安徽建筑大学
ANHUI JIANZHU UNIVERSITY

毕业设计（论文）

题　目：	基于 STM32 的智能温室大棚控制系统设计
姓　名：	年志豪
学　号：	20210040216
学　院：	电子与信息工程学院
专　业：	电子信息工程
指导教师：	邵慧
完成时间：	2024 年 5 月 25 日

摘 要

为了通过环境监测和控制，提高作物生长的质量和效率。本研究设计了一种基于 STM32F407 微处理器的智能温室环境监测与控制系统，完成对温室大棚内部空气湿度、空气温度、土壤湿度、光照强度以及二氧化碳浓度等环境数据的实时测量与监控。主要集成一系列传感器设备与执行控制输出设备，包括温湿度传感器（DHT11）模块、土壤湿度传感器、光敏电阻以及二氧化碳浓度（SGP30）传感器等。

系统具备自动控制和手动控制两种模式，自动模式下能够根据不同农作物的生长需求，设定各自的环境阈值。系统通过自动控制通风口、浇水器、加湿器和通风扇等输出设备工作状态来控制温室温、湿度、土壤湿度、空气湿度和 CO2 浓度等环境参数。手动模式下，可独立控制输出设备工作状态来调节到农作物生长需求环境。通过 0.96 寸 OLED 显示屏，系统可实时展示环境数据和设备状态。蓝牙模块支持数据上传至上位机，实现远程监测和控制。

测试中，光照测试表明光敏电阻能准确响应不同光照强度，设备命令测试实验证实了通风口、浇水机和补光灯的控制指令能被正确接收并执行。土壤湿度从 48%RH 提升至 76%RH 表明浇水机正常工作且可提高土壤湿度，加湿器能将空气湿度从 41%RH 提升至 85%RH。自动控制实验中，系统能根据 CO_2 浓度、光照强度和土壤湿度自动调整设备状态，以维持适宜的作物生长环境。

关键词： 温室大棚；环境监测；STM32；FreeRTOS

ABSTRACT

In order to enhance the quality and efficiency of crop growth through environmental monitoring and control, this study has designed an intelligent greenhouse environmental monitoring and control system based on the STM32F407 microcontroller. The system is capable of real-time measurement and monitoring of various environmental parameters within the greenhouse, such as air humidity, air temperature, soil moisture, light intensity, and carbon dioxide concentration. It integrates a series of sensor devices and actuator control output devices, including a temperature and humidity sensor (DHT11) module, soil moisture sensor, photoresistor, and carbon dioxide concentration (SGP30) sensor.

The system features both automatic and manual control modes. In automatic mode, it can set specific environmental thresholds based on the growth requirements of different crops. The system adjusts the working status of output devices such as ventilation ports, waterers, humidifiers, and exhaust fans to control the greenhouse's temperature, humidity, soil moisture, air humidity, and CO2 concentration. In manual mode, individual control of output devices is possible to regulate the environment to meet the growth needs of crops. An 0.96-inch OLED display allows the system to display real-time environmental data and device status. A Bluetooth module supports data uploading to an upper computer for remote monitoring and control.

During testing, the light intensity test demonstrated that the photoresistor can accurately respond to different light intensities, and the device command test confirmed that control commands for ventilation ports, waterers, and supplemental lights were correctly received and executed. The soil moisture increased from 48% RH to 76% RH, indicating that the waterer operates normally and can improve soil moisture. The humidifier was able to raise the air humidity from 41% RH to 85% RH. In the automatic control experiment, the system was able to automatically adjust the status of devices based on CO2 concentration, light intensity, and soil moisture to maintain a suitable environment for crop growth.

Keywords： Greenhouse; Environmental Monitoring; STM32; FreeRTOS

目 录

第1章　绪论

1.1 研究背景与意义

随着全球气候变化与人口压力所引发的粮食安全问题日渐凸显，提升农产品生产能效和保障农产品品质已成为国际农业领域待解决的关键议题。

智能温室控制系统能够减少人工干预，减少参数调节差错，节约人力成本并且通过物联网平台，可以实现远程监控和控制，提高管理效率，能够对温湿度、光照强度、CO_2 浓度等环境参数进行实时监测和控制，为作物提供适宜的生长环境。

1.2 国内外研究现状

基于 STM32 单片机的智能温室大棚控制系统设计在国内外已取得了一系列研究成果。在国内，黄超等人[1]设计并测试了一种基于 STM32 的在线恒温光谱分析系统，为环境参数精准监测提供了技术支持。夏志昌等人[2]则研究了 STM32 在控制半导体激光器输出功率和工作温度稳定性方面的应用，为智能控制系统的稳定性优化提供了理论依据。

纪建伟等人[3]专门针对温室环境，设计了一款基于 STM32 的 CO_2 浓度自动调控系统，实现实时、精准的环境调控。吴雪雪[4]探讨了基于 NB-IOT 技术的农作物大棚监测系统，进一步丰富了智能温室大棚的远程监控功能。

郭磊[5]详细阐述了如何利用 STM32 单片机设计温室环境监测和控制系统，为全面实现温室环境智能化管理奠定了基础。

Mateusz Smykowski 等人[6]使用 STM32 处理器开发了一种成本较低的 4 自由度并联机器人原型，表明了 STM32 在自动控制机械结构中的可行性。刘磊等人[7]设计了一种基于 STM32 微控制器和 LabVIEW 的多通道气体传感器检测系统，强化了 STM32 在环境监测中的应用领域。

Ionel Zagan 等人[8]针对 STM32 嵌入式设备优化了 Modbus 通信协议以减少数据采集时间，提高了系统整体效率。金睿等人[9]基于 STM32 设计并验证了人体代谢测量系统，展示了 STM32 在跨学科交叉领域的应用。王晓龙和于洪洋[10]设计了智能温室控制系统的四个模块，包括登录管理模块、数据显示模块、远程

控制模块和系统管理模块。经过测试和分析表明系统能够实现温室的预期效果。刘立军和张杨[11]设计的系统能够实时收集和传输温室内部的温度、湿度、光照强度等环境参数，对于提高温室作物的产量和质量具有重要意义。吴勇[12]等人设计了基于LoRa技术的长距离、低功耗无线传感器网络的温室环境监测和控制系统。通过优化LoRa终端节点的设计，降低了能耗，验证了多节点远程控制的可能性。李秀丽[13]等人设计了一个基于无线传感器网络的自动监测系统，用于监测温室蔬菜的生长条件，提供了无线传感器网络的搭建策略。Trinh[14]等人设计了基于 LoRa（Long Range）技术的自动灌溉系统，表明温室大棚长距离通信控制的实用性。Michiel Aernouts[15]等人讨论了如何部署和特性化低功耗的LoRaWAN传感器网络，以便在温室环境中收集关键数据。

虽然现有的基于 STM32 的智能温室大棚控制系统研究已取得了显著成果，但仍存在一些未充分解决的问题与挑战：现有系统往往侧重于单一或部分环境参数的控制，缺乏对温室环境系统的自动化管理，各子系统间的信息孤立。本研究构建了一个集成的智能调控系统，实现对温室环境的多参数综合管理以及利用FreeRTOS操作系统的调度策略，实现各环境因子间的协同调控。此外，系统预留了接口与模块化设计，便于未来功能扩展。

1.3 研究内容

本研究的内容是基于 STM32 的智能温室大棚控制系统设计，研究内容主要包括以下几个核心部分：

硬件设计：构建一个基于 STM32 微控制器的平台方案，实现环境数据的实时采集与输出设备控制。平台集成包括但不限于 DHT11 温湿度传感器、AD 土壤湿度、光敏电阻及二氧化碳浓度在内的多种传感器来获取温室内的环境数据。系统能够对接一系列执行模块，包括舵机控制、电热片调节、智能加湿器运作以及通风扇控制等设备，使系统能够在收集数据的基础上调整温室设备的工作状态。

软件系统开发：本项目选用FreeRTOS实时操作系统作为基础，构建一套高效的温室环境监测控制系统。系统内核围绕环境数据监测与智能化调控展开。数据采集模块依据预定时间间隔，有序地从各类传感器源获取实时环境数据。

策略紧密结合实时环境参数的变化趋势，能够自主判断并动态调整各个执行机构的工作状态，如舵机角度、加热片功率、加湿器输出量以及通风扇转速等。

环境数据显示界面设计：采用 OLED 显示屏，实时显示温室环境的各项参数，同时提供手动模式下的设备独立控制功能。

1.4 论文结构安排

第 1 章绪论简要介绍研究的背景，强调智能温室在提升农业生产效率与资源管理中的重要性。概述研究内容，包括系统设计与实现以及概述论文结构。

第 2 章设备选型及工作原理概述，具体描述 DHT11 温湿度传感器、OLED 显示屏、HC-05 蓝牙模块以及各类执行模块的选择与工作原理。

第 3 章系统搭建与设备组成，从整体架构出发，系统性地阐述各个组成部分的布局与功能。通过传感器模块的环境监测，执行模块的调控，蓝牙模块的远程通讯，以及显示模块的反馈，共同构成一个闭环的智能温室管理系统。软件方案则深入探讨基于 FreeRTOS 的任务调度机制、自动与手动控制逻辑、系统命令集。

第 4 章实验结果与分析，展示系统的准确度、稳定性等关键性能指标，对比改造前后的温室管理效率，证明设计的有效性。

第 5 章总结与展望，内容包括该系统设计的整体工作总结以及设计的不足与改进方案。

第2章　设备选型及工作原理概述

2.1 温湿度模块 (DHT11)

DHT11 是温湿度传感器模块。其内嵌电阻式感湿元件和 NTC 热敏电阻分别作为湿度传感单元与温度传感单元。该传感器基于单总线通信协议（one-wire）与控制设备进行数据交互，通过微控制器数据采集和处理，实现对温湿度的测量与输出。

硬件结构上，DHT11 采用紧凑封装设计，DHT11 包含四个引脚：VCC 用于接入 3.3V 的电源，GND 为接地端，DATA（DQ）引脚负责与主控系统的单线双向通信，在 DATA 线上使用上拉电阻配置。第四个引脚 NC 为空置引脚。

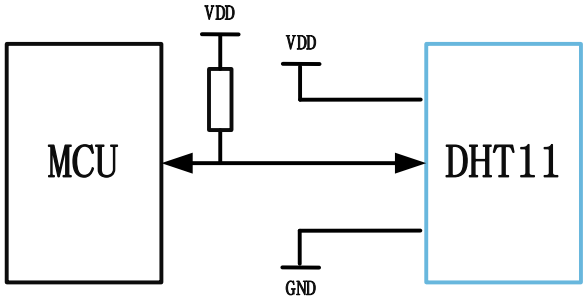


图 2.1 DHT11 典型应用电路

功能特性上，DHT11 相较于仅能提供温度信息的同类传感器有所拓展，能够同时探测温度和湿度，空气温度精度±2℃，测量范围在 0 至 50℃；空气湿度精度±5%RH，测量范围在 20%至 90%RH。其数据传输机制基于严格的时序控制，每次完整传输周期大约 4ms，数据帧由湿度整数、湿度小数、温度整数、温度小数和校验和组成，共计 40 比特。

byte4	byte3	byte2	byte1	byte0
00101101	00000000	00011100	00000000	01001001
整数	小数	整数	小数	校验和
湿度		温度		校验和

图 2.2 DHT11 数据帧

通讯时序上，DHT11 遵循一种定制化的单总线协议，启动通信需通过主控设备发出特定的复位信号。当 DHT11 接收到复位信号后，会由低功耗状态进入活跃状态，发送响应并进行数据传输。

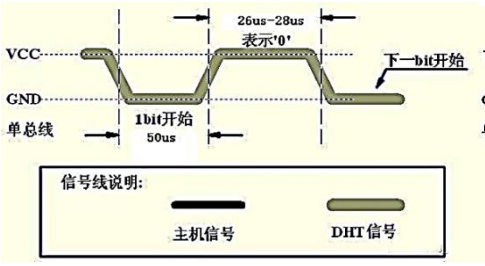


图 2.3 数字“0”时序

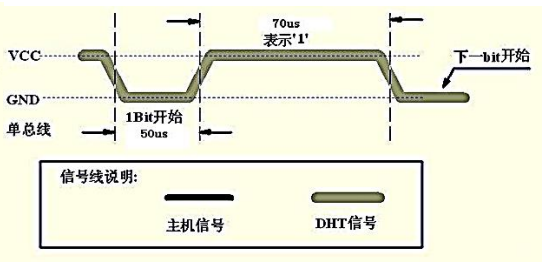


图 2.4 数字“1”时序

其中数据“0”和“1”通过高电平的时间来区分，逻辑“0”对应 26us~28us，逻辑“1”对应 70us。

2.2 显示输出模块 (OLED)

OLED 的核心原理在于利用有机薄膜材料在电场作用下发光的特性，实现对光信息的直接显示。本课题使用 0.96 英寸，分辨率为 128×64 像素，物理尺寸约为 23.7mm×23.8mmOLED 显示器。

在 OLED 显示原理层面，SSD1306 内部集成了一块 128×64bit 的图形 DRAM（GDDRAM），并将该内存空间逻辑划分为 8 个页面，每个页面包含 128 个字节的数据。在与微控制器交互过程中，微控制器(STM32F4)预先建立一个与 OLED 分辨率相匹配的内部缓冲区（实质上利用自身 SRAM 资源）。当需要更新屏幕显示内容时，先在微控制器内部缓冲区进行数据修改操作，而后一次性将整个缓冲区的内容传输至 OLED 的 GDDRAM 中，以此达到更新屏幕显示的目的。

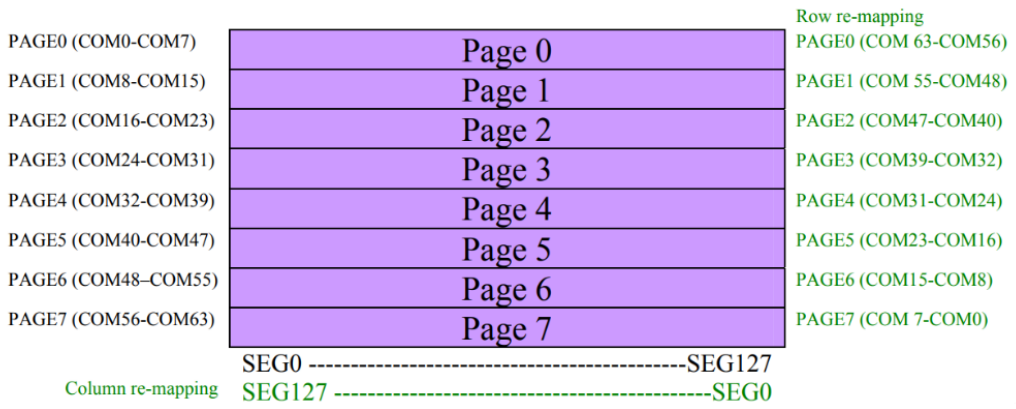


图 2.5 GDDRAM

SSD1306 驱动支持多种通信协议，本文使用 I²C 通信，主设备能够向其发送配置命令和显示数据，从而实现对 OLED 显示屏的有效控制与更新。

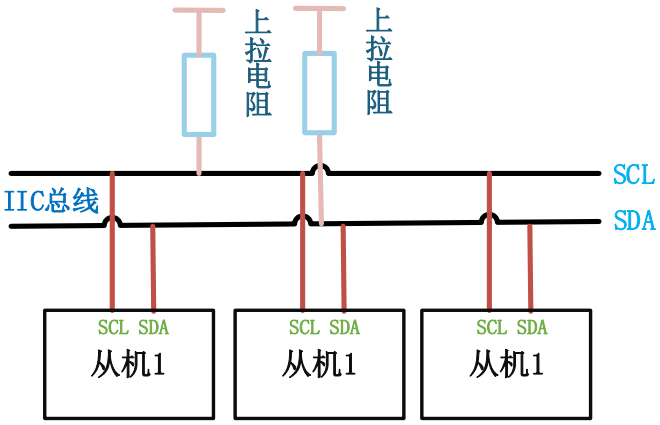


图 2.6 I²C 总线拓扑

I²C 总线上的数据传输主要由四种基本时序元素构成：起始、停止、数据传输和应答。

- 起始条件（Start Condition）：一次传输的开始的标志是在一个周期高电平期间，数据线 SDA 由逻辑“1”向逻辑“0”的跳变。这一变化需发生在 SCL 为逻辑“1”的期间，意味着仅在 SCL 上升沿之前，SDA 才能变为逻辑“0”。
- 停止条件（Stop Condition）：在完成一次数据传输或访问后，通过在 SCL 为逻辑“1”期间，SDA 由逻辑“0”向逻辑“1”的跳变来标识停止条件。类似地，这一变化也应在 SCL 下降沿之后发生，确保总线其他器件正确识别停止信号。
- 数据传输：数据在 SCL 的每个时钟在下降沿切换到下一位数据，上升沿被锁存。每个 Byte（8Bit）的数据传输都是从高位（MSB）传输数据，到低位（LSB）结束。每一 Byte 传输完成后，从设备发送应答信号给主设备，表明从设备接收到数据。
- 应答信号（Acknowledgement Bit）：主设备在每个字节数据传输完成后释放 SDA 线，允许从设备在下一个 SCL 时钟的高电平期间给出应答。若从设备正确接收数据或命令，则在该时钟周期内将 SDA 线拉低表示确认（ACK），反之则保持 SDA 为高电平表示非确认（NACK）。

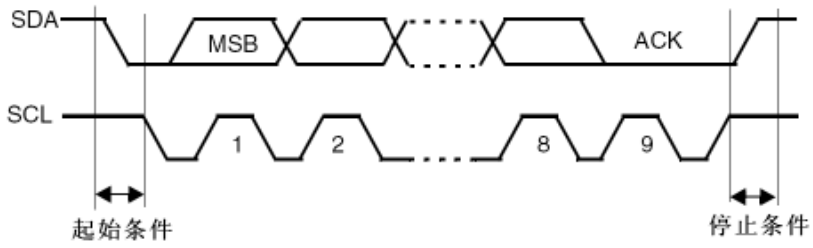


图 2.7 I²C 时序图

2.3 蓝牙无线通信模块 (HC-05)

HC-05 蓝牙模块是一种基于经典蓝牙协议 v2.0 标准的微型无线通信单元，其核心技术架构基于 BC417 单芯片蓝牙集成电路。

HC-05 引脚定义如下：

- HC-05 蓝牙模块有六个管脚
- STATE：状态指示。连接失败时输出逻辑“0”，连接成功输出逻辑“1”。
- RXD：UART 接收引脚
- TXD：UART 发射引脚
- GND：地
- VCC：接电源，可以用+5V。
- EN：使能。接地禁用模块，悬空或接 3.3V 使能。

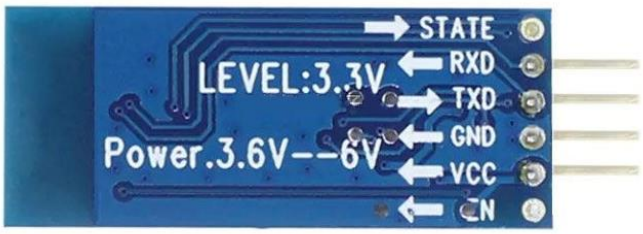


图 2.8 HC-05

本文使用 HC-05 的串口接口(UART)，UART 通信时序严格遵循一套预定义的规则，旨在确保数据在两个异步通信设备之间准确、可靠地传输。在 UART 通信中，数据以字符或字节的形式，通过一对数据线（Tx 和 Rx）进行串行传输，不受公共时钟信号的约束，而是依靠各自的波特率生成器来同步数据流。UART 通信时序如下：

- 起始位（Start Bit）：每次数据传输的开始，数据线（Tx）从高电平切换到低电平，标志着一个新的数据帧开始。起始位的宽度通常为一个单位时间间隔（即一个比特时间），并且所有接收设备在检测到起始位后立即开始时钟同步。
- 数据位（Data Bits）：数据位紧跟在起始位发送后，其数量最常见的是 8 位。每个数据位的持续时间相同，等于选定的波特率所对应的比特时间。
- 奇偶校验位（Parity Bit，可选）：某些 UART 通信可能包含一个奇偶校验位，用于增加传输错误检测的可能性。校验位是对数据位进行算术运算后得出的结果，可以是奇校验（odd parity）或偶校验（even parity），也可以不使用校验位。
- 停止位（Stop Bit(s)）：数据位和奇偶校验位传输完毕后，发送设备将数据线拉高，维持一段时间形成停止位。停止位的数量通常为 1 或 2 个 bit 时间，确保接收设备有足够的时钟周期来准确捕获数据并准备接收下一帧。

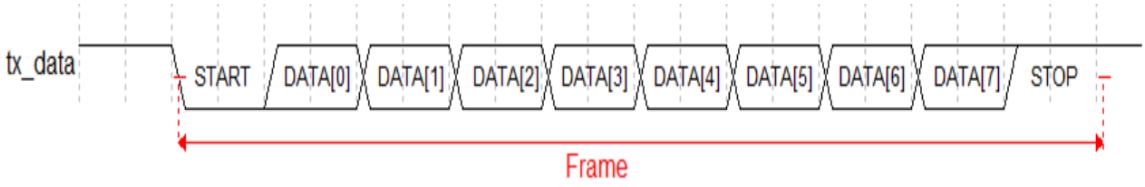


图 2.9 UART 通信时序

在 UART 通信中，主机（Master）与从机（Slave）必须预先设置相同的波特率、数据位数、停止位数以及奇偶校验模式才能确保正确解读对方发送的数据。

2.4 执行模块

执行模块通过一个中央控制器（STM32F4）进行协调，以实现对温室环境的精确控制。每个电路都包含继电器、电阻等基本电子元件，以及一些特定于设备的组件。执行模块如下：

- 加热片电路：包括一个电加热片，通过电流的通断来调节其热量输出，电路中的继电器负责接通或断开电源供应给加热片，而控制器会根据实时监测到的室内温度与预设的目标温度比较后发出相应指令，从而确保温室始终保持在适宜的温度范围内。
- 通风口电路：控制器通过舵机来控制通风口的开启和关闭，进而调节温室内的气体交换、降低气体浓度及调控湿度，传感器反馈的环境条件进行动态调整。
- 浇水机电路：控制器根据土壤湿度传感器的数据决定是否启动浇水机并向农作物提供适量的水分。
- 通风扇电路：通过改变风扇转速或启停状态来促进空气循环和降温。适当的空气流动有助于均匀分布温度和湿度，减少病害发生，并提高植物光合作用效率。
- 加湿器电路：当温室环境过干燥时，主控制器会吸合加湿器控制继电器，来激活加湿设备，增加温室内的空气湿度至适宜水平，这一过程由控制器根据实时湿度监控数据进行自动化控制。

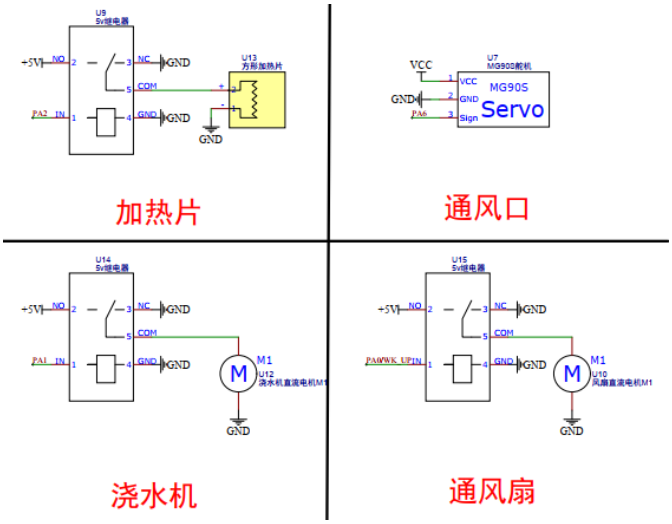


图 2.10 执行模块电路原理图

第3章　系统搭建与软件设计

3.1 系统架构

图 3.1 为基于 STM32 的温室环境监测系统的整体架构。以 STM32 为主控系统，控制多种传感器和执行设备，实现对温室环境的监测和智能控制。主要包括传感器、显示、主控、上位机以及输出控制四个部分。

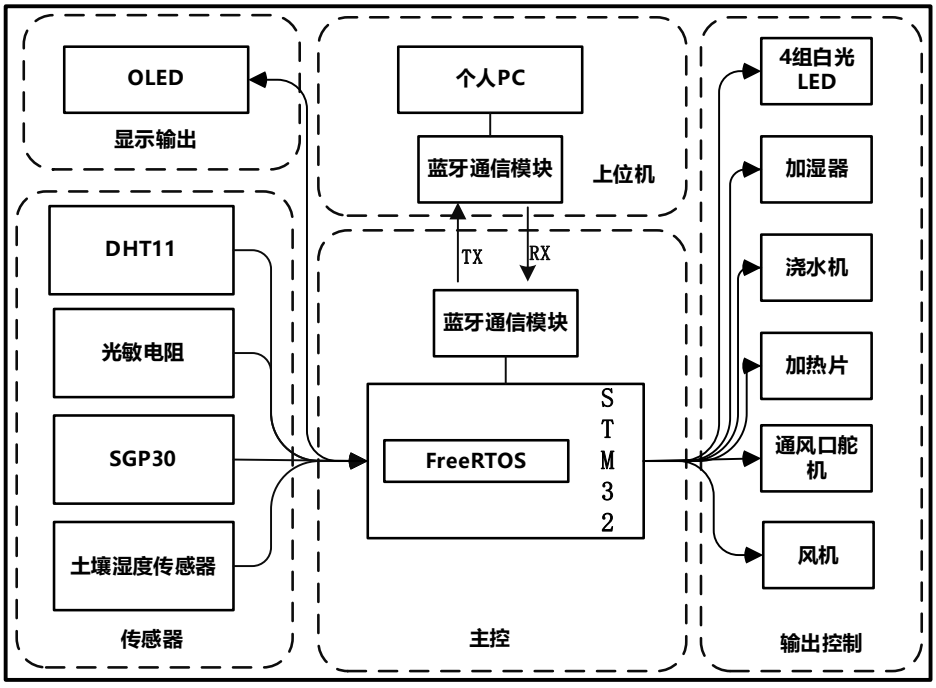


图 3.1 系统架构

其中传感器部分，系统采用温湿度传感器（DHT11）、光敏电阻、土壤湿度传感器以及空气质量传感器（SGP30），用于实时采集温室内的光照强度、温度、土壤湿度、空气质量以及空气湿度等关键参数。系统使用 FreeRTOS，完成处理传感器数据、执行控制逻辑以及与上位机进行通信等任务。系统集成 OLED 显示屏，用于显示当前的环境数据与设备状态。在输出部分，系统连接通风口电机、风机、加热片、加湿器以及浇水机等执行机构。为满足作物生长的需求，该系统可以根据预设的控制策略，自动调节温室环境。系统通过蓝牙通信模块与个人 PC 相连，可通过上位机软件远程查看温室的实时数据、设置控制参数以及发送控制指令。

3.2 组成模块

3.2.1 传感器模块

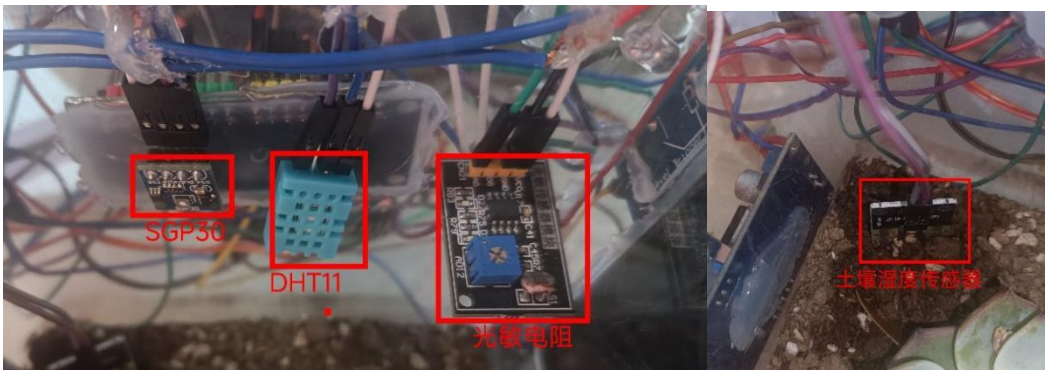


图 3.2 传感器设备

四个传感器模块是智能温室监测系统的重要组成部分：SGP30、DHT11、光敏电阻和土壤湿度传感器，如图 3.2 所示。四个传感器分别用于监测空气质量、温湿度和光照强度。

- SGP30：位于左上角的 SGP30 模块是一种先进的室内空气质量传感器。用于检测总挥发性有机化合物（TVOC）浓度。通过微控制器与 I2C 接口相连，提供实时的二氧化碳浓度数据。
- DHT11：位于中间的是 DHT11。用于测量空气湿度和空气温度，包含一个数字输出，提供湿度与温度数据。
- 光敏电阻：通过测量其阻值，可以间接获得光照强度的信息，辅助系统调整光照条件。
- 土壤湿度传感器：两个金属探针组成。两探针插入泥土中，测量探针间电阻来测量土壤湿度。被埋在土壤中，通过红色框标记的位置可知，探针直接接触土壤以测量土壤湿度。该传感器用于实时监测土壤的湿度情况，系统根据实际环境数据自动控制浇水系统来确保植物得到适当的水分供应。

通过微控制器，传感器的输出数据发送给上位机，在上位机上进行显示。可以实时了解环境数据情况，并根据需要调整浇水频率和量。四个传感器通过不同的接口与 STM32 微控制器相连，共同构成了温室环境监测系统的基础。传

感器采集的数据被实时采集并处理，为后续的控制策略提供依据。

3.2.2 执行模块

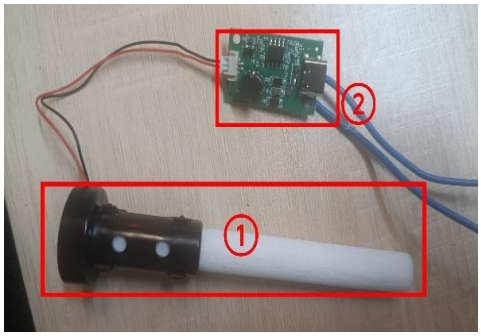


图 3.3 加湿器设备

图 3.3 是一个加湿器的组成部分及其驱动模块。图 3.3 序号 1 黑色的圆柱形装置是加湿器主体，中间有一根白色的棉棒穿过用于输送水。这个部分是加湿器的主要工作组件，负责将水转化为雾气以增加空气湿度。图 3.3 序号 2 为加湿器驱动模块。这个模块是加湿器的控制中心，负责控制加湿器的工作。

图 3.4 SG90 舵机设备

图 3.4 是一个小型舵机。被用来控制通风口的开合。本文选择 Tower Pro 品牌的微伺服电机型号 SG90。电机通过三根彩色线缆连接，分别为电源线、信号线和地线，用于接收控制信号并相应地转动一定的角度。该舵机能够精确地旋转到指定的角度，在系统中用于控制通风口的开合。

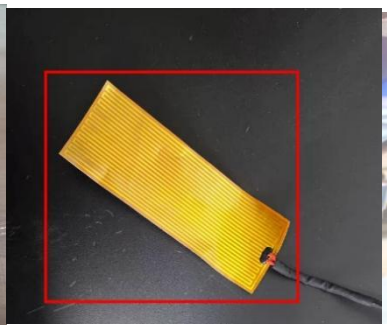




图 3.5 小型水泵设备　　图 3.6 加热器设备　　图 3.7 风机设备

图 3.5 为一个小型潜水泵，用于自动浇水系统。该设备用于维持土壤水分在一个适宜范围。图 3.6 为柔性加热片。该加热片由导电聚合物材料制成，可以在通电后均匀发热。用于控制环境温度。 图 3.7 为通风扇。通风扇是环境控制系统的物理实现，它的运行状态由 AutoModeTask 任务根据实时传感器数据动态调整来实现自动环境控制。

3.2.3 蓝牙模块

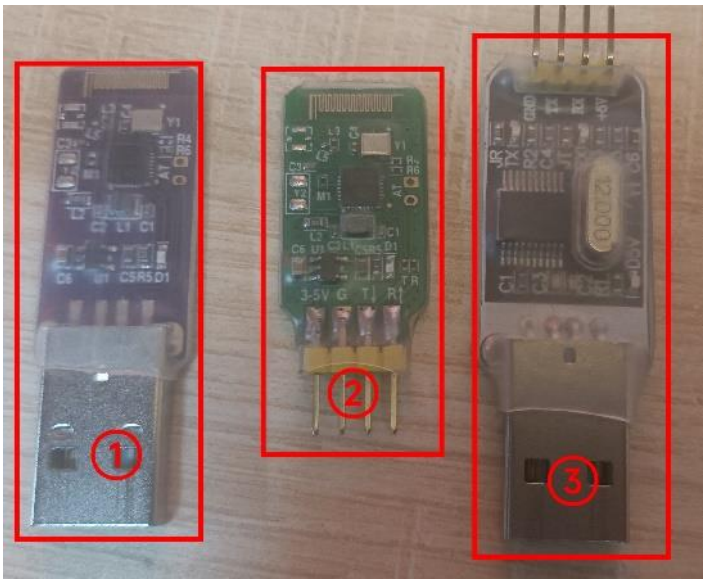


图 3.8 蓝牙设备

蓝牙主机（序号 1）：模块是蓝牙主设备，用于将 PC 连接至蓝牙模块。作为主机，接收环境数据和向从设备发送控制命令。

蓝牙从机（序号 2）：模块是蓝牙从设备，用于接收来自主设备的命令和发送环境数据。连接到 STM32，STM32 控制各种传感器和执行器，以响应来自主设备的命令。

USB 转 TTL 转换器（序号 3）：为使微控制器能与 PC 设备进行通信，模块将 USB 电平转换为 TTL 电平。

3.2.4 显示模块

图 3.9 环境数据

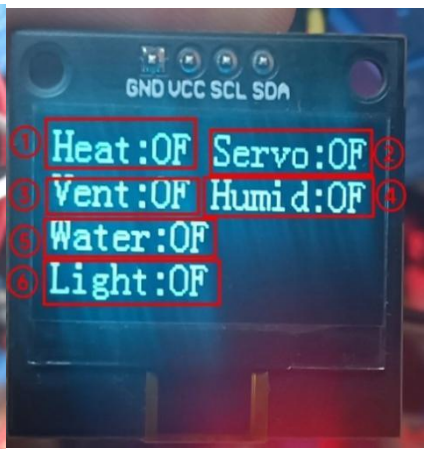


图 3.10 设备状态

图 3.9 为 OLED 显示屏，显示五个环境参数包括：

- 环境温度（Temperature, T）：显示数值为“30.0°C”，指示当前监测区域的气温状态。
- 相对湿度（Relative Humidity, H）：“79%RH”显示温室中的水分量水平。
- 光照强度（Light Intensity）：数值“33”反映了光照资源的充沛度。
- 二氧化碳浓度（CO_2 Concentration）：“1404 ppm”指标反映出温室中的二氧化碳浓度。
- 土壤湿度（Soil Moisture, SM）：“46%RH”展示土壤含水量。

图 3.10 为 OLED 显示屏。可同时显示六个控制设备状态信息，包括：

- 加热片（Heat）：由红色框序号 1 标注，显示为“OFF”，表明当前加热设备处于关闭状态。
- 舵机（Servo）：由红色框序号 2 标注，显示为“OFF”，控制通风口开闭的舵机，其状态影响空气流通和环境气体交换。
- 通风扇（Vent）：由红色框序号 3 标注，显示为“OFF”，通风扇用于促进空气流动，降低温室环境 CO_2 浓度。
- 加湿器（Humid）：由红色框序号 4 标注，显示为“OFF”，表明当前加

湿设备未运行，用于保持农作物生长适宜的空气湿度水平。

- 浇水机（Water）：由红色框序号 5 标注，显示为“OFF”，说明当前自动浇水系统未启动，这对保持土壤湿度、促进作物生长至关重要。
- 补光灯（Light）：由红色框序号 6 标注，显示为“OFF”，表明补光灯未开启，该设备用于补充自然光照。

3.2.5 上位机

图 3.11 上位机输出

图 3.11 为上位机截图，输出信息为系统回传上位机数据。主要包括环境参数和设备运行状态信息。

环境参数：

- 温度(T)：30.2°C（序号 1），温室当前的环境温度为 30.2 摄氏度。
- 湿度（H）：41%RH（序号 2），相对湿度达到 41%的水平。
- 光照强度（L）： 65（序号 3），反应光照强度状况。
- 土壤湿度（SM）：46%RH（序号 4），土壤相对湿度达到 46%的水平。
- CO2 浓度：400 ppm（序号 5），二氧化碳浓度处于 400 parts per million 的水平。

设备运行状态：

- 加热片：OFF(序号 6)，意味着加热设备当前未启动，无需增温。
- 通风口：OFF(序号 7)，显示通风设备未开启，未进行空气流通调节。

- 风机：OFF(序号 8)，风机处于未运行状态，未进行空气搅拌或降温。
- 加湿器：OFF(序号 9)，加湿设备未工作，当前不需要增加空气湿度。
- 浇水机：ON(序号 10)，正在运作，表明系统正对农作物进行适量灌溉。
- LED 灯：ON(序号 11)，灯光照明开启，为植物提供必要的光照条件。

（a）　　（b）

图 3.12 上位机控制界面

如图 3.12（a）包含串口任务定义的命令(序号 1)和自动控制任务定义的命令(序号 2)。如序号 1 红框内所示，定义一系列串口任务指令，旨在通过上位机发送至下位机（STM32 微控制器），功能包括：

- 切换自动模式：此指令使系统从手动模式转换到自动控制模式，允许系统依据预设的环境参数自动调整温室条件。
- 清除缓存区：用于清空串口缓存区数据，确保信息的即时更新与指令的准确传达，防止旧数据干扰当前操作。

序号 2 红框内为自动任务定义的命令，这些命令直接关联于特定农作物生长环境的自动调节策略。命令包括四种作物的环境。

如图3.12（b）为手动控制命令。当系统处于手动控制模式时（发送“切换手动”命令），每一个输出执行设备都可由命令独立控制。

3.3 软件方案设计

3.3.1 任务调度架构设计

使用FreeRTOS任务调度体系架构构建的智能温室环境管理系统。FreeRTOS实时操作系统在资源有限的嵌入式硬件平台上实现多任务并行处理。其设计特点包括：

FreeRTOS 采用抢占式多任务模型，使其能够根据任务优先级进行调度。确保高优先级任务在必要时立即得到执行。

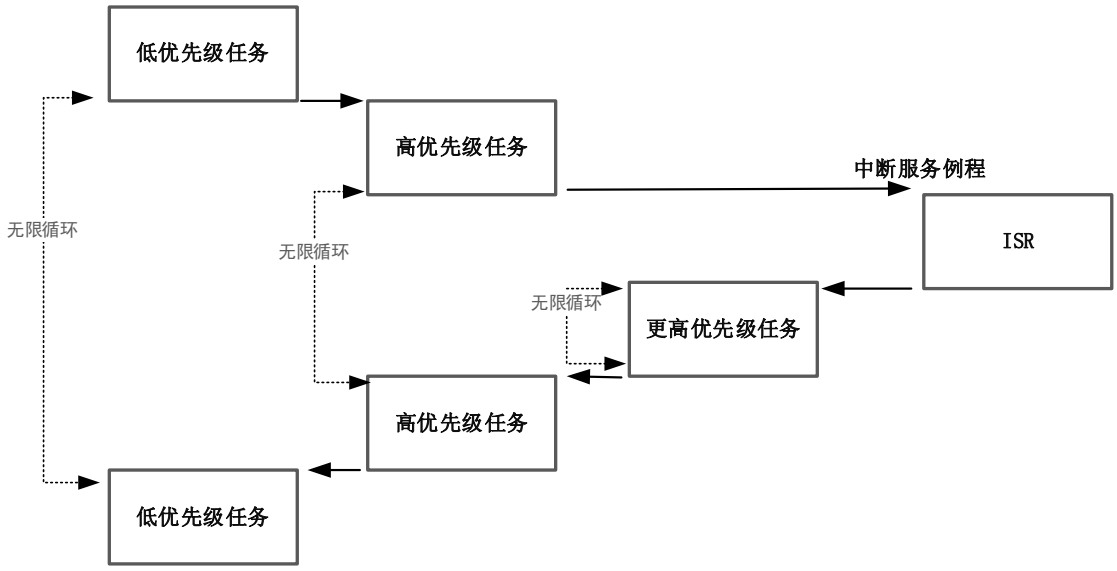


图 3.13 FreeRTOS 任务调度策略

系统内置多种同步对象旨在协助任务间协调合作，解决资源竞争问题。FreeRTOS 允许开发者根据具体应用场景定制内存分配策略。FreeRTOS 内核小巧，可以方便地移植到各种主流嵌入式处理器和微控制器上，适配众多架构和芯片型号。

FreeRTOS 在构建复杂嵌入式系统时，以其低层级抽象和高性能调度机制，降低了任务执行过程中的冗余与延迟，确保在时间约束下完成预定任务。

3.3.2 执行任务设计

该系统设计了一系列专用设备调控子任务，分别对接通风管理任务（VentilationControlTask）、舵机控制任务（ServoControlTask）、湿度调节任务（HumidifierControlTask）、温度调整任务（HeatingControlTask）、灯光控制任务（LightControlTask）以及浇水控制任务（WateringControlTask）。

3.3.3 自动控制进程设计

自动控制进程中，该系统对温室的二氧化碳浓度、空气湿度、空气温度、光照强度以及土壤湿度进行监测，在上位机上实时显示蓝牙发送的数据。

当监测到环境要素偏离所预设的理想阀值范围，系统将控制相应的硬件设备调控环境。当检测到空气中 CO2 浓度过高时，系统会启动通风装置，调整通风口的开闭降低 CO2 浓度。对于湿度波动情况，会依据测量湿度数据，智能控制加湿设备的启停状态。土壤湿度和温度的变化，系统能够通过对浇水策略和温控设备运行的调整，确保温室内部的气候环境条件稳定在适合植物生长的理想范围内。

自动化控制模式下，该系统具备为不同农作物调控生长环境参数的能力。可以根据每种农作物特有的生长需求，可在系统中预先设定各自适宜的环境变量区间。当接收到具体农作物环境切换命令，系统能够按照对应农作物的环境参数阈值对温室内部的各项环境调控设施进行自动控制。

自动控制任务流程如图 3.14 所示，首先检查有无新的串口数据被接收。若有，则根据接收到的数据更新环境参数范围。若无，则比较当前环境数据与温度高于设置范围，任务发送关闭加热器并开启风机和通风口的通知；同时更新参数阈值。若湿度低于最低阈值，任务会发送启动加湿器的通知。若二氧化碳浓度高于预设范围，任务会发送开启通风口的通知。若土壤湿度低于预设范围，任务会发送开启浇水任务的通知。任务进入阻塞状态。

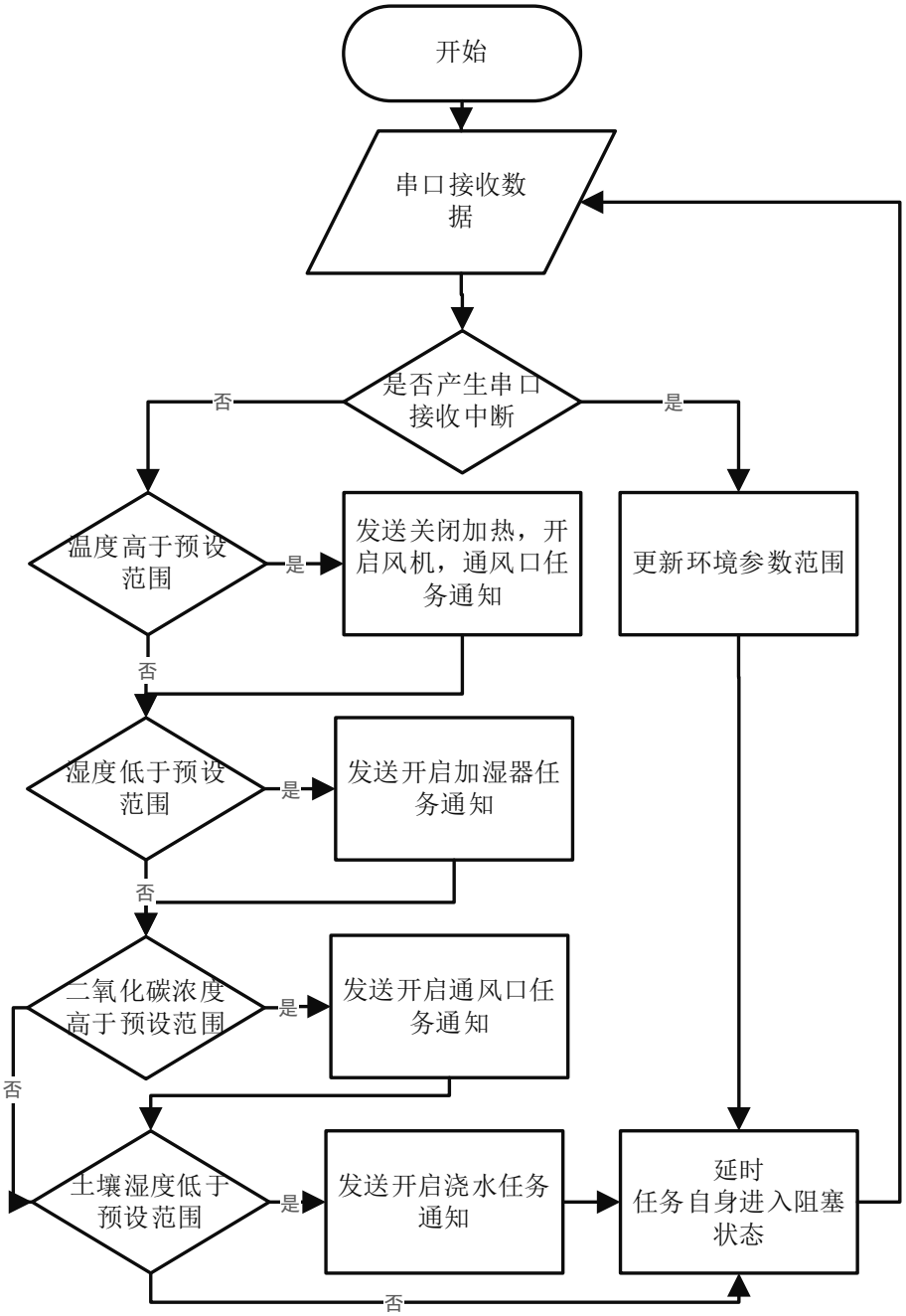


图 3.14 自动控制流程图

3.3.4 手动控制进程设计

手动控制进程中，ManualModTask 任务通过串行通信接口发送定制化指令的能力。可利用上位机，根据实际情况自主开启或关闭输出设备，设备包括通风装置、由舵机驱动的通风口调节系统、湿度调节设备、温度调节装置以及灌

溉系统在内的各类输出控制设备。

在人工指令执行过程中，系统增设了蓝牙回传环境信息的功能。系统不仅能够响应用户的指令，还会同步将设备当前状态以及执行操作后的变化情况通过蓝牙实时回传至控制终端，确保用户能够实时了解到所操作设备的具体状态，从而实现更加透明且可控的手动控制模式。

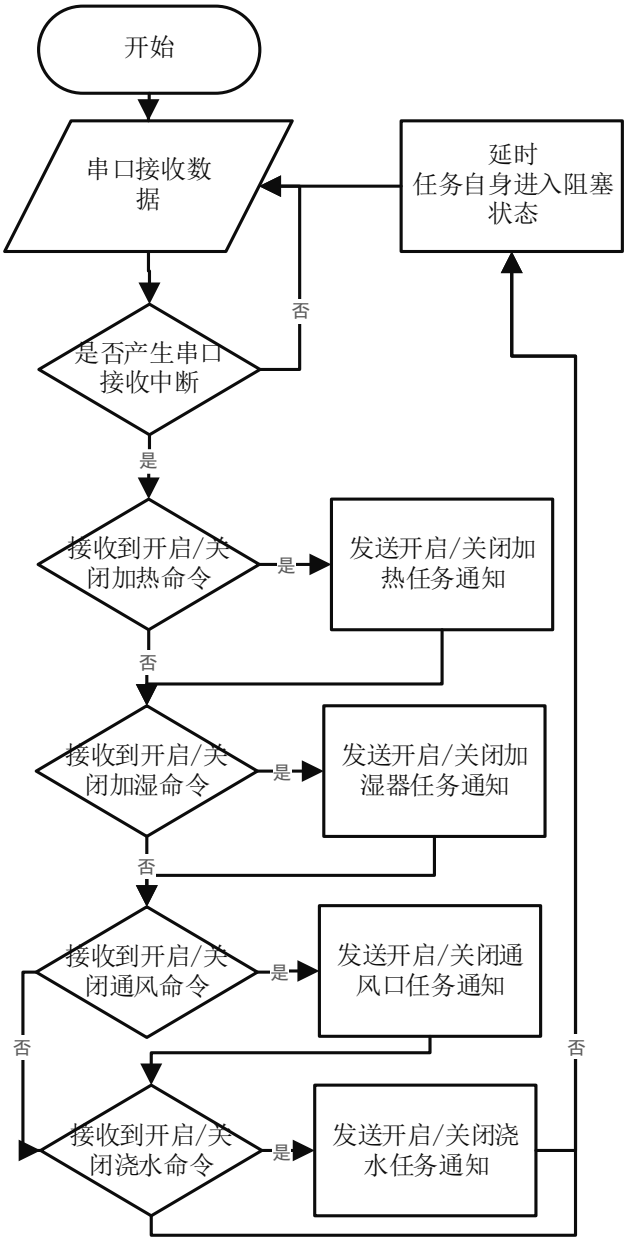


图 3.15 蓝牙控制流程图

3.3.5 系统命令汇总

表 3.2 是智能温室大棚环境监控与控制系统中涉及的主要控制命令及其描述，按照功能和所属进程进行了分类整理。

表 3.2 系统命令汇总

命令	所属进程	描述
switch auto	串口任务	切换自动控制
switch manual	串口任务	切换手动控制
clean	串口任务	清除命令缓冲区
open/close Vent	手动控制进程	打开/关闭通风扇
open/close Servo	手动控制进程	打开/关闭通风口
open/close Humid	手动控制进程	打开/关闭加湿器
open/close Heat	手动控制进程	打开/关闭加热器
open/close Water	手动控制进程	打开/关闭浇水机
open/close Light	手动控制进程	打开/关闭补光灯
checkout X	自动控制进程	切换 X 作物生存环境

第4章　实验结果与分析

4.1 环境测试实验

4.1.1 光照测试

为了验证系统光敏传感器的可靠性，进行实际环境测试，并记录相关数据。图 4.1 显示的为三种不同环境光照强度数据，将系统置于不同的光照条件下，以评估光敏电阻的响应特性，三种不同光照环境分别为：无光环境，自然光环境以及补光环境。

(a) 无光环境　　(b) 自然光环境　　(c) 补光环境

图 4.1 不同环境光敏数据

如图 4.1(a)在无光环境的控制条件下，光敏电阻记录的光照强度读数为基准值“2”。当转换至日常自然光照环境中(图 4.1(b))，光敏电阻的读数显著上升至“31”，与黑暗条件相比，显示出光敏电阻对环境光线增强的预期反应。引入额外的人工光源(图 4.1(c))后，光敏电阻所测得的光照强度进一步提升至“57”。这一数值远高于自然光条件下的读数，也显著区别于初始黑暗状态。

可以看出，光敏电阻在不同光照条件下的响应，光敏电阻在不同光照强度下读数变化验证了光敏电阻对光线变化的追踪能力。

4.1.2 土壤湿度测试

为了验证土壤湿度传感器的有效性以及浇水机提升土壤湿度的能力。记录初始土壤湿度后打开手动控制的浇水机开关(如图 4.2(a)所示)，记录浇水过程中土壤湿度的变化。

(a) 浇水前土壤湿度　　(b) 浇水后土壤湿度

图 4.2 浇水前后土壤湿度

通过对比图 4.2(a)和图 4.2(b)所展示的数据观察到明显的土壤湿度变化。打开浇水机之前(图 4.2(a))，土壤湿度记录为 48%RH。而浇水机开启之后，土壤湿度上升到 76%RH。这一显著的差异(从 48%RH 增至 76%RH)直接反映浇水机工作后土壤水分含量的增加，说明浇水机能够提升土壤的湿度，验证智能温室管理系统中土壤湿度传感器的有效性。

4.1.3 空气湿度测试

为了验证系统中空气湿度传感器的有效性以及加湿器提升空气湿度的能力。关闭加湿器，记录初始空气湿度后打开手动控制的加湿器开关，让其开始工作。记录加湿过程中空气湿度的变化。

(a)加湿前空气湿度　　(b)加湿后空气湿度

图 4.3 加湿前后空气湿度

图 4.3(a)显示实验初始状态下，即加湿器未介入操作前，环境的空气湿度为 41%RH。当加湿器启动并运行一段时间后，图 4.3(b)记录的数据显示空气湿度显著上升至 85%RH。这一从 41%至 85%的大幅度湿度变化，说明了加湿器对环

境湿度的有效调节以及空气湿度传感器在智能环境控制系统中正常的监测作用。

4.2 设备命令测试实验

4.2.1 通风口命令测试

在上位机打开蓝牙功能，并确保蓝牙可见性已开启。成功配对后，在上位机文本输入框中输入指令“open servo”发送，记录设备操作以及上位机接收数据。

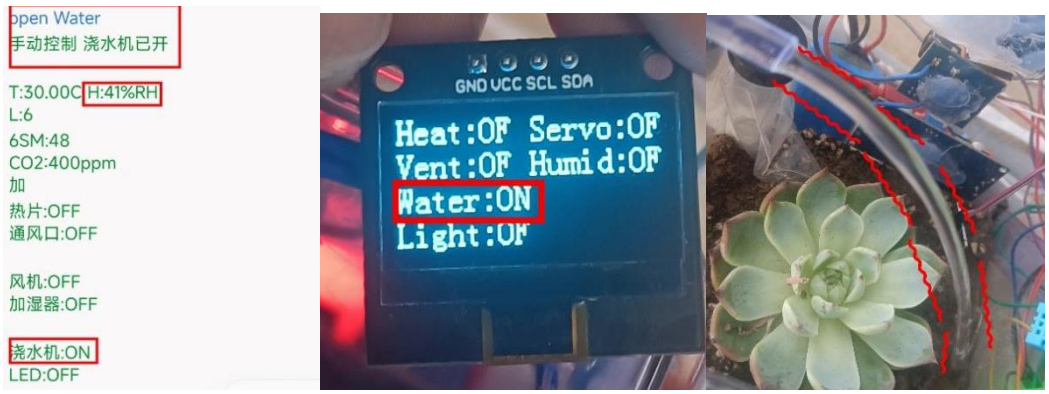
(a)命令窗口　　(b)设备状态信息　　(c) 设备状态

图 4.4 通风口状态

图 4.4(a)可以看到用于手动控制通风口的命令。该命令允许用户通过蓝牙切换通风口的状态。图 4.4(b)为显示屏上的反馈信息，显示通风口已经被打开。信息由系统自动更新，以便用户了解当前的设备状态。图 4.4(c)为通风口的实际照片，即通风口实物图。图片清晰地显示出通风口的开启状态，从而验证了通风口的正确工作。

4.2.2 浇水机命令测试

在图 4.7 所示的控制界面上发出了浇水命令(即“open Water”)，记录设备操作以及上位机接收数据。



(a)命令窗口　　(b)设备状态信息　　(c) 设备状态

图 4.5 浇水机状态

观察到图 4.5(b)中的显示屏上显示出“Water: ON”的状态反馈，这表明浇水指令已被正确接收并执行。图 4.5(c)所示的实际浇灌场景证实浇水机工作状态，表明智能温室控制系统中的浇水机已完成其设计目的。

4.2.3 补光灯命令测试

在图 4.10 所示由串口调试发出 “Light ON”命令，即补光灯开启命令。

(a)命令窗口　　(b)设备状态信息　　(c)设备状态

图 4.6 补光灯状态

系统自动更新补光灯设备状态(图 4.6(b))。4 组补光灯开启(图 4. 4.6(c))，表明补光灯开启指令已被正确接收并执行。

4.3 自动控制实验

4.3.1 二氧化碳浓度控制实验

用户向系统发送控制命令，如图 4.7(a)所示，发送自动控制命令“checkout 1”，用以配置农作物（番茄）的理想生长环境参数。该命令触发系统的自动调整机制，系统随后依据预设参数对环境进行监控与调控。

(a)自动控制命令窗口　　(b) 上位机窗口　　(c) 设备状态信息

图 4.7 二氧化碳浓度控制

当接收到“checkout 1”命令后，系统将二氧化碳浓度（CO_2）的可接受范围调整为 1500ppm 至 750ppm 之间。

如图 4.7(b)所示，当实时监测到的二氧化碳浓度超过了预设的最大值 1500ppm，系统激活通风扇和开启通风口，以促进空气流通和降低环境中二氧化碳浓度水平。图 4.7(c)展示的是通风以及风机设备已启动后的设备状态，表明系统对高二氧化碳浓度的应对措施已实施，确保环境恢复至适宜作物生长的条件。

4.3.2 光照强度控制实验

为了验证系统补光灯自动控制的可靠性，进行了实际环境测试。将系统置于不同的光照条件下，以测试在自动模式下补光灯的控制策略。两种不同光照环境分别为：低光照环境以及高光照环境。

(a)补光前光照强度　　(b)补光后光照强度

图 4.8 光照强度

图 4.8(a)展现系统处于“低光照强度状态”，光照强度为 3，低于预设的最低光照强度阈值。由于实际光照强小于设定的最小光照强度阈值，并且补光灯设备状态标志位为 0，因此系统自动开启补光灯，以补充光照，确保作物得到足够的光照进行光合作用。

在图 4.8(b)所示的“高光照强度状态”中，光照强度为 97，高于预设的最大光照强度阈值，光照过强且补光灯正处于开启状态，系统自动关闭补光灯。

4.3.3 土壤湿度控制实验

番茄生长环境的情况下，土壤湿度的适宜范围被设定为30%RH至60%RH。

(a)浇水前土壤湿度 (b) 浇水后土壤湿度 (c) 设备状态信息

图 4.9 土壤湿度

图 4.9(a)显示当前环境参数为土壤湿度（SM）为 14%RH。系统检测到土壤湿度低于预设阈值（30%RH）且浇水机处于关闭状态，自动开启浇水机（Water:ON）。图 4.9(b) 显示设备响应，表示浇水机工作了一段时间后，土壤湿度上升到了 80%RH。系统检测到土壤湿度达到或超过预设阈值(60%RH) 且浇水机正处在开启状态，自动关闭浇水机（Water:OFF）如图 4.9(c)所示。

第5章　总结与展望

5.1 总结

在本设计中，基于 STM32 的智能温室大棚控制系统的开发成功，实现了对作物生长环境的监测和自动化控制。系统具备自动和手动两种控制模式，这为满足不同作物的特定生长需求提供了灵活性。OLED 显示屏和蓝牙模块的加入，不仅增强了系统的用户交互性，还实现了远程监控的功能，使得用户能够及时获取环境数据和设备状态。FreeRTOS 实时操作系统的引入，确保了系统多任务的高效运行和稳定控制。实验结果表明，系统能够监测环境参数，并根据预设条件自动调节温室内的设备，如通风口、浇水机、加湿器和补光灯等。此外，系统设计中考虑到了未来可能的功能扩展和升级，预留了相应的接口和模块化设计，这为后续增加新的传感器、执行器或改进算法提供了便利。

5.2 后续工作展望

系统仅能通过上位机控制，缺少 UI 交互界面，显示设备仅能输出环境以及设备信息。此外，温室环境参数为预设值，不能应对不同农作物在不同季节对适宜环境要求。

后续工作中，对于控制方式单一，交互界面简陋，系统可利用 TFT 触摸屏取代 OLED 显示设备，提高人机交互便利性。关于农作物在不同季节对于适宜环境的要求，系统可建立农作物的环境参数模型，使得系统能够根据季节变化而调节温室环境，满足农作物对不同季节下对温室环境的要求。

致 谢

未来，我将怀揣着感激与使命，勇往直前，在工程科技领域不断探索创新，满足国家对高素质工程人才需求，为国家繁荣富强添砖加瓦，不负国家的培养与期待。

参考文献

[1] 黄超, 赵宇红, 张洪明等. 基于 STM32 单片机的在线恒温光谱分析系统研制与测试[J]. 光谱学与光谱分析, 2023, 43(09): 2734-2739.

[2] 夏志昌, 于永爱, 尚建华. 基于 STM32 的半导体激光器输出功率和工作温度稳定性研究[J]. 光子学报, 2023, 52(08): 93-104.

[3] 纪建伟, 赵海龙, 李征明等. 基于 STM32 的温室 CO_2 浓度自动调控系统设计[J]. 浙江农业学报, 2015, 27(05): 860-864.

[4] 吴雪雪. 基于 NB-IOT 的农作物大棚监测系统研究[J]. 农机化研究, 2023, 45(11): 122-126.

[5] 郭磊. 基于 STM32 的温室环境监测和控制系统[D].齐鲁工业大学, 2021.

[6] P.Andrzej, M.Smykowski. Using a Development Platform with an STM32 Processor to Prototype an Inexpensive 4-DoF Delta Parallel Robot[J]. Sensors, 2021, 21(23),62-79.

[7] L.Liu, M.Wang, Hussain S, et al. Design of a Multi-Channel Gas Sensor Detection System Based on STM32 Microcontroller and LabVIEW[J]. Journal of Nanoelectronics and Optoelectronics, 2023, 18(1), 17-24.

[8] I.Zagan, V.Gheorghiță. Enhancing the Modbus Communication Protocol to Minimize Acquisition Times Based on an STM32-Embedded Device[J]. Mathematics, 2022, 10(24),46-86.

[9] R.Jin, H.Huang, Y.Wang, et al. Design and Verification of Human Metabolic Measurement System Based on STM32[J]. Chinese Journal of Medical Instrumentation, 2022, 46(3), 273-277.

[10] X.Wang, H.Yu. Research on Control System of Intelligent Greenhouse of IoT Based on Zigbee[C]. 2019 CISAT, Penang, Malaysia, 2019.

[11] L.Liu, Y.Zhang. Design of Greenhouse Environment Monitoring System Based on Wireless Sensor Network[C]. In Proceedings of the 3rd International Conference on Control, Automation and Robotics (ICCAR), Nagoya, 2017, 463-466.

[12] Y.Wu, L.Li, M.Li, et al. Remote-Control System for Greenhouse Based on Open Source

Hardware[C]. IFAC PapersOnLine, 2019, 52(30):178–183.

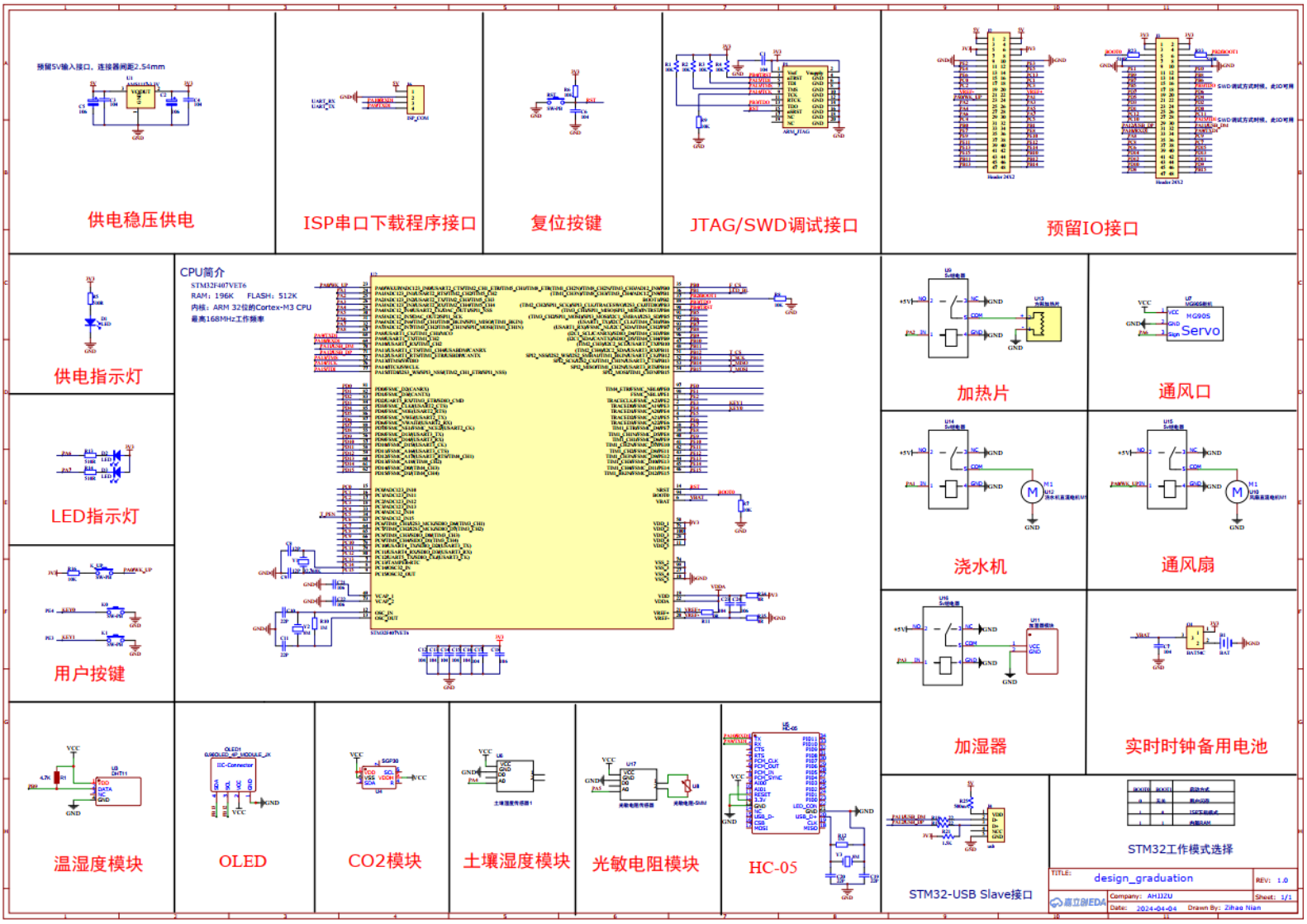
[13] X.Li, X.Cheng, K.Yan, et al. A Monitoring System for Vegetable Greenhouses Based on a Wireless Sensor Network[J]. Sensors, 2010, 10(10), 8963-8980.

[14] V.Anh, D.Trinh, T.Truvant. Design of Automatic Irrigation System for Greenhouse Based on LoRa Technology[C]. In Proceedings of the 2018 International Conference on Advanced Technologies for Communications (ATC), IEEE, 2018, 72-77.

[15] K.Singh, M.Aernouts, M.Meyer, et al. Leveraging LoRaWAN Technology for Precision Agriculture in Greenhouses[J]. Sensors, 2020, 20(7), 18-27.

附录

电路原理图



程序代码

```
1.      /*****************************************************************
2.       *Copyright(C):  ANHJZU
3.       *FileName:      Template
4.      *Author:        Zhihao Nian
5.      *Version:       V1.0
6.      *Description:   This template for STM32F407
7.      *Others:        Non
8.      *Function List: Non
9.     *Date:          2022 9 30
10.     *History:
```

```
      1.Date:
        Author:
        Modification:
      2.Non
**********************************************************************/
#include "stm32f4xx.h"
#include "delay.h"
#include "stdio.h"
#include "string.h"
#include "sys.h"
#include "MyI2C.h"
#include "OLED.h"//PB8（SCL） PB9（SDA）
#include "ADC.h"//PA5(sun_sonsor) PA4(soil_moisture)
#include "DHT11.h"//PB9
#include "pwm.h"//PA6
#include "sgp30.h"//PB0(SCL) PB1(SDA)
#include "usart.h"//PA9 接蓝牙RXD PA10 接蓝牙TXD
#include "timer.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "timers.h"
#include "VentilationControl.h"
#include "HeatingControl.h"
#include "WateringControl.h"
#include "HumidifierControl.h"
#include "Light.h"
#define DEVICE_OPEN   (0x00000001 << 0)  //设置设备开启掩码的位0
#define DEVICE_CLOSE    (0x00000001 << 1)  //设置设备关闭掩码的位1
#define OPEN      1      //设置设备开启掩码的位0
#define CLOSE     0      //设置设备关闭掩码的位1
typedef struct
{
 uint32_t  sgp30_dat ;
 uint32_t  CO2Data ;
 uint16_t  Soil_Moisture ;
 uint16_t  Light_Intensity ;
 float    temperature;
 float    humidity;
}Env_Data_t;         //环境参数结构体
typedef struct
{
```

```
53.    uint32_t  MAX_CO2_Concentration ;
54.    uint32_t  MIN_CO2_Concentration ;
55.    uint16_t  MAX_Soil_Moisture ;
56.    uint16_t  MIN_Soil_Moisture ;
57.    uint16_t  MAX_Light_Intensity ;
58.    uint16_t  MIN_Light_Intensity ;
59.    float     MAX_temperature;
60.    float     MIN_temperature;
61.    float     MAX_humidity;
62.    float     MIN_humidity;
63.  }Env_Para_Range_t;         //环境参数结构体
64.  typedef struct
65.  {
66.    u8 Vent_flag;
67.    u8 Servo_flag;
68.    u8 Humidifier_flag;
69.    u8 Heating_flag;
70.    u8 Watering_flag;
71.    u8  Light_flag;
72.  }Device_status_t;          //环境参数结构体
73.  Env_Data_t      Env_Data_Struct    ;//定义环境参数结构体变量
74.  Env_Para_Range_t  Env_Para_Range_Struct ;//定义环境参数范围结构体变量
75.  Device_status_t   Dev_status__Struct    ;//定义设备状态结构体变量
76.  SemaphoreHandle_t   UsartMuxSem_Handle   ;//定义串口互斥量变量
77.  void DataAcquisition_Callback(void* parameter);
78.  TimerHandle_t DataAcquisition_Handle =NULL;
79.  void startTask       (void *pvParameters);
80.  TaskHandle_t       StartTask_Handler;
81.  void UsartCommandTask    (void *pvParameters);
82.  TaskHandle_t       UsartCommandTask_Handler;
83.  void DataAcquisitionTask   (void *pvParameters);
84.  TaskHandle_t       DataAcquisitionTask_Handler;
85.  void AutoModeTask      (void *pvParameters);
86.  TaskHandle_t       AutoModeTask_Handler;
87.  void ManualModTask      (void *pvParameters);
88.  TaskHandle_t       ManualModTask_Handler;
89.  void VentilationControlTask   (void *pvParameters);
90.  TaskHandle_t       VentilationControlTask_Handler;
91.  void ServoControlTask    (void *pvParameters);
92.  TaskHandle_t       ServoControlTask_Handler;
93.  void HumidifierControlTask   (void *pvParameters);
94.  TaskHandle_t       HumidifierControlTask_Handler;
```

```
95.  void HeatingControlTask    (void *pvParameters);
96.  TaskHandle_t       HeatingControlTask_Handler;
97.  void WateringControlTask   (void *pvParameters);
98.  TaskHandle_t       WateringControlTask_Handler;
99.  void LightControlTask   (void *pvParameters);
100. TaskHandle_t       LightControlTask_Handler;
101. void OledDisplayTask    (void *pvParameters);
102. TaskHandle_t   OledDisplayTask_Handler ;
103. ///*****************主函数******************/
104. int main(void)
105. {
106.  NVIC_PriorityGroupConfig( NVIC_PriorityGroup_4);
107.  delay_init(168);
108.  uart_init(115200);
109.  DWT_Init();
110.  Adc_Init();//光敏电阻 通道5 PA5
111.  Adc2_Init();//土壤湿度 通道4 PA4
112.  SGP30_Init();
113.  DHT11_Init();
114.  IIC_Init();
115.  OLED_Init();
116.  TIM2_PWM_Init(19999,83);//舵机 PA0 500-2000
117.  VentilationControl_Init();
118.  WateringControl_Init();
119.  HeatingControl_Init();
120.  HumidifierControl_Init();
121.  LightControl_Init();
122.  //创建开始任务
123.  xTaskCreate((TaskFunction_t )startTask,    //任务入口地址
124.              (const char*    )"startTask",   //任务名称
125.              (uint16_t       )128,              //任务堆栈大小
126.              (void*          )NULL,   //任务输入参数
127.              (UBaseType_t    )5,             //任务优先级
128.              (TaskHandle_t*  )&StartTask_Handler);    //任务句柄
129.     vTaskStartScheduler();
130. }
131. /*****************?*****************/
132. void startTask(void *pvParameters)
133. {
134.     taskENTER_CRITICAL();           //进入临界区
135.  UsartMuxSem_Handle = xSemaphoreCreateMutex(); //创建互斥量
136.  //创建串口命令任务
```

```
137.  xTaskCreate((TaskFunction_t )UsartCommandTask,  //任务入口地
址
138.      (const char*    )"UsartCommandTask",        //任务名称
139.      (uint16_t       )128,              //任务堆栈大小
140.      (void*          )NULL,             //任务输入参数
141.      (UBaseType_t    )2,             //任务优先级
142.      (TaskHandle_t*  )&UsartCommandTask_Handler);       //任务句柄
143.     //创建数据采集任务
144.  xTaskCreate((TaskFunction_t )DataAcquisitionTask,//任务入口地
址
145.      (const char*    )"DataAcquisitionTask",       //任务名
称
146.      (uint16_t       )512,              //任务堆栈大小
147.      (void*          )NULL,             //任务输入参数
148.      (UBaseType_t    )5,             //任务优先级
149.      (TaskHandle_t*  )&DataAcquisitionTask_Handler);      //任务句柄
150.  //创建自动模式任务
151.  xTaskCreate((TaskFunction_t )AutoModeTask,          //任务入口地
址
152.      (const char*    )"AutoModeTask",         //任务名称
153.      (uint16_t       )512,              //任务堆栈大小
154.      (void*          )NULL,             //任务输入参数
155.      (UBaseType_t    )2,             //任务优先级
156.      (TaskHandle_t*  )&AutoModeTask_Handler);        //任务句柄
157.  //创建手动模式任务
158.  xTaskCreate((TaskFunction_t )ManualModTask,          //任务入口地
址
159.      (const char*    )"ManualModTask",         //任务名称
160.      (uint16_t       )512,              //任务堆栈大小
161.      (void*          )NULL,             //任务输入参数
162.      (UBaseType_t    )2,             //任务优先级
163.      (TaskHandle_t*  )&ManualModTask_Handler);        //任务句柄
164.  //创建通风控制任务
165.  xTaskCreate((TaskFunction_t )VentilationControlTask,       //任务入
口地址
166.      (const char*    )"VentilationControlTask",       //任务名
称
167.      (uint16_t       )128,              //任务堆栈大小
168.      (void*          )NULL,             //任务输入参数
169.      (UBaseType_t    )3,             //任务优先级
170.      (TaskHandle_t*  )&VentilationControlTask_Handler);      //任务句
柄
```

```
171.  //创建舵机控制任务
172.  xTaskCreate((TaskFunction_t )ServoControlTask,         //任务入口地
址
173.      (const char*    )"ServoControlTask",        //任务名称
174.      (uint16_t       )128,               //任务堆栈大小
175.      (void*          )NULL,              //任务输入参数
176.      (UBaseType_t    )3,             //任务优先级
177.      (TaskHandle_t*  )&ServoControlTask_Handler);       //任务句柄
178.  //创建加湿器控制任务
179.  xTaskCreate((TaskFunction_t )HumidifierControlTask,         //任务入
口地址
180.      (const char*    )"HumidifierControlTask",       //任务名
称
181.      (uint16_t       )128,               //任务堆栈大小
182.      (void*          )NULL,              //任务输入参数
183.      (UBaseType_t    )3,             //任务优先级
184.      (TaskHandle_t*  )&HumidifierControlTask_Handler);      //任务句
柄
185.  //创建加热控制任务
186.  xTaskCreate((TaskFunction_t )HeatingControlTask,         //任务入口地
址
187.      (const char*    )"HeatingControlTask",        //任务名称
188.      (uint16_t       )128,               //任务堆栈大小
189.      (void*          )NULL,              //任务输入参数
190.      (UBaseType_t    )3,             //任务优先级
191.      (TaskHandle_t*  )&HeatingControlTask_Handler);       //任务句柄
192.  //创建浇水控制任务
193.  xTaskCreate((TaskFunction_t )WateringControlTask,         //任务入口地
址
194.      (const char*    )"WateringControlTask",       //任务名
称
195.      (uint16_t       )128,               //任务堆栈大小
196.      (void*          )NULL,              //任务输入参数
197.      (UBaseType_t    )3,             //任务优先级
198.      (TaskHandle_t*  )&WateringControlTask_Handler);      //任务句柄
199.  //创建补光控制任务
200.  xTaskCreate((TaskFunction_t )LightControlTask,         //任务入口地
址
201.      (const char*    )"LightControlTask",        //任务名称
202.      (uint16_t       )128,               //任务堆栈大小
203.      (void*          )NULL,              //任务输入参数
204.      (UBaseType_t    )3,             //任务优先级
```

```
205.     (TaskHandle_t*  )&LightControlTask_Handler);  //任务句柄
206.  //创建oled 显示任务
207.  xTaskCreate((TaskFunction_t )OledDisplayTask, //任务入口地
址
208.     (const char*    )"OledDisplayTask",         //任务名称
209.     (uint16_t       )512,              //任务堆栈大小
210.     (void*          )NULL,             //任务输入参数
211.     (UBaseType_t    )4,            //任务优先级
212.     (TaskHandle_t*  )&OledDisplayTask_Handler);      //任务句柄
213.  VentilationControl_Close();
214.  ServoControl_Close();
215.  HumidifierControl_Close();
216.  HeatingControl_Close();
217.  WateringControl_Close();
218.  LightControl_Close();
219.  Dev_status__Struct.Heating_flag  =0;
220.  Dev_status__Struct.Humidifier_flag =0;
221.  Dev_status__Struct.Servo_flag  =0;
222.  Dev_status__Struct.Vent_flag  =0;
223.  Dev_status__Struct.Watering_flag =0;
224.  Dev_status__Struct.Light_flag  =0;
225.  Env_Para_Range_Struct.MAX_CO2_Concentration = 1500  ;
226.  Env_Para_Range_Struct.MIN_CO2_Concentration = 750  ;
227.  Env_Para_Range_Struct.MAX_humidity   = 70  ;
228.  Env_Para_Range_Struct.MIN_humidity   = 60  ;
229.  Env_Para_Range_Struct.MAX_Light_Intensity   = 100  ;
230.  Env_Para_Range_Struct.MIN_Light_Intensity   = 80  ;
231.  Env_Para_Range_Struct.MAX_Soil_Moisture  = 60  ;
232.  Env_Para_Range_Struct.MIN_Soil_Moisture  = 30  ;
233.  Env_Para_Range_Struct.MAX_temperature  = 27  ;
234.  Env_Para_Range_Struct.MIN_temperature  = 23  ;
235.  vTaskSuspend(AutoModeTask_Handler);
236.  vTaskResume(ManualModTask_Handler);
237.     vTaskDelete(StartTask_Handler);            //删除开始任务
238.     taskEXIT_CRITICAL();            //退出临界区
239. }
240. /*****************串口命令任务*****************/
241. void UsartCommandTask(void *pvParameters)
242. {
243.     while(1)
244.     {
245.   if(usart_idle_flag == 1)
```

```
246.   {
247.    if(strcmp((const char*)USART_RX_BUF, "switch auto") ==  0)
248.    {
249.     vTaskSuspend(ManualModTask_Handler);
250.     vTaskResume(AutoModeTask_Handler);
251.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
252.     rx_cnt = 0;
253.     usart_idle_flag = 0 ;
254.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
255.     printf("已切换自动模式\n");
256.     xSemaphoreGive(UsartMuxSem_Handle);
257.    }
258.    else if(strcmp((const char*)USART_RX_BUF, "switch manual") == 0)
259.    {
260.     vTaskSuspend(AutoModeTask_Handler);
261.     vTaskResume(ManualModTask_Handler);
262.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
263.     rx_cnt = 0;
264.     usart_idle_flag = 0 ;
265.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
266.     printf("已切换手动模式\n");
267.     xSemaphoreGive(UsartMuxSem_Handle);
268.    }
269.    else if(strcmp((const char*)USART_RX_BUF, "clean") == 0)
270.    {
271.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
272.     rx_cnt = 0;
273.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
274.     printf("缓冲区已清除\n");
275.     xSemaphoreGive(UsartMuxSem_Handle);
276.     usart_idle_flag = 0 ;
277.    }
278.   }
279.   vTaskDelay(100);
280.  }
281. }
282. /*****************数据采集任务*****************/
283. void DataAcquisitionTask(void *pvParameters)
284. {
285.     while(1)
286.     {
287.   xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
```

```
288.    DHT11_Read_Data(&(Env_Data_Struct.temperature),&(Env_Data_Struct.h
umidity));//采集温度，湿度
289.    SGP30_Write(0x20,0x08);
290.    Env_Data_Struct.sgp30_dat = SGP30_Read();//采集二氧化碳浓度
291.    Env_Data_Struct.CO2Data = (Env_Data_Struct.sgp30_dat & 0xffff0000)
 >> 16;
292.    Env_Data_Struct.Light_Intensity =Get_Adc(5);//采集光照强度
293.    Env_Data_Struct.Soil_Moisture = Get_Adc2(4);//采集土壤湿度
294.
295.    printf("T:%.2fC",Env_Data_Struct.temperature);
296.    printf(" H:%d%%RH\n",(int)Env_Data_Struct.humidity);
297.    printf("L:%d",Env_Data_Struct.Light_Intensity);
298.    printf("SM:%d\n",Env_Data_Struct.Soil_Moisture);
299.    printf("CO2:%dppm\n",Env_Data_Struct.CO2Data);
300.    printf("Heat:");
301.    if(Dev_status__Struct.Heating_flag == 1)
302.    {
303.     printf("ON\n");
304.    }
305.    else
306.    {
307.     printf("OFF\n");
308.    }
309.    printf("Vent:");
310.    if(Dev_status__Struct.Servo_flag == 1)
311.    {
312.     printf("ON\n");
313.    }
314.    else
315.    {
316.     printf("OFF\n");
317.    }
318.
319.    printf("Fan:");
320.    if(Dev_status__Struct.Vent_flag == 1)
321.    {
322.     printf("ON\n");
323.    }
324.    else
325.    {
326.     printf("OFF\n");
327.    }
```

```
328.    printf("Humid:");
329.    if(Dev_status__Struct.Humidifier_flag == 1)
330.    {
331.     printf("ON\n");
332.    }
333.    else
334.    {
335.     printf("OFF\n");
336.    }
337.
338.    printf("Water:");
339.    if(Dev_status__Struct.Watering_flag == 1)
340.    {
341.     printf("ON\n");
342.    }
343.    else
344.    {
345.     printf("OFF\n");
346.    }
347.
348.    printf("LED:");
349.    if(Dev_status__Struct.Light_flag == 1)
350.    {
351.     printf("ON\n");
352.    }
353.    else
354.    {
355.     printf("OFF\n");
356.    }
357.    xSemaphoreGive(UsartMuxSem_Handle);
358.    vTaskDelay(5000);
359.      }
360. }
361. /*****************自动模式任务*****************/
362. void AutoModeTask(void *pvParameters)
363. {
364.
365.      while(1)
366.      { if(usart_idle_flag == 1)
367.    {
368.     if(strcmp((const char*)USART_RX_BUF, "checkout 1 ") == 0)
369.     {
```

```
370.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
371.      rx_cnt = 0;
372.      usart_idle_flag = 0 ;
373.      Env_Para_Range_Struct.MAX_CO2_Concentration = 1500  ;
374.      Env_Para_Range_Struct.MIN_CO2_Concentration = 750  ;
375.      Env_Para_Range_Struct.MAX_humidity    = 70  ;
376.      Env_Para_Range_Struct.MIN_humidity    = 60  ;
377.      Env_Para_Range_Struct.MAX_Light_Intensity    = 100  ;
378.      Env_Para_Range_Struct.MIN_Light_Intensity    = 80  ;
379.      Env_Para_Range_Struct.MAX_Soil_Moisture  = 60  ;
380.      Env_Para_Range_Struct.MIN_Soil_Moisture  = 30  ;
381.      Env_Para_Range_Struct.MAX_temperature  = 27  ;
382.      Env_Para_Range_Struct.MIN_temperature  = 23  ;
383.      xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
384.      printf("自动模式：正在切换至 番茄 环境\n");
385.      xSemaphoreGive(UsartMuxSem_Handle);
386.     }
387.     else if(strcmp((const char*)USART_RX_BUF, "checkout 2 ") == 0)
388.     {
389.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
390.      rx_cnt = 0;
391.      usart_idle_flag = 0 ;
392.      Env_Para_Range_Struct.MAX_CO2_Concentration = 750  ;
393.      Env_Para_Range_Struct.MIN_CO2_Concentration = 550  ;
394.      Env_Para_Range_Struct.MAX_humidity    = 80  ;
395.      Env_Para_Range_Struct.MIN_humidity    = 60  ;
396.      Env_Para_Range_Struct.MAX_Light_Intensity    = 70  ;
397.      Env_Para_Range_Struct.MIN_Light_Intensity    = 50  ;
398.      Env_Para_Range_Struct.MAX_Soil_Moisture  = 60  ;
399.      Env_Para_Range_Struct.MIN_Soil_Moisture  = 30  ;
400.      Env_Para_Range_Struct.MAX_temperature  = 22  ;
401.      Env_Para_Range_Struct.MIN_temperature  = 17  ;
402.      xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
403.      printf("自动模式：正在切换至 草莓 环境\n");
404.      xSemaphoreGive(UsartMuxSem_Handle);
405.     }
406.     else if(strcmp((const char*)USART_RX_BUF, "checkout 3 ") == 0)
407.     {
408.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
409.      rx_cnt = 0;
410.      usart_idle_flag = 0 ;
411.      Env_Para_Range_Struct.MAX_CO2_Concentration = 1200  ;
```

```
412.      Env_Para_Range_Struct.MIN_CO2_Concentration = 800  ;
413.      Env_Para_Range_Struct.MAX_humidity    = 70  ;
414.      Env_Para_Range_Struct.MIN_humidity    = 60  ;
415.      Env_Para_Range_Struct.MAX_Light_Intensity    = 80  ;
416.      Env_Para_Range_Struct.MIN_Light_Intensity    = 60  ;
417.      Env_Para_Range_Struct.MAX_Soil_Moisture  = 60  ;
418.      Env_Para_Range_Struct.MIN_Soil_Moisture  = 30  ;
419.      Env_Para_Range_Struct.MAX_temperature  = 20  ;
420.      Env_Para_Range_Struct.MIN_temperature  = 15  ;
421.      xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY) ;
422.      printf("自动模式：正在切换至 郁金香 环境\n");
423.      xSemaphoreGive(UsartMuxSem_Handle);
424.     }
425.     else if(strcmp((const char*)USART_RX_BUF, "checkout 4 ") == 0)
426.     {
427.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
428.      rx_cnt = 0;
429.      usart_idle_flag = 0 ;
430.      Env_Para_Range_Struct.MAX_CO2_Concentration = 2000  ;
431.      Env_Para_Range_Struct.MIN_CO2_Concentration = 1200 ;
432.      Env_Para_Range_Struct.MAX_humidity    = 50  ;
433.      Env_Para_Range_Struct.MIN_humidity    = 30  ;
434.      Env_Para_Range_Struct.MAX_Light_Intensity    = 100  ;
435.      Env_Para_Range_Struct.MIN_Light_Intensity    = 90  ;
436.      Env_Para_Range_Struct.MAX_Soil_Moisture  = 60  ;
437.      Env_Para_Range_Struct.MIN_Soil_Moisture  = 30  ;
438.      Env_Para_Range_Struct.MAX_temperature  = 20  ;
439.      Env_Para_Range_Struct.MIN_temperature  = 16  ;
440.      xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
441.      printf("自动模式：正在切换至 韭菜 环境\n");
442.      xSemaphoreGive(UsartMuxSem_Handle);
443.     }
444.    }
445.    if(Env_Data_Struct.CO2Data > Env_Para_Range_Struct.MAX_CO2_Concent
ration && (Dev_status__Struct.Servo_flag == 0 || Dev_status__Struct.Vent_
flag == 0))
446.    {
447.     ServoControl_Open();//开启通风口
448.     VentilationControl_Open();//开启通风扇
449.     Dev_status__Struct.Servo_flag = 1;
450.     Dev_status__Struct.Vent_flag  = 1;
451.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
```

```
452.     printf("自动模式：二氧化碳浓度过高，已开启 通风扇与通风口\n");
453.     xSemaphoreGive(UsartMuxSem_Handle);
454.   }
455.   else if(Env_Data_Struct.CO2Data < Env_Para_Range_Struct.MIN_CO2_Co
ncentration && (Dev_status__Struct.Servo_flag == 1 || Dev_status__Struct.
Vent_flag == 1))
456.   {
457.     ServoControl_Close();//关闭通风口
458.     VentilationControl_Close();//关闭通风扇
459.     Dev_status__Struct.Servo_flag = 0;
460.     Dev_status__Struct.Vent_flag  = 0;
461.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
462.     printf("自动模式：二氧化碳浓度过低，已关闭 通风扇与通风口\n");
463.     xSemaphoreGive(UsartMuxSem_Handle);
464.   }
465. //  else if(Dev_status__Struct.Servo_flag == 0 || Dev_status__Struct
.Vent_flag == 1)
466. //  {
467. //    ServoControl_Open();//开启通风口
468. //    VentilationControl_Close();//关闭通风扇
469. //    Dev_status__Struct.Servo_flag = 1;
470. //    Dev_status__Struct.Vent_flag  = 0;
471. //    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
472. //    printf("自动模式：二氧化碳浓度适宜，已关闭 通风扇与通风口\n");
473. //    xSemaphoreGive(UsartMuxSem_Handle);
474. //  }
475.   if((Env_Data_Struct.humidity > Env_Para_Range_Struct.MAX_humidity)
 && (Dev_status__Struct.Humidifier_flag == 1))
476.   {
477.     HumidifierControl_Close();//关闭加湿器
478.     Dev_status__Struct.Humidifier_flag  = 0;
479.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
480.     printf("自动模式：空气湿度过高，已关闭 加湿器 \n");
481.     xSemaphoreGive(UsartMuxSem_Handle);
482.   }
483.   else if((Env_Data_Struct.humidity < Env_Para_Range_Struct.MIN_humi
dity) && (Dev_status__Struct.Humidifier_flag == 0))
484.   {
485.     HumidifierControl_Open();//开启加湿器
486.     Dev_status__Struct.Humidifier_flag  = 1;
487.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
488.     printf("自动模式：空气湿度过低，已开启 加湿器 \n");
```

```
489.     xSemaphoreGive(UsartMuxSem_Handle);
490.    }
491.    if((Env_Data_Struct.Light_Intensity > Env_Para_Range_Struct.MAX_Li
ght_Intensity) && (Dev_status__Struct.Light_flag == 1))
492.    {
493.     LightControl_Close();//关闭补光灯
494.     Dev_status__Struct.Light_flag  = 0;
495.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
496.     printf("自动模式：光照强度过高，已关闭 补光灯 \n");
497.     xSemaphoreGive(UsartMuxSem_Handle);
498.    }
499.    else if((Env_Data_Struct.Light_Intensity < Env_Para_Range_Struct.M
IN_Light_Intensity) && (Dev_status__Struct.Light_flag == 0))
500.    {
501.     LightControl_Open();//开启补光灯
502.     Dev_status__Struct.Light_flag  = 1;
503.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
504.     printf("自动模式：光照强度过低，已开启 补光灯 \n");
505.     xSemaphoreGive(UsartMuxSem_Handle);
506.    }
507.
508.    if((Env_Data_Struct.Soil_Moisture > Env_Para_Range_Struct.MAX_Soil
_Moisture) && (Dev_status__Struct.Watering_flag == 1))
509.    {
510.     WateringControl_Close();//关闭浇水机
511.     Dev_status__Struct.Watering_flag  = 0;
512.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
513.     printf("自动模式：土壤湿度过高，已关闭 浇水机 \n");
514.     xSemaphoreGive(UsartMuxSem_Handle);
515.    }
516.    else if((Env_Data_Struct.Soil_Moisture < Env_Para_Range_Struct.MIN
_Soil_Moisture) && (Dev_status__Struct.Watering_flag == 0))
517.    {
518.     WateringControl_Open();//开启浇水机
519.     Dev_status__Struct.Watering_flag  = 1;
520.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
521.     printf("自动模式：土壤湿度过低，已开启 浇水机 \n");
522.     xSemaphoreGive(UsartMuxSem_Handle);
523.    }
524. //  else if(Dev_status__Struct.Watering_flag == 1)
525. //  {
526. //   WateringControl_Close();//关闭浇水机
```

```
527. //   Dev_status__Struct.Watering_flag  = 0;
528. //   xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
529. //   printf("自动模式：土壤湿度适宜，已关闭 浇水机 \n");
530. //   xSemaphoreGive(UsartMuxSem_Handle);
531. //  }
532.   if((Env_Data_Struct.temperature > Env_Para_Range_Struct.MAX_temper
ature) && Dev_status__Struct.Servo_flag == 0 && Dev_status__Struct.Vent_f
lag == 1 && Dev_status__Struct.Heating_flag == 1)
533.   {
534.    ServoControl_Open();//开启通风口
535.    VentilationControl_Open();//开启通风扇
536.    HeatingControl_Close();//关闭加热器
537.    Dev_status__Struct.Servo_flag  = 1;
538.    Dev_status__Struct.Vent_flag   = 1;
539.    Dev_status__Struct.Heating_flag = 0;
540.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
541.    printf("自动模式：空气温度过高，已开启 通风扇,通风口 已关闭 加热器
\n");
542.    xSemaphoreGive(UsartMuxSem_Handle);
543.   }
544.   else if((Env_Data_Struct.temperature < Env_Para_Range_Struct.MIN_t
emperature) && Dev_status__Struct.Heating_flag == 0)
545.   {
546.    HeatingControl_Open();//开启加热器
547.    Dev_status__Struct.Heating_flag  = 1;
548.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
549.    printf("自动模式：空气温度过低，已开启 加热器 \n");
550.    xSemaphoreGive(UsartMuxSem_Handle);
551.   }
552. //  else if(Dev_status__Struct.Heating_flag == 1)
553. //  {
554. //   HeatingControl_Close();//关闭加热器
555. //   Dev_status__Struct.Heating_flag  = 0;
556. //   xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
557. //   printf("自动模式：空气温度适宜,已关闭 加热器 \n");
558. //   xSemaphoreGive(UsartMuxSem_Handle);
559. //  }
560.
561.
562.   vTaskDelay(1000);
563.     }
564. }
```

```
565. /*****************手动模式任务******************/
566. void ManualModTask(void *pvParameters)
567. {
568.     while(1)
569.     { if(usart_idle_flag == 1)
570.   {
571.    if(strcmp((const char*)USART_RX_BUF, "open Vent") == 0)
572.    {
573.     xTaskNotify( VentilationControlTask_Handler,DEVICE_OPEN, eSetBit
s );
574.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
575.     rx_cnt = 0;
576.     usart_idle_flag = 0 ;
577.    }
578.    else if(strcmp((const char*)USART_RX_BUF, "close Vent") == 0)
579.    {
580.     xTaskNotify( VentilationControlTask_Handler,DEVICE_CLOSE, eSetBi
ts );
581.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
582.     rx_cnt = 0;
583.     usart_idle_flag = 0 ;
584.    }
585.    else if(strcmp((const char*)USART_RX_BUF, "open Servo") == 0)
586.    {
587.     xTaskNotify( ServoControlTask_Handler,DEVICE_OPEN, eSetBits );
588.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
589.     rx_cnt = 0;
590.     usart_idle_flag = 0 ;
591.    }
592.    else if(strcmp((const char*)USART_RX_BUF, "close Servo") == 0)
593.    {
594.     xTaskNotify( ServoControlTask_Handler,DEVICE_CLOSE, eSetBits );
595.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
596.     rx_cnt = 0;
597.     usart_idle_flag = 0 ;
598.    }
599.    else if(strcmp((const char*)USART_RX_BUF, "open Humid") == 0)
600.    {
601.     xTaskNotify( HumidifierControlTask_Handler,DEVICE_OPEN, eSetBits
 );
602.     memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
603.     rx_cnt = 0;
```

```
604.      usart_idle_flag = 0 ;
605.     }
606.     else if(strcmp((const char*)USART_RX_BUF, "close Humid") == 0)
607.     {
608.      xTaskNotify( HumidifierControlTask_Handler,DEVICE_CLOSE, eSetBit
s );
609.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
610.      rx_cnt = 0;
611.      usart_idle_flag = 0 ;
612.     }
613.     else if(strcmp((const char*)USART_RX_BUF, "open Heat") == 0)
614.     {
615.      xTaskNotify( HeatingControlTask_Handler,DEVICE_OPEN, eSetBits );
616.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
617.      rx_cnt = 0;
618.      usart_idle_flag = 0 ;
619.     }
620.     else if(strcmp((const char*)USART_RX_BUF, "close Heat") == 0)
621.     {
622.      xTaskNotify( HeatingControlTask_Handler,DEVICE_CLOSE, eSetBits )
;
623.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
624.      rx_cnt = 0;
625.      usart_idle_flag = 0 ;
626.     }
627.     else if(strcmp((const char*)USART_RX_BUF, "open Water") == 0)
628.     {
629.      xTaskNotify( WateringControlTask_Handler,DEVICE_OPEN, eSetBits )
;
630.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
631.      rx_cnt = 0;
632.      usart_idle_flag = 0 ;
633.     }
634.     else if(strcmp((const char*)USART_RX_BUF, "close Water") == 0)
635.     {
636.      xTaskNotify( WateringControlTask_Handler,DEVICE_CLOSE, eSetBits
);
637.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
638.      rx_cnt = 0;
639.      usart_idle_flag = 0 ;
640.     }
641.     else if(strcmp((const char*)USART_RX_BUF, "open Light") == 0)
```

```
642.     {
643.      xTaskNotify( LightControlTask_Handler,DEVICE_OPEN, eSetBits );
644.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
645.      rx_cnt = 0;
646.      usart_idle_flag = 0 ;
647.     }
648.     else if(strcmp((const char*)USART_RX_BUF, "close Light") == 0)
649.     {
650.      xTaskNotify( LightControlTask_Handler,DEVICE_CLOSE, eSetBits );
651.      memset(USART_RX_BUF,0,sizeof(USART_RX_BUF));
652.      rx_cnt = 0;
653.      usart_idle_flag = 0 ;
654.     }
655.    }
656.    vTaskDelay(100);
657.      }
658. }
659. /*****************通风控制任务*****************/
660. void VentilationControlTask(void *pvParameters)
661. {
662.  uint32_t VentilationControl_Notify = 0;
663.     while(1)
664.     {
665.    xTaskNotifyWait(0,DEVICE_OPEN | DEVICE_CLOSE ,&VentilationControl_
Notify,portMAX_DELAY);
666.    if(VentilationControl_Notify == DEVICE_OPEN)
667.    {
668.     VentilationControl_Open();
669.     Dev_status__Struct.Vent_flag = OPEN;
670.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
671.     printf("手动控制 通风扇已开\n");
672.     xSemaphoreGive(UsartMuxSem_Handle);
673.    }
674.    if(VentilationControl_Notify == DEVICE_CLOSE)
675.    {
676.     VentilationControl_Close();
677.     Dev_status__Struct.Vent_flag = CLOSE;
678.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
679.     printf("手动控制 通风扇已关\n");
680.     xSemaphoreGive(UsartMuxSem_Handle);
681.    }
682.      }
```

```
683. }
684. /*****************舵机控制任务*****************/
685. void ServoControlTask(void *pvParameters)
686. {
687.  uint32_t ServoControl_Notify = 0;
688.     while(1)
689.     {
690.   xTaskNotifyWait(0,DEVICE_OPEN | DEVICE_CLOSE ,&ServoControl_Notify
,portMAX_DELAY);
691.   if(ServoControl_Notify == DEVICE_OPEN)
692.   {
693.    ServoControl_Open();
694.    Dev_status__Struct.Servo_flag = OPEN;
695.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
696.    printf("手动控制 通风口已开\n");
697.    xSemaphoreGive(UsartMuxSem_Handle);
698.   }
699.   if(ServoControl_Notify == DEVICE_CLOSE)
700.   {
701.    ServoControl_Close();
702.    Dev_status__Struct.Servo_flag = CLOSE;
703.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
704.    printf("手动控制 通风口已关\n");
705.    xSemaphoreGive(UsartMuxSem_Handle);
706.   }
707.     }
708. }
709. /*****************加湿器控制任务****************/
710. void HumidifierControlTask(void *pvParameters)
711. {
712.  uint32_t HumidifierControl_Notify = 0;
713.     while(1)
714.     {
715.   xTaskNotifyWait(0,DEVICE_OPEN | DEVICE_CLOSE ,&HumidifierControl_N
otify,portMAX_DELAY);
716.   if(HumidifierControl_Notify == DEVICE_OPEN)
717.   {
718.    HumidifierControl_Open();
719.    Dev_status__Struct.Humidifier_flag = OPEN;
720.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
721.    printf("手动控制 加湿器已开\n");
722.    xSemaphoreGive(UsartMuxSem_Handle);
```

```
723.    }
724.    if(HumidifierControl_Notify == DEVICE_CLOSE)
725.    {
726.     HumidifierControl_Close();
727.     Dev_status__Struct.Humidifier_flag = CLOSE;
728.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
729.     printf("手动控制 加湿器已关\n");
730.     xSemaphoreGive(UsartMuxSem_Handle);
731.    }
732.      }
733. }
734. /*****************加热控制任务****************/
735. void HeatingControlTask(void *pvParameters)
736. {
737.  uint32_t HeatingControl_Notify = 0;
738.     while(1)
739.     {
740.   xTaskNotifyWait(0,DEVICE_OPEN | DEVICE_CLOSE ,&HeatingControl_Noti
fy,portMAX_DELAY);
741.    if(HeatingControl_Notify == DEVICE_OPEN)
742.    {
743.     HeatingControl_Open();
744.     Dev_status__Struct.Heating_flag = OPEN;
745.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
746.     printf("手动控制 加热器已开\n");
747.     xSemaphoreGive(UsartMuxSem_Handle);
748.    }
749.    if(HeatingControl_Notify == DEVICE_CLOSE)
750.    {
751.     HeatingControl_Close();
752.     Dev_status__Struct.Heating_flag = CLOSE;
753.     xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
754.     printf("手动控制 加热器已关\n");
755.     xSemaphoreGive(UsartMuxSem_Handle);
756.    }
757.      }
758. }
759. /*****************浇水控制任务****************/
760. void WateringControlTask(void *pvParameters)
761. {
762.  uint32_t WateringControl_Notify = 0;
763.     while(1)
```

```
764.     {
765.   xTaskNotifyWait(0,DEVICE_OPEN | DEVICE_CLOSE ,&WateringControl_Not
ify,portMAX_DELAY);
766.   if(WateringControl_Notify == DEVICE_OPEN)
767.   {
768.    WateringControl_Open();
769.    Dev_status__Struct.Watering_flag = OPEN;
770.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
771.    printf("手动控制 浇水机已开\n");
772.    xSemaphoreGive(UsartMuxSem_Handle);
773.   }
774.   if(WateringControl_Notify == DEVICE_CLOSE)
775.   {
776.    WateringControl_Close();
777.    Dev_status__Struct.Watering_flag = CLOSE;
778.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
779.    printf("手动控制 浇水机已关\n");
780.    xSemaphoreGive(UsartMuxSem_Handle);
781.    }
782.     }
783. }
784. /****************补光控制任务****************/
785. void LightControlTask(void *pvParameters)
786. {
787.  uint32_t LightControl_Notify = 0;
788.     while(1)
789.     {
790.   xTaskNotifyWait(0,DEVICE_OPEN | DEVICE_CLOSE ,&LightControl_Notify
,portMAX_DELAY);
791.   if(LightControl_Notify == DEVICE_OPEN)
792.   {
793.    LightControl_Open();
794.    Dev_status__Struct.Light_flag = OPEN;
795.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
796.    printf("手动控制 补光灯已开\n");
797.    xSemaphoreGive(UsartMuxSem_Handle);
798.   }
799.   if(LightControl_Notify == DEVICE_CLOSE)
800.   {
801.    LightControl_Close();
802.    Dev_status__Struct.Light_flag = CLOSE;
803.    xSemaphoreTake(UsartMuxSem_Handle,portMAX_DELAY);
```

```
804.     printf("手动控制 补光灯已关\n");
805.     xSemaphoreGive(UsartMuxSem_Handle);
806.    }
807.      }
808. }
809. /*****************OLED 显示任务****************/
810. void OledDisplayTask(void *pvParameters)
811. {
812.  uint16_t cnt = 0;
813.     while(1)
814.     { cnt ++ ;
815.   if(cnt < 11)
816.   {
817.    OLED_ShowString(1, 1,"T:");
818.    OLED_ShowString(1, 5,".");
819.    OLED_ShowString(1, 7,"C  ");
820.    OLED_ShowString(1, 10,"H:");
821.    OLED_ShowString(1, 14,"%RH");
822.    OLED_ShowString(2, 1,"Light:");
823.    OLED_ShowString(2, 9,"        ");
824.    OLED_ShowString(3, 1,"CO2  :");
825.    OLED_ShowString(3, 11," ppm  ");
826.    OLED_ShowString(4, 1,"SM   :");
827.    OLED_ShowString(4, 9,"   %RH  ");
828.    OLED_ShowNum(1, 3, (int)Env_Data_Struct.temperature, 2);
829.    OLED_ShowNum(1, 6, (int)(Env_Data_Struct.temperature*100)%100, 1)
;
830.    OLED_ShowNum(1, 12, Env_Data_Struct.humidity, 2);
831.    OLED_ShowNum(2, 7, Env_Data_Struct.Light_Intensity, 2);
832.    OLED_ShowNum(3, 7, Env_Data_Struct.CO2Data, 4);
833.    OLED_ShowNum(4, 7, Env_Data_Struct.Soil_Moisture, 2);
834.    vTaskDelay(500);
835.   }
836.   else
837.   {
838.    OLED_ShowString(1, 1,"Heat:");
839.    if(Dev_status__Struct.Heating_flag == 1)
840.    {
841.     OLED_ShowString(1, 6,"ON ");
842.    }
843.    else
844.    {
```

```
845.     OLED_ShowString(1, 6,"OF ");
846.    }
847.    OLED_ShowString(1, 9,"Servo:");
848.    if(Dev_status__Struct.Servo_flag == 1)
849.    {
850.     OLED_ShowString(1, 15,"ON");
851.    }
852.    else
853.    {
854.     OLED_ShowString(1, 15,"OF");
855.    }
856.
857.    OLED_ShowString(2, 1,"Vent:");
858.    if(Dev_status__Struct.Vent_flag == 1)
859.    {
860.     OLED_ShowString(2, 6,"ON ");
861.    }
862.    else
863.    {
864.     OLED_ShowString(2, 6,"OF ");
865.    }
866.    OLED_ShowString(2, 9,"Humid:");
867.    if(Dev_status__Struct.Humidifier_flag == 1)
868.    {
869.     OLED_ShowString(2, 15,"ON");
870.    }
871.    else
872.    {
873.     OLED_ShowString(2, 15,"OF");
874.    }
875.
876.    OLED_ShowString(3, 1,"Water:");
877.    if(Dev_status__Struct.Watering_flag == 1)
878.    {
879.     OLED_ShowString(3, 7,"ON        ");
880.    }
881.    else
882.    {
883.     OLED_ShowString(3, 7,"OF        ");
884.    }
885.
886.    OLED_ShowString(4, 1,"Light:");
```

```
887.    if(Dev_status__Struct.Light_flag == 1)
888.    {
889.     OLED_ShowString(4, 7,"ON        ");
890.    }
891.    else
892.    {
893.     OLED_ShowString(4, 7,"OF        ");
894.    }
895.    vTaskDelay(500);
896.   }
897.   if(cnt == 19)
898.   {
899.    cnt = 0 ;
900.   }
901.
902.     }
903.  }
```