实 训 课 程 报 告

课程名称：接口与通信实训

专业：电子信息工程 班级：20 电信 1 班

姓名：王恩缘 学号：2031020090

指导教师：贾婷、史秀男

成绩：

评语:

完成日期：2022 年 06 月 09 日

摘 要

本次实训我完成了基于 STM32 单片机的无线遥控智能时钟设计，其中的电路包括 5 个部分，分别为：STM32 单片机主控模块、按键模块、nRF24L01 无线通信模块、LCD1602 显示模块、DHT11 温湿度传感器模块。

STM32 为本项目的主控 MCU，在 MDK5 开发环境中利用 ST 为 F1 系列的 32 单片机编写的库函数，对单片机的各个管脚、内部时钟、中断以及各种外设进行配置与设定。按键用于向 STM32 的 GPIO 端口输入信号，便于调试以及人工操控。nRF24L01 可以发送与接收无线信号，实现与上位机的半双工通信。LCD1602 是方便易用，成本低廉的液晶显示模块，因此项目采用此显示器件，通过八个数据端口进行并行通信来接收字符信号，以及 3 个控制端口实现字符显示的功能。最后，DHT11 通过 DATA 管脚向 32 输出温湿度传感器检测到的数据。

以 Python 为接口的上位机与单片机进行通信，在不同运行平台、开发环境、程序语言的情形下，用串口通信进行数据的交换。串口方面使用 VSPD 虚拟串口，利用 MDK5 自带调试功能将串口数据发送至 COM 口。数据接收采用 pyserial 库读取数据，并使用 tkinter 库进行图形化 UI 界面的绘制，通过按钮与窗口等交互方式，完成收发数据、改变上位机的配置信息等功能。

关键词：单片机；串口通信；上位机

Abstract

In this training, I have accomplished the design of wireless remote control intelligent clock based on STM32 microcontroller. The circuit includes 5 parts, Including: STM32 microcontroller main control module, button module, nRF24L01 wireless communication module, LCD1602 display module, DHT11 temperature and humidity sensor module.

STM32 is the main control MCU of this project. In the MDK5 development environment, the library functions written by ST for 32 single-chip microcomputers of the F1 series are used to configure and set the various pins, internal clocks, interrupts and various peripherals of the single-chip microcomputer. The button is used to input signals to the GPIO port of STM32, which is convenient for debugging and manual control. nRF24L01 can send and receive wireless signals to achieve half-duplex communication with the host computer. LCD1602 is an easy-to-use, low-cost liquid crystal display module, so the project uses this display device to receive character signals through eight data ports for parallel communication, and three control ports to realize the function of character display. Finally, DHT11 outputs the data detected by the temperature and humidity sensor to 32 through the DATA pin.

The host computer with Python as the interface communicates with the single-chip microcomputer. In the case of different operating platforms, development environments, and programming languages, serial communication is used to exchange data. The serial port uses the VSPD virtual serial port, and uses the MDK5's own debugging function to send the serial port data to the COM port. For data reception, the pyserial library is used to read the data, and the tkinter library is used to draw the graphical UI interface. The functions such as sending and receiving data and changing the configuration information of the host computer are completed through interaction methods such as buttons and windows.

Key words: MCU; serial communication; upper computer

目 录

1　总体方案设计

1.1 需求分析

我们现在处在一个逐步向智能化迈进的时代，人工智能、物联网等概念已深入千家万户的认知之中。疫情影响下，近两年我国社会生产力收到相应程度的影响，22 年上半年在辽宁省疫情更是反反复复不得安宁。这对于本就在时代冲洗下日暮西山的传统的钟表制造行业雪上加霜，企业生存问题得到极大程度的挑战。因为智能手机的普及，加之疫情期间对于健康码和行程卡的硬性要求，我国居民智能手机市场进一步向中老年人群普及。因此现在已经很少会有人想用传统意义的“闹钟”这种形态的产品。

这便是我们本项目的研究成果的实际意义，我们将闹钟接入无线模块，配备了 STM32F103ZET6 这颗具有一定智慧的“大脑”，不但可以迎合现代人对于物联网产品的消费热情，ZET6 强大的扩展性也可以完成其中的智能化设计以及温湿度显示等丰富的功能。本项目的集成化程度较高，设备的轻巧、便携也更适合做出各式各样小巧精致的工业外形设计，满足不同消费人群的审美需求。

1.2 系统的构成

结构框图如图。DHT11 采集环境温湿度信息并发送给 STM32，32 自带的通用定时器满足时钟的计时的精度与条件。整合了时间信息与温湿度信息后，STM32 一边将信息传给 LCD1602 显示，一边将数据传给 nRF24L01 进行无线信号的发送。

nRF24L01 将 32 发送的信息转换为无线数据信号，以串口通信的方式发送给 PC。PC 端则使用 Python 利用 pyserial 库与 tkinter 库等编写上位机程序，与 nRF24L01 进行通信。在上位机界面中可以通过 UI 交互显示收发信息，并通过上位机发送的信息，可对时钟进行设置及控制。

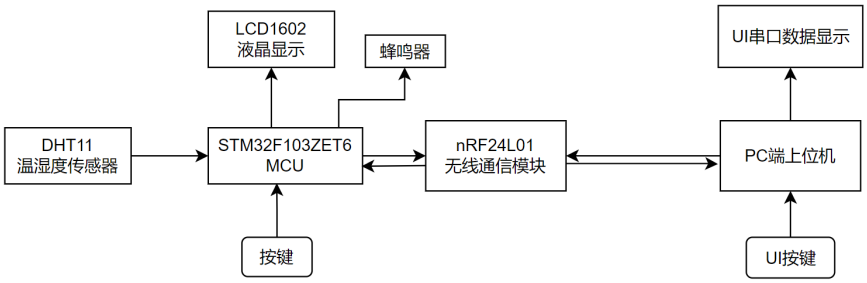


图 1.1 整体结构框图

各模块之间的关系，及其与 STM32F103ZET6 的管脚的具体配置的整体思维导图如图 1.2 所示：

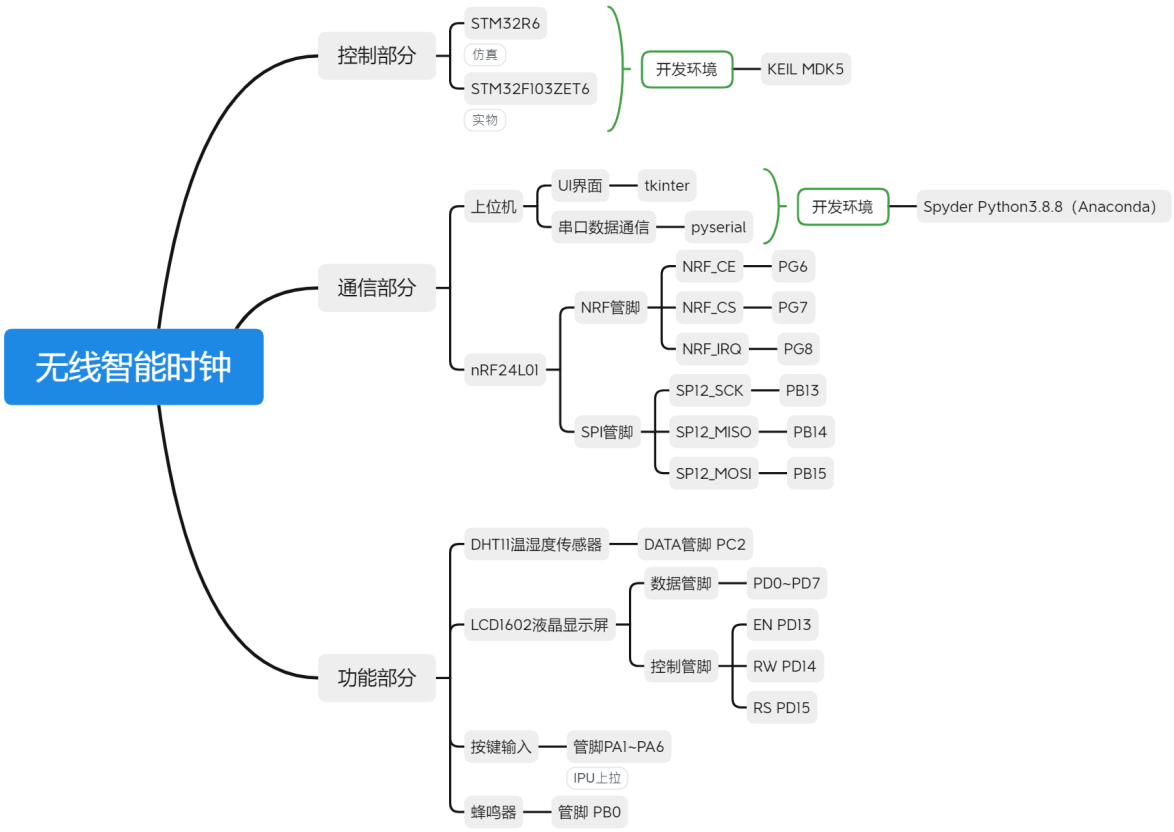
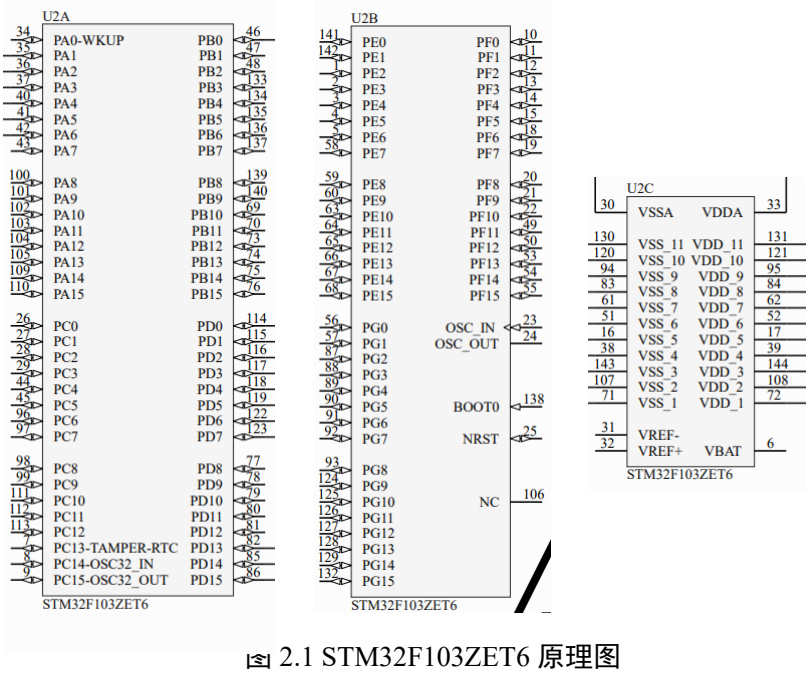


图 1.2 全局思维导图

2 智能时钟装置电路设计

2.1 智能时钟装置主控电路设计

本项目采用了 ST 官方给出的 Altium Designer 原理图及其封装方案，ZET6 被分隔成为三个部分，我在这里将其命名，分别为 GPIO 部分（UA），GPIO 及其晶振部分（UB），以及供电及其复位部分（UC）。三个部分如图 2.1 所示。



2.1 STM32F103ZET6 原理图

下面我将一一阐述该项目中，32 单片机最小系统的构成及其设计。

2.1.1 时钟电路

晶振是由石英晶体组成的，石英晶体之所以能当为振荡器使用，是基于它的压电效应：在晶片的两个极上加一电场，会使晶体产生机械变形；在石英晶片上加上交变电压，晶体就会产生机械振动，同时机械变形振动又会产生交变电场，虽然这种交变电场的电压极其微弱，但其振动频率是十分稳定的。当外加交变电压的频率与晶片的固有频率（由晶片的尺寸和形状决定）相等时，机械振动的幅度将急剧增加，这种现象称为“压电谐振”。

晶振电路为主控芯片提供系统时钟，所有的外设工作，CPU 工作都要基于该时钟，类似于整个系统的“心跳节拍”。本项目的时钟电路设计如图 2.2 所示。

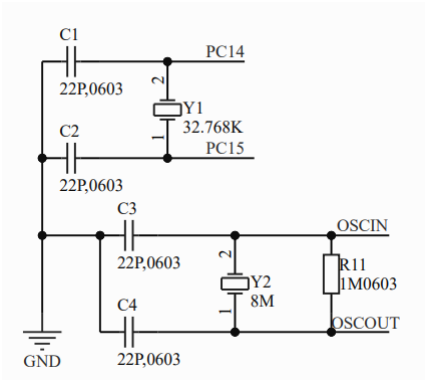


图 2.2 时钟电路设计

晶振分为无源和有源，但是本质上都是皮尔斯震荡电路（反相放大器+电阻+电容+晶体+电源），只不过对于单片机而言，单片机内部集成了反相放大器和电阻以及电源，外接晶体和电容就可以了，这里的晶体就称之为无源晶振。

而有源晶振是将皮尔斯振荡器作成一个整体，直接加电源即可工作，当然，价格也会比无源的贵一些。

2.1.2 供电电路

仔细观察一下可以发现，ST 将 VDD、VSS、VSSA、VREF+、VDDA、VSSA 等单独提出来做一个部件，但其实封装还是共用的。类似于大一下半学期学习数电模电在 Multisim 仿真中，与门、非门等元器件名称后面加 ABC，代表集成电路的其中一组。

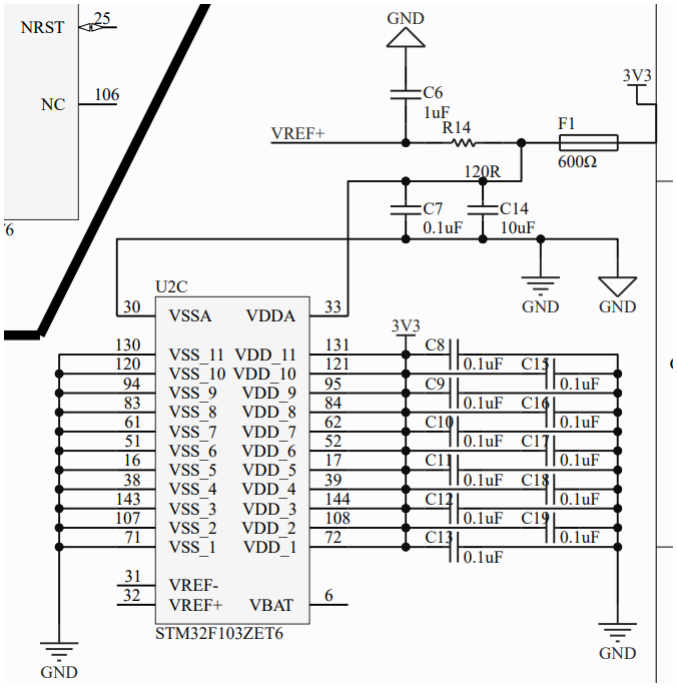


图 2.3 供电电路设计

VCC、VDD、VSS、VREF+、VSSA、VDDA 管脚的意义与作用如下：

1）VCC：接入电路的电压。（V 代表电压，C 字母代表电路）。

2）VDD: 器件内部的工作电压，一般直接接 3.3V。（D 为 device 器件的首字母，表示器件）。

3）VSS: 通常指电路公共接地端电压，所以可以看到将其接到 GND 上去。（S 为 series 表示公共连接）。

4）VDDA： 通对比 VDD 多了个 A，代表与模拟电路有关，所以它是对所有的模拟电路部分供电。需要注意的是 VDDA 和 VDD 之间的电压差不能超过 300mV，VDD 与 VDDA 应该同时上电或调电。

5）VSSA：后面跟了个 A，毫无疑问，这是 VDDA 的地，所以 VDDA 和 VSSA 是单片机内部模拟电路的正负（电源）。

6）VREF+、VREF-：看到 ref，代表这是个参考电压，结合 STM32 的引脚具备 A/D 转换的功能这点，就能清楚它是用来提高 ADC 精度的。在引脚数目上了 100 后，为了保证更好的低电压输入精度，连接一个单独的参考电压输入到 VREF+中，VREF+输入电压范围为 2.0V 到 VDDA，VREF-可用时，必须绑定到 VSSA。在引脚数为 64 时，将没有这个 VREF（VREF+、VREF-在内部被接到 VDDA、VSSA 上）。

2.1.3 复位电路

主控芯片是低电平复位(引脚 NRST)，硬件按键复位属于系统复位之一(另外还有软件复位，看门狗计数终止复位等)。其中的电容 C5 的目的是按键硬件消抖，防止在按键刚刚接触/松开时的电平抖动引发误动作(按键闭合/松开的接触过程大约有 10ms 的抖动，这对于主控芯片 I/O 控制来说已经是很长的时间，足以执行多次复位动作。由于电容电压不会突变，所以采用电容滤波，防止抖动复位误动作)。

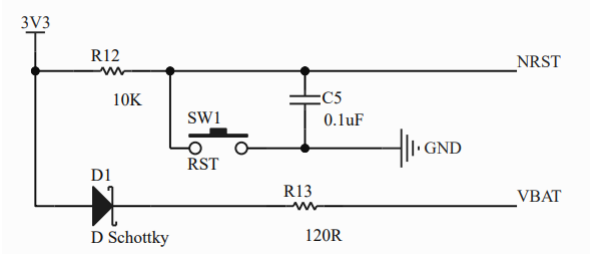


图 2.4 复位电路设计

VBAT 的作用: 使用电池或其他电源连接到 VBAT 脚上时，当 VDD 断电时，可以

保存备份寄存器的内容和维持 RTC 的功能。

2.2 外设电路设计

2.2.1 按键电路

本项目中按键采用上拉输入，上拉就是把电位拉高。上拉就是将不确定的信号通过一个电阻嵌位在高电平，电阻同时起限流作用。按键电路的设计如图 2.5 所示。

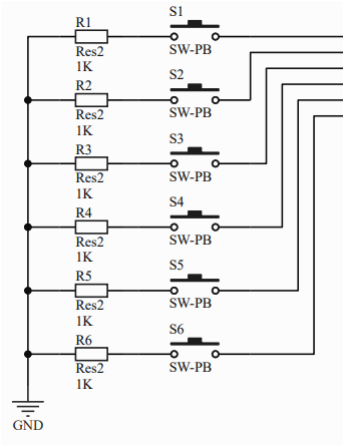


图 2.5 按键电路设计

本项目中按键 1~6 分别接到 GPIOA 所对应的 PA1~PA6 管脚，在初始化配置时注意将输入模式改为 IPU。

2.2.2 蜂鸣器电路

蜂鸣器是电路设计中常用的器件，广泛用于工业控制报警、机房监控、门禁控制、计算机等电子产品作预警发声器件，驱动电路也非常简单。蜂鸣器电路设计图如图 2.6。

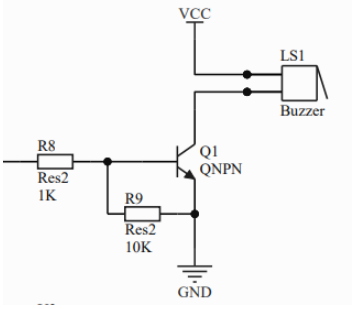


图 2.6 蜂鸣器电路设计

连接至管脚 PB0，将该管脚配置为推挽输出 Mode_Out_PP。设定为高电平时工作。

2.2.3 液晶屏显示电路

LCD1602 是很多单片机爱好者较早接触的字符型液晶显示器，它的主控芯片是 HD44780 或者其它兼容芯片。

表 2.1 液晶屏 LCD1602 引脚示意表

引脚号	符号	引脚说明
0	GND	电源地
1	VCC	电源正极
2	VL	偏压信号
3	RS	命令/数据
4	RW	读/写
5	EN	使能
6	DB0	数据端口
7	DB1	数据端口
8	DB2	数据端口
9	DB3	数据端口
10	DB4	数据端口
11	DB5	数据端口
12	DB6	数据端口
13	DB7	数据端口
14	LED+	背光正极
15	LED-	背光负极

各个管脚的具体说明如下：

1）VSS 接电源地。

2）VDD 接+5V。

3）VO 是液晶显示的偏压信号，可接 10K 的 3296 精密电位器，或同样阻值的 RM065/RM063 蓝白可调电阻。

4）RS 是命令/数据选择引脚，接单片机的一个 I/O，当 RS 为低电平时，选择命令；当 RS 为高电平时，选择数据。

5）RW 是读/写选择引脚，接单片机的一个 I/O，当 RW 为低电平时，向 LCD1602 写入命令或数据；当 RW 为高电平时，从 LCD1602 读取状态或数据。如果不需要进行

读取操作，可以直接将其接 VSS。

6）E，执行命令的使能引脚，接单片机的一个 I/O。

7）D0—D7，并行数据输入/输出引脚，接入单片机的 GPIO 任意的 8 个 I/O 口。如果接 P0 口，P0 口应该接 4.7K—10K 的上拉电阻。如果是 4 线并行驱动，只须接 4 个 I/O 口。

8）A 背光正极，可接一个 10—47 欧的限流电阻到 VDD。本项目中接入可调电阻 R10 直接进行背光亮度的调节。

9）K 背光负极，接 GND。设计图见图 2.7 所示。

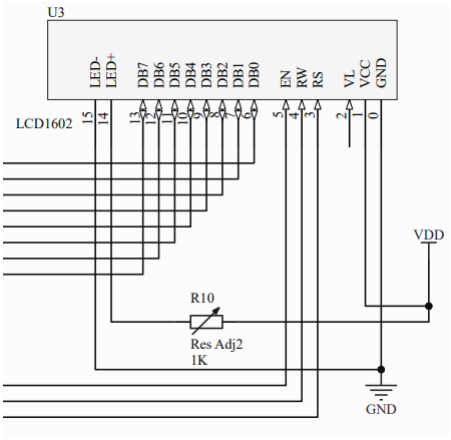


图 2.7 液晶屏 LCD1602 电路设计

2.2.4 通信接口电路

NRF24L01 是 NORDIC 公司最近生产的一款无线通信通信芯片，采用 FSK 调制，内部集成 NORDIC 自己的 EnhancedShortBurst 协议。可以实现点对点或是 1 对 6 的无线通信。无线通信速度可以达到 2M（bps）。NORDIC 公司提供通信模块的 GERBER 文件，可以直接加工生产。嵌入式工程师或是单片机爱好者只需要为单片机系统预留 5 个 GPIO，1 个中断输入引脚，就可以很容易实现无线通信的功能，非常适合用来为 MCU 系统构建无线通信功能。，原理图设计如图 2.8 所示。

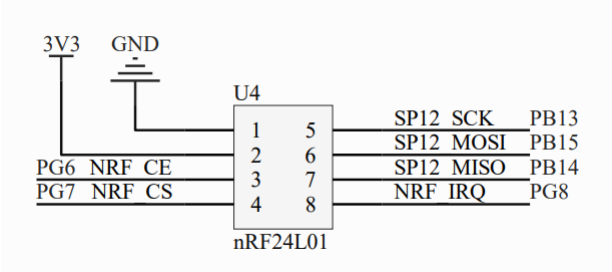


图 2.8 nTF24L01 电路设计

2.2.5 温湿度检测电路

DHT11 数据处理：

串行接口（单线双向）DATA 用于微处理器与 DHT11 之间的通讯和同步，采用单总线数据格式，一次通讯时间 4ms 左右，数据分小数部分和整数部分，具体格式在下面说明，当前小数部分用于以后扩展，现读出为零.操作流程如下：

一次完整的数据传输为 40bit，高位先出。

数据格式：8bit 湿度整数数据+8bit 湿度小数数据+8bi 温度整数数据+8bit 温度小数数据+8bit 校验和数据传送正确时校验和数据等于“8bit 湿度整数数据+8bit 湿度小数数据+8bit 温度整数数据+8bit”温度小数数据所得结果的末 8 位。

用户 MCU 发送一次开始信号后，DHT11 从低功耗模式转换到高速模式，等待主机开始信号结束后，DHT11 发送响应信号，送出 40bit 的数据，并触发一次信号采集，用户可选择读取部分数据模式，DHT11 接收到开始信号触发一次温湿度采集，如果没有接收到主机发送开始信号，DHT11 不会主动进行温湿度采集。采集数据后转换到低速模式。电路设计如图 2.9 所示。

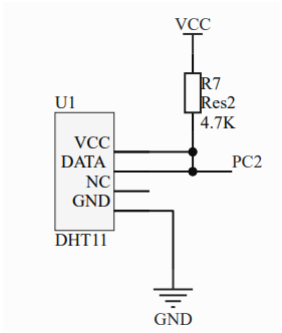


图 2.9 DHT11 电路设计

总线空闲状态为高电平，主机把总线拉低等待 DHT11 响应，主机把总线拉低必须大于 18 毫秒，保证 DHT11 能检测到起始信号。DHT11 接收到主机的开始信号后，等待主机开始信号结束，然后发送 80us 低电平响应信号。主机发送开始信号结束后,延时等待 20-40us 后，读取 DHT11 的响应信号，主机发送开始信号后，可以切换到输入模式,或者输出高电平均可，总线由上拉电阻拉高。

3 智能时钟程序设计

3.1 主程序设计

在“main.c”主程序中，首先进行全局变量的定义：

```
extern int Rcount,minute,hour;
int Stop_Rcount=20,Stop_minute=0,Stop_hour=0;
u8 LCD_Display_Stop[16];
u8 temprature[8];
```

其中，“extern int”为外部变量引用，在这里使用此处是调用了“timer.c”中定义的变量。由于中断服务函数在“timer.c”中，时分秒计数以及显示不在主程序中，因此使用外部引用的方式，使得主函数中对应的“Rcount”、“minute”、“hour”变量与时钟同步。

完成相关“.h”文件引用以及各种全局变量的定义、子函数的声明后，进入主函数部分。主函数的程序流程图如图 3.1 所示。

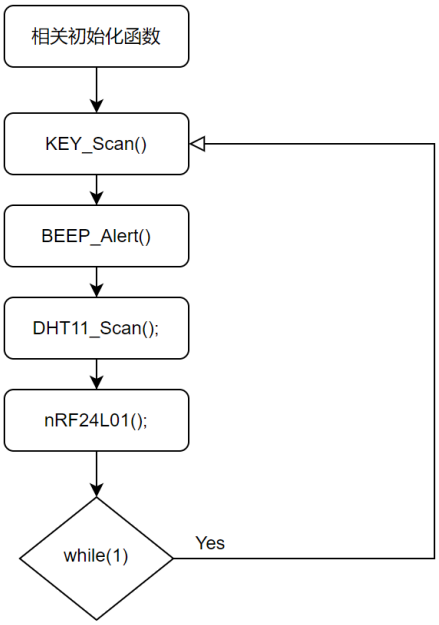


图 3.1 主函数程序流程图

接着，在 main()函数中，调用 TEACHING 文件夹中各个功能所对应的初始化函数，如下所示：

```
uart_init(115200);
Timerx_Init(1999,35999);      //计数一次 1s
```

```
BEEP_Init();
LCD1602_Init();
NRF24L01_Init();
Dht11_Init();
```

初始化完成后，在 while(1)死循环中反复运行以下子函数。

```
while(1){
    KEY_Scan();
    BEEP_Alert();
    DHT11_Scan();
    nRF24L01();   }
```

其中，KEY_Scan()函数为按键扫描函数，可以通过六个按键实现对定时时分秒进行设置；BEEP_Alert()函数中实现了音乐播放的功能；DHT11_Scan()函数实现了温湿度检测并显示在 LCD1602 上；nRF24L01()函数实现了与上位机的通信。

3.2 按键程序设计

在 KEY_Scan 函数中，由于六个按键的配置大同小异，这里只展示 KEY1 按键的程序设计。程序流程图见图

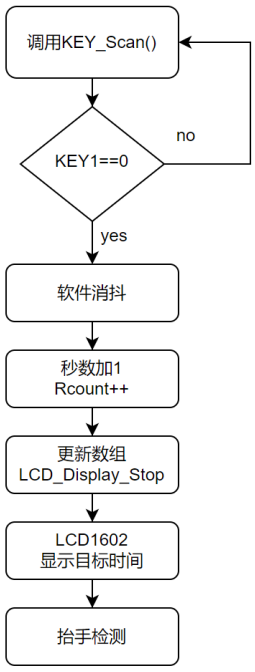


图 3.2 按键扫描程序流程图

本项目中的按键均以宏定义的方式调用，优点是变得更加直观的同时提高代码的复用率。宏定义内容如下：

```
#define KEY1 GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_1)
```

在上文提到的按键设计电路图（图 2.5）可以看出，本项目的按键并没有硬件消抖的设计。因此要进行软件消抖，因为当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键开关在闭合时不会马上就稳定的接通，在断开时也不会一下子彻底断开，而是在闭合和断开的瞬间伴随了一连串的抖动。程序代码设计如下所示：

```
if(KEY1==0){
    delay_ms(10);                    //软件消抖
    if(KEY1==0){
```

完成消抖操作后，在 if 中执行如下语句。Stop_Rcount++为自定停止时间再加一秒，同理，KEY2 为减一秒，KEY3 为加一分钟……以此类推。停止时间修改后，在液晶屏 LCD1602 的第一行显示目前设定的停止时间。

```
Stop_Rcount++;                //秒数+
sprintf((char*)LCD_Display_Stop,"STOP++%2d:%2d:%2d",Stop_hour,Stop_minute
,Stop_Rcount);
LCD1602_Show_Str(1, 0, (u8*)"                    ");
LCD1602_Show_Str(1, 0, LCD_Display_Stop);        //显示更改的时间
while(KEY1==0);                  //抬手检测
```

最后一行的抬手检测，是由于人手的操作存在随机因素，且按下的时间多为几十到几百毫秒才会抬手。为了避免一次按下，在这几十毫秒期间判断语句里的内容重复不断地进行，造成连按的效果，我在本次判断语句程序的最后加上这句抬手检测，当手没有抬起时，程序会停留在 while(KEY1==0)一句中等待抬手，再继续执行下一个语句。

3.3 液晶屏显示程序设计

事实上，在图 3.5 中，就已经出现了液晶屏显示的语句。

先用“stdio.h”里的 sprintf 函数将已发送变化的三个代表停止的时分秒变量“写入”到 LCD1602_Display_Stop 数组中，再通过 1602 的特性利用十六个空格字符将对应行的屏幕清空，再使用 LCD1602_Show_Str 函数将 LCD1602_Display_Stop 中的字符型变量，依照 ASCII 码对应的字符显示在屏幕上。其控制端口的管脚配置如图所示：

```
//1602 液晶指令/数据选择引脚
#define LCD_RS_Set()   GPIO_SetBits(GPIOD,GPIO_Pin_15)
#define LCD_RS_Clr()   GPIO_ResetBits(GPIOD, GPIO_Pin_15)
//1602 液晶读写引脚
#define LCD_RW_Set()  GPIO_SetBits(GPIOD, GPIO_Pin_14)
#define LCD_RW_Clr()  GPIO_ResetBits(GPIOD, GPIO_Pin_14)
//1602 液晶使能引脚
#define LCD_EN_Set()   GPIO_SetBits(GPIOD, GPIO_Pin_13)
#define LCD_EN_Clr()   GPIO_ResetBits(GPIOD, GPIO_Pin_13)
```

LCD1602 的基本操作分为四种：

读状态。输入 RS=0，RW=1，E=高脉冲。输出：D0—D7 为状态字。

读数据。输入 RS=1，RW=1，E=高脉冲。输出：D0—D7 为数据。

写命令。输入 RS=0，RW=0，E=高脉冲。输出：无。

写数据。输入 RS=1，RW=0，E=高脉冲。输出：无。

这四种操作加上八个数据端口的操作构成了 TEACHING 文件夹里的“lcd1602.c”中绝大部分的操作。

3.4 温湿度检测程序设计

DHT11 温湿度传感器模块对应的程序流程图设计如下：

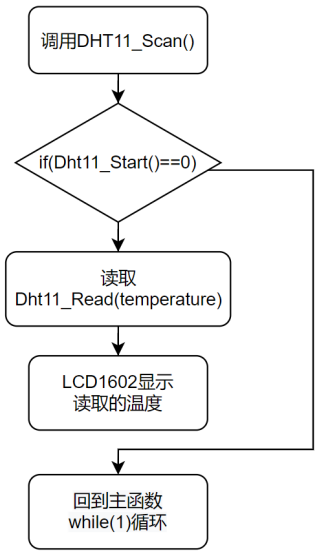


图 3.3 按键扫描程序流程图

代码设计相较于其它模块较为简单，只需判断 Dht11_Start()是否为 0，如果为 0 则调用 Dht11_Read 函数，输出温湿度至 temperature 数组变量。

```
void DHT11_Scan() {
    if(Dht11_Start()==0) {
        Dht11_Read(temperature);
        LCD1602_Show_Str(1, 0, (u8*)"                    ");
        LCD1602_Show_Str(1, 0, temperature);              //显示温度
    }
}
```

3.5 通信程序设计

首先，STM32 需要收集数据，并和 nRF24L01 交换数据；nRF24L01 需要与 PC 端的上位机交换数据。下面我将分别从 STM32 的角度——即下位机的设计，以及上位机的设计进行阐述。

3.5.1 下位机通信模块设计

从单片机控制的角度来看，我们只需要关注六个控制和数据信号，分别为 CSN、SCK、MISO、MOSI、IRQ、CE。

CSN：芯片的片选线，CSN 为低电平芯片工作。

SCK：芯片控制的时钟线（SPI 时钟）

MISO：芯片控制数据线（Master input slave output）

MOSI：芯片控制数据线（Master output slave input）

IRQ：中断信号。无线通信过程中 MCU 主要是通过 IRQ 与 NRF24L01 进行通信。

CE：芯片的模式控制线。在 CSN 为低的情况下，CE 协同 NRF24L01 的 CONFIG 寄存器共同决定 NRF24L01 的状态。

程序设计框图如图 3.3 所示：

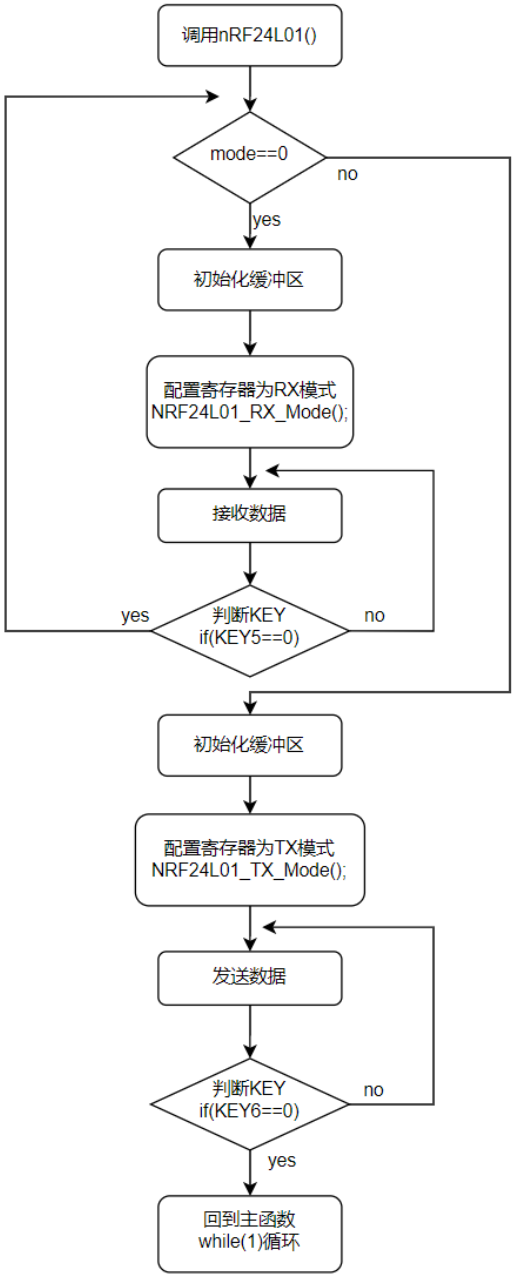


图 3.3 无线通信模块程序流程图

nRF24L01 通信方式为半双工串口通信。由于串口通信的需要，我在全局变量中定义了 tmp_buf 数组，作为缓冲区临时存放串口通信数据，共 64 位：

u8 tmp_buf[64];

在 nRF24L01()子函数中，具体代码设计如下：

接收模式：和无线串口模块通信的时候，无线数据缓冲区的第一字节是数据帧有效数据长度。

```
if(NRF24L01_RxPacket(tmp_buf)==0) {   //一旦接收到信息,则显示出来
    tmp_buf[32]=0;        //加入字符串结束符
    LCD1602_Show_Str(1, 0, (u8*)"                    ");
    LCD1602_Show_Str(1, 0, (u8*)"    Received DATA");
}else delay_us(100);
```

发送模式：将数据接收至缓冲区 tmp_buf 数组中：

```
if(NRF24L01_TxPacket(tmp_buf)==TX_OK)   {
    LCD1602_Show_Str(1, 0, (u8*)"                    ");
    LCD1602_Show_Str(1, 0, (u8*)"    Sended DATA:");
    for(t=0;t<29;t++){
        key++;
        if(key>('~'))key=' ';       //此处用来填充一下缓冲区 ASCII
        tmp_buf[t+1]=00000001; } mode++;
```

3.5.2 上位机设计

本次实训我选择使用 tkinter 设计 GUI，tkinter 界面简单，但也满足本项目的需求。串口通信用 pyserial 实现串口数据即时（自动）接收。而实现的关键在于需要有像单片机中的中断那样的响应机制。在 Python 中自然不可能直接接收到硬件中断，但我们可以通过为串口接收加一个轮询串口的线程。Python 的线程可以用 threading 库实现。

关于 pyserial：

要用 pyserial 建立串口通信，首先要先建一个 Serial 类对象，然后定义端口波特率等参数，然后就可以启动。

Pyserial 的初始化配置代码如下：

```
ser = serial.Serial()
ser.port = 'COM4'                #串口号 COM4
ser.baudrate = 115200            #波特率
ser.bytesize = 8                 #8 数据位
ser.stopbits = 1                 #1 停止位
```

完成初始化配置后，便可运行串口上位机：

```
ser.open() #  开启串口
while True:
```

```
ch = ser.read() #  只收一个 bytes
print(ch.decode(encoding='ascii'),end=' ')
```

在测试代码的时候，当然要有数据给它。本项目使用 nRF24L01 发送数据，通过 VSPD 虚拟串口连接至 pyserial 实现通信。

还要注意的是，pyserial 接收上来的数据是 bytes 类，不是标准的 str 类，所以接收后要用 bytes 的 decode 方法解码，用 ser.write()写入前也要用 str 的 encode 方法编码。

关于 tkinter：

tkinter 的界面设计我参考了梁勇的《Python 语言程序设计》中的教学与例程，在这次编程时也基本仿照书本例子编写。这里从我总代码中截出一点介绍几个关键点：

1、tkinter 控件中数据的变量

比如这里用一个输入框类(tk.Entry)接收设定波特率，默认 9600。这个变量 self.Baudrate)的类型是 tk.IntVar。Tkinter 提供了一种变量类，和普通变量对应，但又可以与 GUI 控件连接，动态变化：

```
labelBaudrate = tk.Label(frame_COMinf,text="Baudrate: ")
self.Baudrate = tk.IntVar(value = 9600)
ertryBaudrate = tk.Entry(frame_COMinf, textvariable = self.Baudrate)
```

我们要用这些变量类数据进行计算时，必须把他们转成普通变量，通常是用 get()函数，如 self.Baudrate.get()，比如在按键的响应函数中：

```
def processButtonSS(self):
    self.ser.port = self.COM.get()      #  获取数据
    self.ser.baudrate = self.Baudrate.get()
```

2、tkinter 的 grid 函数布局和 pack 函数布局

以用于展示接收信号的文本框为例，grid 函数布局更像是划分网格空间填充，pack 则是直接装。相比之下 grid 更常用：

```
frameRecv = tk.Frame(window) # frameRecv 的父节点是 window
frameRecv.grid(row = 2, column = 1) #  整个窗口用 grid 布局，接收文本框在第二列
labelOutText = tk.Label(frameRecv,text="Received Data:")
labelOutText.grid(row = 1, column = 1, padx = 3, pady = 2, sticky = tk.W)
```

上位机的图形化 UI 界面如图 3.4 所示。停止位和数据位按照发送的数据结构自由调整，点击“Strat”按钮打开串口即可与 nRF24L01 产生连接。

图 3.4 上位机图形化 UI 界面图

左侧按键 KEY1~KEY6 对应中代码配置的 KEY1~KEY6，随机检测并点击其中三个按键，结果如图 3.5 所示。每个按键按下后将弹出与该按键功能的提示框。

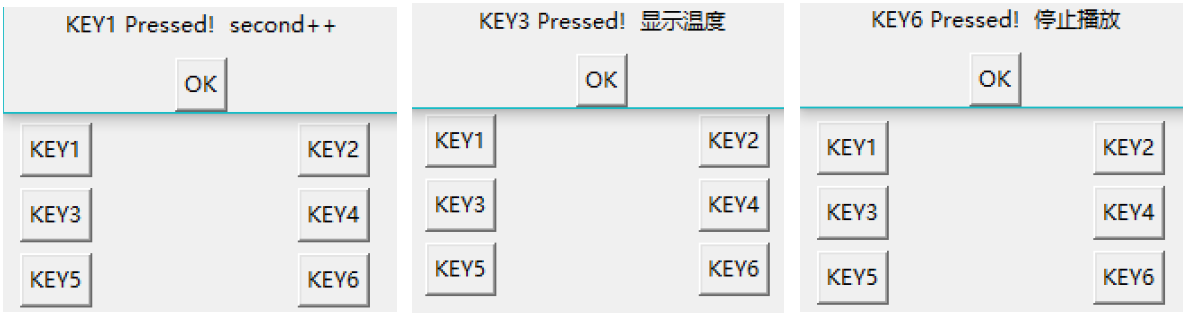


图 3.4 上位机按键功能测试

单片机在接收到数据后，32 单片机将对应数据位的数据转换为整型，并赋给指定的变量中。这样得到了与实体按键 KEY1~KEY6 起到相同效果的三个标志位。闹钟预设时间发送给 nRF24L01 后，STM32 将三个数据位的数据分别从 tmp_buf 数组中提取并强制转换位整型，分别赋给停止的时分秒变量中（Stop_hour，Stop_minute，Stop_Rcount），便完成了定时更改。

4 运行与调试

4.1 硬件功能仿真与调试

按照所画 AD 原理图画出所需 Proteus 仿真图，使用单片机代码进行仿真。硬件仿真图，如图 4.1 所示。

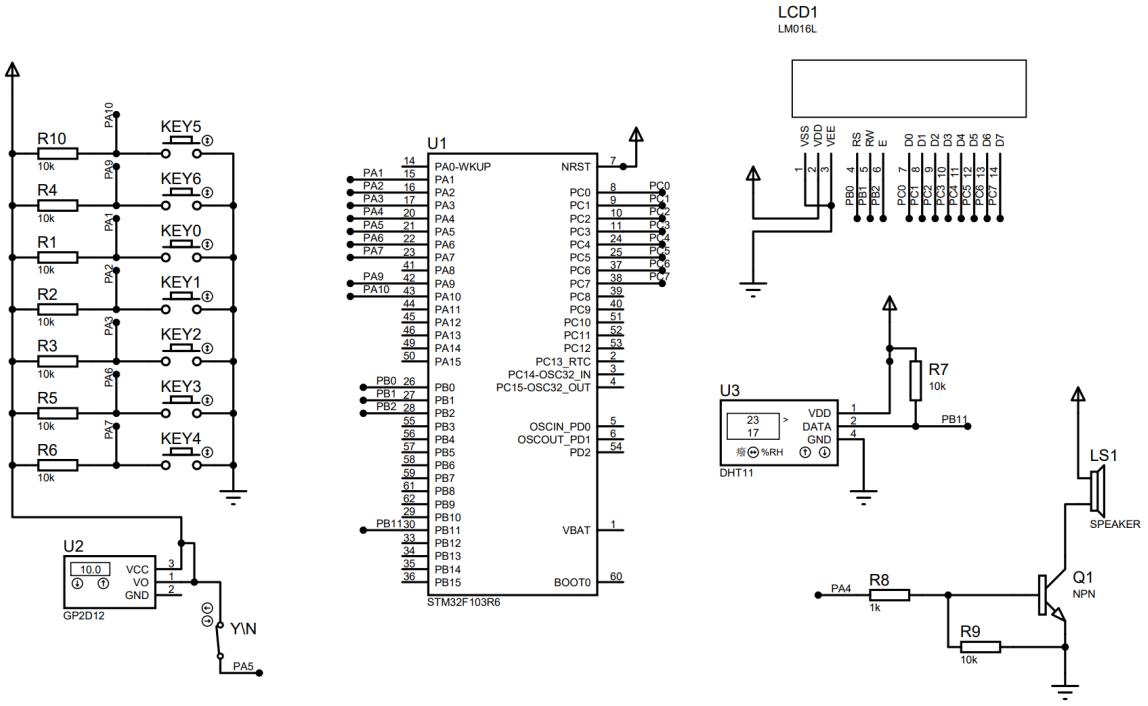


图 4.1 硬件仿真电路图

图中实现了按键仿真、DHT11 仿真、蜂鸣器以及 LCD1602 的功能仿真。开机上电以后，LCD1602 的第二行显示实时时间。按下第一个按键 KEY0，第一行显示定时目标时间的同时，定时时间增加一秒；如图 4.2：

图 4.2 液晶显示 KEY0 按下

第二个按键 KEY1，第一行显示的定时时间减少一秒；如图 4.3：

图 4.3 液晶显示 KEY1 按下

STM32 在接收到 DHT11 发送的温湿度信息数据时，LCD1602 的第一行显示当前温度湿度信息数据。

图 4.4 液晶显示温湿度信息

KEY2，KEY3 为调整分钟，KEY4，KEY5 为调整小时。等到第一行的定时时间与第二行的实时时间重合后，蜂鸣器工作并发出周期性鸣叫。

图 4.5 时钟到达指定时间

4.2 通信系统调试

本次调试 VSPD 提供的串口接口为 COM3，波特率与 nRF24L01 初始化配置的波特率保持一致，即 9600 Baud Rate。由于 Proteus 仿真中难以实现 nRF24L01 的复杂通信机制，本次实训我使用了 MDK5 自带的 Debug 功能，将输出的串口数据经过 VSPD 虚拟串口传入 Python 编写的上位机程序中。VSPD 虚拟串口配置如图 4.6 所示，此时 PC 端物理 COM 端口连接成功。

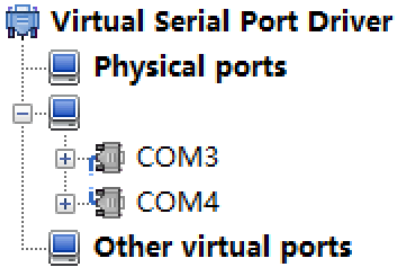


图 4.6 VSPD 虚拟串口配置

接着进行 MDK5 中的配置。在 Keil5 界面上方工具栏中点击 Debug 功能键，进入 MDK5 中的 Debug 模式。

在调试界面中，左下角“Command”提示框为指令面板。如图 4.7，提示框内显示“MODE COM3 9600,0,8,1”对应的意义依次为 COM3 串口，波特率 9600，无奇偶校验，8 数据位，1 终止位。

图 4.7 Debug 界面信息窗口

而调试界面的右下角为 UART1（串口 1）收发信息显示框。点击左上角的运行按钮或按 F5 快捷键开始仿真调试。如图 4.8，上位机从串口接收到的信息与调试界面显示的串口 1 发送信息一致且同步，证明下位机与上位机成功建立连接并完成了数据收发。

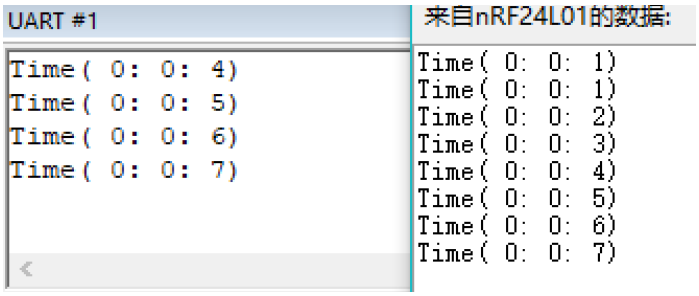


图 4.8 时钟即时显示

按下按钮 KEY3 和 KEY4，分别打开温度、湿度显示功能，串口信息改变为如图 4.9 所示的内容——摄氏度和湿度百分比。

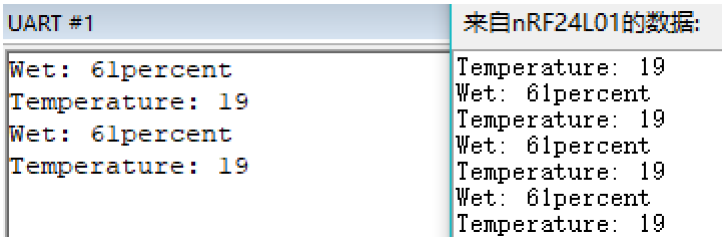


图 4.9 温湿度即时显示

结 论

在本次实训我完成了 AD 原理图的绘制。在 KEIL5 上完成了对智能无线液晶时钟的下位机代码程序设计，并使用 Python 为接口设计出对应的上位机。

这次实训是对我们学习的一个检验，项目中很多知识我们在日常的学习中都没有遇到过，例如以 Python 为接口的上位机与单片机的通信，在不同运行平台、开发环境、程序语言的情形下，使用共同的通信协议（串口）进行数据的交换。

从底层方面看，在本学期单片机接口技术这一课程中，学习重点在于 STM32 库函数版本程序代码的编写，并没有真正地从硬件开始对单片机进行如此全面细致的设计，要求是理解即可，并没有对应的考核形式。因此本次实训我也更加全面具体地掌握了“单片机接口”这一技术，为将来科研也好，就业也罢，都会有所帮助。从产品应用层面看，不仅仅是单片机这一课程，我所学习过的所有课程都没有往用户使用与体验方面靠拢，仅仅是理解理论知识完成指定的任务；而本次实训要求编写的上位机程序涉及到了图形化 UI 界面，开始在应用层对电子产品进行设计和改造，更贴合实际。

这同时提醒我：要想成为一个合格的硬件工程师就应该具备强大自学能力，在工作中会遇到很多从未接触过的问题，当有了问题时要去解决，在你不断努力，寻找答案的过程中，自己的能力也在潜移默化的提升。有时遇到问题时可能有很多想法但却不知道那个正确，这就让我们不断地去探索，不断地尝试。

参考文献

[1] 梁勇．Python 语言程序设计[M]．北京：机械工业出版社，2017：35-41．

[2] 陈佳林．智能硬件与机器视觉[M]．北京：机械工业出版社，2020：186-210．

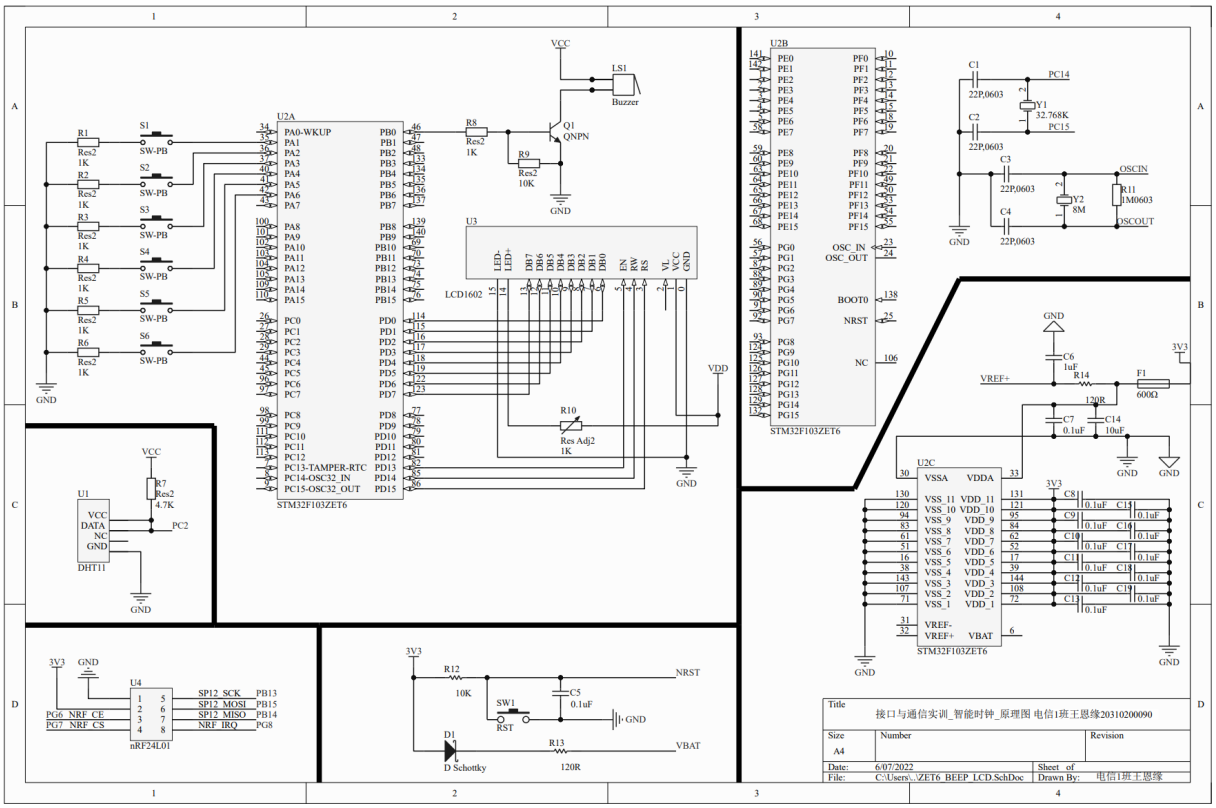
[3] 刘小鲁．智能家居对中国未来家庭的影响 [R]．宏观经济研究报告，2017．

[4] 刘火良，杨森．STM32 库开发实战指南：基于 STM32F103[M]．北京：机械工业出版社，2019：85-96．

[5] STM32F103 datasheet [EB/OL]．www.st.com/en/microcontrollers /stm32f103.html

附 录

AD 原理图总览：



参考代码：

<main.c>

```
u8 RunSYS=0;
u8 key,mode;
u16 t=0;
u8 tmp_buf[64];
void Init_Buffer (unsigned char *P,unsigned int Count,unsigned char Type);
void nRF24L01(void);

void KEY_Scan(void);
void BEEP_Alert(void);
void DHT11_Scan(void);

int main(void)
{
    uart_init(115200);
    Timerx_Init(1999,35999);      //计数一次 1s
```

```
    NVIC_Configuration();

    BEEP_Init();
    LCD1602_Init();
    NRF24L01_Init();
    Dht11_Init();

    while(1)
    {
        KEY_Scan();
        BEEP_Alert();
        DHT11_Scan();
        nRF24L01();
    }
}

void DHT11_Scan()                //判断 Dht11_Start()是否为 0，如果为 0 则调用 Dht11_Read 函数，并输出温湿度。
{
    if(Dht11_Start()==0)
    {
        Dht11_Read(temperature);

        LCD1602_Show_Str(1, 0, (u8*)"                    ");
        LCD1602_Show_Str(1, 0, temperature);             //显示温度
    }
}

void KEY_Scan()
{
    if(KEY1==0)
    {
        delay_ms(10);                    //软件消抖
        if(KEY1==0)
        {
            Stop_Rcount++;               //秒数+
    sprintf((char*)LCD_Display_Stop,"STOP++%2d:%2d:%2d",Stop_hour,Stop_minute,Stop_Rcount);
            LCD1602_Show_Str(1, 0, (u8*)"                    ");
            LCD1602_Show_Str(1, 0, LCD_Display_Stop);        //显示更改的时间

            while(KEY1==0);              //抬手检测
        }
```

```
    }
    else if(KEY2==0)
    {
        delay_ms(10);                       //软件消抖
        if(KEY2==0)
        {
            Stop_Rcount--;                  //秒数-

            sprintf((char*)LCD_Display_Stop,"STOP-
-%2d:%2d:%2d",Stop_hour,Stop_minute,Stop_Rcount);
            LCD1602_Show_Str(1, 0, (u8*)"                    ");
            LCD1602_Show_Str(1, 0, LCD_Display_Stop);          //显示更改的时间

            while(KEY2==0);
        }
    }
    else if(KEY3==0)
    {
        ……
    }
    ……
}
void BEEP_Alert()
{
    if(Stop_Rcount==Rcount && Stop_minute==minute && Stop_hour==hour)
    {
        while(1)
        {
            BEEP_1;
            delay_ms(1000);
            BEEP_0;
            delay_ms(1000);

            if(KEY1==0)         //KEY1 取消报警（长按）
            {
                delay_ms(10);
                if(KEY1==0)
                    break;
            }
        }
    }
}
```