生产型气泵驱动板以及上位机开发

项目背景

泵项目是为实现江苏蚂蚁动力科技有限公司的超声波泵压电陶瓷稳定驱动，使设备工作在最佳状态，同时开发上位机程序可以实时显示下位机的各种状态信息以及异常控制，并且为了满足大规模生产的需要，利用上下位通信同时上位机将数据处理为想要的 Excel 数据文件，进而快速完成各设备间的参数设置。

承担角色

我在该 5 人以上的团队中，一开始接触时负责利用 C#进行上位机的开发，通过阅读源代码了解架构、理解需求、确定实施方案、调试运行完成上位机的开发，调试发现下位机可能存在问题，便与下位机开发同事进行交流解决问题完成当前的开发需求。由于前程序员采用的通信协议并无帧头帧尾且判断数据帧结束方式是依靠一段时间内不在接收到数据，这边会导致通信效率难以在此基础上提高，且重复性代码较多 main 函数臃肿阅读性以及可维护性较差，为此便交由我进行下位机程序的优化。针对上面的问题逐个解决，并讨论规范了代码的命名以及文件管理风格，使通信效率大大提高，数据丢包率从 30%降低到 1%以下。完成下位机代码优化后之后这个项目则等待客服提出新的需求进行改进。

项目成果

1. 实现以上位机和 PLC 为主站，n 个不同型号的驱动板为下位机从机的多主多从的稳定通信，且规范了通信帧，利用状态机高校通信并且提高功能可扩展性。
2. 上位机通过 Modbus TCP 协议获取 PLC 指定位置的数据，即判断是否获取到 RS485 总线控制权，如获取到控制权上位机将按照自定义的协议发送给下位机获取相关数据，而当上位机释放总线后也会利用 Modbus TCP 协议告诉 PLC 使能总线控制权。
3. 上位机可以选择是否将接收到的数据处理为 Excel 表格并且将峰值附近指定长度的数据绘制成折线图便于展示分析，并且通过多线程编程提高处理速度避免操作界面卡顿影响卡顿。
4. 由于 RS485 是单双工通信而波特率限制在 19200 且要求下位机在扫频阶段时间很快但数据量却比较大，想要通过上位机精准控制扫频频率的变化与数据传输无法达到要求，为此我提出消息队列机制，同时优化上下位机通信机制，使其可以通过应答连续获取高速进行数据传输，而不需要上位机每次都发一个完整的数据帧。这套方案使上下位机的通信效率大幅提升，丢包率直线下降。
5. 将下位机程序划分为 3 层：驱动层、应用层、用户层，降低代码耦合性提高代码易读性，为之后的用户型气泵驱动板的移植降低大量难度。（图 4，图 5）
6. 解决期间产生的各种 BUG，比如因为字节对齐愿意导致成员无法正确访问（通过空成员和__attribute__((packed))强制单字节对齐等）、解决由于在上下位机通信阶段导致长时间的数据计算产生的异常值等等问题。
7. 优化以前程序的运行速度：以空间换时间的思想实现将实时计算 CRC 验证的方法替换

为利用查表计算的方法，避免耗时的重复计算；通过滑动窗口方法替换原本数组整体移动更新方法实现数据图的移动显示更新提高显示速度；通过定时器事件控制下位机在指定时间后进行休眠避免长时间运行导致温度过高；通过多线程以及同步信号的方式将分离出 Excel 处理线程、串口接收线程，数据发送线程、图形绘制线程，进一步提高通信效率；数据帧中包含唯一标识，用于上位机进行丢包检测，如有丢包则即使进行补包发送；

8. 更新上位机的操作界面，使其能更美观的显示所需要的数据，并且实现工具栏的效果将参数配置放置其中。

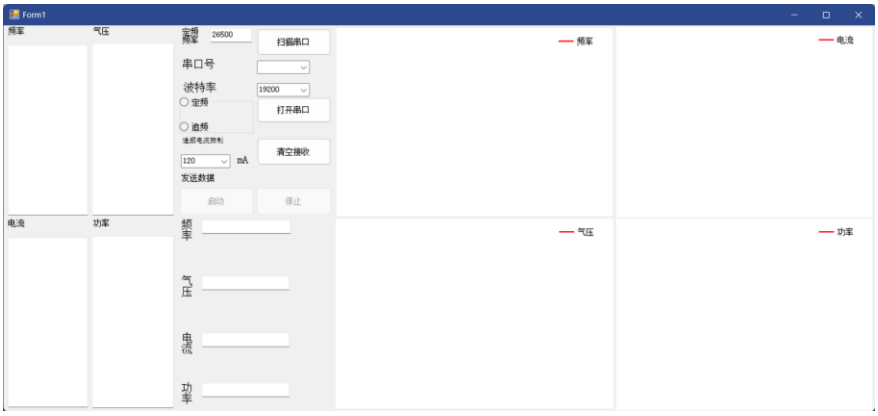


图 1 刚接手时应用程序界面

图 2 第一次更新后的界面

图 3 目前最新版操作界面

项目价值

1. 工作效率显著提升：系统的自动化控制和实时监测功能，减少了人工操作工作量，提高了气泵管理的效率。操作人员可通过可视化界面随时掌握气泵运行状态，及时发现并解决问题。
2. 数据管理能力增强：数据的可视化展示和文件保存功能，为气泵运行数据分析提供了有力支持。通过对历史数据的分析，可优化气泵运行参数，提高能源利用效率。
3. 系统稳定性提高：通过解决技术难题，系统在长时间运行过程中保持稳定，减少了故障发生的概率，降低了因设备故障导致的生产中断风险。

项目总结

在该项目中，通过从需求理解到架构设计、问题解决的全流程参与，我不仅完成了上下位机通信与驱动板开发的核心目标，在了解用户需求并提出解决方案上也有所进步，更在技术深度和工程思维上获得了显著成长，具体体现在以下方面：

1. 对通信协议的设计有了更深的理解，能独立实现通信协议模块的相关上下位机程序。在面对不熟悉得通信协议程序编写时也能与其他人进行高效交流，实现上手开发。
2. 拥有较强学习新知识的能力，接收适应能力快。在该工程中面对不熟悉的 MCU 与 C#程序开发，通过看手册文档以及咨询了解，快速掌握核心并进行开发。
3. 对程序设计有了更深的理解，实现分层编写降低耦合度。将一个复杂的功能才分为相互独立的子功能，对子功能实现开发调试完毕后，在组合起来实现完整的功能。
4. 大型代码阅读理解能力显著提高。由于该项目的代码量十分庞大且在优化前比较臃肿，同时是多人开发慢慢留下的代码，导致前期理解比较复杂，但好在中期掌握一定信息后在 AI 的辅助下大幅度的加快了阅读能力，并且将其整理留下笔记。
5. 面对问题可以提出多种解决方案的能力。在客户提出需要在极短时间内完成数据采集而发送数据不缺失但数据传输时间可以适当延长的需求时，提出消息队列的管理机制、尝试提高通信波特率等方案；面对需要多主多从的通信场景下，提出总线控制管理机制，避免通信干扰。

该项目不仅是技术实现的过程，更是工程思维与问题解决能力的锻炼。从最初面对 30% 的丢包率、臃肿的代码、界面卡顿等问题，到通过架构设计、机制优化、细节调试逐一攻克，我深刻体会到“以终为始”的设计理念 —— 每个技术方案都应服务于实际需求（如稳定通信、高效处理、易维护）。

通过上位机开发个项目，我在多线程编程、串口通信、数据可视化和 Excel 文件操作等方面积累了丰富经验，提升了自己的技术水平。同时，在面对各种技术难题时，锻炼了问题解决能力和创新思维。此外，负责项目的全流程开发，让我对项目管理有了更深入的理解，学会了如何合理安排时间、制定计划和控制进度。

上位机设计简介

在系统初始化方面，程序定义了大量全局变量用于记录系统状态、参数及线程同步，如定义串口通信的波特率、端口名等串口参数，还有运行时间、停止时间等时间相关变量，以及多线程同步使用的 AutoResetEvent 对象。界面布局使用 TableLayoutPanel 进行设计，包含多个文本框、按钮、下拉框等控件，用于用户输入参数和执行操作，例如扫描串口、打开串口、设置运行模式等。

串口通信是系统的重要组成部分。程序会自动搜索可用串口并将其添加到下拉列表中，用户可选择串口并设置参数后打开串口。接收数据时，通过状态机来解析数据帧，确保数据的完整性和准确性，同时添加超时处理机制，当接收超时时，会重置状态机。在发送数据时，根据 SendFLAGGroup 标志位的不同，程序会发送请求帧或进行连续数据获取，并且在发送请求帧后会等待数据包成功接收信号。

气泵运行控制功能上，系统支持定时运行和休息模式。用户在文本框中输入运行时间和停止时间后，程序会根据定时器控制气泵的启停。在运行过程中，系统会实时监测气泵的运行状态，如频率、气压、电流、功率等，并根据预设条件进行控制。

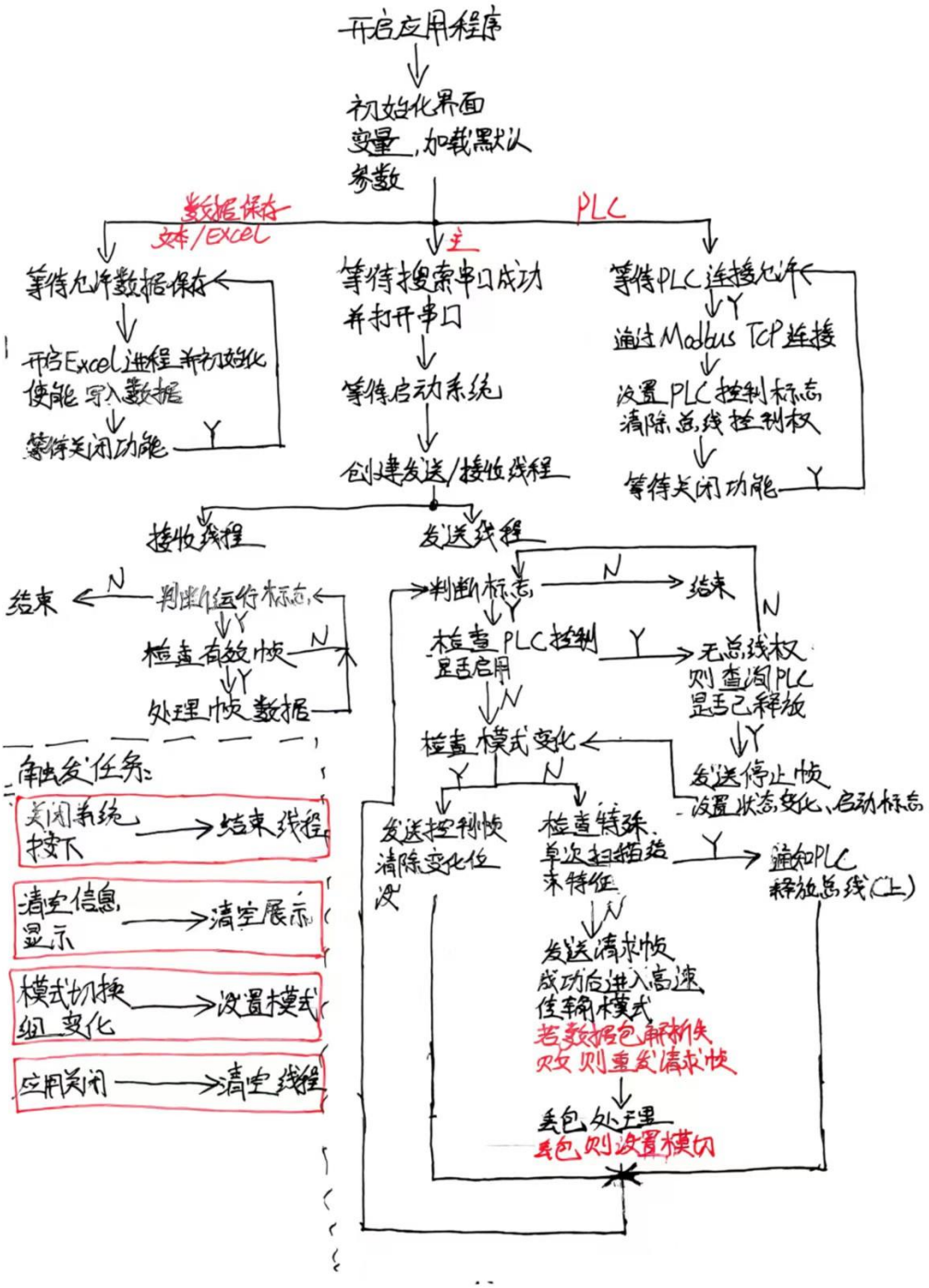
数据可视化方面，程序使用 Chart 控件将采集到的气泵运行数据以折线图的形式展示出来，包括频率、气压、电流和功率四个图表。为了保证图表数据的实时性，会定时更新图表。

数据保存功能允许用户将采集到的数据保存到 Excel 文件中。在保存数据时，程序会计算最大电流值，并根据最大电流对应的频率值确定一个范围，将该范围内的数据背景色改变，方便用户分析数据。同时，程序还提供了释放 Excel COM 对象的辅助函数，避免内存泄漏。

PLC 通信部分，程序可以与 PLC 建立连接，用户输入 PLC 的 IP 地址和端口号即可实现通信。在系统运行过程中，根据条件判断是否需要向 PLC 发送控制信号，例如在单次扫描完成且 PLC 控制功能启用时，会向 PLC 发送上位机完成信号。

上位机框架

一、节点流程图：



附录

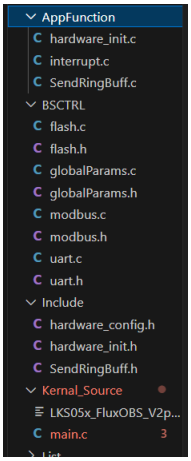


图 4 旧程序文件目录

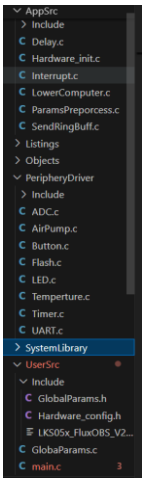


图 5 新程序文件目录

```c
void uartRecvCh(uint8_t ch)
{
    uint8_t Res;

    gUsartRecvTime = 0;

    Res = ch;   //读取接收到的数据
    if((USART_RX_STA&0x80)==0x00)//接收未完成
    {
        if(USART_RX_STA>=USART_REC_LEN)
        {
            USART_RX_STA=0;//接收数据错误,重新开始接收
        }
        else
        {
            if(USART_RX_STA == 0x00)
            {
                if(Res != gAddr && Res != 0xff)//不等于本地地址或广播地址
                {
                    return;
                }
            }
            USART_RX_BUF[USART_RX_STA&0x7F]=Res ;
            USART_RX_STA++;
        }
    }
}
```

```c
void UTIMER1_IRQHandler(void)
{
    static uint8_t i_fdd=0;
    /* 时基500us */
    if (UTIMER_IF & TIMER1_IF_ZERO)
    {
        UTIMER_IF = TIMER1_IF_ZERO;
        i_fdd++;

        if (gUTimerCount > 0 ) ···
        if(gUTimerCountForRS > 0) ···

        if( ( USART_RX_STA != 0x00 ) && ( (USART_RX_STA&0x80) == 0x00 ) )
        {
            gUsartRecvTime++;
            if(gUsartRecvTime > 3)
            {
                USART_RX_STA |= 0x80;
                gUsartRecvTime = 0;
            }
        }
        else

//      if(gCalibrateTime > 0) ···

        gUTimer1s++;//2000=1s
        switch(gUTimer1s) ···
    }
}
```

图 6 旧程序通信帧结束判断

```c
uint8_t  UartReceiveStateMachineHandler(u8 Data){
    static u16 CrcEstimate = 0;// CRC计算结果
    static u8 ReciveNum = 0;// 当前接收个数
    static u8 ReciveBuffIndex = 0;// 当前接收缓冲区索引
    static ReciveFrame* CurReciveDataFrame = &UartFrameData[0];
    // 接收状态机处理
    switch (State){
        case ReciveHead:{// 接收数据头 ···
        case ReciveAddress:{// 接收地址 ···
        case ReciveMark:{// 接收数据标识 ···
        case ReciveFuncIndex:{ ···
        case ReciveDataLength:{// 接收数据长度 ···
        case ReciveData:{// 接收数据 ···
        case ReciveCRCH:{// 接收CRC高字节 ···
        case ReciveCRCL:{// 接收CRC低字节 ···
        case ReciveEnd:{// 接收结束
            if (Data == END_BYTE){
                State = ReciveHead;// 接收结束,进入接收数据头状态
                gUartOutTimes = 0;// 暂停超时计数
                // 判断是否切换缓存区
                if (((ReciveComplish >> ((1 - ReciveBuffIndex) * 2)) & 0x03) != 2){···
                ReciveEndError = false;
                return ReciveSucc;// 返回接收成功状态
            } else if (Data == HEAD_BYTE){
                State = ReciveAddress;// 如果是0xAA,则表明现在是另一个数据帧的开始,继续接收地址
            } else{
                State = ReciveHead;// 并未接收到结束符,进入接收错误状态
                gUartOutTimes = 0;// 暂停超时计数
            }
            break;
        }
        default: State = ReciveHead;break;//多余状态，正常情况下不可能出现
    }
    // 非接收数据头状态,都会重置超时时间
    if (State != ReciveHead){
        gUartOutTimes = OutTime;
    }
    return State;// 返回当前状态
```

```c
/**
 * @brief  串口接收中断函数
 * @param  None
 * @retval None
 */
void UART1_IRQHandler(void){
    static uint8_t UART1_DATA = 0;
    if (UART1_IF & BIT1){//接收完成事件
        UART1_IF = BIT1; //清除接收完成标志位
        UART1_DATA = UART_ReadData(UART1);
        UartReciveCh(&UART1_DATA);
    }
}
```

图 7 新程序通信协议结束判断