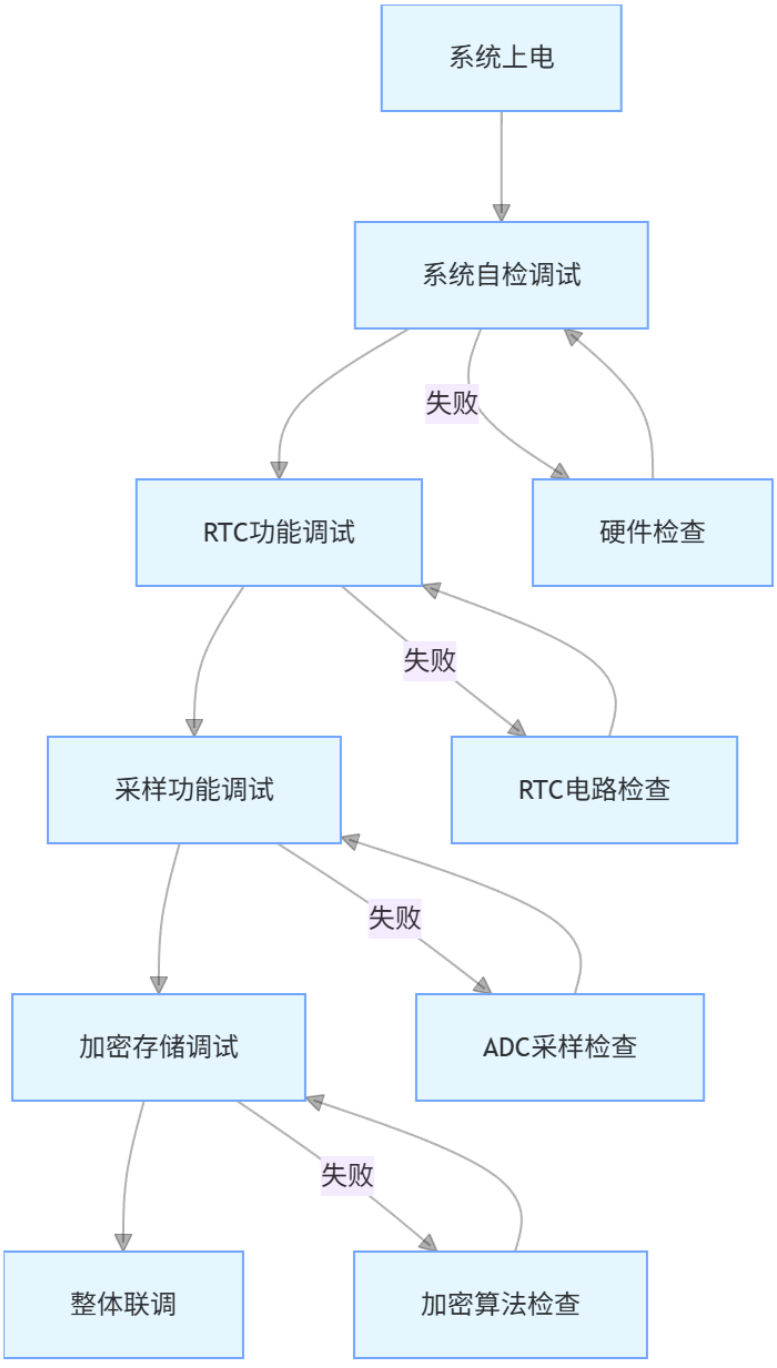
步骤 3：超限测试

输入"limit 5.0"，调整输入电压超过 5V，检查 LED2 是否点亮，串口是否显示"OverLimit"。

采样功能调试用例

测试场景	输入	预期输出
正常采样	电压=3.3V	"2025-06-19 10:00:00 ch0=3.30V"
超限报警	电压=12.6V, 阈值=10.0	"ch0=12.60V OverLimit(10.00)!"
周期切换	按下 KEY3	"sample cycle adjust:10s"

5.3 调试流程示意图


系统上电
系统自检调试
失败
RTC功能调试
硬件检查
失败
采样功能调试
RTC电路检查
失败
加密存储调试
ADC采样检查
失败
整体联调
加密算法检查

5.4 整体联调

（1）参数配置

设置变比为 10.0，阈值为 30.0，保存到 flash。

（2）启动采样

输入 "start"，调整输入电压，观察采样数据是否按变比缩放，超限提示是否正确。

（3）加密测试

输入 "hide"，检查加密数据格式是否正确，解码后是否恢复原始值。

（4）存储验证

断电后插入 TF 卡，查看 sample、overLimit、hideData 文件夹是否生成对应文件。

加密存储验证流程：

发送 hide 启用加密

输入电压 12.5V → 串口输出 6774C4F5000C8000

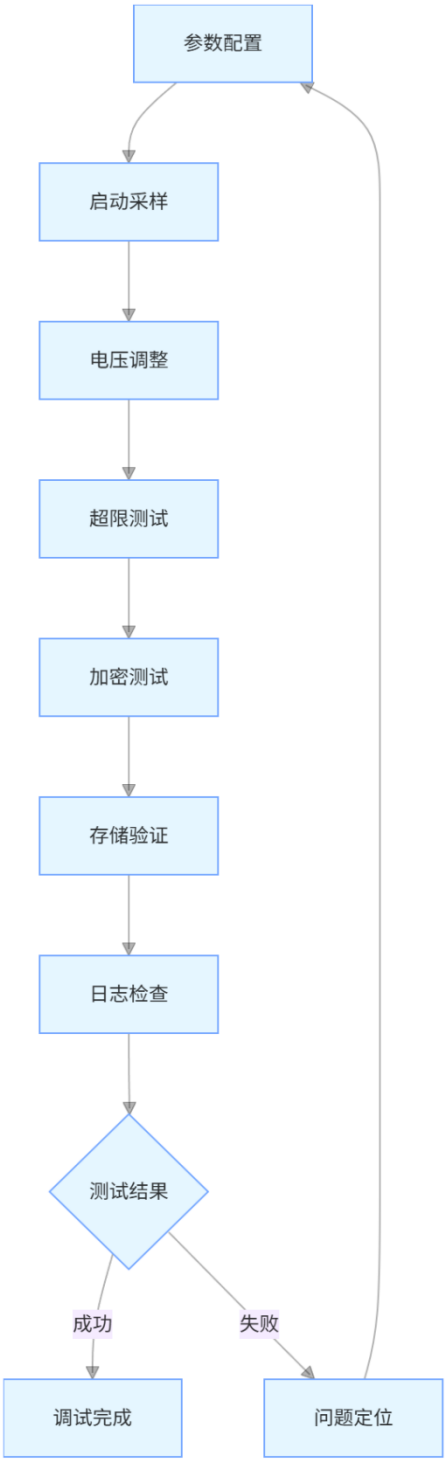
检查 TF 卡：

/hideData/hideData20250619120000.txt 存在加密数据

/sample/ 为空（符合设计）

发送 unhide → 恢复显示 2025-06-19 12:00:00 ch0=12.50V

5.5 联调模式示意图



附：产品使用手册

1. 系统简介

1.1 系统功能

本系统是一款工业嵌入式电压数据采集系统，具备以下功能：

（1）电压数据采集、处理、显示与存储

（2）系统自检与实时时钟管理

（3）变比和阈值参数配置

（4）数据加密与多类型存储

1.2 系统组成

（1）主控模块：MCU 控制器

（2）采样模块：ADC 采样电路

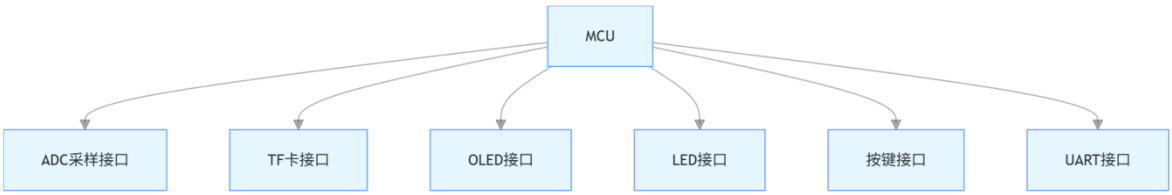
（3）存储模块：flash 和 TF 卡

（4）显示模块：OLED 显示屏

（5）指示模块：LED 指示灯

（6）输入模块：按键和串口

1.3 系统接口示意图



2. 安装与连接

2.1 硬件安装

（1）将 TF 卡插入 TF 卡插槽，注意方向正确。

（2）安装 CR2032 电池（可选，用于保持 RTC 时间）。

（3）连接电源（5V DC）。

2.2 串口连接

（1）使用 USB 转串口线连接设备 UART 接口。

（2）在电脑上安装串口驱动，设置波特率为 115200bps，数据位 8，停止位 1，无校验。

3. 操作指南

3.1 系统启动

（1）接通电源，系统自动初始化，串口输出：

```
====system init====
Device_ID:2025-CIMC-队伍编号
====system ready====
```

（2）OLED 显示 "system idle"，表示系统就绪。

3.2 时间设置

（1）串口输入："RTC Config 2025-06-19 15:30:00"

（2）系统返回："CIMC'西门子杯'中国智能制造挑战赛 RTC Config success Time:2025-06-1915:30:00"

（3）输入 "RTC now" 查询当前时间。

3.3 采样操作

（1）启动采样：串口输入 "start"，LED1 闪烁，OLED 显示时间和电压值。

（2）停止采样：输入 "stop"，LED1 熄灭，OLED 显示 "system idle"。

（3）按键控制：按下 KEY1 切换采样状态，KEY2/KEY3/KEY4 调整采样周期（5s/10s/15s）。

3.4 参数设置

（1）变比设置：

① 输入 "ratio"，系统显示当前变比

② 输入新值（0-100），如 "10.5"

③ 系统返回 "ratio modified success Ratio=10.5"

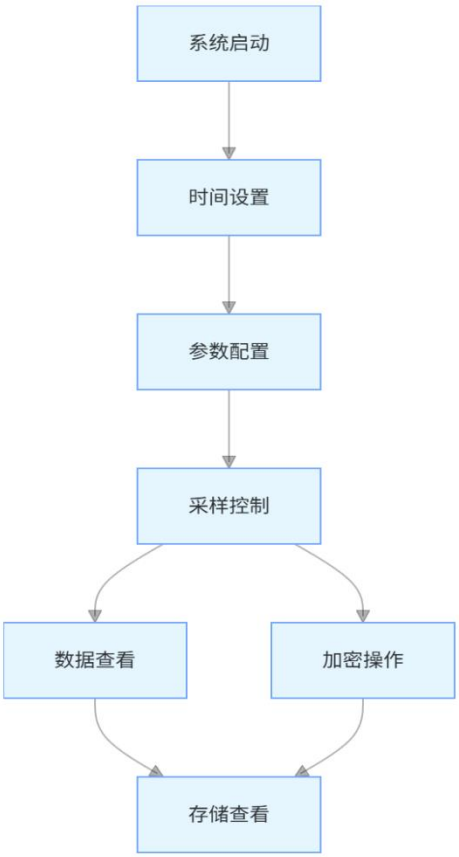
（2）阈值设置：

① 输入 "limit"，系统显示当前阈值

② 输入新值（0-500），如 "50.0"

③ 系统返回 "limit modified success limit=50.0"

3.5 操作流程图



4. 数据查看与管理

4.1 串口数据查看

采样数据：启动采样后，串口按周期输出时间和电压值。

加密数据：输入 "hide"，查看加密后的十六进制数据。

4.2 TF 卡数据管理

（1）采样数据：存储在 sample 文件夹，文件名格式 "sampleData {datetime}.txt"。

（2）超阈值数据：存储在 overLimit 文件夹，文件名格式 "overLimit {datetime}.txt"。

（3）日志数据：存储在 log 文件夹，文件名 "log {id}.txt"，id 按上电次数递增。

（4）加密数据：存储在 hideData 文件夹，文件名 "hideData {datetime}.txt"。

4.3 数据存储目录结构示意图

plaintext

```
TF 卡根目录
|
├── sample
```

```
│   ├── sampleData20250619153000.txt
│   └── sampleData20250619153500.txt
│
├── overLimit
│   ├── overLimit20250619153030.txt
│   └── overLimit20250619153530.txt
│
├── log
│   ├── log0.txt
│   └── log1.txt
│
└── hideData
    ├── hideData20250619153000.txt
    └── hideData20250619153500.txt
```

5. 故障排除

5.1 常见问题处理

（1）串口无输出

① 检查串口线连接和波特率设置

② 重启系统，确保初始化完成

（2）TF 卡检测失败

① 检查 TF 卡是否插入正确

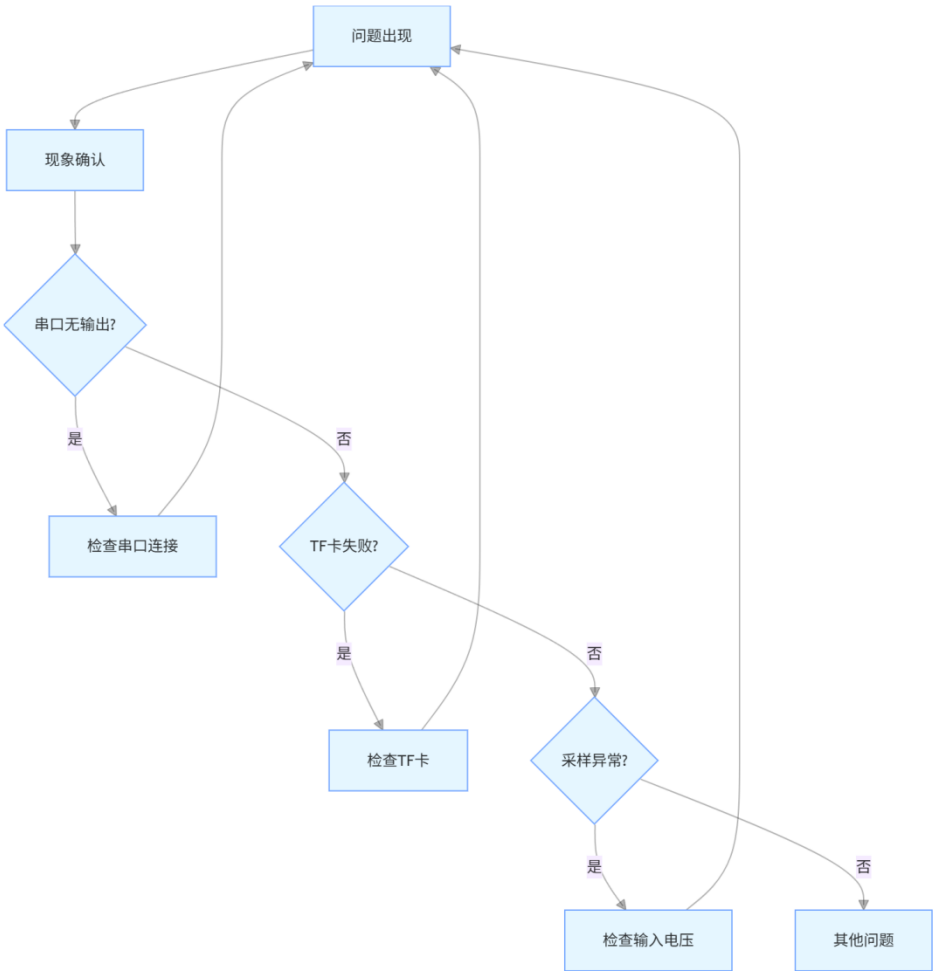
② 尝试更换 TF 卡，确保格式为 FAT32

（3）采样数据异常

① 检查输入电压是否在范围内

② 验证变比设置是否正确

5.2 故障排查流程图


问题出现
现象确认
串口无输出?
是
否
检查串口连接
TF卡失败?
是
否
检查TF卡
采样异常?
是
否
检查输入电压
其他问题