

**毕业论文**

基于STM32的智能空调调节

系统设计

**学生姓名**： **学号**：

235033102

杨嘉

自动化系

**系 部：**

电气工程及其自动化

**专 业：**

朱珊（讲师）

**指导教师：**

二○二五年六月

**诚信声明**

本人郑重声明：

所呈交的毕业论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

本人签名： 年 月 日

基于STM32的智能空调调节系统设计

摘 要

本论文设计并实现了一种基于STM32F103C8T6微控制器的智能空调控制系统，旨在提升室内环境舒适性、降低能源消耗并实现远程管理。系统集成温湿度传感器、继电器控制模块、OLED显示模块及WiFi通信模块，通过DHT22传感器采集室内环境数据，主控芯片结合预设阈值自动调节空调运行状态，实现智能控制。系统支持自动控制、本地手动控制及远程控制三种工作模式，其中远程控制基于MQTT协议接入中国移动OneNET平台，支持移动端和网页端远程操控。软件架构采用模块化设计，功能分层清晰，主程序采用状态机机制结合定时调度实现多任务协作与系统稳定运行。本系统具备部署灵活、控制精度高、用户交互友好等优点，具备良好的应用前景和推广价值。

关键词: 智能空调控制，MQTT协议，OneNET平台，状态机调度

**Design of an Intelligent Air Conditioning Regulation System Based on STM32**

**Abstract**

This thesis designs and implements an intelligent air conditioning control system based on the STM32F103C8T6 microcontroller, aiming to improve indoor comfort, optimize energy consumption, and enable remote intelligent management. The system adopts a modular architecture and mainly consists of a central control unit, a temperature and humidity sensing module, a relay control module, an OLED display module, and a WiFi communication module. A DHT22 sensor is used to collect real-time indoor temperature and humidity data. The main controller automatically adjusts the operating status of the air conditioner according to preset thresholds. The software supports three working modes: automatic control, local manual control, and remote control. The remote control function is implemented using the MQTT protocol and connects to China Mobile’s OneNET platform, supporting mobile and web-based remote operation. The system uses a modular software design and a state machine combined with timed scheduling to achieve multitask coordination and stable operation. The proposed system features flexible deployment, high control accuracy, and a user-friendly interface, showing good prospects for practical application and promotion.

**Keywords:** Intelligent air conditioning control, MQTT protocol, OneNET platform, state machine scheduling

目 录

[第1章 绪论 1](#_Toc196948735)

[1.1 研究目的及意义 1](#_Toc196948736)

[1.2 国内外研究现状 1](#_Toc196948737)

[1.2.1 国外研究进展 1](#_Toc196948738)

[1.2.2 国内研究进展 2](#_Toc196948739)

[1.3 本文主要研究的内容 2](#_Toc196948740)

[第2章 系统总体设计方案 4](#_Toc196948741)

[第3章 系统硬件设计 6](#_Toc196948742)

[3.1 STM32微控制器 6](#_Toc196948743)

[3.2 温湿度模块 7](#_Toc196948744)

[3.3 通信模块 7](#_Toc196948745)

[3.4 液晶显示模块 7](#_Toc196948746)

[3.5 输入模块 8](#_Toc196948747)

[3.6 继电器模块 8](#_Toc196948748)

[第4章 系统软件设计 10](#_Toc196948749)

[4.1 软件架构与控制状态机设计 10](#_Toc196948750)

[4.1.1 软件模块化结构设计 10](#_Toc196948751)

[4.1.2 控制状态机的实现逻辑 11](#_Toc196948752)

[4.2 系统主程序设计 12](#_Toc196948753)

[4.3 DHT22 温湿度检测模块 13](#_Toc196948754)

[4.4 OLED屏幕实现方案 13](#_Toc196948755)

[4.5 WIFI通信实现方案 14](#_Toc196948756)

[4.6 MQTT通信模块设计 14](#_Toc196948757)

[4.6.1 设备属性上报机制设计 15](#_Toc196948758)

[4.6.2 指令下发与消息响应流程 15](#_Toc196948759)

[4.6.3 MQTT 消息解析与反馈机制设计 16](#_Toc196948760)

[4.7 自动模式实现方案 17](#_Toc196948761)

[4.8 手动模式实现方案 17](#_Toc196948762)

[4.9 远程模式实现方案 18](#_Toc196948763)

[4.10 安卓APP 19](#_Toc196948764)

[4.10.1 控制指令下发 19](#_Toc196948765)

[4.10.2 传感器信息显示 19](#_Toc196948766)

[第5章 系统调试 20](#_Toc196948767)

[5.1 模块功能测试 20](#_Toc196948768)

[5.2 仿真效果图 21](#_Toc196948769)

[5.3 实物效果图 22](#_Toc196948770)

[第6章 总结与展望 23](#_Toc196948771)

[6.1 总结 23](#_Toc196948772)

[6.2 展望 23](#_Toc196948773)

[参考文献 24](#_Toc196948774)

[致谢 25](#_Toc196948775)

[附录 26](#_Toc196948776)

[附录一 程序 26](#_Toc196948777)

[附录二 元件清单 36](#_Toc196948778)

# 绪论

## 研究目的及意义

近些年来，我国国内经济飞速发展，新办公楼宇数量逐年增加，同时楼宇建筑物越来越多地采用中央空调，使建筑能耗大幅度增加。而且经济发展所带来的能源短缺问题越来越严重，如何高效节约能源和提高能源利用效率已经成为一个亟待解决的公众问题，而中央空调系统能耗占建筑物总能耗的50%以上，有效地采取措施节省中央空调系统能耗是建筑物节能面临的当务之急[1]。

提升室内环境舒适性：传统的空调系统在室内温度调节方面存在一定的局限性，无法根据实际需求进行精确控制。智能空调系统的设计旨在通过实时监测和智能控制算法，能够根据室内环境参数和用户需求，自动调节空调运行状态，提供更加舒适的室内温度和湿度，改善用户的生活质量[2]。

节能和环保：空调系统是家庭和商业场所中耗能较高的设备之一。智能空调系统的设计旨在通过优化能耗，减少能源的浪费和碳排放，以达到节能和环保的目的。通过精确控制和智能调节，系统可以根据实际需求和环境状况，合理调整空调的运行，降低能源消耗，减少对环境的负面影响[3]。

实现远程监控和管理：智能空调系统的设计融入了互联网技术，可以实现远程监控和管理。用户可以通过智能手机或其他终端设备，随时随地对空调系统进行监测和控制。这种远程控制的便利性使用户能够灵活地管理室内环境，提高使用便利性和用户体验[4]。

探索智能家居技术的应用：智能空调系统作为智能家居系统的重要组成部分，具有广阔的应用前景。通过本论文的研究，可以探索智能家居技术在空调系统中的应用，为智能家居领域的研究和发展提供有价值的实践经验。

## 国内外研究现状

### 国外研究进展

在国外，智能空调控制系统的研究起步较早，技术成熟。较为典型的是美国、德国、日本和韩国等国家，它们通过将智能控制技术与空调系统深度融合，提升了设备的智能水平和能源利用效率。2019年，Lin LC等人在《Research on data transmission based on CC2530 of ZigBee》中指出，利用ZigBee协议转换器配合集中控制器，可实现空调系统的集中控制，并通过智能网关与监控主机组成统一的管理系统，不仅支持同一网关下的空调控制，还可实现跨网关的协调控制[5]。

同年，Ata E等学者在《ZigBee wireless sensor and control network》研究中提出了一种基于ZigBee协议的无线传感与控制系统，系统结构由VRV空调系统（包括室外主机、室内机、线控器等）、协议转换器、智能网关、集中控制器及监控主机等构成，展示了空调系统的高集成化和灵活性[6]。

此外，2022年Yu CT等人在《Wireless sensor networks for emergency navigation》中指出，在大型办公楼宇中，VRV多联空调系统相比传统的水冷螺杆式冷水机组，虽然初始投资较高，但在后期维护、能源效率和控制灵活性等方面具有明显优势，因此成为许多高端建筑空调设计的优选方案[7]。

### 国内研究进展

在国内，虽然起步稍晚，但随着智能建筑与绿色节能理念的推广，智能空调系统的研究也取得了显著进展。2021年，王念欧在《基于人流分布感知的智能空调控制系统设计》中提出将监控软件部署在云平台上，用户可通过PC、笔记本或平板终端进行远程管理，同时系统还能为第三方BMS系统提供空调控制接口服务，增强了系统的开放性与可扩展性[8]。

同年，马鹏宇等人则在《智能空调系统在办公楼宇中的应用》一文中介绍了温湿度控制电路的基本功能，即当环境温度或湿度超出设定范围时，系统将自动启动加热、加湿、制冷或除湿等功能模块，实现闭环控制。这类系统多结合嵌入式技术与传感器融合，实现了温湿度的动态感知与控制响应[9]。

2020年，汤斯茗在《基于ZigBee的智能空调控制系统的研究与设计》中设计了一套利用ZigBee无线通信实现的智能空调系统，通过对室内环境温湿度的实时采集、上下限值设定与显示，实现了系统对空调设备的智能调控。该系统具有通信稳定、部署灵活等优点，适用于多种复杂环境[10]。

此外，随着国产嵌入式平台（如STM32、瑞芯微RK系列）与云平台（如阿里云、OneNET）的普及，越来越多的研究开始关注如何实现本地边缘计算与云端远程管理的结合。例如，利用MQTT协议进行远程数据交互、使用移动APP或微信小程序进行空调控制，已经成为国内高校与企业智能空调系统设计的热点方向。

## 本文主要研究的内容

本系统以 STM32F103C8T6 微控制器为核心控制单元，集成了温湿度传感器、继电器驱动模块、OLED 显示模块及 WiFi 通信模块，通过嵌入式 C 语言完成系统控制逻辑的实现。系统使用 DHT22 传感器采集室内温湿度数据，STM32 在获取数据后根据预设的温湿度上下限进行判断，并控制风扇、压缩机或加热装置的启停，以实现空调设备的自动化环境调节功能。

在实现基础控制功能的基础上，系统设计了三种工作模式：自动控制模式、本地手动控制模式和远程控制模式。自动模式下，系统根据传感器数据自动判断是否开启空调设备，实现无人值守的智能调节；手动模式通过物理按键由用户直接控制空调的开关状态；远程控制模式通过 WiFi 模块连接至中国移动 OneNET 平台，接收用户从移动端或网页端下发的 MQTT 控制命令。系统在接收到 JSON 格式的控制指令后，解析出控制内容，并执行相应的控制操作。同时，系统也会通过 MQTT 协议周期性地向云平台上报当前温湿度数据及设备运行状态，实现远程监控与管理功能。

为了提升本地用户的交互体验，系统集成 OLED 显示模块，用于实时显示温湿度数值、当前控制模式以及设备运行状态，提供直观的界面反馈。软件架构方面，系统采用模块化设计思路，将传感器采集、设备控制、通信处理、状态上报和指令解析等功能分别封装为独立模块，使得代码结构清晰、逻辑分明，便于调试和后续维护。在程序调度方面，主循环结合状态机机制对系统各个功能模块进行合理调控，保证了控制逻辑的实时性和系统运行的稳定性。

# 系统总体设计方案

本次设计的智能空调调节系统以STM32F103C8T6单片机作为主控核心，依据其强大的数据处理能力、丰富的外设资源以及低功耗特性，构建了一个具备高度集成性与良好扩展性的智能控制平台。系统遵循模块化设计思想，结合嵌入式控制技术与物联网通信技术，整体采用感知——决策——执行——通信——交互的五层功能架构，不仅提升了程序的可维护性与可扩展性，也为后续的功能拓展与系统升级提供了坚实基础。系统通过软硬件的高效协同，旨在实现空调设备的智能调节、远程控制与环境状态可视化功能。

在硬件结构方面，系统主要由温湿度传感器、继电器驱动模块、OLED显示模块、按键输入模块和WIFI通信模块构成，各模块围绕STM32核心主控单元有机协调、协同运行。DHT22温湿度传感器用于感知室内环境变化，并将数据实时传送至STM32进行处理；继电器模块受STM32控制，实现对空调制冷、加热、加湿和通风设备的开关控制；OLED模块提供运行信息本地显示界面，用户可以直接查看温湿度数据、工作模式与设备状态；WIFI模块作为通信接口，与中国移动OneNET平台建立连接，实现MQTT协议下的远程控制与设备状态回传；按键输入模块支持用户在本地无网络环境下对空调进行手动控制，提高系统的适用性和安全性。

系统的软件架构同样基于模块化构建，包含初始化模块、数据采集模块、判断控制模块、通信交互模块、状态反馈模块与用户输入处理模块等，每个模块功能独立，协作有序。在功能实现方面，系统设计了三种工作模式：自动模式、本地控制模式与远程控制模式。其中，自动模式下，系统根据设定的温湿度阈值实现自主判断与设备联动控制，满足无人值守场景需求；本地模式允许用户通过按键手动控制继电器状态，保障在通信异常或调试过程中的操作灵活性；远程控制模式依托MQTT通信机制，用户可通过手机或Web网页向OneNET平台发布指令，设备在接收到下发命令后解析JSON数据并执行对应控制操作，同时将运行状态与温湿度信息通过属性上报机制反馈至云端，构建完整的信息交互闭环。

此外，系统主程序基于状态机机制结合定时器调度方式构建，确保多任务环境下各模块高效协作和系统的实时性。在运行过程中，主控芯片定期采集环境数据，判断当前状态与设定阈值的关系，选择合适的控制策略并驱动相应模块动作，同时同步完成数据上传与本地显示，整体运行流程清晰、响应及时。系统整体设计框图如图 2.1所示，展示了各模块之间的逻辑关系与功能协作路径，直观体现出本系统从数据感知、逻辑判断、动作控制到远程通信的完整闭环结构。

综上所述，本设计以STM32为核心，构建了一个集环境监测、智能控制、远程通信与人机交互于一体的智能空调调节系统。系统设计合理、功能完整，能够有效提升空调系统的智能化程度与能效管理水平，适用于住宅、办公室、实验室等多种对温湿度环境有较高要求的场所，具有良好的工程实用价值与推广潜力。

图 2.1 系统总体设计框图

# 系统硬件设计

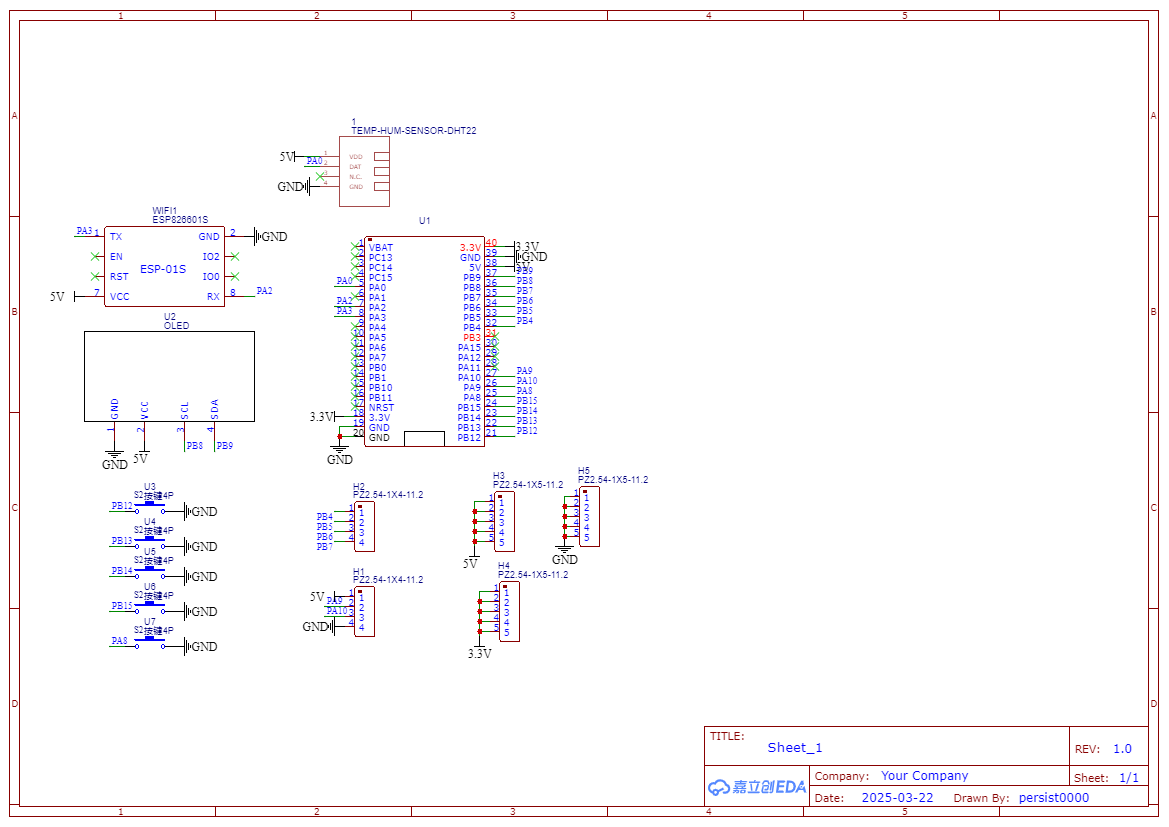
本系统的硬件部分以STM32F103C8T6单片机为核心控制器，配合温湿度检测模块、继电器执行模块、WIFI通信模块、OLED液晶显示模块及按键输入模块，共同构建起完整的智能空调调节平台。各功能模块围绕主控芯片分工明确、相互协作，完成环境感知、状态判断、控制执行、远程通信与信息交互等核心功能，系统硬件结构如图 3.1所示。以下对主要硬件模块进行详细介绍。

图 3.1 系统硬件结构图

## STM32微控制器

如图 3.2所示本系统选用的STM32F103C8T6单片机基于ARM Cortex-M3内核，具有72MHz主频、64KB Flash和20KB SRAM，具备强大的运算能力和丰富的外设资源。其集成多个USART、SPI、I2C、ADC和定时器等接口，满足多种外设接入与数据处理的需求。在本系统中，STM32负责完成环境数据采集、逻辑判断、继电器驱动控制、OLED数据显示、通信数据处理等任务，是整个系统的核心控制单元。

图 3.2 STM32F103C8T6单片机

## 温湿度模块

如图 3.3所示为了实现对环境参数的实时监测，系统采用了DHT22温湿度传感器。该传感器内部集成温度与湿度采集功能，具备输出稳定、响应速度快、性价比高等优点。其通过单总线方式与STM32连接，周期性采集室内温湿度信息，并作为自动控制逻辑的关键判断依据。STM32主控定时读取数据，通过判断与设定阈值的关系，决定是否进行制冷、加热或通风等操作。

图 3.3 DHT22温湿度传感器

## 通信模块

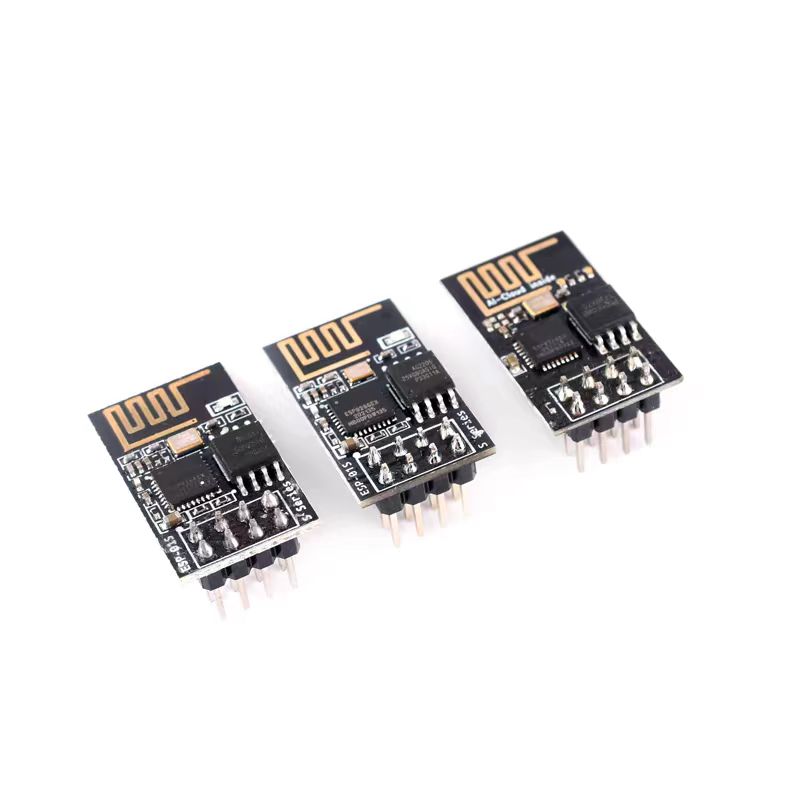
如图 3.4所示系统使用ESP8266 WiFi通信模块，与中国移动OneNET平台建立MQTT通信连接，实现远程控制与状态反馈功能。该模块通过串口与STM32进行数据交互，主控芯片接收平台下发的JSON格式控制指令，并据此控制继电器动作，同时将当前设备状态与环境信息以JSON格式打包后上报至平台，实现智能空调的远程管理和双向交互功能。通信模块的引入使得系统具备良好的扩展性和物联网接入能力。

图 3.4 WIFI模块

## 液晶显示模块

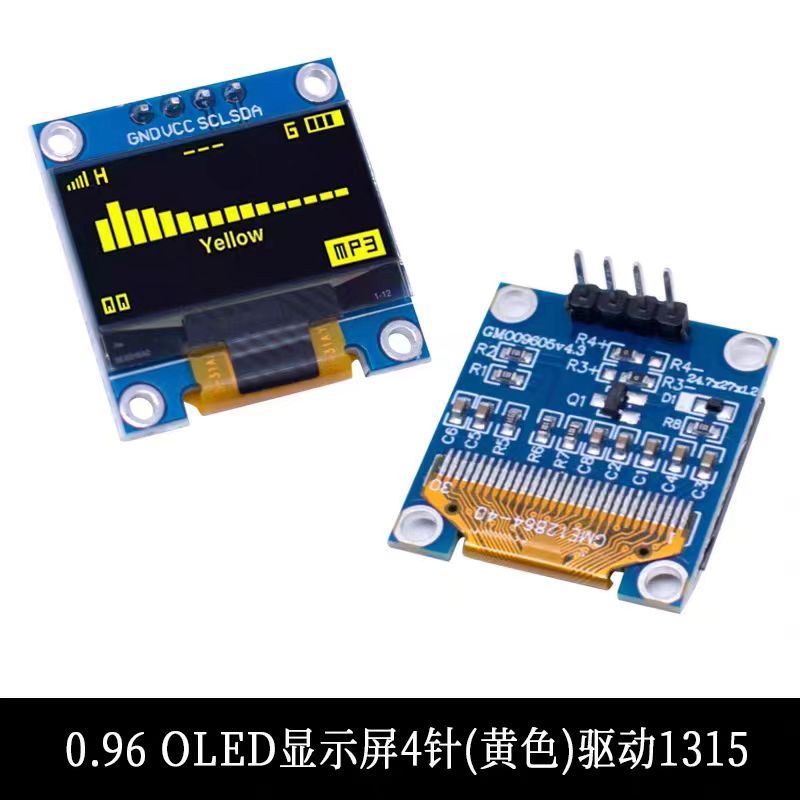
如图 3.5所示为实现用户对系统当前运行状态的直观掌握，系统配备了一块0.96英寸的OLED液晶屏，采用I2C通信方式与STM32连接，具有功耗低、显示清晰的特点。显示内容包括当前温湿度数值、工作模式（自动/本地/远程）、空调运行状态（制冷/加热/关闭）等。通过动态更新机制，用户可实时获知系统运行状况，提升人机交互体验。

图 3.5 OLED显示模块

## 输入模块

手机屏幕截图

AI 生成的内容可能不正确。如图 3.6所示本系统设置了按键输入模块，用于支持本地控制与功能切换。在本地控制模式下，用户可通过按键实现继电器开关控制，实现对空调设备的直接操作。同时，按键还可用于模式切换、系统复位等基本功能。该模块通过GPIO与STM32相连，结合中断机制或定时扫描方式实现可靠识别与响应，提高系统的灵活性与可控性。

图 3.6 按键输入模块

## 继电器模块

如图 3.7所示继电器模块是系统的执行单元，主要负责空调设备的电源控制。系统采用低电平触发的继电器模块，控制端由STM32输出GPIO信号驱动，当继电器闭合时，电路接通，空调设备开始运行。通过多通道继电器的控制，可以灵活地实现对制冷、加热、风扇等不同功能单元的独立管理。该模块具备负载能力

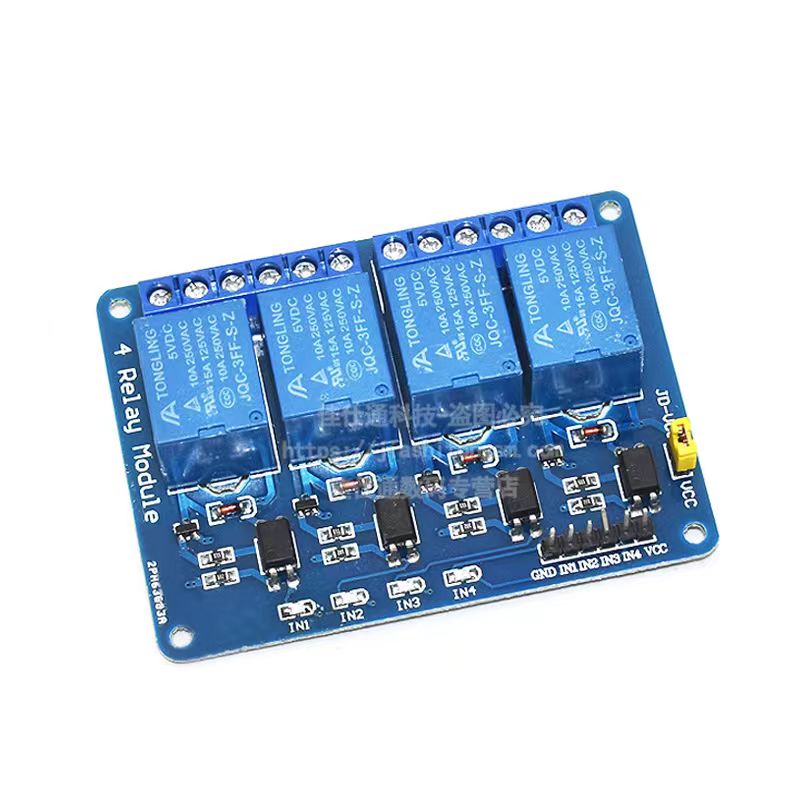
强、抗干扰性好等特点，确保系统运行的安全性与稳定性。

图 3.7 继电器模块

# 系统软件设计

## 软件架构与控制状态机设计

### 软件模块化结构设计

本智能空调控制系统的软件设计遵循模块化结构原则，将整个系统划分为若干功能独立的逻辑单元，每个模块负责处理特定的功能任务。这种设计不仅能够增强代码的可读性和可维护性，同时也便于系统功能的扩展和后期升级。系统主要包括传感器采集模块、控制判断模块、信息显示模块、通信传输模块以及用户交互模块，各模块协同运行，构成了一个完整的智能控制闭环。

在传感器采集模块中，系统采用 DHT22 数字温湿度传感器作为环境数据采集元件，通过单总线方式与 STM32F103C8T6 主控芯片通信，实时采集当前环境的温度和湿度数据。采集的数据经处理后，将被传递至控制判断模块用于后续决策。控制判断模块是系统的核心，包含多个工作模式的判定逻辑，包括自动控制、手动控制与远程控制等。该模块根据温湿度数据变化、用户按键操作或远程命令做出响应，控制继电器的开合状态，从而实现对空调设备的启停控制。所有控制逻辑均采用 C 语言在裸机环境下实现，无需操作系统支持，具有运行效率高、响应速度快的特点[11]。

显示模块负责将系统的实时运行状态直观地呈现给用户。该模块采用 OLED 显示屏作为输出设备，通过 I2C 总线与主控芯片连接，能够动态显示当前的温湿度值、空调运行状态、控制模式等关键信息。通过简洁的界面设计，用户可以快速掌握系统的当前状态，提升使用便捷性。

通信模块实现了系统与外部网络的连接功能。本系统通过 ESP8266 WiFi 模块与 OneNET 云平台建立 MQTT 通信连接，实现设备运行状态的实时上报与平台控制命令的接收。通信协议采用轻量级的 MQTT，适用于资源受限的嵌入式系统。STM32 通过串口与 ESP8266 进行数据交互，通信模块在接收到平台下发的 JSON 格式指令后，将其解析并传入控制模块执行相应操作，从而实现远程控制能力[12]。

用户交互模块则为本地操作提供接口。系统设计了按键输入功能，允许用户在本地切换工作模式或直接控制空调的启停状态，即使在断网或平台不可用的情况下，也能保证系统的基本控制功能。该模块通过对按键状态的扫描识别，实现不同操作指令的判断与响应，增强了系统的实用性与鲁棒性。

整个系统的软件架构以模块之间的函数调用为基本交互手段，采用顺序执行和状态判断的方式调度各模块逻辑。由于不依赖 RTOS 或消息队列机制，系统架构更加紧凑高效，适合资源受限的 STM32 单片机平台。模块间通过结构体、标志变量及函数封装进行耦合与数据传递，保持了良好的层次结构与功能分离，便于调试和后续优化。通过上述设计，本系统实现了从传感器数据采集到智能判断、控制执行、信息展示、远程通信和用户交互等多环节的完整流程，体现了嵌入式系统在智能家居场景下的应用优势。

### 控制状态机的实现逻辑

为了实现智能空调系统的多模式控制功能，本系统采用了基于状态机的控制逻辑来管理不同的运行模式。状态机的引入使得系统在面对不同的输入源（如传感器数据、按键输入、远程指令）时，能够具备清晰、稳定且可维护的运行状态切换机制，从而确保系统在各类情境下均能做出准确响应。

整个系统在上电启动后，首先进入初始状态，此时系统处于空闲或待命模式，主要完成初始化配置，包括硬件接口初始化、OLED 显示模块启动、WiFi 模块连接检测、MQTT 客户端配置等操作，并持续等待来自传感器或用户的输入事件。一旦系统完成初始化，即开始进入具体的控制状态。控制状态主要分为三类，分别是自动控制模式、手动控制模式以及远程控制模式，三者之间通过条件判断进行切换。

在自动控制模式下，系统定期读取由 DHT22 传感器采集的环境温湿度数据，并与设定的阈值进行比较。当温度或湿度超过设定范围，例如环境温度高于28°C，系统即自动发出控制信号驱动继电器，启动空调设备；而当环境条件回落至设定范围内，系统则会自动关闭空调，从而实现动态调节与节能控制的目的。该模式下，系统主要依赖传感器数据变化作为触发条件，并根据环境状态自主进行控制决策，具有无需人工干预的特点，适用于无人值守的运行场景。

手动控制模式则通过用户在本地按键的操作进行触发。当用户通过按钮输入请求，例如按下“开机”或“切换模式”键时，系统中断当前状态并切换至手动控制状态。在该模式下，用户拥有对空调设备直接控制的优先权，不再依赖传感器数据进行判断。系统将执行用户意图所指示的动作，例如强制开启或关闭空调。此模式特别适用于用户希望临时调整空调状态、绕过自动逻辑的场景，例如环境变化尚未达到自动触发条件但用户已感不适时，可通过此模式快速响应需求。

远程控制模式的进入则依赖于 OneNET 云平台下发的控制命令。当系统通过 MQTT 协议接收到平台发送的远程控制指令后，会立即对消息进行解析，并据此调整空调设备状态，例如执行远程开关指令或模式切换命令。该模式通常用于用户通过移动端 APP 进行远程操控时的场景，实现了用户在异地情况下对空调运行状态的远程干预能力，提升了系统的智能化水平与实际应用的便利性。

上述三种控制模式之间的切换机制基于事件驱动和状态判断相结合的方式进行实现。当系统检测到传感器数据超过阈值时，优先进入自动控制状态；当用户按键输入发生时，将打断当前控制流程并转入手动控制状态；而一旦接收到远程 MQTT 控制命令，则系统立即切换至远程控制状态并执行相应命令。在状态切换过程中，系统还会更新 OLED 显示模块的内容，以实时反映当前的控制模式和执行状态，增强用户的可视化体验。

通过上述状态机的设计，系统实现了模式之间的稳定切换与互不干扰，使得智能空调系统在面对复杂环境变化和多源输入时，仍能保持清晰、可靠的运行逻辑，充分体现出嵌入式系统在智能家居场景中的适应性和实用价值。

## 系统主程序设计

本系统以 STM32F103 为核心控制器，主程序围绕硬件初始化、网络连接、数据采集与上报、界面交互及智能控制五大功能模块展开，采用模块化设计思想，通过主循环结构实现对各个子系统的协调管理与逻辑调度。

系统启动后，首先完成各类外设资源的初始化，包括定时器配置、串口通信通道建立、显示模块准备以及控制逻辑的初始状态设置。其中，OLED 屏幕在系统初始化阶段承担状态提示作用，为用户提供硬件加载成功与否的可视化反馈。此外，系统还预设了本地按键交互功能，便于手动模式切换与状态调试。

硬件准备完成后，系统进入网络连接阶段。通过无线通信模块与云平台建立 MQTT 协议连接，并在连接成功后完成设备注册与主题订阅。整个过程由 OLED 屏幕同步显示连接进度，确保网络初始化过程可控、透明。在建立通信通道后，系统具备了从平台接收指令并回传数据的能力。

进入主循环后，系统按照设定周期不断采集环境温湿度数据，并进行本地显示和远程上报。同时，根据用户当前所选控制模式，系统能够在自动与手动之间进行灵活切换：自动模式下，系统根据传感器数据动态调节设备运行状态，提升智能响应能力；手动模式则允许用户通过按键或远程指令直接干预设备状态，增强交互灵活性。

在保持数据上传与设备控制的同时，系统还持续监听来自云平台的指令信息，确保下发命令能够被及时解析并响应执行。为提升系统运行稳定性，每轮循环均对通信缓存进行清理，避免串口数据拥堵对系统造成影响。整体程序运行逻辑清晰、任务划分明确，能够稳定支持多模块并行协作的系统运行需求。

## DHT22 温湿度检测模块

本系统选用 DHT22 作为环境温湿度检测传感器，其具备高精度、数字信号输出、响应速度快等特点，适用于对环境状态变化要求较高的智能控制系统中。在本设计中，DHT22 的数据引脚通过单总线方式连接至 STM32 的 GPIO 端口，采用轮询方式定时采集当前环境的温度与湿度数据，为系统自动控制与远程状态上报提供基础支撑。

传感器与主控之间的通信过程遵循 DHT22 官方通信时序，包括起始信号发送、设备应答、数据传输及校验确认四个阶段。主控 MCU 首先通过输出低电平启动信号请求数据传输，DHT22 响应后发送共 40 位的传输数据，其中包含 16 位湿度信息、16 位温度信息以及 8 位校验码。采集完成后，系统将根据接收数据进行校验判断，以确认通信过程的可靠性。

在程序实现上，DHT22 驱动模块以结构体封装形式返回湿度和温度的整数及小数部分，同时计算并存储浮点数值便于后续处理与显示。数据采集频率可由主循环中的时间计数器控制，通常设定为每 2 秒一次。为了避免因硬件时序误差导致的数据不稳定问题，系统通过精确延时函数对信号变化过程进行微秒级控制，确保通信过程的时效性与准确性。

采集到的温湿度数据除了在本地 OLED 屏幕进行实时显示，还会通过 MQTT 协议上传至 OneNET 云平台，实现远程环境监控。在自动控制模式下，系统还将实时根据温度变化动态调整设备运行状态，从而达到智能控制与节能优化的目的。

## OLED屏幕实现方案

OLED屏幕作为本项目的显示终端，主要用于显示设备的状态信息、传感器数据以及控制界面。屏幕通过I2C协议与STM32连接，简化了硬件布线。OLED的初始化主要包括对显示控制寄存器的设置，如显示模式、对比度以及电荷泵等，以确保屏幕能够正常工作。通过I2C通信，STM32向OLED发送相应的命令和数据，实时更新显示内容。显示内容的管理使用显存进行存储，每个点阵的显示状态由相应的内存地址表示，更新时通过I2C协议将显存数据发送至OLED屏幕。显示内容包括字符、图形以及动态效果，支持自定义图像的加载。为了提高系统性能，屏幕刷新和更新仅在必要时进行，避免了不必要的I2C通信和处理。总的来说，OLED屏幕通过高效的控制方式和合理的显示更新策略，实现了设备状态和传感器数据的清晰显示，增强了用户的交互体验。

## WIFI通信实现方案

在本设计中，使用ESP8266模块实现与外部网络的WIFI通信功能。ESP8266模块通过串口与STM32微控制器进行通信，主要实现设备的网络连接和数据发送功能。初始化过程中，首先通过AT命令与ESP8266模块进行通信，设置模块工作模式为客户端模式，并连接指定的WIFI网络。具体的AT命令包括设置工作模式（AT+CWMODE=1）、获取IP地址（AT+CWJAP）等。

ESP8266模块通过串口接收数据并处理，串口通信的配置通过USART2进行，USART中断用于接收ESP8266返回的数据。数据通过USART发送至ESP8266模块，模块再将数据通过WIFI发送至网络。为了确保通信的可靠性，使用了命令响应机制，即在发送每一条命令后，系统会等待ESP8266的响应，确认命令是否执行成功。如果响应结果符合预期（如返回“OK”或“GOT IP”），则继续进行下一步操作。若响应不符，则重试发送命令，直到成功为止[13]。

为了更好的调试和显示设备状态，系统将ESP8266的状态信息通过OLED显示屏进行实时反馈，帮助用户监控设备的连接状态。ESP8266的通信数据接收部分通过缓冲区进行存储，确保接收的数据不会丢失，并通过解析得到有效的数据包。对于收到的数据，通过解析获取相应的指令或消息，进一步进行处理。

该实现方案为设备的WIFI通信提供了稳定的基础，确保设备可以通过网络与外界进行数据交换，支持设备状态的远程监控和控制。

## MQTT通信模块设计

MQTT通信模块设计是系统与OneNET云平台进行数据交互的核心。此模块实现了设备的状态上报、指令接收与响应、以及设备控制命令的处理。整个模块采用了原生的MQTT协议，通过自定义的MQTT库来处理与云平台的连接、消息的发布与订阅。设计重点是确保设备能够稳定地与平台进行数据传输，同时保证数据交互的实时性和可靠性[14]。

该模块通过TCP连接与OneNET平台建立通信通道，采用主题发布/订阅模式进行数据交换。设备通过定期向平台报告状态信息，同时订阅平台下发的控制指令，并根据指令修改设备状态。通过线程机制和消息队列的设计，确保了设备状态的及时更新与响应，使得设备能在最短的时间内与平台同步。

### 设备属性上报机制设计

设备属性上报是MQTT通信模块的核心功能之一，主要用于将设备的状态信息上传到OneNET平台。设备在运行过程中，会定期采集自身状态并通过MQTT协议发布消息。上报的主题采用了标准的格式，例如“$sys/{pid}/{device-name}/thing/property/set”，并通过JSON格式的数据传输设备的属性数据。

设计时，系统采用了链表结构来管理设备和其属性，保证不同类型的设备可以根据自身特性上报不同的属性信息。设备的状态信息通过状态机进行管理，设备每当状态发生变化时，会立即触发上报机制。上报的内容包括设备的基本状态（如开关、温湿度等）、设备运行数据以及其他重要参数。这一机制确保了平台始终能够接收到设备的最新状态，进而实现实时监控和控制。

### 指令下发与消息响应流程

在智能空调控制系统的通信模块设计中，指令下发与消息响应流程是 MQTT 通信机制的核心功能之一，承担着设备远程控制和状态反馈的双重任务。该功能模块确保了 OneNET 云平台与终端设备之间的实时信息交互，是实现远程智能控制不可或缺的关键环节。

为了实现云平台向终端设备的精准控制，系统在初始化阶段便完成对 MQTT 控制主题 $sys/{product\_id}/{device\_name}/thing/property/set 的订阅。一旦平台向该主题发布控制消息，系统内部 MQTT 客户端立即接收到该消息，并触发对应的回调处理函数。接收到的 MQTT 消息一般采用 JSON 格式封装，内容中包含了目标设备的控制命令及相关参数，例如开关指令、温湿度设定值等。系统使用内置的 cJSON 库对该消息进行解析，提取出具体的控制字段，并将其映射到系统内部的控制接口。

在指令解析完成后，系统根据指令类型进入不同的执行流程。例如，当收到“开启空调”指令时，控制模块会立即驱动继电器吸合，完成空调设备的通电操作；若指令为“设定温度阈值”，则系统将更新自动控制逻辑中使用的阈值参数。整个控制过程基于事先设定的状态机机制，能够自动判断当前工作模式，并决定是否允许执行某些类型的操作，从而保证系统运行的安全性和逻辑一致性。

完成操作后，系统将执行结果打包成一条 MQTT 响应消息并发布至平台指定的状态上报主题，如 $sys/{product\_id}/{device\_name}/thing/property/post，该消息中包含设备当前的工作状态、操作执行结果、以及可能出现的异常或错误提示。这种双向通信机制确保了平台在发送指令后能够获得明确反馈，不仅增强了平台对设备的管控能力，也为系统的后续数据分析、异常预警和远程诊断提供了可靠依据。

为进一步提升系统的并发处理能力和指令响应效率，指令处理流程采用了同步与异步相结合的设计思路。对于简单、实时性强的控制指令，如开关控制等，系统优先采用同步执行方式，确保指令在最短时间内生效；而对于复杂的设置操作或需要状态确认的流程，系统则使用异步处理机制，将指令执行任务挂起到后台执行，待完成后再统一返回结果。这种策略有效避免了主控线程被长时间阻塞的问题，提升了系统整体响应速度和稳定性。

尽管系统整体运行在 STM32F103C8T6 微控制器上，资源受限，但仍通过合理的结构设计与任务分配，实现了对远程控制命令的稳定接收与快速响应。在实际测试中，系统在高频率指令下发场景中表现良好，能够连续无误地处理来自 OneNET 平台的多条控制指令，证明了该指令下发与响应机制的可靠性

与实用性。该机制的实现不仅保障了用户的远程控制体验，也为系统后续向多设备协同、云端批量管理等复杂场景扩展提供了技术基础。

### MQTT 消息解析与反馈机制设计

在本智能空调控制系统中，MQTT 协议被选为核心通信协议，是实现设备与 OneNET 云平台之间双向数据交互的关键技术。MQTT 具有轻量级、发布/订阅模型、低带宽占用等特点，非常适合应用于资源受限的嵌入式系统中。系统的主要通信逻辑围绕设备属性的上报与平台指令的下发展开，确保云端能够准确掌握设备的运行状态，同时也能实现远程控制指令的快速响应。

系统运行过程中，首先会通过 MQTT 协议建立与 OneNET 云平台的连接，并在初始化阶段完成指定主题的订阅。具体而言，系统订阅了 $sys/{product\_id}/{device\_name}/thing/property/set 这一标准控制主题，用于接收平台下发的设备属性控制指令。一旦平台向该主题发布控制命令，系统会立即收到 MQTT 消息并进入解析流程。为了解析这些包含控制信息的 JSON 格式数据，系统内部集成了 cJSON 库，通过调用相关函数动态提取出消息中的关键词字段，例如控制对象（如“空调”）、操作类型（如“开”或“关”）、目标参数（如温度设定值）等。解析完成后，系统依据解析结果执行相应的硬件控制操作。例如，当平台发出“开启空调”命令时，系统将通过 GPIO 输出高电平信号驱动继电器吸合，从而实现空调的电源接通；如果指令为“关闭空调”，则系统会执行反向操作。

为了提高系统的交互性与可管控性，系统还构建了一套设备状态的上报机制。每当空调的工作状态发生变化，或周期性地经过设定的时间间隔后，系统便会主动收集包括当前温湿度值、继电器开关状态等信息，并使用 cJSON 将这些状态信息封装为符合 OneNET 协议要求的 JSON 数据。随后，这些数据会通过 MQTT 协议发布至 $sys/{product\_id}/{device\_name}/thing/property/post 主题，实现设备状态的主动上报。这不仅有助于用户在手机客户端或 Web 平台上实时查看设备状态，也为后续的设备数据分析与智能化优化提供了基础数据支撑。

## 自动模式实现方案

自动模式是本系统智能控制逻辑的核心组成部分，旨在实现基于环境温湿度的自主调节功能。在该模式下，系统无需用户干预即可持续监测室内温度与湿度，并根据预先设定的环境参数阈值自动控制相应的设备运行状态，实现真正意义上的无人值守智能控制。

具体而言，系统周期性地读取 DHT22 传感器采集到的环境温湿度数据，并将当前数值与设定的上下限进行比较。当温度高于预设上限时，系统自动开启制冷装置；当温度低于下限时，启动加热设备；湿度高于设定范围则触发通风系统以降低湿度，而湿度过低则通过控制加湿设备恢复环境湿度至合理区间。所有控制指令均通过 STM32 控制继电器实现执行，确保对各类空调设备的精准启停控制。

系统支持用户通过物理按键实现手动与自动模式的切换，模式切换后，OLED 显示模块将实时更新界面内容，显示当前工作模式和关键状态参数，提升用户对系统运行状态的感知能力。在自动模式下，系统各控制环节均处于闭环状态，即采集—判断—控制—反馈形成循环，具有较强的实时性和自适应性。

## 手动模式实现方案

手动模式作为系统的辅助控制方式，为用户提供了直接干预设备运行的能力，尤其适用于用户需对设备进行临时调节或调试的场景。在该模式下，用户可通过系统前面板的物理按键对空调系统中的子模块进行独立控制，包括制冷、加热、通风及加湿等功能，每个按键均对应一个特定的控制功能。

系统在初始化阶段即完成按键与继电器的 GPIO 引脚配置，将对应的引脚设为上拉输入，并启用外部中断功能，以便于及时捕捉用户的按键操作。按键一旦被按下，STM32 检测到电平变化即触发中断服务函数，执行对应的控制逻辑，控制目标设备通过继电器通断实现开关操作。同时，为提升用户交互体验，系统配套使用 LED 指示灯，反映当前设备的运行状态，确保用户在操作过程中的可见性与可控性。

为了避免按键抖动或误触引发的重复响应，系统在中断响应中加入软件消抖机制，通过短时间延时判断按键稳定状态，进一步增强了系统的操作可靠性。在每次按键触发后，系统还会实时更新内部状态记录，以便后续在切换模式或状态上报时保持一致性。

## 远程模式实现方案

远程控制模式是本系统实现智能化与信息化的重要体现，借助 WIFI 通信模块与中国移动 OneNET 云平台之间的 MQTT 协议通信机制，实现用户通过移动端或网页端对空调系统的远程控制与状态监测。在该模式下，用户无需接触设备本体即可实现设备的开关控制、参数调节及运行状态的实时查看，极大地拓展了系统的使用场景与操作便利性。

系统在初始化过程中，首先配置 ESP8266 模块连接本地 WiFi 网络，并完成与 OneNET 平台的 MQTT 客户端连接。连接成功后，系统订阅平台下发指令的专用主题 $sys/{ProductID}/{DeviceName}/thing/property/set，实时监听来自云端的控制命令。当用户通过手机或网页端发布控制指令后，消息经由 MQTT 通道传输至设备端，STM32 接收到消息后，使用 cJSON 库对其 JSON 格式内容进行解析，从中提取控制目标、操作类型与参数值等关键信息。

根据解析结果，STM32 控制对应的继电器输出高低电平，实现如打开制冷设备、关闭加湿器等实际操作。同时，系统会在操作完成后将当前环境温湿度数据、控制模式、设备运行状态等信息重新封装为 JSON 格式的数据包，并通过 $sys/{ProductID}/{DeviceName}/thing/event/property/post 主题向云平台进行状态上报，确保用户端可以获取准确的反馈信息，实现完整的闭环控制。

在远程模式下，系统既保留了本地 OLED 实时显示能力，也兼容用户通过 OneNET 平台远程查看状态信息，增强了系统的多端交互能力。此外，为提高通信可靠性与响应效率，系统设计了接收超时检测、断线重连机制与命令缓存处理策略，确保在网络波动等复杂环境下仍能保持稳定运行。

综上所述，远程控制模式不仅提升了用户对设备的控制自由度，也体现了物联网技术在传统家电控制系统中的深度融合与应用价值，是现代智能家居系统不可或缺的关键模块。

## 安卓APP

本系统配套开发了一款基于 UniApp 框架的 Android 智能家居控制 APP，主要实现对空调系统的远程控制、温湿度信息的实时查看以及设备状态反馈等功能。APP 通过调用中国移动 OneNET 物联网平台提供的 REST 接口，与部署在终端的控制系统进行交互，实现远程智能化管理[15]。

### 控制指令下发

在控制方面，APP 提供了多个按钮与开关控件，用于控制空调子系统中如制冷、加热、加湿与通风等功能模块。用户在 Android 手机端点击某一控件后，APP 会立即封装控制参数，以 POST 方法向 OneNET 提供的设备属性设置接口发送 JSON 数据，包含 product\_id、device\_name 与所需控制字段，同时携带有效的 token 进行身份认证。OneNET 在接收到指令后会将其通过 MQTT 协议转发至终端设备，终端解析并执行对应的控制任务。

该流程响应迅速、结构清晰，用户每一次操作都能稳定传达至设备端，有效提升了控制的实时性与准确性。

### 传感器信息显示

传感器数据显示部分采用定时轮询机制，APP 每隔 2 秒通过 GET 请求主动查询 OneNET 平台中对应设备的属性状态。返回的 JSON 数据中包含当前温湿度、风扇运行状态、加湿状态等信息，系统通过解析这些字段后，实时更新界面中相应的数值和图标显示模块。

这种方式保证了用户可以在手机端随时了解空调系统的运行状态和室内环境情况，同时为进一步的操作提供参考。即使在网络不稳定的情况下，该机制也能保持较好的刷新与反馈体验，提升系统的可用性和用户体验。

# 系统调试

## 模块功能测试

为验证智能空调调节系统中各功能模块能否稳定、准确地完成预期控制任务，本文分别对环境数据采集、继电器控制、自动调节逻辑、MQTT远程通信及OLED状态显示等功能进行了详细测试。测试方法采用人工模拟环境变化、本地手动触发与云端远程控制相结合，评估各模块响应时间、控制准确率与系统联动效果。测试结果如下所示：

1. 环境温湿度采集模块测试

本模块采用DHT22传感器周期性采集环境温湿度，并在OLED屏显示。测试中通过模拟不同温湿度环境，记录传感器响应时间与读取准确率，结果如表 5.1所示：

表 5.1 温湿度采集模块测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试环境** | **温度读取值** | **湿度读取值** | **是否稳定** | **采集周期（ms）** |
| 正常室温（25℃/45%） | 25℃ | 45% | 是 | 1000 |
| 高温高湿环境 | 36℃ | 70% | 是 | 1000 |
| 低温干燥环境 | 12℃ | 25% | 是 | 1000 |
| **测试环境** | **温度读取值** | **湿度读取值** | **是否稳定** | **采集周期（ms）** |
| 正常室温（25℃/45%） | 25℃ | 45% | 是 | 1000 |

1. 自动调节与继电器控制模块测试

继电器控制风扇、制冷和加湿设备，系统根据设定阈值自动切换设备状态。测试中手动设置不同温湿度值观察自动调节响应情况。测试结果如表 5.2所示：

表 5.2 自动调节功能测试结果

|  |  |  |  |
| --- | --- | --- | --- |
| **测试条件（温/湿）** | **控制动作** | **继电器响应时间（ms）** | **是否正确执行** |
| 30℃ / 40% | 打开风扇 | 250 | 是 |
| 18℃ / 60% | 关闭风扇 | 240 | 是 |
| 28℃ / 20% | 打开加湿器 | 260 | 是 |
| 22℃ / 50% | 所有设备关闭 | 230 | 是 |

1. OLED显示模块测试

OLED用于显示当前温湿度、设备状态与控制模式。测试在状态变化时观察OLED是否及时更新，结果如表 5.3所示：

表 5.3 OLED显示功能测试结果

|  |  |  |
| --- | --- | --- |
| **状态变化触发事件** | **显示刷新是否及时** | **显示内容是否正确** |
| 空调由关闭变为开启 | 是 | 是 |
| 加湿器启动 | 是 | 是 |
| 模式切换为手动控制 | 是 | 是 |
| 温湿度变化 | 是 | 是 |

**MQTT远程控制模块测试**

系统通过ESP8266模块连接OneNET平台，接受远程控制指令并反馈状态。测试通过MQTT.fx发送命令并测量系统响应情况，结果如表 5.4所示：

表 5.4 MQTT远程控制模块测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试项目** | **测试次数** | **成功次数** | **成功率** | **平均响应时间（ms）** | **备注** |
| 接收远程开关空调指令 | 20 | 20 | 100% | 300 | 控制与本地一致 |
| 状态上报至云端 | 20 | 20 | 100% | 250 | 支持主动与被动上报 |
| 远程切换自动/手动模式 | 20 | 19 | 95% | 320 | 一次丢包重发后成功 |

## 仿真效果图

为验证智能空调调节系统在实际硬件部署前的功能完整性与逻辑正确性，本设计在 Proteus 仿真环境中对主要模块进行了电路级仿真测试。仿真模型包括 STM32F103C8T6 主控芯片、DHT22 温湿度传感器、电机与LED灯模拟负载、电源模块以及 OLED 显示模块。通过按钮输入模拟用户控制行为，系统可在设定温湿度阈值下自动切换设备状态，并实时更新 OLED 显示内容。

仿真过程中，电机驱动逻辑正常，LED灯的状态切换与 DHT22 数据变化保持一致。仿真还验证了电源连接、外围器件时序、显示刷新等细节逻辑的合理性。仿真结果显示系统在虚拟环境下功能稳定，各模块运行状态与实际预期一致，为后续实物部署提供了理论依据与电路正确性验证。

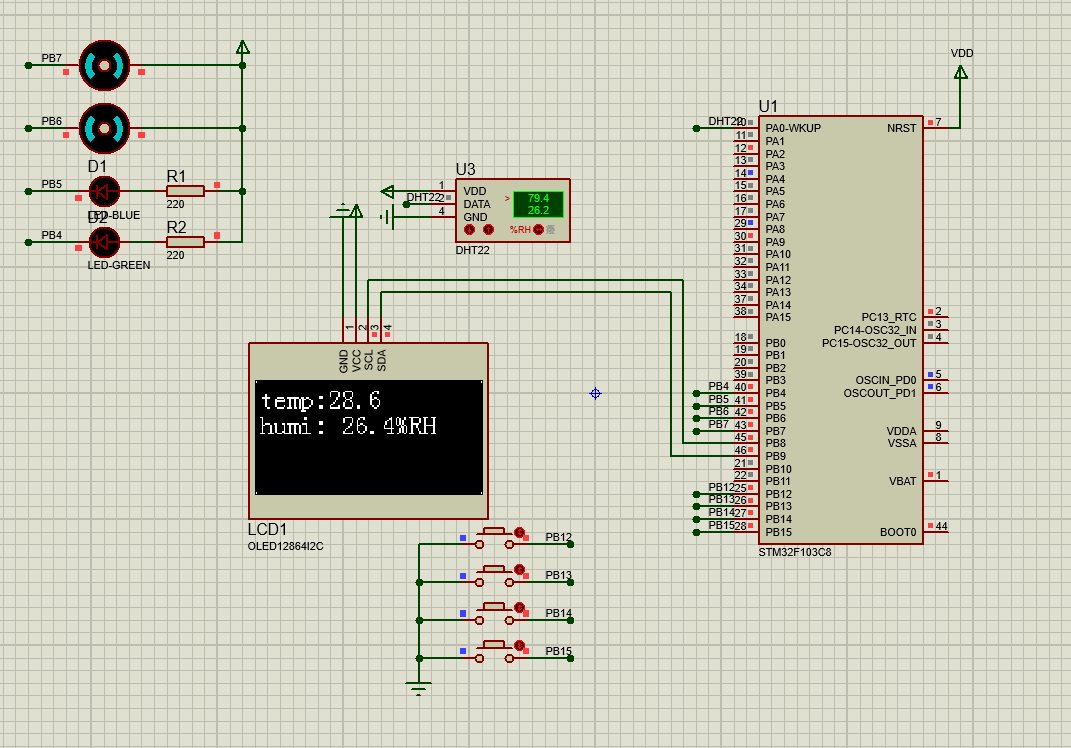
系统仿真界面如图 5.1所示：

图 5.1 仿真效果图

## 实物效果图

在完成系统软硬件集成与功能测试后，搭建了智能空调调节系统的完整实物平台。系统基于 STM32F103C8T6 微控制器，外接 DHT22 温湿度传感器、继电器模块、风扇、加湿器、OLED 显示器、ESP8266 无线模块，整体布线紧凑、结构合理。

实物展示过程中，用户可APP控制或 MQTT 平台远程指令实现空调系统的启停、模式切换以及温湿度调节。系统实时采集环境数据，根据温湿度变化自动控制风扇与加湿器运行状态，并通过 OLED 屏显示当前工作模式、设备状态和环境数据。如图所示可直观观察到系统各模块联动工作过程，实物演示效果良好，功能齐全，运行稳定，较好地达到了项目设计目标。

# 总结与展望

## 总结

本课题围绕智能空调调节系统展开设计与实现，依托 STM32F103C8T6 作为核心控制器，结合 DHT22 传感器、继电器模块、风扇、加湿器等外设，实现了对环境温湿度的实时监测与自动调节控制。同时，系统集成了本地按键、OLED 显示和 MQTT 网络通信等模块，支持多种控制方式，并通过 ESP8266 无线模块连接 OneNET 平台，完成远程数据同步、控制指令下发与设备状态上报功能。

系统采用标准库进行 STM32 编程，具有结构清晰、响应实时、易于维护等优点。构建了模块化、可扩展的软件架构。经调试验证，系统在环境感知、自动调节、远程控制、状态反馈等方面功能完整，运行稳定，能够有效提升空调控制的智能化程度，满足现代智能家居中的应用需求。

## 展望

本系统虽然实现了温湿度自动调节、远程控制、状态反馈等核心功能，但在整体智能化水平、系统稳定性和用户体验方面仍有进一步优化的空间。未来可以考虑引入更高精度的传感器，提高环境感知的准确性，同时通过改进控制算法，使风扇、加湿器等设备的启停更加柔性、智能。在控制策略方面，可尝试应用模糊控制或 PID 等方法，实现对环境变化更平滑、响应更迅速的调节过程。为提升用户交互体验，移动端 APP 可增加图表可视化、设备状态日志记录及定时控制等实用功能，并进一步优化界面布局与操作逻辑。此外，为增强系统的稳定性与安全性，可加强网络通信的异常处理机制及数据加密措施，提升系统在复杂网络环境下的可靠性。在平台拓展层面，未来系统可以整合更多智能家居设备，朝着统一控制、集中管理的方向演进，逐步构建功能更全面、响应更智能、交互更友好的智能家居综合控制系统。

# 参考文献

1. 李强. 智能建筑暖通空调系统的节能和优化探究[J]. 智能城市应用, 2023, 6(5): 108.
2. 蒋敏政. 空调能耗对建筑碳排放的影响[J]. 化学建材, 2022, 14(3): 89-95.
3. 张峰. 碳中和下中央空调智能化及节能减排技术[J]. 中国期刊网, 2022.​
4. 丁兆勇. 基于双碳背景下某建筑空调系统节能减排研究[J]. 环境与发展, 2023, 5(11).
5. Lin L C, Li Y X, Wang Y. Research on data transmission based on CC2530 of ZigBee[C]// 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). Chengdu: IEEE, 2019: 1438–1442.
6. Yu C T, Lin Y L, Lee Y S. Wireless sensor networks for emergency navigation: Advanced technologies and real-life applications[J]. Computer Communications, 2022, 182: 30–42.
7. Elahi A, Gschwender A. ZigBee Wireless Sensor and Control Network[M]. Upper Saddle River, NJ: Prentice Hall, 2009.
8. 王念欧. 基于人流分布感知的智能空调控制系统设计[D]. 重庆: 重庆邮电大学, 2021.
9. 马鹏宇, 王晓东, 刘建超. 智能空调系统在办公楼宇中的应用[J]. 智能建筑与城市信息, 2021(3): 36–39.
10. 汤斯茗. 基于ZigBee的智能空调控制系统的研究与设计[D]. 芜湖: 安徽工程大学, 2020.
11. 李聪聪.智能空调测控系统的设计与实现[D].安徽大学,2022.
12. 陈民植,郑正斌,张萍.基于OneNET云平台的智能空调系统[J].物联网技术,2024,14(07):139-143.
13. 崔翔宇,赵红,袁焕涛,等.基于Wi-Fi的智能空调双向监控系统[J].青岛大学学报(工程技术版),2020,35(04):55-61.
14. Anne L, Nandan T, Kunj M, et al. MQTT-Based Android Chat Application for IoT[J]. SN Computer Science, 2022, 3(5): 402.​
15. Anne L, Nandan T, Kunj M, 等. MQTT-Based Android Chat Application for IoT[J]. SN Computer Science, 2022, 3(402): 1–10.

# 致谢

在完成本论文的写作和研究过程中，我获得了许多人的帮助和支持，在此向他们表示最诚挚的感谢。

首先，我要感谢我的导师朱老师，她在论文的各个阶段都给予了我宝贵的指导和建议。她不仅在学术上为我指明了研究方向，而且在遇到困难时给予了我鼓励和支持，为我的研究提供了坚实的基础。在她的悉心指导下，我在研究方法、理论分析和写作技巧等方面都有了显著的提高。

其次，我要感谢我的家人，无论是物质上的资助还是精神上的鼓励，他们都始终如一地支持着我，给予我力量和动力。特别是我的父母，他们无微不至的关怀和殷切期望是我不断前进的源泉。

最后，我还要感谢我的朋友们，他们在我研究和写作过程中给予了我巨大的帮助，在我遇到难题时总是不遗余力地帮助我解决问题，为我的研究提供了许多有价值的建议和灵感，在我感到压力和困惑时，给予我宝贵的建议和支持。

再次感谢所有在本论文完成过程中给予我帮助和支持的人！你们的支持和鼓励是我前进的动力，是我顺利完成本论文的关键。希望在今后的学术道路上，能够继续得到你们的指导和帮助，让我们共同进步，共同成长。

# 附录

# 附录一 程序

|  |
| --- |
| //单片机头文件  #include "stm32f10x.h"  //网络协议层  #include "onenet.h"  //网络设备  #include "esp8266.h"  //硬件驱动  #include "delay.h"  #include "usart.h"  #include "LED.h"  #include "key.h"  #include "dht22.h"  #include "oled.h"  #include "manual.h"  #include "auto\_mode.h"  //C库  #include <string.h>  //mqtts.heclouds.com:1883  #define ESP8266\_ONENET\_INFO "AT+CIPSTART=\"TCP\",\"mqtts.heclouds.com\",1883\r\n"  /\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： Hardware\_Init  \*  \* 函数功能： 硬件初始化  \*  \* 入口参数： 无  \*  \* 返回参数： 无  \*  \* 说明： 初始化单片机功能以及外接设备  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  void Hardware\_Init**(**void**)**  **{**    NVIC\_PriorityGroupConfig**(**NVIC\_PriorityGroup\_2**);** //中断控制器分组设置  Delay\_Init**();** //systick初始化    Usart1\_Init**(**115200**);** //串口1，打印信息用    Usart2\_Init**(**115200**);** //串口2，驱动ESP8266用    //IIC\_Init(); //软件IIC初始化  OLED\_Init**();**  // LED\_Init(); //蜂鸣器初始化  // KEY\_Init();  // Relay\_Init();  Manual\_Control\_Init**();**  Mode\_Switch\_Init**();**  //UsartPrintf(USART\_DEBUG, " Hardware init OK\r\n");  OLED\_ShowString**(**0**,** 0**,** **(**u8 **\*)**"Hardware init OK"**,** 16**,** 1**);**  OLED\_Refresh**();**  DelayMs**(**1000**);**  **}**  /\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： main  \*  \* 函数功能：  \*  \* 入口参数： 无  \*  \* 返回参数： 0  \*  \* 说明：  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  DHT22\_Data\_TypeDef Data **=** **{**0**};**  int main**(**void**)**  **{**  unsigned short timeCount **=** 0**;**  unsigned char **\***dataPtr **=** **NULL;**  Hardware\_Init**();**  ESP8266\_Init**();**    OLED\_DisplayWrappedText**(**0**,** 0**,** "Connect MQTTs Server..."**,** 16**);**  **while(**ESP8266\_SendCmd**(**ESP8266\_ONENET\_INFO**,** "CONNECT"**))**  DelayXms**(**500**);**    OLED\_DisplayWrappedText**(**0**,** 0**,** "Successfully Connect MQTTs Server..."**,** 16**);**  **while(**OneNet\_DevLink**())**  **{**  ESP8266\_SendCmd**(**ESP8266\_ONENET\_INFO**,** "CONNECT"**);**  DelayXms**(**500**);**  **}**    OLED\_DisplayWrappedText**(**0**,** 0**,** "Successfully In MQTT..."**,** 16**);**  OLED\_DisplayWrappedText**(**0**,** 0**,** "Device login..."**,** 16**);**  DelayMs**(**500**);**  OneNET\_Subscribe**();**  OLED\_Clear**();**  OLED\_Show\_Character**();**  **while(**1**)**  **{**  Check\_Mode\_Switch**();** // 轮询检测按键状态  **if(++**timeCount **>=** 50**)**  **{**  Read\_DHT22\_Data**(&**Data**);**  OLED\_Show\_Temp\_Humi**(&**Data**);**  OneNet\_SendData**();** // 发送数据  Auto\_Mode\_Control**(**Data**.**temperature**,** Data**.**humidity**);**  timeCount **=** 0**;**  ESP8266\_Clear**();**  **}**    dataPtr **=** ESP8266\_GetIPD**(**0**);**  **if(**dataPtr **!=** **NULL)**  OneNet\_RevPro**(**dataPtr**);**    DelayMs**(**50**);**  **}**  **}**  //usart.c  #include "usart.h"  #include "delay.h"  #include <stdarg.h>  #include <string.h>  #include <stdio.h>  void Usart1\_Init(unsigned int baud)  {  GPIO\_InitTypeDef GPIO\_InitStruct;  USART\_InitTypeDef usart\_initstruct;  NVIC\_InitTypeDef nvic\_initstruct;    RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1, ENABLE);    //PA9 TXD  GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_AF\_PP;  GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_9;  GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA, &GPIO\_InitStruct);    //PA10 RXD  GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;  GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_10;  GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA, &GPIO\_InitStruct);    usart\_initstruct.USART\_BaudRate = baud;  usart\_initstruct.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;  usart\_initstruct.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx;  usart\_initstruct.USART\_Parity = USART\_Parity\_No;  usart\_initstruct.USART\_StopBits = USART\_StopBits\_1;  usart\_initstruct.USART\_WordLength = USART\_WordLength\_8b;  USART\_Init(USART1, &usart\_initstruct);    USART\_Cmd(USART1, ENABLE);    USART\_ITConfig(USART1, USART\_IT\_RXNE, ENABLE);    nvic\_initstruct.NVIC\_IRQChannel = USART1\_IRQn;  nvic\_initstruct.NVIC\_IRQChannelCmd = ENABLE;  nvic\_initstruct.NVIC\_IRQChannelPreemptionPriority = 0;  nvic\_initstruct.NVIC\_IRQChannelSubPriority = 2;  NVIC\_Init(&nvic\_initstruct);  }  void Usart2\_Init(unsigned int baud)  {  GPIO\_InitTypeDef GPIO\_InitStruct;  USART\_InitTypeDef usart\_initstruct;  NVIC\_InitTypeDef nvic\_initstruct;    RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);  RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_USART2, ENABLE);    //PA2 TXD  GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_AF\_PP;  GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_2;  GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA, &GPIO\_InitStruct);    //PA3 RXD  GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;  GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_3;  GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;  GPIO\_Init(GPIOA, &GPIO\_InitStruct);    usart\_initstruct.USART\_BaudRate = baud;  usart\_initstruct.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;  usart\_initstruct.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx;  usart\_initstruct.USART\_Parity = USART\_Parity\_No;  usart\_initstruct.USART\_StopBits = USART\_StopBits\_1;  usart\_initstruct.USART\_WordLength = USART\_WordLength\_8b;  USART\_Init(USART2, &usart\_initstruct);    USART\_Cmd(USART2, ENABLE);    USART\_ITConfig(USART2, USART\_IT\_RXNE, ENABLE);    nvic\_initstruct.NVIC\_IRQChannel = USART2\_IRQn;  nvic\_initstruct.NVIC\_IRQChannelCmd = ENABLE;  nvic\_initstruct.NVIC\_IRQChannelPreemptionPriority = 0;  nvic\_initstruct.NVIC\_IRQChannelSubPriority = 0;  NVIC\_Init(&nvic\_initstruct);  }  void Usart\_SendString(USART\_TypeDef \*USARTx, unsigned char \*str, unsigned short len)  {  unsigned short count = 0;    for(; count < len; count++)  {  USART\_SendData(USARTx, \*str++);  while(USART\_GetFlagStatus(USARTx, USART\_FLAG\_TC) == RESET);  }  }  void UsartPrintf(USART\_TypeDef \*USARTx, char \*fmt,...)  {  unsigned char UsartPrintfBuf[296];  va\_list ap;  unsigned char \*pStr = UsartPrintfBuf;    va\_start(ap, fmt);  vsnprintf((char \*)UsartPrintfBuf, sizeof(UsartPrintfBuf), fmt, ap);  va\_end(ap);    while(\*pStr != 0)  {  USART\_SendData(USARTx, \*pStr++);  while(USART\_GetFlagStatus(USARTx, USART\_FLAG\_TC) == RESET);  }  }  void USART1\_IRQHandler(void)  {  if(USART\_GetITStatus(USART1, USART\_IT\_RXNE) != RESET)  {  USART\_ClearFlag(USART1, USART\_FLAG\_RXNE);  }  }  //dht22.c  #include "dht22.h"  #include "delay.h"  static void DHT22\_SetOutput(void)  {  RCC\_APB2PeriphClockCmd(DHT22\_RCC\_CLOCK, ENABLE);    GPIO\_InitTypeDef GPIO\_InitStruct = {0};    GPIO\_InitStruct.GPIO\_Pin = DHT22\_GPIO\_PIN;  GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_Out\_PP;  GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;    GPIO\_Init(DHT22\_GPIO\_PORT, &GPIO\_InitStruct);  DHT22\_SetPin(1);  }  static void DHT22\_SetInput(void)  {  RCC\_APB2PeriphClockCmd(DHT22\_RCC\_CLOCK, ENABLE);    GPIO\_InitTypeDef GPIO\_InitStruct = {0};    GPIO\_InitStruct.GPIO\_Pin = DHT22\_GPIO\_PIN;  GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;    GPIO\_Init(DHT22\_GPIO\_PORT, &GPIO\_InitStruct);  }  static void DHT22\_Start(void)  {  DHT22\_SetOutput(); //将DHT22引脚配置为输出模式并拉高  DHT22\_SetPin(0);  DelayMs(1); //延时20ms，确保DHT22进入响应状态  DHT22\_SetPin(1);  DelayUs(30); //延时30us，准备接收应答信号  DHT22\_SetInput(); //切换为输入模式等待响应  }  int8\_t DHT22\_Check\_Response(void)  {  uint8\_t cnt = 0;  //DHT22\_SetInput();//设置IO为输入模式  if (DHT\_DQ\_IN() == Bit\_RESET) //检测是否有低电平  {  while((!DHT\_DQ\_IN()) && (cnt <= 85))//计算低电平持续时间，判断低电平是否应答超时，手册最大应答时间是85us  {  cnt ++; //每1us自加一次  DelayUs(1);  }  if(cnt > 85) //大于85us则应答超时  {  return ACK\_OVER\_TIME;//响应超时  }  else //应答正常  {  cnt = 0;  }  /\*低电平应答正常后到高电平 \*/  while(DHT\_DQ\_IN() && (cnt <= 85))//计算高电平持续时间，判断高电平是否应答超时，最大应答时间也是85us  {  cnt ++;  DelayUs(1);  }  if(cnt > 85) //响应超时  {  return ACK\_OVER\_TIME;  }  else  {  cnt = 0;  return ACK\_SUCCESS;  }  }  else  {  return ACK\_ERROR;//应答错误，起始信号后没有应答信号返回  }  }  uint8\_t DHT22\_Read\_Byte(void)  {  uint8\_t i;  uint8\_t DHT22\_Byte = 0;  for(i = 0; i < 8; i++)  {  while(DHT\_DQ\_IN() == Bit\_RESET);//等待低电平结束  DelayUs(40); //等待40us，再判断IO口电平状态  if(DHT\_DQ\_IN() == Bit\_SET)// 40 us后仍为高电平则表示数据“1”  {  /\* 等待数据1的高电平结束 \*/  while(DHT\_DQ\_IN() == Bit\_SET);    DHT22\_Byte |= (uint8\_t)(0x01 << (7-i)); //把第7-i位置1，MSB先行  }  else // 40 us后为低电平则表示数据“0”  {  DHT22\_Byte &= (uint8\_t)~(0x01 << (7-i)); //把第7-i位置0，MSB先行  }  }    return DHT22\_Byte;//返回当前读取到的字节  }  uint8\_t Read\_DHT22\_Data(DHT22\_Data\_TypeDef \*Data)  {    DHT22\_Start();//发送起始信号  if(DHT22\_Check\_Response() == ACK\_SUCCESS) //接收应答成功  {  Data->humi\_int = DHT22\_Read\_Byte();  Data->humi\_dec = DHT22\_Read\_Byte();  Data->temp\_int = DHT22\_Read\_Byte();  Data->temp\_dec = DHT22\_Read\_Byte();  Data->check\_num = DHT22\_Read\_Byte();  }    DHT22\_SetOutput();//设置IO为输出模式  GPIO\_SetBits(GPIOA, GPIO\_Pin\_0);//释放总线      if(Data->check\_num == Data->humi\_int + Data->humi\_dec +  Data->temp\_int + Data->temp\_dec)//校验正确  {  Data->humidity = (256 \* Data->humi\_int + Data->humi\_dec) / 10.0f;  Data->temperature = (256 \* Data->temp\_int + Data->temp\_dec) / 10.0f;  return SUCCESS; //校验通过  }  else  return ERROR;//校验不过  } |

# 附录二 元件清单

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 元件名称 | 型号 | 数量 |
| 1 | 主控芯片 | STM32F103C8T6 | 1 |
| 2 | 温湿度传感器 | DHT22 | 1 |
| 3 | 四路继电器 | 5V继电器模块 | 1 |
| 4 | WIFI模块 | ESP8266 | 1 |
| 5 | 液晶显示模块 | OLED模块 | 1 |
| 6 | 按键模块 | 6\*6\*6轻触按键 | 5 |