个人信息脱敏

题目: 基于 STM32 的智能物联网家居系统

一、实验目的

(1) 本实验要求基于 ARM Cortex 嵌入式系统完成综合项目设计；

(2) 将理论知识与实际应用相结合，通过实际操作检验嵌入式系统知识的理解和掌握。

(3) 鼓励学生发挥创造力，设计新颖的解决方案，培养创新思维和独立思考的能力;

(4) 通过项目设计，学生可以培养和提高解决复杂问题的能力。

二、实验环境

软件平台：Windows10 操作系统、Keil 5 IDE、HBuilder X IDE、STM32 CubeMX、EMQX Server.

硬件设备：STM32F103C8T6 开发板、ESP01-S WIFI 模块、CH340 USB to TTL 模块、DT11 温湿度传感器、LED 灯、2*2 开关、有源蜂鸣器、ST7735S TFT 显示器、PCB 板、直插式排母若干。

三、实验方案设计

本次实验我要实现的系统是《基于 STM32 的智能物联网家居系统》。

智能家居系统是指利用物联网技术，通过传感器、控制器、执行器等设备，实现家庭环境监测、设备控制、安全监控等功能的自动化和智能化的家居解决方案，兼具实用性和创新型。

智能家居系统应该体现两个方面的智能：其一，智能家居自身能够采集数据，并根据数据自动做出响应，如自动调节温度、照明、发出警报等，提高家庭安全；其二，用户应该可以通过网络远程控制智能设备，监控并管理家中设备，提高居住的舒适度和便利性。也就是说，我们的系统应该分为设备端和用户端两个模块开发，用户端通过网络与设备端交互。

3.1 设备端硬件方案设计

作为智能家居系统，其基本的功能和要求包括：

① 房间环境测量、警报与照明控制功能，以及高温报警、高温断电等功能；

② 用户可以通过 Android 客户端查看测量数据，控制警报与照明开关；

③ 通过输出设备进行数据显示；

④ 具有网络通信功能，能够连接至服务器。

因此，我们根据功能，将硬件分为三个模块：数据采集和处理模块、输出模块、通信模块。

① 数据采集和处理模块：由 STM32F103C6 单片机、DT11 温湿度传感器、LED 灯、蜂鸣报警器组成；

② 输出模块：我们使用 ST7735S TFT 屏幕作为数据显示接口；

③ 通信模块：考虑到单片机性能，我们使用物联网设备最常用的 MQTT 协议与用户端进行通信，使用 ESP-01S WIFI 模块作为通信元件。

根据外设端口要求，我设计的智能家居系统的硬件接线图如图 1 所示：

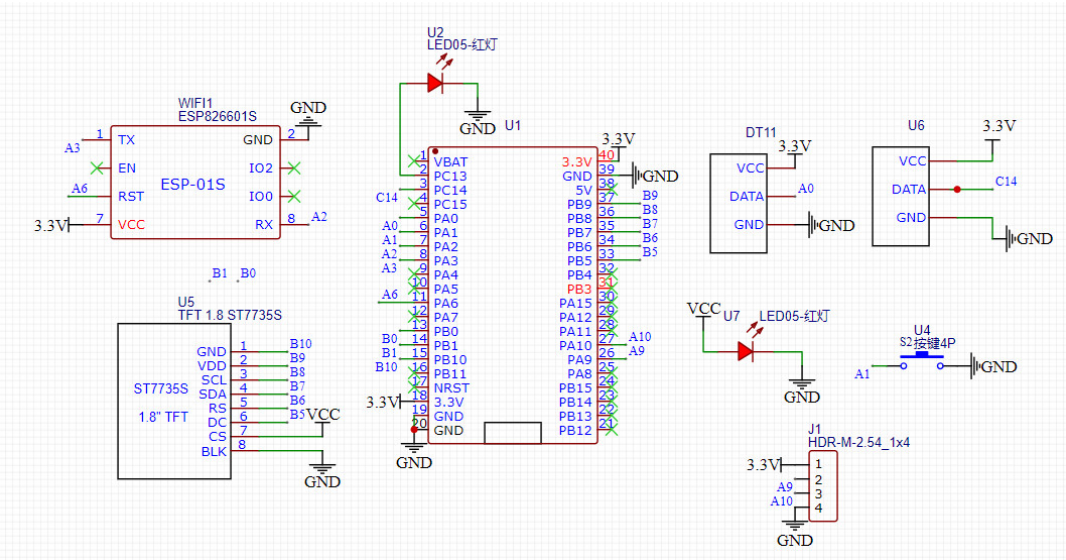


图 1 硬件接线图

3.2 用户端软件方案设计

考虑到真实使用场景，我们认为移动端小程序适合作为智能家居系统的载体，便于用户随时查看和操控房间内的物联智能设备。因此，我们的小程序应该具备的功能和要求包括：

① 提供简单易用的查看和操作页面；

② 能够连接至服务器，通过服务器与设备端通信；

③ 能够查看环境数据、控制照明和报警系统；

④ 记录设备的历史数据，并进行呈现。

3.3 系统框架图

我们使用 MQTT Server 实现设备与用户连接，最终形成的系统框架图如图 2 所示：

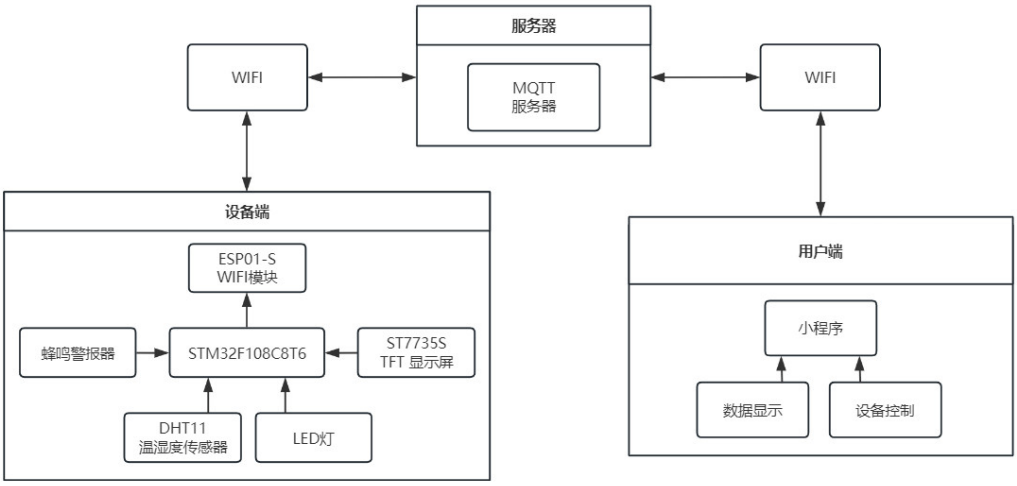


图 2 系统框架图

四、项目硬件实现过程

项目的硬件独立开发，最后整合为完整系统。

4.1 DHT11 温湿度传感器

4.1.1 设备概述

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器，内部由一个 8 位单片机控制一个电阻式感湿元件和一个 NTC 测温元件。

其引脚较为简单，VDD 接 3.3V 供电，GND 接负极，DATA 为串行数据总线，接 PA0。

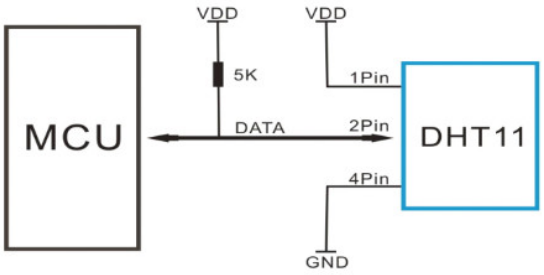


图 3 DHT11 引脚图

4.1.2 控制原理

DHT11 采用单总线协议通信与单片机通信单片机发送一次复位信号后，DHT11 从低功耗模式转换到高速模式，等待主机复位结束后，DHT11 发送响应信号，并拉高总线准备传输数据。

数据格式为 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 + 8bit 校验和，共 40bit 数据，由于 DHT11 的精度值只达到个位，因此小数部分全为 0。

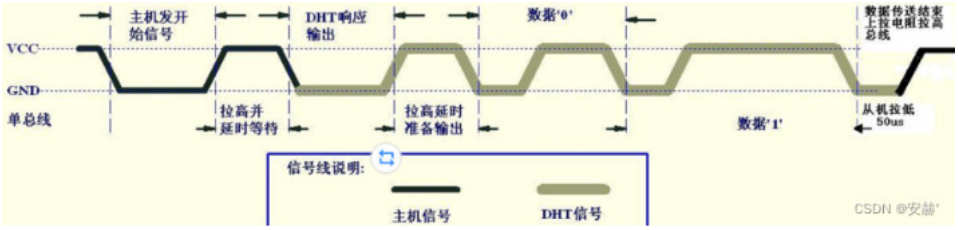
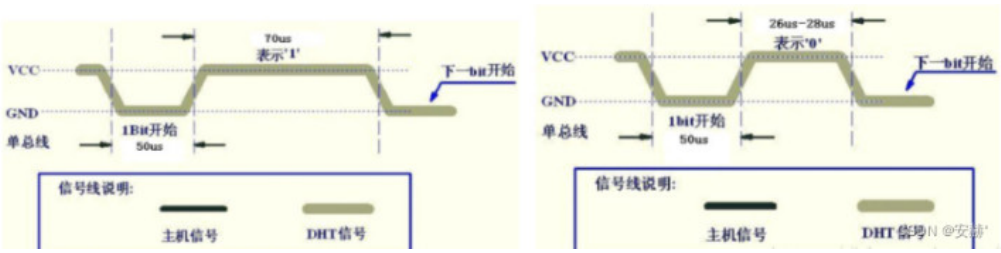


图 4 DHT11 操作时序图

DHT11 的操作时序图如图 4 所示： DHT11 接收到主机开始信号后，会触发一次温湿度采集；然后拉低总线 80us 表示数据准备完成，再拉高总线 80us 后开始传输数据。

也就是说主机发送开始信号后，如果检测到总线被拉低，就每隔 1us 计数一次，直至计数至 80 次；然后总线被拉高，当总线被拉高后重新计数检测 80us 的高电平，主机就准备开始接收数据。

实际上 DHT11 的响应时间并不是标准的 80us，往往存在误差，当响应时间处于 20~100us 之间时就可以认定响应成功。



DHT11 以高电平的长短定义数据位是 0 还是 1,每 1bit 数据都以 50us 低电平时隙开始，当 50us 低电平时隙过后拉高总线，高电平持续 26~28us 表示数据“0”；持续 70us 表示数据“1”。当最后 1bit 数据传送完毕后，DHT11 拉低总线 50us，表示数据传输完毕，随后总线由上拉电阻拉高进入空闲状态。

4.1.3 代码实现

我们知道每个数据以 50us 低电平时隙开始，然后拉高电平，数据“0”的高电平持续 26~28us，数据“1”的高电平持续 70us。

因此，当数据位之前的 50us 低电平时隙过后，总线肯定会拉高，此时延时 40us 后检测总线状态，如果为高，说明此时处于 70us 的时隙，则数据为“1”； 如果为低，说明此时已经过了 26-28us 间隙，数据为“0”。

也就是说，要分辨 0/1，只需要检测电平由低变高后，等待 40us 再检测高低，即可判断当前数据为 0/1，代码如下图所示：

```
//从DHT11读取一个位
//返回值：1/0
uint8_t DHT11_Read_Bit(void)
{
  uint8_t retry=0;
  while(DHT11_DQ_IN&&retry<100)//等待变为低电平
  {
    retry++;
    DelayUs(1);
  }
  retry=0;
  while(!DHT11_DQ_IN&&retry<100)//等待变高电平
  {
    retry++;
    DelayUs(1);
  }
  DelayUs(40);//等待40us
  if(DHT11_DQ_IN)return 1;
  else return 0;
}
```

图 5 DHT11 获取 bit 数据

在读取到 40bit 数据后，我们组合为 5byte 数据，若前 4 个 byte 数据之和等于第 5 个，说明符合校验和，数据无误，然后将传感器采集结果赋值到 temp 和 humi 变量中。代码如下图所示：

```
uint8_t DHT11_Read_Data(uint8_t *temp,uint8_t *humi)
{
  uint8_t buf[5];
  uint8_t i;
  DHT11_Rst();
  if(DHT11_Check()==0)
  {
    for(i=0;i<5;i++)//读取40位数据
    {
      buf[i]=DHT11_Read_Byte();
    }
    if((buf[0]+buf[1]+buf[2]+buf[3])==buf[4])
    {
      *humi=buf[0];
      *temp=buf[2];
    }
  }
  else return 1;
  return 0;
}
```

图 6 DHT11 获取温湿度数据

4.2 ST7735S TFT 屏幕

4.2.1 设备概述

ST7735S 是一块 1.8 英寸采用 SPI 通信的 TFT 全彩屏。其引脚定义为：

① BLK：背光调节；② CS：片选信号；③ DC：读/写模式选择信号；

④ RST：复位信号；⑤ SDA：SPI 数据线；⑥ SCL：SPI 时钟线；

⑦ VDD：兼容+5V 和+3.3V；⑧ GND：接地；

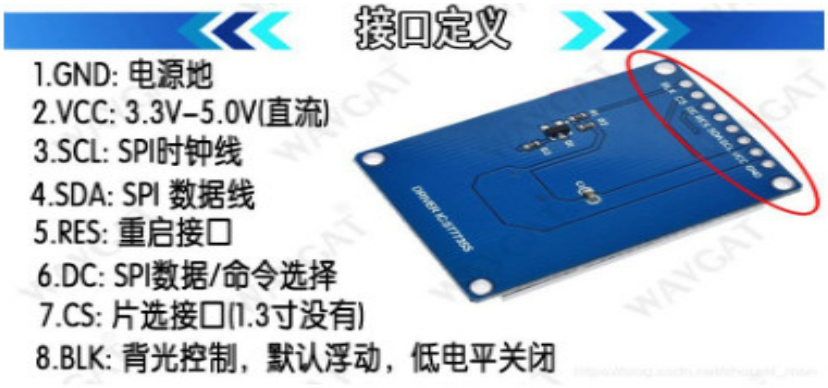


图 7 ST7735S 引脚定义

4.2.2 控制原理

SPI 是一种高速的全双工通信协议，用于微控制器和各种外围设备之间的数据交换，可以在同一时间发送和接收数据。通常采用主从模式，即有一个主设备（通常是微控制器）控制通信的时序，而一个或多个从设备（如传感器、显示器等）响应主设备的命令。

SPI 通信使用四根线：

① MOSI：主设备数据输出，从设备数据输入线。

② MISO：主设备数据输入，从设备数据输出线。

③ SCLK：时钟线，由主设备生成，用于同步数据传输。

④ CS：片选线，用于激活特定的从设备。

SPI 主设备和从设备都有一个串行移位寄存器，主设备通过向它的 SPI 串行寄存器写入一个字节来发起一次传输，其通信流程可以简单分为如下步骤:

① 主设备发起信号，将 CS 拉低，启动通信；

② 主设备通过发送时钟信号，来告诉从设备进行写数据或者读数据操作；

③ 主机将要发送的数据写到发送数据缓存区，缓存区经过移位寄存器通过 MOSI 信号线送至从机，同时 MISO 接口接收到的数据经过移位寄存器移到接收缓存区；

④ 从机也进行相同的操作，从而实现主从机移位寄存器内容的交换。

在 SPI 协议中，没有实际的读写之分，读和写操作是同步完成的。读写操作通过对移位寄存器的使用进行区分。

图 8 SPI 主从模式示意图

4.2.3 代码实现

我们使用 HAL 库模拟 SPI 通讯驱动 ST7735S 屏幕显示内容。基本思路是通过配置 GPIO 引脚模拟 SPI 通讯中的 SCK、MOSI、MISO 和 SS 信号，然后利用 HAL 库函数模拟 SPI 信号传输。

ST7735S 屏幕的使用流程包括：启动-复位-初始化-控制内容。

初始化主要是退出睡眠模式，配置电压和刷新率等操作，其具体内容可通过查阅文档得到，其部分代码如下图所示，不再赘述：

```
void LCD_Init(void)
{
  LCD_GPIO_Init();//初始化GPIO

  LCD_RES_Clr();//复位
  DelayMs(100);
  LCD_RES_Set();
  DelayMs(100);


  //************* Start Initial Sequence **********//
  LCD_WR_REG(0x11); //Sleep out
  DelayMs(120);              //Delay 120ms
  //------------------------------------ST7735S Frame Rate-----------------------------------------//
  LCD_WR_REG(0xB1);
  LCD_WR_DATA8(0x01);
  LCD_WR_DATA8(0x2C);
  LCD_WR_DATA8(0x2D);

  LCD_WR_REG(0xB2);
  LCD_WR_DATA8(0x01);
  LCD_WR_DATA8(0x2C);
  LCD_WR_DATA8(0x2D);

  LCD_WR_REG(0xB3);
  LCD_WR_DATA8(0x01);
  LCD_WR_DATA8(0x2C);
  LCD_WR_DATA8(0x2D);
  LCD_WR_DATA8(0x01);
  LCD_WR_DATA8(0x2C);
  LCD_WR_DATA8(0x2D);
  //------------------------------------End ST7735S Frame Rate---------------------------------//
```

图 9 ST7735S 初始化代码（部分）

ST7735S 的核心函数是控制内容显示，其实现思路是：用四个八位的数据表示写入区域，分别是 X 的起始坐标和终点坐标,Y 的起始坐标和终点坐标。通过写入 0X2A 和 0X2B 寄存器将四个八位坐标发送至屏幕，之后配置 0X2C 寄存器，进行坐标范围内的颜色配置。

而要在坐标区域内绘画出各种图形和文字，我们可以使用取模工具提前获取颜色参数值，而不必单个像素设置。

以字符显示函数为例，输入颜色、字号、坐标，然后程序遍历字体数组，对像素点进行渲染。

```
void LCD_ShowChar(u16 x,u16 y,u8 num,u16 fc,u16 bc,u8 sizey,u8 mode)
{
  u8 temp,sizex,t;
  u16 i,TypefaceNum;//一个字符所占字节大小
  u16 x0=x;
  sizex=sizey/2;
  TypefaceNum=sizex/8*sizey;
  num=num-' ';    //得到偏移后的值
  LCD_Address_Set(x,y,x+sizex-1,y+sizey-1);  //设置光标位置
  for(i=0;i<TypefaceNum;i++)
  {
    if(sizey==16)temp=ascii_1608[num][i];          //调用8x16字体
    else if(sizey==32)temp=ascii_3216[num][i];     //调用16x32字体
    else return;
    for(t=0;t<8;t++)
    {
      if(!mode)//非叠加模式
      {
        if(temp&(0x01<<t))LCD_WR_DATA(fc);
        else LCD_WR_DATA(bc);
      }
      else//叠加模式
      {
        if(temp&(0x01<<t))LCD_DrawPoint(x,y,fc);//画一个点
        x++;
        if((x-x0)==sizex)
        {
          x=x0;
          y++;
          break;
        }
      }
```

4.3 ESP01-S WIFI 模块

4.3.1 设备概述

ESP01-S 是一款基于 ESP8266 芯片的 Wi-Fi 模块，支持 802.11b/g/n 标准的 WIFI 网络，可以无线传输数据；另外，ESP01-S 具有睡眠模式，可以在不工作时降低功耗，非常适合智能家居应用场景。

ESP01-S 使用串口（TX、RX）与单片机通信，传输速率有限，不适合大型数据。

4.3.2 通信规则

ESP01 通过 AT 指令与设备通信，其规则如下：

- 测试命令：仅仅发送指令 “AT” 即测试 ESP01-S 模块是否准备好，若准备好则响应 “OK”。
- 读取命令：在 AT 指令后面加上 “=?”即构成测试命令。例如 “AT+MODE?”，会列举当前是什么模式。
- 执行命令：在 AT 指令后面加上 “=”再接上相应的参数即可，例如 “AT+MODE=NORMAL”，将当前模式设置为正常模式。
- ESP01-S 支持 AP 和 STA 两种工作模式，分别用于开启热点让其他设备连接，或者连接其他设备热点两种工作模式。在本系统中，我们使用 STA 模式工作。

注：命令响应处理通过烧录 ESP01-S 的相关固件实现，此处我们烧录了 AT-MQTT 固件，ESP01-S 接收到相关指令后，会做出相应的响应，主机根据返回值进行处理即可。

相关代码如下图所示，我们通过串口向 ESP01-S 发送命令，获取并检验其返回值：

```
_Bool ESP8266_SendCmd(char *cmd, char *res)
{

  unsigned char timeOut = 200;

  Usart_SendString(USART2, (unsigned char *)cmd, strlen((const char *)cmd));

  while(timeOut--)
  {
    if(ESP8266_WaitRecive() == REV_OK)              //如果收到数据
    {
      if(strstr((const char *)esp8266_buf, res) != NULL)    //如果检索到关键词
      {
        ESP8266_Clear();                  //清空缓存

        return 0;
      }
    }

    DelayXms(10);
  }

  return 1;

}
```

图 10 ESP01-S 命令

4.3.3 ESP01-S 初始化

我们在初始化时通过串口向模块依次发送如下命令：

1. AT：建立连接；
2. AT+CWMODE=1：将 ESP01-S 设置为 Station 模式，使其能连接 WIFI 热点；
3. AT+CWDHCP=1：启动 DHCP 协议，为 ESP01S 配置一个 IP 地址；
4. AT+CWJAP=WIFI 账号/密码：让 ESP01-S 连接到指定的 WIFI 热点；

至此，WIFI 模块的初始化工作完成，ESP01-S 成功连接至 WIFI 热点。

4.4 MQTT 协议

4.4.1 MQTT 协议概述

MQTT 是一种基于发布/订阅模式的轻量级通讯协议，构建于 TCP/IP 协议上，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。由于其轻量级、低带宽的特性（协议的固定头部仅 2 字节），很适合用于智能物联网设备通信场景。

MQTT 协议中有三种身份：代理、发布者、订阅者。每个订阅者订阅一个或多个主题，发布者指定主题，并将消息发布至消息代理服务器，服务器根据主题转发给对应的订阅者。

其工作模式图，如下图所示：

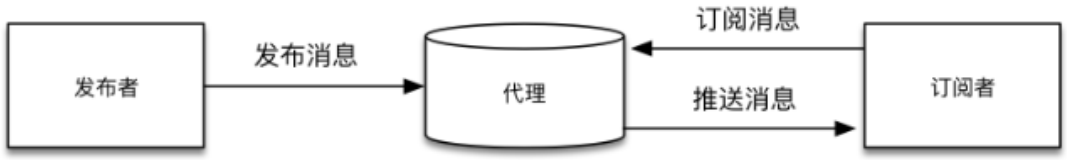


图 11 MQTT 工作模式图

4.4.2 MQTT 服务器连接

ESP-01S 初始化后，需要连接至我们的 MQTT 服务器，并订阅主题 Operate，以接收用户传来的指令，代码如下图所示：

```
#define ESP8266_ONENET_INFO   "AT+CIPSTART=\"TCP\",\"wf111287.ala.dedicated.aliyun.emqxcloud.cn\",1883\r\n"
while(ESP8266_SendCmd(ESP8266_ONENET_INFO, "CONNECT"))
DelayXms(500);
OneNet_Subscribe(devPubTopic, 1);
```

同时，我们的移动设备也要作为发布者，向用户端发送采集到的传感器信息。

MQTT 传输的消息分为：主题和负载两部分，主题我们设置为 GetData，payload 设置为传感器数据，用户订阅主题 GetDatas 后可以获取所需数据。

采集到 DHT11 数据后，我们每 5s 向服务器发送一次传感器信息。

```
DHT11_Read_Data(&temp,&humi);
if(++timeCount >= 100)                    //发送间隔5s
{
  sprintf(PUB_BUF,"{\"DeviceID\":3,\"Humi\":%d,\"Temp\":%d,\"Led\":%d,\"Beep\":%d}",
    humi,temp,led_info.Led_Status,Beep_info.Beep_Status);
  OneNet_Publish("getData/3", PUB_BUF);
  timeCount = 0;
  ESP8266_Clear();
}
```

```
void OneNet_Publish(const char *topic, const char *msg)
{

  MQTT_PACKET_STRUCTURE mqttPacket = {NULL, 0, 0, 0};              //协议包

  UsartPrintf(USART_DEBUG, "Publish Topic: %s, Msg: \r\n", topic);

  if(MQTT_PacketPublish(MQTT_PUBLISH_ID, topic, msg, strlen(msg), MQTT_QOS_LEVEL0, 0, 1, &mqttPacket) == 0)
  {
    ESP8266_SendData(mqttPacket._data, mqttPacket._len);           //向平台发送订阅请求

    MQTT_DeleteBuffer(&mqttPacket);                       //删包
  }

}
```

图 12 MQTT 发送数据

4.5 STM32 单片机

单片机作为所有外围设备的主机，需要负责硬件初始化、硬件配置、硬件控制和数据处理等操作。硬件初始化和硬件配置不再赘述，我们只讲数据处理部分的操作。

在主循环中，单片机不断获取 DHT11 的数据，然后每隔 5s 像服务器发送一次数据。同时，要监听用户发送至主题 Operate 的控制指令，并作出响应：

```
case MQTT_PKT_PUBLISH:                          //接收的Publish消息
  result = MQTT_UnPacketPublish(cmd, &cmdid_topic, &topic_len, &req_payload, &req_len, &qos, &pkt_id);

    if(result == 0)
    {
      char *data_ptr = NULL;

      UsartPrintf(USART_DEBUG, "topic: %s, topic_len: %d, payload:, payload_len: %d\r\n",
                                cmdid_topic, topic_len, req_len);
      json=cJSON_Parse(req_payload);

      if(json!=NULL)
      {
        json_led = cJSON_GetObjectItem(json,"LED");
        json_beep = cJSON_GetObjectItem(json,"BEEP");
        if(json_led!=NULL){
          if(json_led->valueint)
          {
              Led_Set(LED_ON);

          }
          else Led_Set(LED_OFF);
          cJSON_Delete(json_led);
        }
```

图 13 监听指令

同时，STM32 采集到数据后，不仅要显示数据，还要根据环境温度，主动报警机制：

```
void Refresh_Data(void)
{
  if(temp>29){
    BEEP_Set(BEEP_ON);
  }else{
    BEEP_Set(BEEP_OFF);
  }
  if(temp>29){
    LCD_ShowIntNum(55,20,temp,4, RED ,WHITE,16);
    LCD_ShowString(70,95,"ON ",RED,WHITE,16,0);
    LCD_ShowString(20,120,"Temp Too High",RED,WHITE,16,0);
  }else{
    LCD_ShowIntNum(55,20,temp,4, DARKBLUE ,WHITE,16);
    LCD_ShowString(70,95,Beep_info.Beep_Status ? "ON " : "OFF",DARKBLUE,WHITE,16,0);
    LCD_ShowString(20,120,"Temp Comfort ",DARKBLUE,WHITE,16,0);
  }
  LCD_ShowIntNum(55,45,humi,4,DARKBLUE,WHITE,16);

  LCD_ShowString(70,70,led_info.Led_Status ? "ON " : "OFF",DARKBLUE,WHITE,16,0);
}
```

图 14 数据响应与屏幕刷新

4.6 PCB 板设计

使用嘉立创 EDA 绘制硬件接线图，然后转化为 PCB 板，选择 6 层板进行布线，如下图所示：

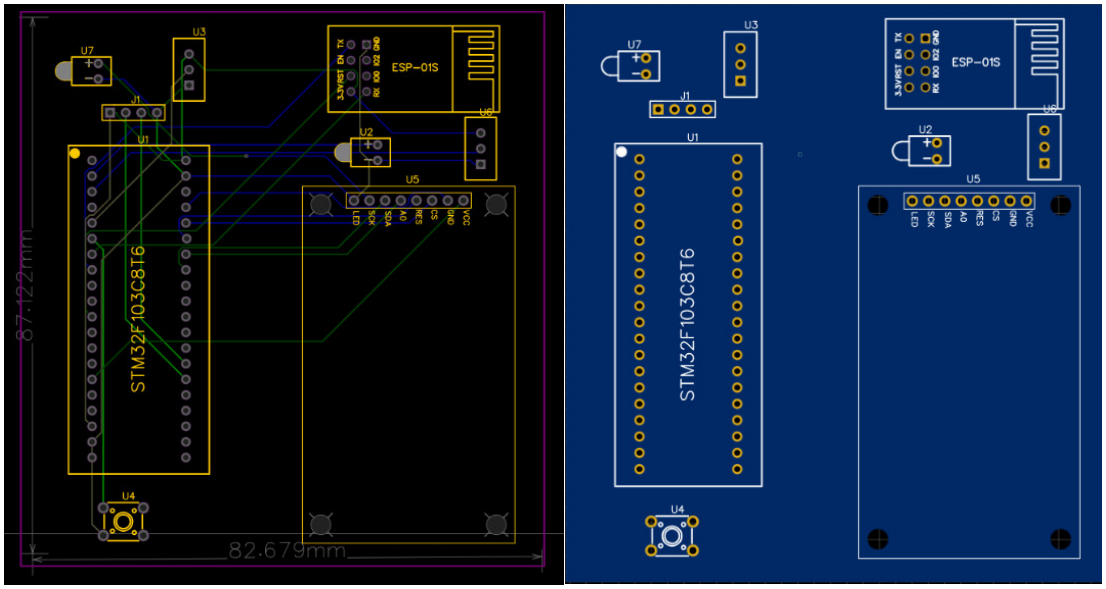


图 15 PCB 设计

注意的是，PCB 板层数越多，越容易进行布线，线路越不容易发生冲突。

可以将排母焊接至 PCB 板中替代直接焊接硬件，方便后期替换硬件；面包板比 PCB 板更为灵活，能满足一般的实验需求。

五、项目软件实现过程

小程序端设计的过程并非本课程的重点，因此我们只挑核心内容进行讲解，小程序以最终的效果呈现为主。

作为智能家居系统的配套程序，其核心功能是查看房间环境数据，以及控制物照明和报警开关等功能，还要将设备发送的数据进行存储，以日志的形式呈现。

在连接到 MQTT 服务器后，小程序需要订阅 GetData 主题，监听来自设备端的消息并同步更新至用户界面：

```
    _this.client.on(
        "message",
        function (topic, message) {
            var sensorData = JSON.parse(message)

            //获取数据 绑定数据
            _this.temp = sensorData.Temp + "℃"
            _this.humi = sensorData.Humi + "%"
            _this.led = sensorData.Led==1?true:false
            _this.beep = sensorData.Beep==1?true:false
            _this.logs.push(`${_this.getFormattedDate()}  设备数据: { Temp: ${_this.temp} ,
            //_this.alarm = false
        }
    )
},
```

图 16 获取设备信息

除此之外，用户的小程序端应该能控制照明、警报器等开关，其做法是像 Operate 主题发送对应的指令，设备收到指令后会作出响应：

```
// TODO LED开关执行方法
onLedChange(event) {
    var _this = this
    var switchState = event.mp.detail.value
    console.log(switchState)
    if (switchState) {
        console.log("下发指令: 开灯")
        _this.client.publish(
            'Operate/3',
            '{"LED":1}',
            function (error) {
                if (!error) {
                    console.log("成功下发指令到设备端: 开灯")
                    _this.logs.push(`${_this.getFormattedDate()} 下发指令: 开灯`);
                }
            })
```

图 17 控制物联设备

至此，系统的软硬件实现介绍完毕。

六、实物展示与运行效果演示

6.1 PCB 板与实物

如图 18 所示，可以看到我们的硬件系统集成了 STM32 开发板、TFT 全彩屏、ESP01-S WIFI 模块、DHT11 温湿度传感器、蜂鸣警报器、LED 灯等硬件：

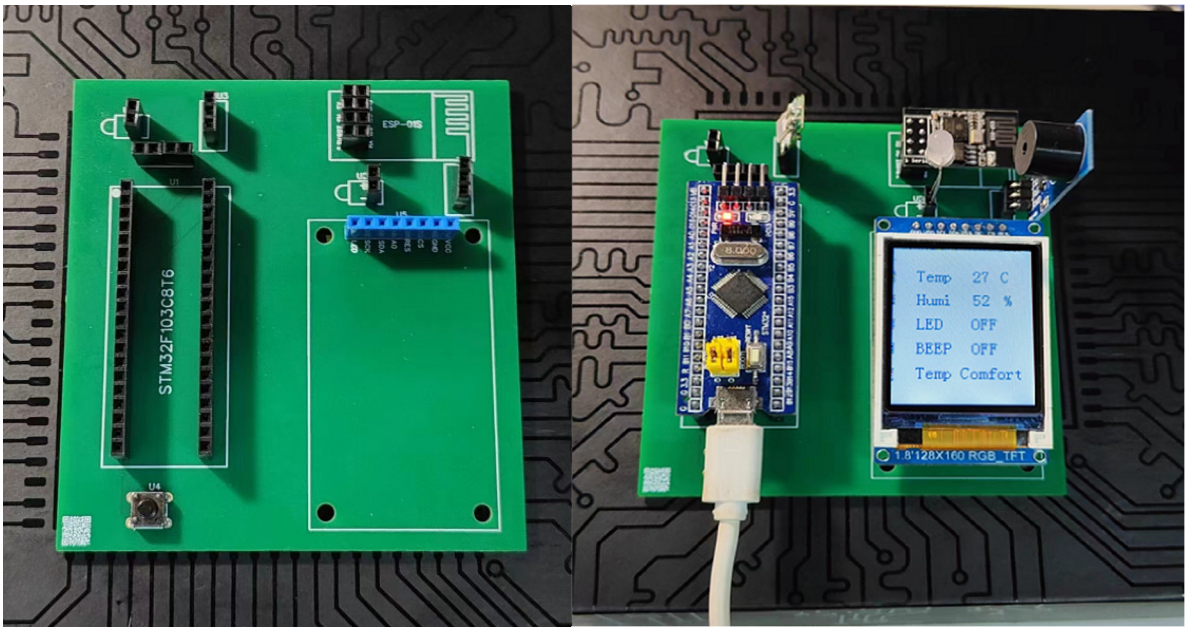


图 18 实物演示

6.2 温度采集+过热警报

过热时超过阈值（32℃）会触发蜂鸣器警报，温度降下来时会关闭蜂鸣器警报。

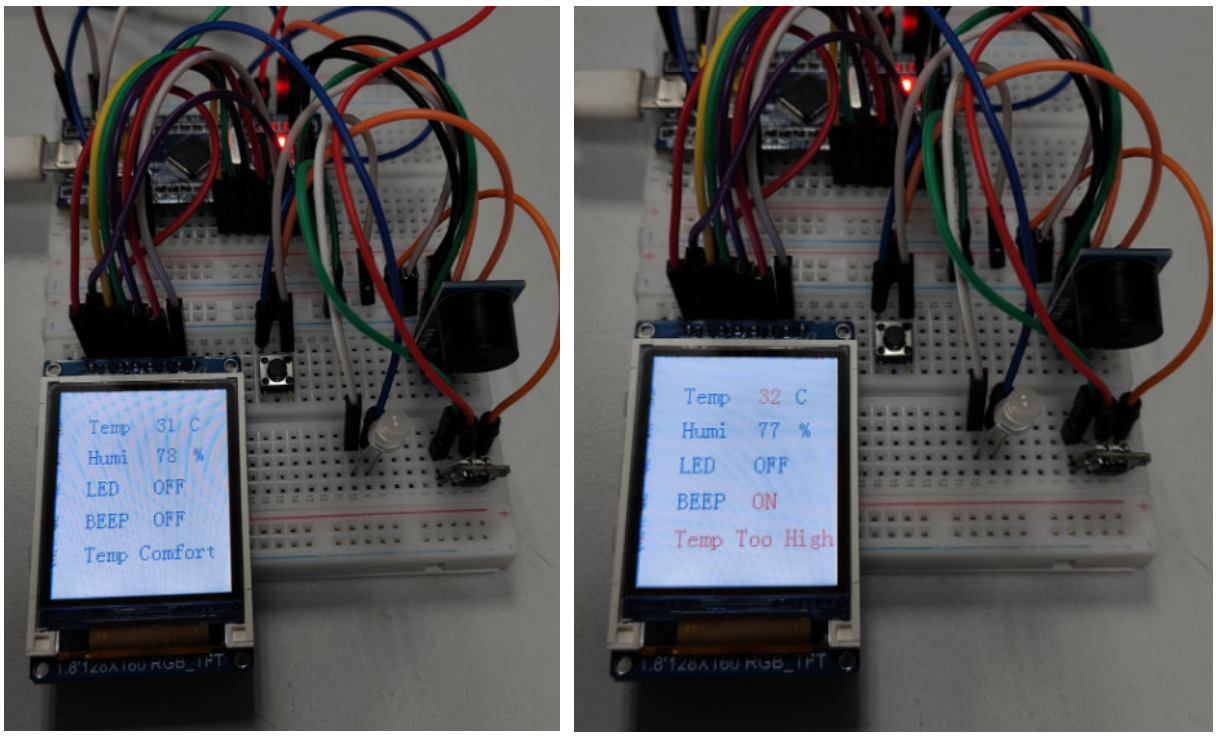


图 19 温度采集与过热警报

6.3 手机端程序、设备控制、日志显示

在手机程序中，我们可以看到设备发送的温湿度数据以及开关灯状态，我们也可以通过小程序向设备发送指令，控制其 LED 开关和警报器开关。

另外，设备的数据日志会被呈现在小程序下方，以便用户查看。

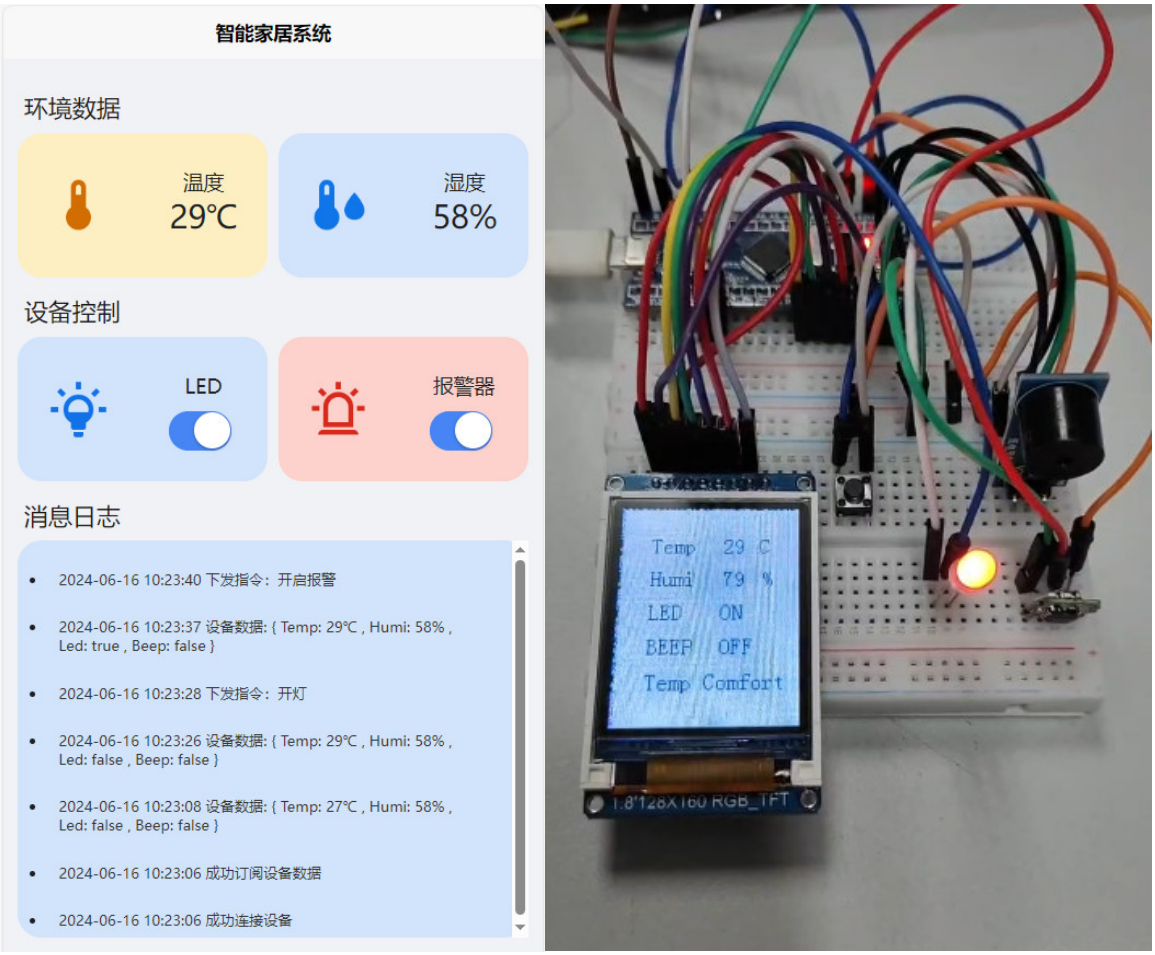


图 20 手机端程序、设备控制、日志显示

更详细的效果展示的 PPT 的视频演示中，此处不再展开。

七、实验总结

通过本次实验，我搭建了一套基于 STM32 单片机的智能家居系统。

该系统整合了 DHT11 温湿度传感器、ST7735S TFT 屏幕、ESP-01S WIFI 模块等多种硬件，实现了房间环境测量、警报与照明控制功能，以及高温报警、高温断电等功能，硬件设备被集成至一块 PCB 板上。硬件整合的过程涉及硬件设计、PCB 设计，还涉及 SPI、串口、MQTT 等各种通信协议，对能力是极大的考验。

此外，我们还为智能家居系统编写了移动程序。设备与用户程序通过 MQTT 协议连接至同一个服务器上，用户可以通过移动程序与硬件进行交互，实现了数据查看和设备控制的功能，符合智能家居的真实使用场景。

这次实验不仅加深了我对嵌入式系统设计的理解，提升了解决实际问题的能力。通过理论与实践的结合，我们成功构建了一个实用的智能物联网家居软硬件系统，完成了物联网设备的主流全流程开发。为我以后的职业道路选择（物联网方向）提供了指引。