

**本科毕业设计（论文）**

题 目： 基于STM32单片机的物联网农业检测控制与控制系统的设计

学 院： 机械电子工程学院

专 业： 电气工程及其自动化

班 级： 1班

学生姓名： 赵国承

学生学号： 1602120130

指导教师： 潘莹月

完成日期： 2020.03.10

山东农业工程学院制

毕业设计（论文）诚信声明书

本人提交的毕业设计（论文）《基于STM32单片机的物联网农业检测控制与控制系统的设计》是在指导教师指导下独立研究、写作的成果，论文中引用他人的研究成果，均已在论文中加以注释说明；指导教师和其他人员对本文提出，并被本文采纳的修改意见和建议，均已在我的谢辞中加以说明并深致谢意。

设计（论文）作者（签字）                   年    月    日

指导教师（签字）                       年    月    日

毕业设计（论文）版权使用授权书

本毕业设计（论文）《基于STM32单片机的物联网农业检测控制与控制系统的设计》是本人在校期间所完成学业的组成部分，是在山东农业工程学院教师的指导下完成的。本人特授权山东农业工程学院可将本毕业设计（论文）的全部或部分内容编入有关书籍、数据库保存，可采用复制、印刷、网页制作等方式将设计（论文）文本和经过编辑、批注等处理的设计（论文）文本提供给读者查阅、参考，可向有关学术部门和国家有关教育主管部门呈送复印件和电子文档。

设计（论文）作者 （签字）               年    月    日

指导教师（签字）                  年    月    日

目  录

[摘要 I](#_Toc34748035)

[关键词 I](#_Toc34748036)

[Abstract II](#_Toc34748037)

[Key words II](#_Toc34748038)

[1.绪论 1](#_Toc34748039)

[1.1 选题背景和意义 1](#_Toc34748040)

[1.2 农业物联网的国内外研究现状 1](#_Toc34748041)

[1.3 设计研究内容 2](#_Toc34748042)

[1.4 研究方法及技术路线 2](#_Toc34748043)

[1.4.1 研究方法 2](#_Toc34748044)

[1.4.2 技术路线 3](#_Toc34748045)

[1.5 系统的组成 4](#_Toc34748046)

[2.系统硬件的设计 5](#_Toc34748047)

[2.1 STM32单片机 5](#_Toc34748048)

[2.1.1 单片机选型 5](#_Toc34748049)

[2.1.2 STM32F103C8T6芯片简介 6](#_Toc34748050)

[2.2温度传感器 7](#_Toc34748051)

[2.2.1 DHT11温湿度传感器简介 7](#_Toc34748052)

[2.2.2 DHT11温湿度传感器硬件电路设计 8](#_Toc34748053)

[2.3 光强传感器 8](#_Toc34748054)

[2.3.1 光强传感器简介 8](#_Toc34748055)

[2.3.2 光强传感器硬件电路设计 8](#_Toc34748056)

[2.4 WIFI模块 9](#_Toc34748057)

[2.4.1 无线通信模块选型 9](#_Toc34748058)

[2.4.2 ESP8266硬件电路设计 10](#_Toc34748059)

[2.5 OLED显示屏 10](#_Toc34748060)

[2.5.1 显示屏选型 10](#_Toc34748061)

[2.5.2 OLED硬件电路设计 11](#_Toc34748062)

[2.5 继电器 11](#_Toc34748063)

[3.系统软件的设计 13](#_Toc34748064)

[3.1 系统软件设计总体方案 13](#_Toc34748065)

[3.2 系统软件开发环境 14](#_Toc34748066)

[3.2.1 STM32单片机开发环境 14](#_Toc34748067)

[3.2.2 APP开发环境 14](#_Toc34748068)

[3.3 云平台的应用 16](#_Toc34748069)

[3.3.1 机智云平台简介 16](#_Toc34748070)

[3.3.2 设备接入云平台开发 17](#_Toc34748071)

[3.4 主要功能模块程序设计 18](#_Toc34748072)

[3.4.1 温湿度传感器模块程序 18](#_Toc34748073)

[3.4.2 光强传感器模块程序 19](#_Toc34748074)

[3.4.3 OLED显示屏模块程序 20](#_Toc34748075)

[3.4.4 WIFI通信模块程序 20](#_Toc34748076)

[3.5 手机APP设计 21](#_Toc34748077)

[4.实验测试与结果分析 25](#_Toc34748078)

[4.1 虚拟设备测试 25](#_Toc34748079)

[4.2 真实设备调试 26](#_Toc34748080)

[4.3 测试结果与分析 27](#_Toc34748081)

[5.总结与展望 29](#_Toc34748082)

[5.1 主要设计成果总结 29](#_Toc34748083)

[5.2 系统特点总结 29](#_Toc34748084)

[5.3 展望 30](#_Toc34748085)

[谢辞 31](#_Toc34748086)

[参考文献 32](#_Toc34748087)

[附录A 硬件电路图 34](#_Toc34748088)

[附录B 程序 35](#_Toc34748089)

基于STM32单片机的物联网农业检测控制与控制系统的

设计

摘要

：随着物联网、云平台以及智能手机的快速发展，传统的农业生产方式已经难以满足当今时代进步的需要。农业作为基础支撑产业，如何能够实现实时获取农作物生长环境参数是确保农作物产量增长的关键。

基于此，本文设计了一款基于云平台的远程农业检测控制系统，该系统借助于云平台实现了手机APP对设备端的远程控制和实时监测。该系统主要由三个部分组成：手机APP、云平台、STM32单片机和ESP8266 WIFI模块相结合的设备端。首先，手机APP作为人机交互终端，提供了可视化操作界面，可以远程控制设备上led灯的亮灭以及蜂鸣器的开关；同时还可以接收来自设备端的光照强度信息和温湿度信息在APP界面上动态显示。其次，设备端使用STM32F103C8T6芯片作为核心控制终端，采用模块化设计，使用WIFI通信模块配置入网，利用温湿度和光强传感器对当前环境进行探测。为了使信息更加方便显示，外加了OLED显示屏，可以在显示屏上显示温湿度以及光强等信息。

最后，本设计通过使用虚拟设备和真实设备对系统可靠性进行了测试验证，验证结果满足本次设计的目的。

关键词

：远程监控  云平台  STM32  APP

Design of IoT agricultural detection control and control system based on STM32 microcontroller

Abstract

：With the rapid development of the Internet of Things, cloud platforms, and smart phones, traditional agricultural production methods have been difficult to meet the needs of the modern era. As a basic supporting industry, how to achieve real-time acquisition of crop growth environment parameters is the key to ensure the growth of crop yield.

Based on this, this paper designs a remote agricultural detection and control system based on the cloud platform. The system uses the cloud platform to realize the remote control and real-time monitoring of the device end by the mobile phone APP. The system is mainly composed of three parts: a mobile APP, a cloud platform, an STM32 microcontroller and an ESP8266 WIFI module combined device side. First, the mobile phone APP, as a human-computer interaction terminal, provides a visual operation interface, which can remotely control the on and off of the LED light on the device and the switch of the buzzer; at the same time, it can also receive the light intensity information and temperature and humidity information from the device end on the APP interface On dynamic display. Secondly, the device uses the STM32F103C8T6 chip as the core control terminal, adopts a modular design, uses a WIFI communication module to configure the network, and uses the temperature, humidity and light intensity sensors to detect the current environment. In order to make the information more convenient to display, an OLED display is added, which can display information such as temperature, humidity and light intensity on the display.

Finally, the design verified the system reliability by using virtual equipment and real equipment, and the verification results met the purpose of this design.

Key words

: Remote monitoring；cloud platform；STM32；APP

1.绪论

1.1 选题背景和意义

近年来，我国农业农村经济发展的国内和国际环境发生了很大变化，中央对“十二五”时期农业农村经济发展又提出了更高的要求，为适应工业化快速推进，必须加快农业现代化进程。“十二五”时期我国工业化、进程将继续加快，产业结构调整、城镇规模扩大和人口的持续增加等必将促使农产品需求刚性持续增长以及对农产品质量的更高要求。面对不断加大的农产品供给和质量安全的压力，农业发展面临着农村劳动力素质结构性下降、耕地约束加剧等突出问题，需要切实转变农业发展方式，加快农业现代化步伐。

对于传统的农业监测，都是需要有专门的人去察看农田信息。在此过程中所获的信息基本上都是靠农户自己以往的经验来确定农作物的实际情况，这种方法一是费时费力，需要农户专门去农田查看信息，占用农户完成其他农活的时间；二是得到的信息不够准确，只能根据以往的经验进行大概估测农田的情况[1]。所以通过物联网来远程进行农业监测与控制就显得很重要，它很好的解决了费时费力以及获得信息不准确的问题，它对于农作物质量的保证起到了十分关键的作用。

用现代信息技术改造传统农业、装备农业是实现农业现代化的重要途径。物联网技术是现代信息技术的新生力量，是推动信息化与农业现代化融合的重要切入点[2]。本设计是改变了传统的农业监测方式，运用了基于STM32单片机以及物联网远程通信的方式，可对目标控制农田实现准确感知、及时反馈、远程控制、提升农业决策指挥水平的作用。对于实现农业监测与控制、增加产量、提高农户收入具有不可或缺的重要意义。

1.2 农业物联网的国内外研究现状

近些年来，美国、日本、欧洲等一些发达国家和地区相继开展了一系列物联网技术在农业生活领域的应用研究，取得了很多优秀的成果，因此实现了物联网技术与农业生产、流通、加工等领域的关联实践，带动了物联网与相关产业的发展。例如在美国很多的农场主们利用物联网技术在防治自然灾害、虫患等方面已有投入的产品、并且表现优异。在以色列，土地资源稀缺，气候炎热，但是以其发达的物联网农业科技提高生产效率，其中完善的农业服务体系和高度发达的农业科技是以色列农业物联网发展具有不可比拟的优势，通过建立物联网滴灌技术，利用传感器技术回传滴灌的土壤数据，决定合适灌溉的时间，灌溉量的多少[3]。

在国内，我国大部分农户仍采用人工作业管理的方式进行农业活动，科技含量低，生产效率低，生产方式落后。在我国广大的农村地区，人们在从事农业生产活动中获取农作物与周围环境的信息有限，其主要方式是通过人力活动去获取。虽然近年来，我国通过在农业上运用物联网技术取得了相当不错的成果。但是与国外相比各方面还相对比较落后，总体还处于较低的水平。虽然已经有能够完全实现农业监测的产品被研发成功，例如我国研制了地面监测站和遥感技术结合的墒情监测系统等，但其应用范围、控制精度较国际先进水平相比还是有一定的差距。实现物联网农业系统在农业生产领域的广泛应用的目标仍旧需要努力[4]。

1.3 设计研究内容

根据当前我国农业自动化的程度和现实需求，本设计能够使农业监测控制水平实现高度的自动控制。即通过使用温湿度及光强传感器将数据反馈给STM32，STM32通过esp8266与云端进行通信，云端将数据下发到APP，进而实现实时检测控制的目的[5]。本设计主要的研究内容如下：

（1）系统概念层次设计：进行功能分析，对检测控制系统的构成、预计要实现的功能进行简单的规划设计。

（2）温湿度及光强传感器的选择：对各种适合测量温湿度及光强传感器进行分析，最终选择各方面相对性能较好的DHT11和GY-30传感器。

（3）硬件方面设计：对系统主控制器的选择、各种核心元器件的选择、在 Altium Designer当中进行硬件电路的设计。

（4）软件方面设计：利用 Keil uVision5为开发环境对单片机程序进行编写，利用Android Studio为开发环境编写APP端软件程序。

（5）检测与调试：在条件允许的范围内对系统进行检测与调试，对其传输的数据准确值进行评估，分析误差的来源，寻找能够优化测量数据的方案。

1.4 研究方法及技术路线

1.4.1 研究方法

文献资料法、经验总结法、归纳总结法。

通过采用上述方法能够很好的研究问题，具体如下：

（1）采用文献资料法分析农业监测控制系统的研究背景及意义、国内外应用研究现状。

（2）通过对农业监测控制系统中的具体情况进行归纳与分析，使之系统化、理论化。

（3）使用归纳总结法对监测控制内容总结归纳，并及时改进设计的不足。

1.4.2 技术路线

本设计的技术路线如图 1-1 所示

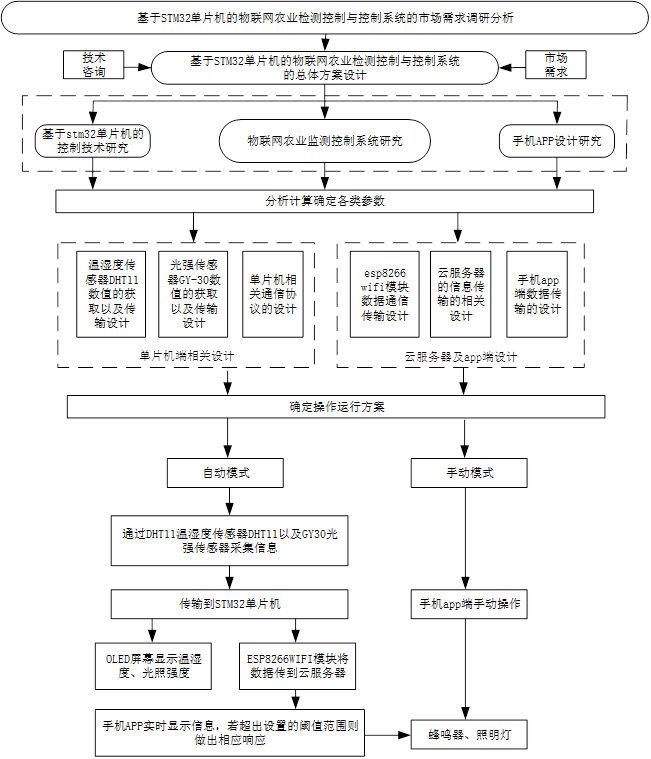


图 1-1 技术路线图

1.5 系统的组成

本设计主要由以下几个部分组成：数据采集端（DHT11、GY-30）、单片机检测平台、OLED显示部分、ESP8266数据传输部分、云平台处理以及手机APP实时显示控制部分。具体如图1-2所示：



图1-2 系统装置流程图

2.系统硬件的设计

本文设计的基于物联网的智能农业监测系统，目标是准确实时的获取农作物生长的环境信息，为科学研究提供全面的参考，实现农业的精准操作[6]。其硬件总体框图如图2-1所示：



图2-1 系统硬件总体框图

从图2-1可知，系统的硬件部分采用了两种传感器：DHT11温湿度传感器以及GY-30光强传感器，除此外STM32单片机还接有OLED屏幕和ESP8266 WIFI模块，还可以通过与单片机相连的继电器去控制蜂鸣器和灯。

2.1 STM32单片机

2.1.1 单片机选型

将运算器、控制器、存储器和各种输入输出接口等计算机的主要部件集成在一块芯片上,就能得到一个单芯片的微型计算机,它虽然只是一个芯片,但在组成和功能上己经具有了计算机系统的特点,因此称之为单片微型计算机简称单片机[7]。在当前使用的单片机控制芯片系列当中，STM32、51、TMS、MSP430、PIC、AVR、STC单片机被广泛使用。其比较信息见表2-1。

表2-1 主流单片机信息比较

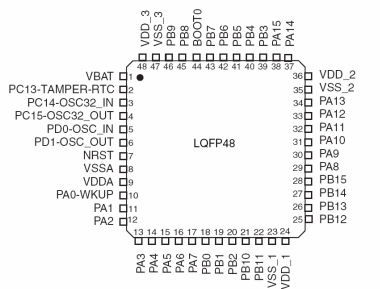
|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 特点/缺点 | 应用范围 | 代表产品 |
| 51单片机 | 特点：操作简单、价格低廉  缺点：运行速度过慢 | 在对性能要求不高的场合使用 | 8051、80C51 |
| MSP430单片机 | 特点：低功耗、速度快  缺点：资料较少、指令占空间较大 | 在低功耗及超低功耗的工业场合应用的比较多 | MSP430F系列等 |
| PIC单片机 | 特点：低工作电压、低功耗、驱动能力强  缺点：需要反复地选择对应的存储体 | 工业场合应用 | PIC16F873、PIC16F877 |
| AVR单片机 | 特点：高性能、高速度、低功耗  缺点：通用性较低 | 医疗设备、GPS等 | ATUC64L3U、AT90S8515 |
| STC单片机 | 特点：高速、低功耗、超强抗干扰  缺点：功能不及STM32 | 在对性能要求不高的场合使用 | STC12C2052AD |
| Freescale单片机 | 特点：低成本，高性能 | 主要应用在汽车领域 | MC9S12G系列 |
| STM32单片机 | 特点：高性能、 低成本、低功耗、高性价比 | 较其他单片机而言，应用极广泛 | STM32F103  系列 |

从表2-1我们不难看出，七款主流的单片机不管是性能还是应用领域都有所或多或少的差异，但是综合来看STM32系列的单片机，不仅目前市场价位非常低，而且其拥有丰富的外设接口、大型存储空间，使得该款单片机具有强大扩展和存储功能。最为关键的是，该款单片机技术成熟，所以这款单片机有丰富的资料可以查阅，这大大节约了开发的成本。故选择STM32系列的单片机。

2.1.2 STM32F103C8T6芯片简介

本次设计中我们选用的是STM32系列的STM32F103C8T6为主控芯片，其引脚分布如图2-2所示。STM32F103C8T6是一款基于ARM Cortex-M3内核STM32系列的32位的微控制器，程序存储器容量是64KB，包含2个12位的ADC、 3个通用16位定时器和1个PWM定时器，还包含标准和先进的通信接口：多达2个I2C接口和SPI接口、3个USART接口、一个USB接口和一个CAN接口。且由于其性价比很高，故可以完全使用其满足本设计的所有需求。

具体的硬件设计图见附录A。



2-2 STM32F103C8T6引脚分布

2.2温度传感器

2.2.1 DHT11温湿度传感器简介

DHT11数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性与卓越的长期稳定性。其相对湿度精度为±5%，温度为±2℃，相对湿度量程为20%～90%，温度量程为0～50℃[8]。

传感器包括一个NTC测温元件和一个电阻式感湿元件，并与一个高性能8位单片机相连接。其内部结构图如图2-3所示。DHT11内含两个重要电子元件,一是电阻式感湿元件用于测试周围环境的湿度参数,另一个是NTC测温元件用于测试温度参数。另外,DHT11中有一块一次性可编程OTP内存,用来存放校准系数。感湿元件和测温元件检测到参数后,调用OTP中的校准系数,进而得出精准的温度湿度检测值[9]。



图2-3 DHT11内部结构图和引脚说明

2.2.2 DHT11温湿度传感器硬件电路设计

如图2-4所示，DHT11外接3.3V电源，通过PB9引脚在4.7k上拉电阻的作用下与单片机相连。



图2-4 DHT11硬件连接图

2.3 光强传感器

2.3.1 光强传感器简介

GY-30光强传感器，内部使用的是BH1750FVI芯片，是一种用于两线式串行总线接口的数字型光强度传感器集成电路。BH1750的通信方式是采用双线 IIC总线通信，一个用于数据传输处理，另一个用于控制模块的时钟，通过时钟和脉冲变化决定数据传输的应答和接收[10]。这种集成电路可以根据收集的光线强度数据来调整液晶或者键盘背景灯的亮度。利用它的高分辨率可以探测较大光强度范围在1到65535lx，由于其检测范围较大，故可以满足本次设计的需要，其内部结构图如图2-5所示：



图2-5 GY-30内部结构图和引脚说明

2.3.2 光强传感器硬件电路设计

如图2-6所示，该传感器的外接3.3V电源，通过PB10和PB11引脚在10k上拉电阻的作用下与单片机相连。



图2-6 GY-30硬件连接图

2.4 WIFI模块

2.4.1 无线通信模块选型

根据本次设计的要求，选择的通信模块必须具有以下两个基本功能：一个是实现接收STM32通过串口发送过来的数据，一个是实现通过路由器发送接收的数据到云平台，综合考虑以后选择的是市面上比较流行的无线通信模块—正点原子公司生产的ATK-ESP8266模块。ATK-ESP8266是的一款高性能的UART-WIFI模块，板载安信可公司的ESP8266模块。

ESP8266是一个完整且自成体系的WIFI网络解决方案，能够独立运行，也可以作为从机搭载于其他主机运行[11]。而ATK-ESP8266模块采用串口与MCU通信，内置TCP/IP协议栈，能够实现串口与WIFI之间的转换[12]。通过ATK-ESP8266模块，传统的串口设备只是需要简单的串口配置，即可通过网络传输自己的数据。该模块可以兼容3.3V和5V的单片机系统，这样可以很方便和主控芯片进行连接，此外，它还支持 WIFI STA、串口转AP和WIFI STA+WIFI AP三种模式，这有利于开发。

除了上述所说的优点之外，下面还列举了一些优点：

（1）价格便宜

ESP8266模块目前是主流WIFI模块产品中价格相对便宜的一款WIFI模块，它以最低的成本实现最大的功能，非常实用，这非常符合本设计的研发定位。

（2）高集合程度

ESP8266具有很多部件，包括电源管理组件等，它们全部都高度集合在ESP8266模块上，这样高度集合使ESP8266模块体积小，便于设计使用。

（3）应用多领域样化

ESP8266应用非常广泛，比如可以用在智能电源插头、家庭自动化等等，如此广泛的应用使ESP8266有大量的资料，这样可以加快研发周期，进而降低研发成本[13]。

2.4.2 ESP8266 硬件电路设计

如图2-7所示，ATK-ESP8266模块的RXD和TXD管脚分别与STM32单片机的PA9和PA10引脚相连接。



图2-7 ESP8266硬件连接图

2.5 OLED显示屏

2.5.1 显示屏选型

目前市场上比较常见的三种显示屏分别是：LED点阵显示屏、LCD液晶显示屏以及OLED显示屏，其各自特点分别介绍如下：

（1）LED显示屏是一种平板显示器，由一个个小的LED模块面板组成，用来显示文字、图像、视频等各种信息的设备。LED电子显示屏集微电子技术、计算机技术、信息处理于一体，具有色彩鲜艳、动态范围广、亮度高、寿命长、工作稳定可靠等优点。可惜 LED灯体积较大，因此设计的显示屏也很大，多被用于户外招牌、广告等。

（2 ）LCD显示屏因为使用的是液晶材料，像素集成度高、亮度好、对比度高，因此可以在相同的面积内显示更多的数据，而且功耗低、不伤眼，因此它被广泛的使用到现在的电子设备中，如电视、移动产品等屏幕。

（3） OLED显示屏即有机发光二极管。 OLED 由于同时具备自发光，不需背光源、对比度高、厚度薄、视角广、反应速度快、可用于挠曲性面板、使用温度范围广、构造及制程较简单等优异之特性，被认为是下一代的平面显示器新兴应用技术。

综上可以看出，OLED 显示屏各项性能更为突出，故我们选择OLED显示屏。

2.5.2 OLED硬件电路设计

如图2-8所示，OLED具有四个引脚，分别是GND、VCC、SCL、SDA，且GCC需要接地，VCC接电源（3-5V），SCL接STM32单片机的PB6管脚，SDA接STM32单片机的PB7管脚。



图2-8 OLED硬件电路

2.5 继电器

为了达到对本设计中蜂鸣器和灯进行控制的目的，需要在系统中加入两个一路继电器设备。继电器可以起到开关使用，能够自动调节和安全保护的作用[14]。该继电器模块能耗低，体积小，最大负载250V交流电压，采用贴片光耦隔离，驱动能力强，性能稳定，非常适合本系统使用。其工作原理是利用电磁感应，通过给一个小电流的回路通电，使模块内部带铁芯的线圈产生磁场，磁场吸附开关触点，从而控制大电流回路的导通，具体电路连接方法如下。

以控制灯为例，将继电器的输入端DC+与5V直流电源的正极相连，DC-与5V直流电源的负极相连，IN与STM32单片机的PB12引脚相连，将继电器的输出端NO与灯的正极相连，灯另一端接地，COM端引入电源。由于本设计采用的接常开端接法，且NC端为继电器的常闭端口，故悬空处理。当单片机输入高电平时，继电器NO端与COM口短接，灯亮。当IN端输入低电平时，NC端与COM口短接，NO端悬空，灯灭。继电器控制电路图如图2-9所示：



图2-9 继电器控制电路

3.系统软件的设计

3.1 系统软件设计总体方案

系统的软件设计包括检测控制各个模块的软件设计，为了配合系统硬件电路工作，本次设计还需设计出与之配套的软件设计，程序见附录B。软件部分包含单片机系统、GY-30光强数据的采集、DHT11温湿度数据的采集、OLED液晶显示、WIFI模块的通信以及继电器的驱动设计。

本系统软件运行流程图如图3-1所示，首先对系统的延时函数初始化、配置中断分组、LED初始化、DHT11初始化、OLED显示屏初始化等操作，然后温湿度传感器及光强传感器开始进行探测采集数据。采集到的数据会传输给STM332单片机，STM32单片机接收到数据后分别将数据发送给OLED显示屏和ESP8266 WIFI模块，这样OLED就能显示温湿度以及光强。而ESP8266连接路由器后进一步发送数据到云平台，云平台会将数据传输至远程控制端，设计将收到的数据进行分析比较，超出阈值则进行报警。反之手机向云平台发送控制请求，云平台将请求返回至单片机，进而实现远程控制继电器。



图3-1 软件运行流程图

3.2 系统软件开发环境

3.2.1 STM32单片机开发环境

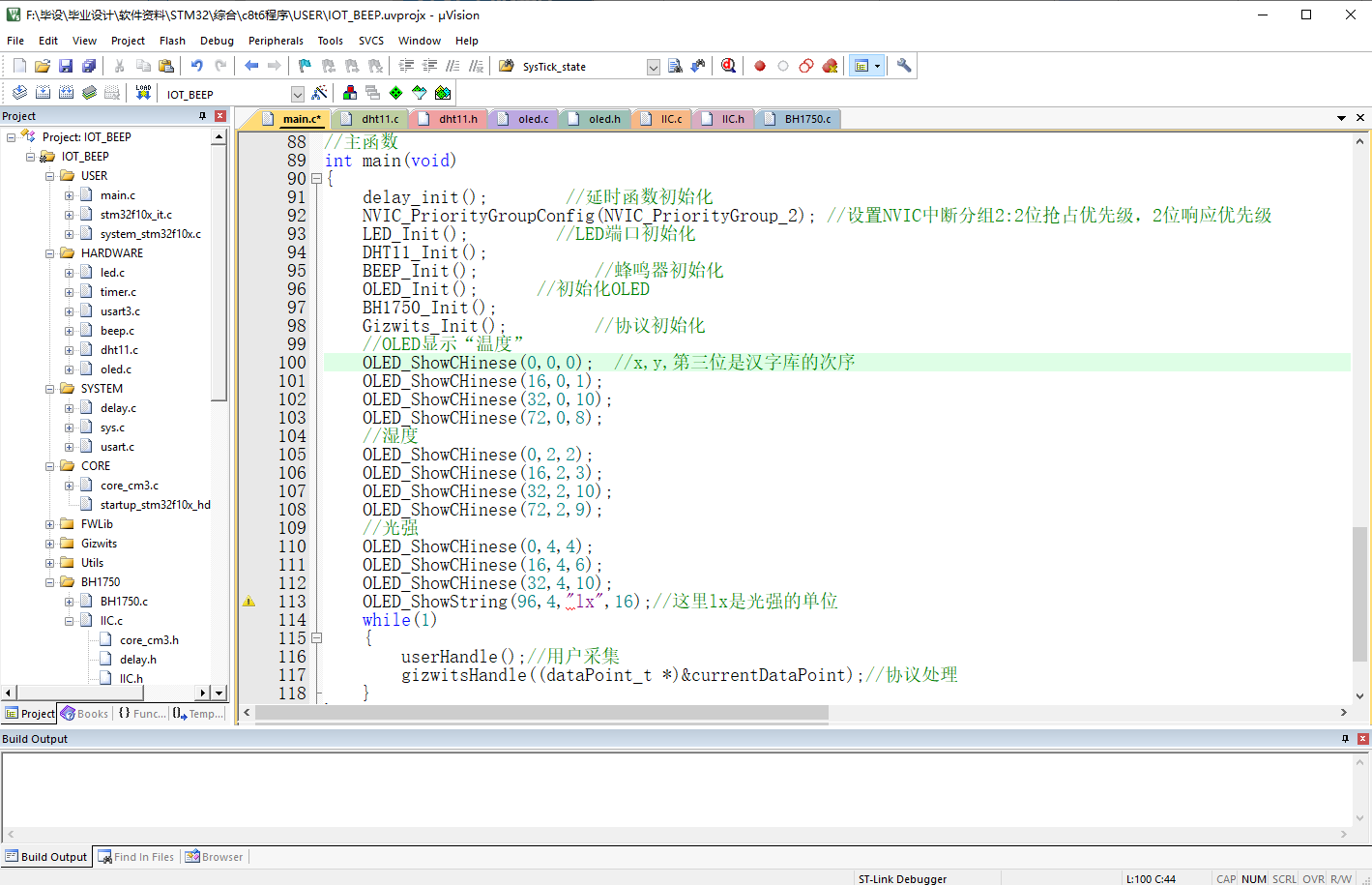


图3-2 Keil开发环境界面

本检测控制系统中各传感器的开发工具如图3-2所示，使用的是KEIL软件。Keil uVision5程序开发软件是由德国著名的嵌入式系统开发工具和Keil公司设计研发的，是目前STM32单片机程序开发的首选工具。该软件提供了C语言的编译、下载和仿真调试等一系列开发功能，是开发STM32最实用的开发工具[15]。在打开软件界面后，首先要新建一个工程，然后将在STM32单片机官方网站下载的库文件添加至新建工程内，最后选择芯片进行相应的软件配置。

3.2.2 APP开发环境

系统控制端的开发环境使用的是Android Studio，是一款综合的Java编程环境,被许多开发人员和行业专家誉为市场上最好的集成开发环境,尤其在智能代码助手、代码自动提示、重构等方面的功能可以说是超常的。Android Studio提供了集成的Android开发工具用于开发和调试 [16]。其开发环境界面如3-3所示：

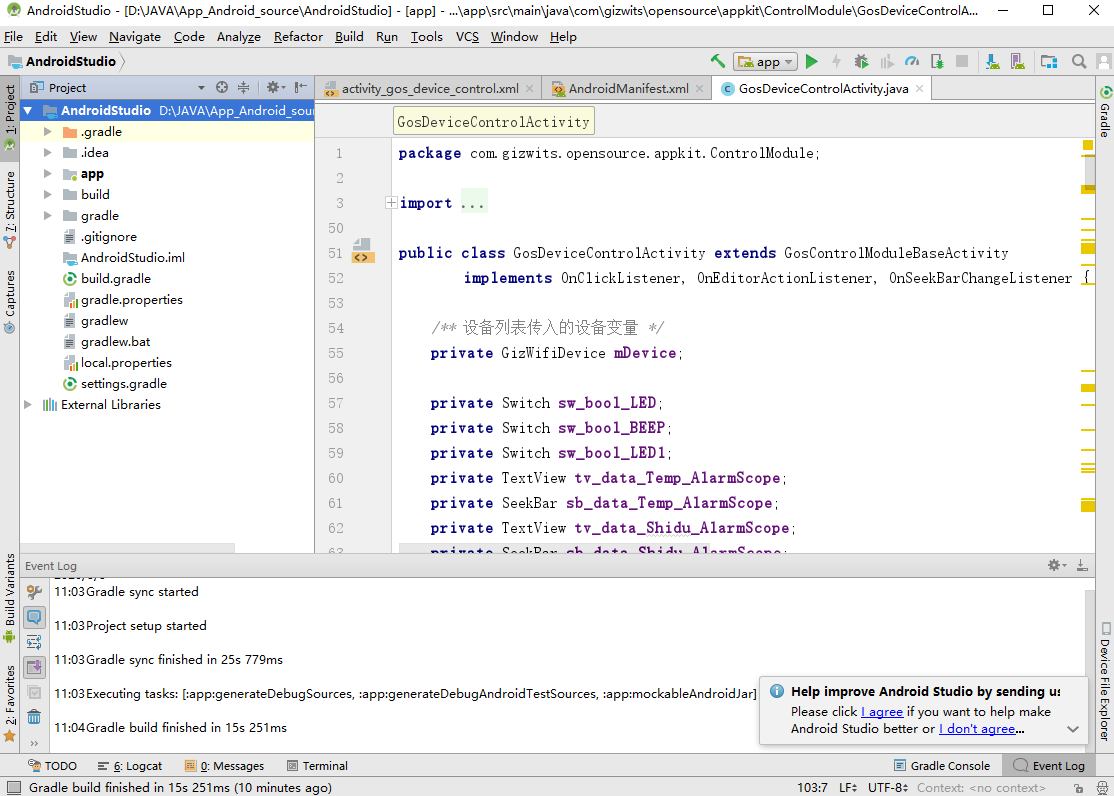


图3-3 Android Studio 开发环境界面

其优势如下：

（1）稳定速度快：Android Studio使用了单项目管理模式，在启动速度上比Eclipse快。

（2）功能强大的UI编辑器：集合了Eclipse+ADT的优点，并且能更实时的展示界面布局效果。

（3）完善的插件管理：Android Studio支持了多种插件，可直接在插件管理中下载所需的插件，。

（4）整合了Gradle构建工具：Gradle继承了Ant的灵活性和Maven的生命周期管理，采用了DSL格式，使得脚本更加灵活简洁。

（5）智能：智能保存，智能补齐，在实际的编辑代码中熟练使用后，可极大提高代码编写效率。

（6）内置终端：不需要自己打开一个终端来使用ADB等工具。

（7）谷歌官方支持：是Google官方专门为Android应用开发打造的利器，也是目前Google官方唯一推荐，并且不再支持其他IDE。

3.3 云平台的应用

近年来，由于物联网的快速普及和云计算的飞速发展，涌现出了一大批物联网云平台。云平台作为本文设计的物联网农业检测控制系统的枢纽，发挥了承上启下的作用，连接着下位机与手机APP端，是本系统数据传输的中心点，云平设计的可靠与否影响着整个系统的运行。考虑到完全自主设计云服务器开发难度大、周期长、成本较高，因此本设计选用了云服务平台——机智云作为本系统云平台的架构进行配置开发。

3.3.1 机智云平台简介

机智云是致力于物联网、智能硬件云服务的开发平台，机智云平台为开发者提供了自助式智能硬件开发工具与开放的云端服务，能够最大限度降低了物联网硬件开发的技术门槛，降低研发的成本。其主要包括机智云云端、机智云设备端和SDK三部分，如图3-4所示：

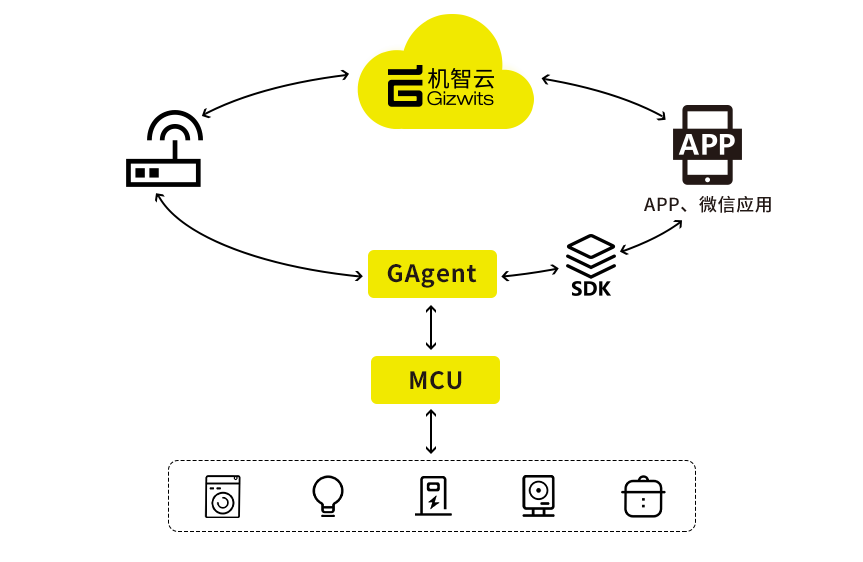


图3-4 机智云平台的基本构造

使用机智云平台开发产品，一般需要以下这些步骤：

（1）登录机智云官网，注册账号；

（2）进入开发者中心，创建产品；

（3）创建云平台数据点；

（4）对虚拟设备调试、生成云平台协议和产品代码；

（5）向WIFI模块烧录机智云GAgent固件；

（6）手机APP与机智云对接通信；

（7）APP绑定设备完成开发。

3.3.2 设备接入云平台开发

物联网农业检测控制与控制系统接入云平台是与手机APP建立通信的基本前提，本文选用了ESP8266 WIFI模块接入机智云的方案，机智云提供了与本文使用的ESP8266模块相匹配的固件。设备接入机智云首先做的是的将机智云固件GAgent移植到WIFI模块中。使用GAgent固件将农业检测控制与控制系统接入机智云并与APP建立通信，需要以下几个步骤：

（1）设备上电。WIFI模块中GAgent固件与MCU建立连接，如果设备能够正常工作，MCU会回复GAgent固件发出的心跳包。

（2）配置农业检测控制与控制系统入网。机智云API函数gizwitsSetMode()提供了softAP与AirLink两种模式配置入网[17]。本文选择的是AirLink模式配置入网。首先在手机连接路由器后，在APP内输入所连路由器的SSID和密码，然后利用机智云串口助手打开AirLink配网模式后进行配网，若配网成功，则WIFI模块自动连接路由器。AirLink入网流程见图3-5。

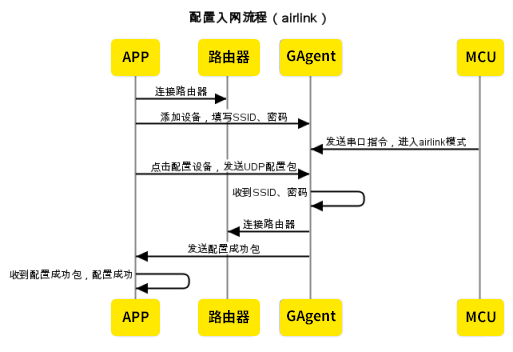


图3-5 AirLink模式入网流程图

（3）手机APP绑定农业检测控制与控制系统。在第二步完成配网的同时，直接使用已经设计好的APP 在局域网中发现并绑定设备。

在设备绑定完成后，手机APP除了可以在广域网中通过云平台来控制设备，还可以在同一局域网但无其他外网的情况下直接控制设备。

3.4 主要功能模块程序设计

3.4.1 温湿度传感器模块程序

DHT11传感器在本农业检测控制与控制系统设计中起到采集周围温湿度数据的作用，采用的是单总线数据格式，传输时序如图3-6所示：

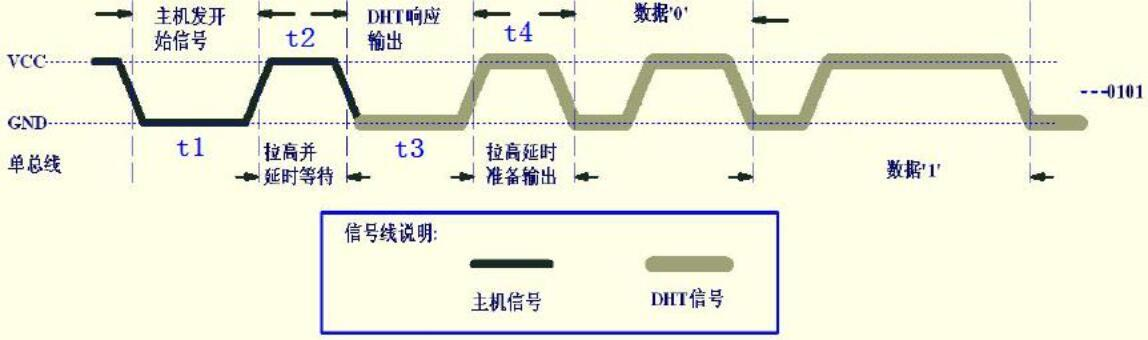


图3-6 DHT11数据时序

而MCU复位信号和DHT11响应信号如图3-7所示。MCU发送开始信号时，MCU至少拉低18ms，在拉高延时等待20us-40us后读取DH11T 的回应信号，读取总线为低电平，说明DHT11发送响应信号，DHT11发送响应信号后，再把总线拉高，准备发送数据，每一bit数据都以低电平开始[18]。

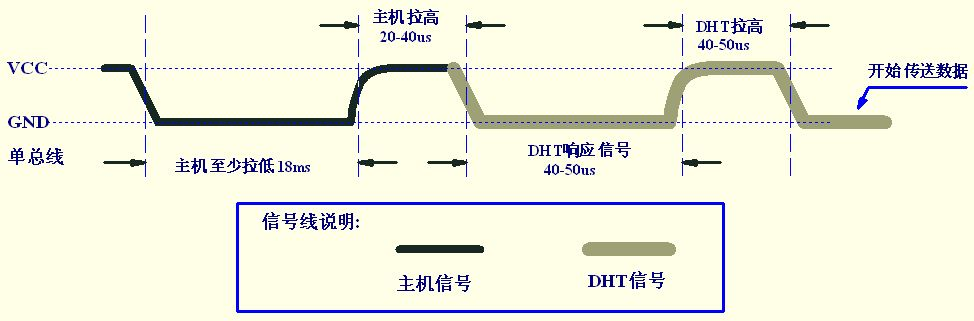


图3-7 MCU复位信号和DHT11响应信号图

DHT11和MCU每传输一次完整的数据为40bit，第一个8bit是湿度整数数据，第二个8bit是湿度小数数据，第三个8bit是温度整数数据，第四个8bit是温度小数数据，第五个8bit是校验位。

故根据其工作原理便可以写出控制DHT11的程序，其中部分代码示例如图3-8所示，完整代码见附录B。

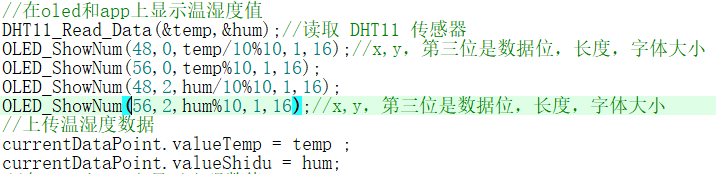
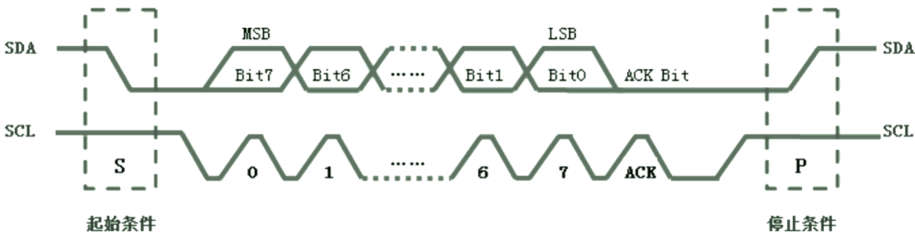


图3-8 DHT11代码示例

3.4.2 光强传感器模块程序

光强传感器GY-30本农业检测控制与控制系统设计中起到采集周围光照强度的作用，与STM32单片机之间以标准IIC总线协议[19]进行连接通讯。如图3-9为GY-30的IIC时序图。

图3-9 GY-30的IIC时序图

当SCL为高电平时，SDA从高电平变为低电平，表示产生起始信号，开始数据传输，这个阶段主设备在SCL时钟线上的每一个脉冲，都会同时在SDA数据线上传输一位的数据，数据位由高到低传输。IIC总线上的在SCL时钟线高电平期间，SDA传输数据有效。在数据传送时，SDA数据线上的数据允许在时钟线SCL为低电平期间进行数据改变，此时SDA的数据无效，一般在这个时候SDA进行电平切换，为下一次表示数据做好准备。每个字节完成后，跟着一个应答位。当不想再进行数据传输时，主机产生一个停止信号，总线释放，SCL和SDA线都回到空闲状态。根据其工作原理便可以写出控制GY-30的程序，其中部分代码示例如图3-10所示，完整代码见附录B。

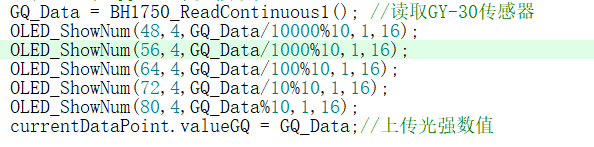


图 3-10 GY-30 代码示例

3.4.3 OLED显示屏模块程序

OLED显示屏和GY-30传感器都是用的IIC通信协议，其具体通信原理就不在赘述。OLED模块的控制器是SSD1306，驱动OLED显示器件[20],要使OLED可以正常使用，可以根据以下三步执行：首先将初始化OLED的IO引脚，然后将OLED模块初始化，最后向SSD1306显存写值，将内容在OLED上显示出来。其工作流程如图3-11所示。



图3-11 OLED工作流程

除此之外，因为OLED不能自动识别出我们的汉字，所以我们还需要借助取模软件PCtoLCD2002帮助我们取模，只需要将点阵格式选为“阴码”，取模方式选为“列行式”，取模走向设置为“逆向”，自定义格式里面选择C51格式，取出的字模数据便可以在OLED显示屏上正常显示，否则有可能出现“倒像”等情况。其中部分代码示例如图3-12所示，完整代码见附录B。

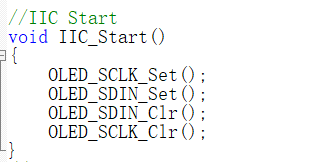


图3-12 OLED代码示例

3.4.4 WIFI通信模块程序

ESP8266 WIFI模块能够实现用户串口数据到无线网络之间的交换[21]。因此如果只是想驱动WIFI模块工作，我们只需要完成对串口的初始化即可。串口通信初始化的一般步骤可以分为如下几步：

（1）串口时钟使能，GPIO时钟使能；

（2）串口复位；

（3）GPIO 端口模式设置；

（4）串口参数初始化；

（5）开启中断并且初始化 NVIC；

（6）使能串口；

（7）写中断处理函数。

这样WIFI模块便能够正常的和MCU通信了，其中部分代码示例如图3-13所示，完整代码见附录B。

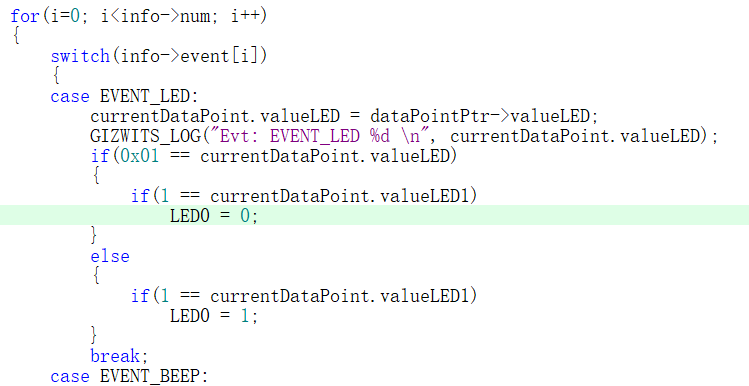


图3-13 WIFI模块代码示例

3.5 手机APP设计

为了快速接入云平台建立通信，本设计使用了机智云提供的开源框架。该框架不但集合了APP与机智云数据点通信的协议，还含有设备与APP通信的协议，包括设备配置入网的softAP与AirLink两种模式，因此开发APP的主要工作只需是进行一些简单的代码修改和UI设计，实现APP快速开发[22]。总体工作流程如图3-14所示：



图3-14 APP总体工作流程

对于本设计而言，此APP的主要功能需求是可以远程控制监视产品设备。当在手机中打开APP后，首先进入登录界面，如图3-15所示。这里可以选择注册新用户或者直接跳过就可以。



图3-15 用户登录界面

然后进入监测控制页面UI如图3-16所示，监测控制界面使用了EditText控件、ImageView控件、Switch控件、SeekBar控件等。在工程的XML文件中可以设置控件属性，比如文字大小、控件颜色、控件背景、控件高度与宽度、控件之间的布局关系等。布局使用了两种较常用的线性布局（LinearLayout）和相对布局RelativeLayout。



图3-16 监测控制界面

本设计的监测控制界面的作用是清晰的展示给用户可以使用的主要功能。在该界面用户可以看到温湿度以及光照强度的数值，可以在范围内自由设置温湿度以及光照强度的阈值，同时可以控制蜂鸣器和灯。其中APP下发数据的基本代码如图3-17所示：

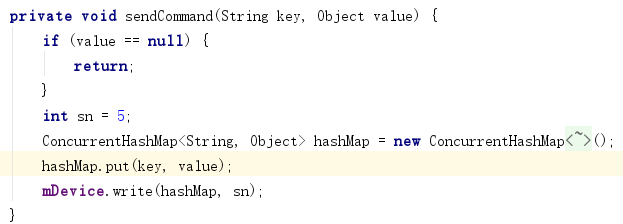


图3-17 下发数据代码

其中key是准备写入数据的数据点对应的标识名，value是要发送的数据值，key与value存储在ConcurrentHashMap中，保证线程安全并提高并发传输能力。而sn是传输数据协议的命令序号，用于回调时判断数据是否成功发送。

对于一个APP而言往往少不了全局配置文件，appConfig.json作为全局配置文件，其主要代码如图3-18所示：



图3-18 全局配置文件

根据图3-18我们可以看到appId、appSecert、productKey、productSecret这四个参数是必须要填写的，其他的参数可以选填。其中AppId是应用标识码，当开发者需要为一款产品开发应用时，平台会自动为其生成一个appId并自动关联设备。appSecert是应用密匙，与appId一起生成。productKey是产品唯一标识码，开发者通过机智云开发者平台创建新产品后，随机生成的32位字符串，将productKey写入主控设备以后，机智云通过此标识码对设备进行识别并自动完成注册。productSecret和productKey有同样功能，也起着身份识别作用。

4.实验测试与结果分析

4.1 虚拟设备测试

如图4-1所示是机智云平台提供的在线调试界面，使用虚拟设备对APP进行调试。测试方法是在云平台的虚拟设备窗口里找到模拟设备上报数据一栏，修改指定数据点的值后点击推送，在通信日志一栏可以看到虚拟设备上报的数据，也可以通过手机APP查看上报的数据[23]。具体步骤如下所示：

首先我们进入机智云的开发者中心，启动本次产品所对应的虚拟设备，扫描对应的二维码，扫描成功后，在通信日志中可以看到设备成功上线。



图4-1 虚拟设备测试界面

然后我们开始测试APP端与虚拟设备之间的通信：在虚拟设备中，分别填入测试下发的数据点的数据，之后点击“推送”按钮向APP下发数据。其中虚拟设备下发的各数据点的测试值如图4-2（a）所示，手机APP端接收的数据如图4-2（b）所示：

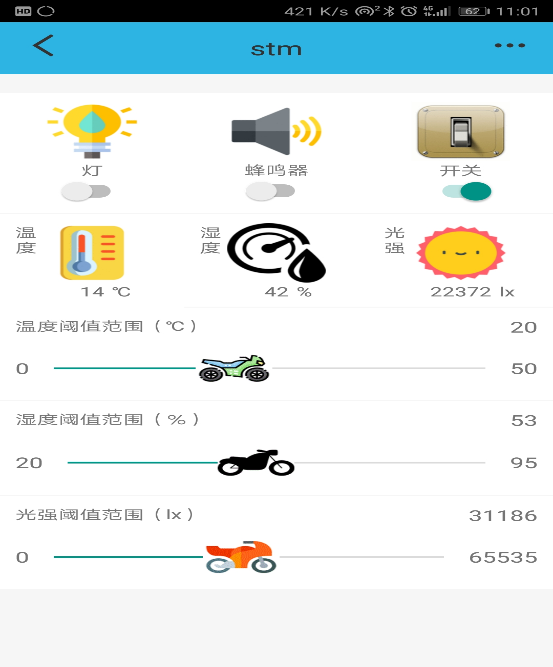
 (a)虚拟设备下发数据 (b)APP端接收数据

图4-2 APP与虚拟设备通信测试

然后在使用APP向虚拟设备上传数据，查看虚拟设备端的接收数值是否与APP端一致。经过数百次的测试实验以后，发现虚拟设备和手机APP的各数据点完全一致，符合本系统功能要求。

4.2 真实设备调试

通过虚拟设备调试后，能够确定手机APP端和云端的数据通信正常。在此基础上验证整个系统的功能是否符合预期的要求，确保数据能够准确传输，保证农业检测控制系统的可靠性和实用性。测试分为以下几个部分:

一、对农业检测控制系统进行功能测试与稳定性测试，设备能够可靠工作是整个控制系统正常工作的基础，一旦出现问题或者故障会使整个系统瘫痪，因此设备测试是重中之重。设备端各项功能测试包括：报警测试、显示屏页面测试等。稳定性测试主要用于检验设备能否长时间正常工作，让设备持续几天处于开机状态下是否会出现死机、重启等问题。

二、在不同的网络状态下测试APP与农业检测控制系统设备的通信。测试手机分别在接入局域网和4G网络情况下能否与设备正常通信。

三、测试系统通信的稳定性与成功率。主要包括测试设备断网后重新连接的成功率、APP正确接收设备端发送数据的成功率以及设备端正确接收APP端数据的成功率。

四、APP功能测试。主要测试设备检测控制模块的功能，包括能否实时的显示设备端发送的数据，能否控制继电器的动作，能否准确设置传感器的阈值等。

4.3 测试结果与分析

本文对进行的测试结果进行了统计分析：

一、虚拟设备测试时数据在双向之间传输无误，进而验证了APP与云平台之间的数据传输是可靠的。

二、如图4-3所示，当农业检测控制系统处于手动模式时，手机APP端能够控制继电器进而控制灯和蜂鸣器，且OLED上能够正常显示温湿度以及光照强度数据。在自动模式下，若数据超过设定阈值则会自动发出警报，OLED屏幕上会闪烁警告图标，设备蜂鸣器发出声音并且灯亮。

  
图4-3 OLED屏幕显示

三、当APP和设备接入同一个局域网时，即使所连接的路由器没有网络也不影响APP对设备的控制；当APP与设备不属于一个局域网时，手机通过4G或者WIFI网络连接物联网，此时手机可以实现对设备的远程监视控制，但是若设备所连接的路由器没有网络则不可以进行控制[24]。

四、本设计对农业检测控制系统的通信进行了三个方面的测试，在经过大量次数的测试与统计后得出的数据如表4-1所示，由此可以得出系统的通信是具有很高的稳定性的。

表4-1 系统通信测试表

|  |  |  |
| --- | --- | --- |
| 测试方式 | 测试次数 | 成功率（%） |
| 设备断网重连 | 200 | 99.7 |
| APP接收数据 | 300 | 99.9 |
| 设备接收数据 | 300 | 99.8 |

五、APP进入设备控制界面。此时设备端已经将数据上传到APP，并且不断实时变化。如图4-4所示，APP监控界面显示了已绑定设备当前上报的温湿度以及光照强度数据，手机端可以在阈值范围内自由的设置阈值，并且若传感器探测数据超过设定阈值则会自动发出警报，此时继电器会控制灯闪烁和蜂鸣器响。

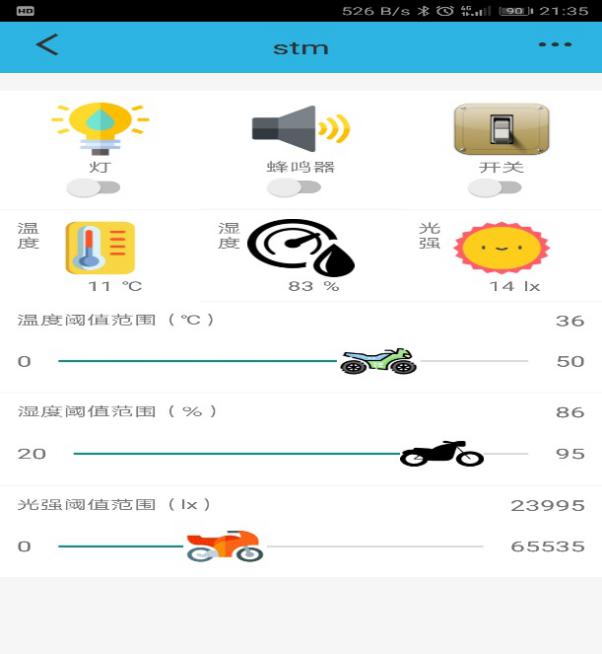


图4-4 APP监控界面

5.总结与展望

5.1 主要设计成果总结

本文设计了一款基于STM32单片机的物联网农业检测控制与控制系统。其中硬件平台的是以STM32F103C8T6作为核心处理器，结合现代物联网技术和传感器技术的发展，能够实时的监测控制中温湿度、光照强度的变化，又通过ESP8266 WIFI模块接入了云平台，实现了可在手机APP侧查看设备当前状态和远程控制设备的目的。主要设计成果包括：

（1）分析了物联网农业检测控制系统的国内外研究现状，结合市场需求和新技术的发展，给出了物联网农业检测控制系统的技术路线。

（2）通过分析结合本设计需要实现的功能，确定了物联网农业检测控制系统的基本结构、核心元器件的选型等。

（3）选用DHT11传感器和GY-30传感器能够很好的探测周围环境温湿度及光照强度的变化，并能够在OLED上显示出来，方便观察。

（4）制作的手机APP在功能上分为手动模式和自动模式。手机APP在自动模式下，系统将采集的温湿度及光照数据与手机APP上设置的阈值做比较，如果超出则报警，并作出相应响应。手机APP在手动模式下，可以通过手机APP远程控制照明灯、继电器、蜂鸣器等。

（5）实现了STM32单片机利用WIFI模块通过云平台与手机APP的通信，能够比较稳定的将数据进行传输。

（6）对整个系统进行了实验测试，证明此设计具有一定的实用性。

5.2 系统特点总结

本设计的特点主要由以下所示：

（1）系统性价比高

在充分考虑能够完成设计任务要求的前提下，本设计尽可能选择性价比高的器件，比如本设计选择的主控器STM32F103C8T6就具有很高的性价比，且能够很好的完成设计需求。

（2）系统稳定性好

本设计经过了数据传输测试以及传输成功率的统计分析，测试证明本设计系统数据传输稳定性好。

（3）系统传输速度快

本设计的农业检测控制系统，能够将传感器检测到数据实时的在OLED和手机APP上显示，传输速度快。

（4）通信无距离限制

本设计在理论上说，只要手机和设备端能够正常连接网络，不论距离多远，手机端也能够实现对设备端的检测控制。

5.3 展望

物联网农业检测控制系统虽然已经完成了设计，并经过实验测试具备了远程检测控制的能力，但是仍然还存在许多问题有待进一步研究：

（1）本设计系统的功能还不够完善。对应农业环境的检测集中于温度、湿度、光照强度三个层面上，这导致设备获取的环境信息有限，从而无法为用户提供更加全面性的指导建议。

（2）本设计虽然能够检测农业环境温湿度以及光照强度，但是并未提出有效的环境调控策略，未来研究需要在调控环境策略方面做更加深入的研究。

（3）由于时间和工作量的原因，本文只设计了基于Android系统的APP，未来可以设计基于IOS系统的APP或者微信小程序，使农业检测控制系统应用更加广泛、适用性更强。

谢辞

寒来暑往，斗转星移，转眼间大学学习生活已经悄然结束。回首大学四年，从懵懵懂懂到变得逐渐成熟，期间经历了太多的事情，在这里我留下了自己最宝贵的青春年华，带走了不仅是美好的回忆还有各位老师对我的谆谆教诲。

首先，我要感谢我的指导老师潘莹月老师，在我完成毕业设计时遇见的问题，潘老师都能耐心及时的帮我解答，正因如此，我才能顺利的完成毕业设计。

在此，我向尊敬的潘老师表示最衷心的感谢！

然后我要感谢大学四年的同学们，感谢你们在我学习生活中给我提供的许多帮助，尤其是我的室友们，感谢你们在宿舍生活中的包容和陪伴。

感谢我的父母，在我的求学生涯中，是你们给了我全力的支持，你们的殷殷期望是我进步的动力。

最后，对我论文中所引用的文章作者一并致以诚挚的感谢。

参考文献

1. 陈韵秋,李峥. 基于STM32和Android系统的智能农业大棚设计[J]. 淮北师范大学学报(自然科学版),2019,40(01):43-48.
2. 李国刚,李旭文,温香彩. 物联网技术发展与环境自动监控系统建设[J]. 中国环境监测,2011,27(1):5-10.
3. 豁保强. 智能大棚监测与控制关键技术的研究[D]. 天津科技大学,2014.
4. Hongwei Zhu. Research on Agricultural Internet of Things Information Collection and Control System[P]. Proceedings of the 2018 3rd International Workshop on Materials Engineering and Computer Sciences (IWMECS 2018),2018.
5. Zhuang Miao. Research on Intelligent Agriculture Monitoring System Based on Internet of Things[P]. Proceedings of the 2018 3rd International Workshop on Materials Engineering and Computer Sciences (IWMECS 2018),2018.
6. 王冬. 基于物联网的智能农业监测系统的设计与实现[D]. 大连理工大学,2013.
7. 韩毓. 基于单片机的蔬菜大棚温度控制系统[D]. 中国海洋大学,2010.
8. 廖建尚. 基于物联网的温室大棚环境监控系统设计方法[J]. 农业工程学报,2016,32(11):233-243.
9. 陈建新.DHT11数字温湿度传感器在温室控制系统中的应用[J].山东工业技术,2016(18):120.
10. 邹曙光. 基于Android的嵌入式农业环境采集系统设计与实现[D]. 江西农业大学,2016.
11. 范兴隆. ESP8266在智能家居监控系统中的应用[J]. 单片机与嵌入式系统应用,2016,16(09):52-56.
12. 王小娟. 基于ESP8266无线传输的温湿度检测仪设计[J]. 九江职业技术学院学报,2017(04):22-24+32.
13. 刘振. 基于STM32智能家居的无线网关设计与实现[D].浙江理工大学,2017.
14. 郑婷婷. 基于物联网的智慧农业控制管理系统[D]. 西安工程大学,2018.
15. 莫先. 基于STM32单片机家电控制及家居环境监测系统设计与实现[D]. 重庆理工大学,2016.陶镇. 基于机智云的物联网移动终端SDK的设计与实现[D].中国科学院大学(中国科学院工程管理与信息技术学院),2017.
16. 尹孟征.基于Android的APP开发平台综述[J].通信电源技术,2016,33(04):154-155+213.
17. 韩丹翱,王菲. DHT11数字式温湿度传感器的应用性研究[J]. 电子设计工程,2013,21(13):83-85+88.
18. 李升红. 基于STM32和WIFI技术的家居盆栽植物智能监控系统[D].武汉轻工大学,2018.
19. 赵晶. 单片机控制OLED显示系统研究[D]. 重庆大学,2006.
20. 薛翔,王琰. 基于ESP8266的智能开关控制系统设计[J]. 电子世界,2018(21):147-148.
21. 程峥. 基于嵌入式和设备云平台的家庭植物工厂系统设计[D].天津工业大学,2019.
22. 高蒙. 基于机智云平台的远程监控系统开发关键技术研究[D].西安理工大学,2019.
23. 曾建清. 基于STM32的多功能空气质量监测系统设计[D].电子科技大学,2019.
24. 许锰. 基于云平台的家庭安防系统研究与设计[D].山东大学,2018.

附录A 硬件电路图



附录B 程序

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

农业检测控制系统主程序

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "sys.h"

#include "delay.h"

#include "usart.h"

#include "dht11.h"

#include "timer.h"

#include "usart3.h"

#include "oled.h"

#include "BH1750.h"

#include "gizwits\_product.h"

//名称替换

#define TScope currentDataPoint.valueTemp\_AlarmScope

#define HScope currentDataPoint.valueShidu\_AlarmScope

#define GQScope currentDataPoint.valueGQ\_AlarmScope

/\* 用户区当前设备状态结构体\*/

dataPoint\_t currentDataPoint;

//机智云协议初始化

void Gizwits\_Init(void)

{

TIM3\_Int\_Init(9,7199);//1MS系统定时

uart\_init(9600);//这里的波特率只能是9600

memset((uint8\_t\*)&currentDataPoint, 0, sizeof(dataPoint\_t));//设备状态结构体初始化

gizwitsInit();//缓冲区初始化

}

//数据采集、并在OLED显示数据、上报数据、判断是否触发警报

void userHandle(void)

{

static u16 GQ\_Data;

static u8 temp, hum;

if(timeuser > 1000)//每1秒上报一次数据

{

timeuser=0;//在.h文件里面已经声明过

//在oled和app上显示温湿度值

DHT11\_Read\_Data(&temp,&hum);//读取 DHT11 传感器

//x,y，第三位是数据位，长度，字体大小

OLED\_ShowNum(48,0,temp/10%10,1,16);

OLED\_ShowNum(56,0,temp%10,1,16);

OLED\_ShowNum(48,2,hum/10%10,1,16);

OLED\_ShowNum(56,2,hum%10,1,16);

//上传温湿度数据

currentDataPoint.valueTemp = temp ;

currentDataPoint.valueShidu = hum;

//在oled和app上显示光强数值

GQ\_Data = BH1750\_ReadContinuous1(); //读取GY-30传感器

OLED\_ShowNum(48,4,GQ\_Data/10000%10,1,16);

OLED\_ShowNum(56,4,GQ\_Data/1000%10,1,16);

OLED\_ShowNum(64,4,GQ\_Data/100%10,1,16);

OLED\_ShowNum(72,4,GQ\_Data/10%10,1,16);

OLED\_ShowNum(80,4,GQ\_Data%10,1,16);

currentDataPoint.valueGQ = GQ\_Data;//上传光强数值

if(0 == currentDataPoint.valueLED1)//判断led1发送数据

{

OLED\_ShowCHinese(48,6,13);

//判断温湿度以及光强是否超出给定值，若超出蜂鸣器响，led闪烁if((temp>=TScope||hum>=HScope||GQ\_Data>=GQScope)&&(TScope!=0&&HScope!=0&&GQScope!=0))

{

OLED\_ShowCHinese(0,6,17);

BEEP = 1;

LED0 = 0;

delay\_ms(500);

OLED\_ShowCHinese(0,6,18);

BEEP = 0;

LED0 = 1;

}

currentDataPoint.valueLED = currentDataPoint.valueBEEP = 0;

}

else

{

OLED\_ShowCHinese(48,6,11);

}

}

}

//主函数

int main(void)

{

delay\_init(); //延时函数初始化

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_2); //设NVIC中断分组

DHT11\_Init();

OLED\_Init(); //初始化OLED

BH1750\_Init();

Gizwits\_Init(); //协议初始化

//OLED显示“温度”

OLED\_ShowCHinese(0,0,0); //x,y,第三位是汉字库的次序

OLED\_ShowCHinese(16,0,1);

OLED\_ShowCHinese(32,0,10);

OLED\_ShowCHinese(72,0,8);

//湿度

OLED\_ShowCHinese(0,2,2);

OLED\_ShowCHinese(16,2,3);

OLED\_ShowCHinese(32,2,10);

OLED\_ShowCHinese(72,2,9);

//光强

OLED\_ShowCHinese(0,4,4);

OLED\_ShowCHinese(16,4,6);

OLED\_ShowCHinese(32,4,10);

OLED\_ShowString(96,4,"lx",16);//这里lx是光强的单位

OLED\_ShowCHinese(48+16,6,14);

OLED\_ShowCHinese(48+32,6,15);

OLED\_ShowCHinese(48+48,6,16);

while(1)

{

userHandle();//用户采集函数

gizwitsHandle((dataPoint\_t \*)&currentDataPoint);//协议处理

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DHT11程序——dht11.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "dht11.h"

#include "delay.h"

//复位DHT11

void DHT11\_Rst(void)

{

DHT11\_IO\_OUT(); //SET OUTPUT

DHT11\_DQ\_OUT=0; //拉低DQ

delay\_ms(20); //拉低至少18ms

DHT11\_DQ\_OUT=1; //DQ=1

delay\_us(30); //主机拉高20~40us

}

//等待DHT11的回应

u8 DHT11\_Check(void)

{

u8 reply=0;

DHT11\_IO\_IN();//SET INPUT

while (DHT11\_DQ\_IN&&reply<100)//DHT11会拉低40~80us

{

reply++;

delay\_us(1);

};

if(reply>=100)return 1;

else reply=0;

while (!DHT11\_DQ\_IN&&reply<100)//DHT11拉低后会再次拉高40~80us

{

reply++;

delay\_us(1);

};

if(reply>=100)return 1;

return 0;

}

//从DHT11读取一个位

u8 DHT11\_Read\_Bit(void)

{

u8 reply=0;

while(DHT11\_DQ\_IN&&reply<100)//等待变为低电平

{

reply++;

delay\_us(1);

}

reply=0;

while(!DHT11\_DQ\_IN&&reply<100)//等待变高电平

{

reply++;

delay\_us(1);

}

delay\_us(40);//等待40us

if(DHT11\_DQ\_IN)return 1;

else return 0;

}

//从DHT11读取一个字节

u8 DHT11\_Read\_Byte(void)

{

u8 i,dat;

dat=0;

for (i=0; i<8; i++)

{

dat<<=1;

dat|=DHT11\_Read\_Bit();

}

return dat;

}

//从DHT11读取一次数据

u8 DHT11\_Read\_Data(u8 \*temp,u8 \*humi)

{

u8 buf[5];

u8 i;

DHT11\_Rst();

if(DHT11\_Check()==0)

{

for(i=0; i<5; i++) //读取40位数据

{

buf[i]=DHT11\_Read\_Byte();

}

if((buf[0]+buf[1]+buf[2]+buf[3])==buf[4])

{

\*humi=buf[0];

\*temp=buf[2];

}

} else return 1;

return 0;

}

//初始化DHT11的IO口 DQ 同时检测DHT11的存在

u8 DHT11\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

//使能PB,PE端口时钟

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB,ENABLE);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_9;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_Init(GPIOB, &GPIO\_InitStructure);

DHT11\_Rst();

return DHT11\_Check();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DHT11程序——dht11.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_DHT11\_H

#define \_\_DHT11\_H

#include "sys.h"

//IO方向设置

//#define DHT11\_IO\_IN() {GPIOA->CRH&=0XFFFF0FFF;GPIOA->CRH|=8<<12;}

//#define DHT11\_IO\_OUT() {GPIOA->CRH&=0XFFFF0FFF;GPIOA->CRH|=3<<12;}

//这里是高八位设置，若是低八位改成L即可

#define DHT11\_IO\_IN() {GPIOB->CRH&=0XFFFFFF0F;GPIOB->CRH|=8<<4;}

#define DHT11\_IO\_OUT() {GPIOB->CRH&=0XFFFFFF0F;GPIOB->CRH|=3<<4;}

////IO操作函数

#define DHT11\_DQ\_OUT PBout(9) //数据端口 PB9

#define DHT11\_DQ\_IN PBin(9) //数据端口 PB9

u8 DHT11\_Init(void); //初始化DHT11

u8 DHT11\_Read\_Data(u8 \*temp,u8 \*humi);//读取温湿度

u8 DHT11\_Read\_Byte(void); //读出一个字节

u8 DHT11\_Read\_Bit(void); //读出一个位

u8 DHT11\_Check(void); //检测是否存在DHT11

void DHT11\_Rst(void); //复位DHT11

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GY-30程序——BH1750.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "BH1750.h"

#include "delay.h"

#include "sys.h"

#include "stm32f10x.h"

#include "stdint.h"

u8 BUF[8];

u16 temp2=0;

static u32 lux=0;

/\*\*\*\*向BH1750单次写命令\*\*\*\*/

void Single\_Write\_BH1750(unsigned char Reg\_Address)

{

IIC\_Start1();

IIC\_Send\_Byte(0x46); //发送器件地址0100 0110 最后一位0，表示写

IIC\_Send\_Byte(Reg\_Address);

IIC\_Stop1();

}

/\*\*\*\*BH1750初始化\*\*\*\*/

void BH1750\_Init(void)

{

IIC\_Config();

Single\_Write\_BH1750(0x01);

}

//从BH1750单次读lx

void BH1750\_ReadOneTime(void)

{

IIC\_Start1();

IIC\_Send\_Byte(0x46); //发送器件地址0100 0110 最后一位0，表示写

IIC\_Wait\_Ack1();

IIC\_Send\_Byte(0x20);

IIC\_Wait\_Ack1();

IIC\_Stop1();

delay\_ms(200);

IIC\_Start1();

IIC\_Send\_Byte(0x47);

IIC\_Wait\_Ack1();

BUF[0]=IIC\_Read\_Byte(1);

BUF[1]=IIC\_Read\_Byte(0);

IIC\_Stop1();

}

//从BH1750连续读lx

u16 BH1750\_ReadContinuous1(void)

{

u16 temp=0,temp1=0;

IIC\_Start1();

IIC\_Send\_Byte(0x46); //发送器件地址0100 0110 最后一位0，表示写

IIC\_Wait\_Ack1();

IIC\_Send\_Byte(0x10);

IIC\_Wait\_Ack1();

IIC\_Stop1();

delay\_ms(200);

IIC\_Start1();

IIC\_Send\_Byte(0x47);

IIC\_Wait\_Ack1();

temp=IIC\_Read\_Byte(1);

temp1=IIC\_Read\_Byte(0);

IIC\_Stop1();

temp2=temp1+(temp<<8);

lux=temp2;

return lux;

}

void Conversion(void)

{

temp2=BUF[0];

temp2=(temp2<<8)+BUF[1];

lux=(float)temp2/1.2; //lux是float型

}

void IIC\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB,ENABLE);

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_10|GPIO\_Pin\_11;

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_Out\_PP; //推挽输出

GPIO\_Init(GPIOB,&GPIO\_InitStructure);

GPIO\_SetBits(GPIOB,GPIO\_Pin\_10|GPIO\_Pin\_11); //将总线拉高释放总线

}

//总线初始化

void IIC\_Start1(void)

{

SDA\_OUT() //设置SDA线为输出

//在开始数据传输前，先让SDA和SCL都拉高

IIC\_SDA=1; //发送起始条件的数据信号，释放总线

delay\_us(2);

IIC\_SCL=1;

delay\_us(5); //Tsu;STA：起始条件的建立时间大于4.7us。

IIC\_SDA=0; //SDA由高变为低表示开始信号

delay\_us(4); //起始条件的保持时间大于4us

IIC\_SCL=0; //准备发送或者接收数据

delay\_us(2);

}

//停止信号

void IIC\_Stop1(void)

{

SDA\_OUT() //设置SDA线为输出

IIC\_SDA=0; //发送停止信号的数据信号

delay\_us(2);

IIC\_SCL=1; //发送停止信号的时钟信号

delay\_us(5); //停止信号的建立时间大于4us

IIC\_SDA=1; //发送停止信号

delay\_us(4); //

}

//等待应答信号到来

u8 IIC\_Wait\_Ack1(void)

{

u8 ucErrorTime=0;

SDA\_IN();

IIC\_SDA=1;

delay\_us(2);

IIC\_SCL=1;

delay\_us(2);

while(READ\_SDA)

{

ucErrorTime++;

if(ucErrorTime>=250)

{

IIC\_Stop1();

return 1;

}

}

IIC\_SCL=0;

return 0;

}

//产生Ack应答

void IIC\_Ack(void)

{

IIC\_SCL=0;

SDA\_OUT();

IIC\_SDA=0; //主器件应答

delay\_us(2);

IIC\_SCL=1;

delay\_us(4); //SCL高电平周期大于4us

IIC\_SCL=0; //清时钟线，钳住IIC总线以便继续接收

delay\_us(2);

}

//不产生Ack应答

void IIC\_NAck(void)

{

IIC\_SCL=0;

SDA\_OUT();

IIC\_SDA=1; //主器件不应答

delay\_us(2);

IIC\_SCL=1;

delay\_us(4); //SCL高电平周期大于4us

IIC\_SCL=0; //清时钟线，钳住IIC总线以便继续接收

delay\_us(2);

}

//发送一个字节

void IIC\_Send\_Byte(unsigned char c) //要传送的数据长度为8位

{

u8 i;

SDA\_OUT();

for(i=0; i<8; i++)

{

if((c<<i)&0x80) //判断发送位

IIC\_SDA=1;

else IIC\_SDA=0;

delay\_us(2);

IIC\_SCL=1; //拉高SCL，通知被控器开始接收数据位

delay\_us(4); //保证SCL高电平周期大于4us

IIC\_SCL=0; //拉低SCL，允许SDA传输下一位数据。

delay\_us(2);

}

IIC\_SCL=0; //拉低SCL，为下次数据传输做好准备

delay\_us(2);

}

//接收一个字节

u8 IIC\_Read\_Byte(unsigned char ack)

{

unsigned char i,receive=0;

SDA\_IN();

for(i=0; i<8; i++)

{

IIC\_SCL=0; //置SCL为低，准备接收数据位

delay\_us(5); //时钟低电平周期大于4.7us

IIC\_SCL=1; //置SCL为高，使SDA上数据有效

delay\_us(2);

receive=receive<<1;

//读取SDA，把接收的数据位放入receve中

if(READ\_SDA) receive=receive+1;

delay\_us(2);

}

if(!ack) IIC\_NAck(); //发送NAck

else IIC\_Ack(); //发送Ack

return receive;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GY-30程序——BH1750.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_BH1750\_H

#define \_\_BH1750\_H

#include "sys.h"

//IO操作函数

#define IIC\_SCL PBout(10) //SCL

#define IIC\_SDA PBout(11) //SDA

#define READ\_SDA PBin(11) //输入SDA

//IO方向设置，设置输入还是输出，半双工就这样

#define SDA\_IN() {GPIOB->CRH&=0XFFFF0FFF;GPIOB->CRH|=8<<12;}

#define SDA\_OUT() {GPIOB->CRH&=0XFFFF0FFF;GPIOB->CRH|=3<<12;}

void IIC\_Config(void);

void IIC\_Init(void) ;

void IIC\_Start1(void); //起始信号

void IIC\_Stop1(void); //停止信号

u8 IIC\_Wait\_Ack1(void);

void IIC\_Ack(void);

void IIC\_NAck(void);

void IIC\_Send\_Byte(unsigned char c); //要传送的数据长度为8位

u8 IIC\_Read\_Byte(unsigned char ack);

void Single\_Write\_BH1750(unsigned char Reg\_Address);

void BH1750\_Init(void);

void BH1750\_ReadOneTime(void);

u16 BH1750\_ReadContinuous1(void);

void BH1750\_ReadContinuous2(void);

void BH1750\_Multiple\_Read(void);

void Conversion(void);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OLED程序——oled.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "oled.h"

#include "stdlib.h"

#include "codetab.h"

#include "delay.h"

//IIC Start

void IIC\_Start()

{

OLED\_SCLK\_Set();

OLED\_SDIN\_Set();

OLED\_SDIN\_Clr();

OLED\_SCLK\_Clr();//注意一下

}

//IIC Stop

void IIC\_Stop()

{

OLED\_SCLK\_Set() ;

OLED\_SDIN\_Clr();

OLED\_SDIN\_Set();

}

void IIC\_Wait\_Ack()

{

OLED\_SCLK\_Set() ;

OLED\_SCLK\_Clr();

}

// IIC Write byte

void Write\_IIC\_Byte(unsigned char IIC\_Byte)

{

unsigned char i;

unsigned char m,da;

da=IIC\_Byte;

OLED\_SCLK\_Clr();

for(i=0; i<8; i++)

{

m=da;

m=m&0x80;

if(m==0x80)

{

OLED\_SDIN\_Set();

}

else OLED\_SDIN\_Clr();

da=da<<1;

OLED\_SCLK\_Set();

OLED\_SCLK\_Clr();

}

}

// IIC Write Command

void Write\_IIC\_Command(unsigned char IIC\_Command)

{

IIC\_Start();

Write\_IIC\_Byte(0x78); //Slave address,SA0=0

IIC\_Wait\_Ack();

Write\_IIC\_Byte(0x00); //write command

IIC\_Wait\_Ack();

Write\_IIC\_Byte(IIC\_Command);

IIC\_Wait\_Ack();

IIC\_Stop();

}

// IIC Write Data

void Write\_IIC\_Data(unsigned char IIC\_Data)

{

IIC\_Start();

Write\_IIC\_Byte(0x78); //D/C#=0; R/W#=0

IIC\_Wait\_Ack();

Write\_IIC\_Byte(0x40); //write data

IIC\_Wait\_Ack();

Write\_IIC\_Byte(IIC\_Data);

IIC\_Wait\_Ack();

IIC\_Stop();

}

void OLED\_WR\_Byte(unsigned dat,unsigned cmd)

{

if(cmd)

{

Write\_IIC\_Data(dat);

}

else

{

Write\_IIC\_Command(dat);

}

}

//坐标设置

void OLED\_Set\_Pos(unsigned char x, unsigned char y)

{

OLED\_WR\_Byte(0xb0+y,OLED\_CMD);

OLED\_WR\_Byte(((x&0xf0)>>4)|0x10,OLED\_CMD);

OLED\_WR\_Byte((x&0x0f),OLED\_CMD);

}

//清屏函数,清完屏,整个屏幕是黑色的

void OLED\_Clear(void)

{

u8 i,n;

for(i=0; i<8; i++)

{

OLED\_WR\_Byte (0xb0+i,OLED\_CMD); //设置页地址（0~7）

OLED\_WR\_Byte (0x00,OLED\_CMD); //设置显示位置—列低地址

OLED\_WR\_Byte (0x10,OLED\_CMD); //设置显示位置—列高地址

for(n=0; n<128; n++)OLED\_WR\_Byte(0,OLED\_DATA);

} //更新显示

}

//在指定位置显示一个字符,包括部分字符

//x:0~127

//y:0~8

//mode:0,反白显示;1,正常显示

//size:选择字体 16/12

void OLED\_ShowChar(u8 x,u8 y,u8 chr,u8 Char\_Size)

{

unsigned char c=0,i=0;

c=chr-' ';//得到偏移后的值

if(x>128-1) {

x=0;

y=y+2;

}

if(Char\_Size ==16)

{

OLED\_Set\_Pos(x,y);

for(i=0; i<8; i++)

OLED\_WR\_Byte(F8X16[c\*16+i],OLED\_DATA);

OLED\_Set\_Pos(x,y+1);

for(i=0; i<8; i++)

OLED\_WR\_Byte(F8X16[c\*16+i+8],OLED\_DATA);

}

else {

OLED\_Set\_Pos(x,y);

for(i=0; i<6; i++)

OLED\_WR\_Byte(F6x8[c][i],OLED\_DATA);

}

}

//m^n函数

u32 oled\_pow(u8 m,u8 n)

{

u32 result=1;

while(n--)

result\*=m;

return result;

}

//显示2个数字

void OLED\_ShowNum(u8 x,u8 y,u32 num,u8 len,u8 size2)

{

u8 t,temp;

u8 enshow=0;

for(t=0; t<len; t++)

{

temp=(num/oled\_pow(10,len-t-1))%10;

if(enshow==0&&t<(len-1))

{

if(temp==0)

{

OLED\_ShowChar(x+(size2/2)\*t,y,' ',size2);

continue;

} else enshow=1;

}

OLED\_ShowChar(x+(size2/2)\*t,y,temp+'0',size2);

}

}

//显示一个字符号串

void OLED\_ShowString(u8 x,u8 y,u8 \*chr,u8 Char\_Size)

{

unsigned char j=0;

while (chr[j]!='\0')

{ OLED\_ShowChar(x,y,chr[j],Char\_Size);

x+=8;

if(x>120) {

x=0;

y+=2;

}

j++;

}

}

//显示汉字

void OLED\_ShowCHinese(u8 x,u8 y,u8 no)

{

u8 t,adder=0;

OLED\_Set\_Pos(x,y);

for(t=0; t<16; t++)

{

OLED\_WR\_Byte(Hzk[2\*no][t],OLED\_DATA);

adder+=1;

}

OLED\_Set\_Pos(x,y+1);

for(t=0; t<16; t++)

{

OLED\_WR\_Byte(Hzk[2\*no+1][t],OLED\_DATA);

adder+=1;

}

}

//初始化SSD1306

void OLED\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructer;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);

GPIO\_InitStructer.GPIO\_Pin=GPIO\_Pin\_6|GPIO\_Pin\_7; //6--SCL 7--SDA

GPIO\_InitStructer.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO\_InitStructer.GPIO\_Mode=GPIO\_Mode\_Out\_PP;

GPIO\_Init(GPIOB, &GPIO\_InitStructer);

delay\_ms(200);

OLED\_WR\_Byte(0xAE,OLED\_CMD);//--display off

OLED\_WR\_Byte(0x00,OLED\_CMD);//---set low column address

OLED\_WR\_Byte(0x10,OLED\_CMD);//---set high column address

OLED\_WR\_Byte(0x40,OLED\_CMD);//--set start line address

OLED\_WR\_Byte(0xB0,OLED\_CMD);//--set page address

OLED\_WR\_Byte(0x81,OLED\_CMD); // contract control

OLED\_WR\_Byte(0xFF,OLED\_CMD);//--128

OLED\_WR\_Byte(0xA1,OLED\_CMD);//set segment remap

OLED\_WR\_Byte(0xA6,OLED\_CMD);//--normal / reverse

OLED\_WR\_Byte(0xA8,OLED\_CMD);//--set multiplex ratio(1 to 64)

OLED\_WR\_Byte(0x3F,OLED\_CMD);//--1/32 duty

OLED\_WR\_Byte(0xC8,OLED\_CMD);//Com scan direction

OLED\_WR\_Byte(0xD3,OLED\_CMD);//-set display offset

OLED\_WR\_Byte(0x00,OLED\_CMD);//

OLED\_WR\_Byte(0xD5,OLED\_CMD);//set osc division

OLED\_WR\_Byte(0x80,OLED\_CMD);//

OLED\_WR\_Byte(0xD8,OLED\_CMD);//set area color mode off

OLED\_WR\_Byte(0x05,OLED\_CMD);//

OLED\_WR\_Byte(0xD9,OLED\_CMD);//Set Pre-Charge Period

OLED\_WR\_Byte(0xF1,OLED\_CMD);//

OLED\_WR\_Byte(0xDA,OLED\_CMD);//set com pin configuartion

OLED\_WR\_Byte(0x12,OLED\_CMD);//

OLED\_WR\_Byte(0xDB,OLED\_CMD);//set Vcomh

OLED\_WR\_Byte(0x30,OLED\_CMD);//

OLED\_WR\_Byte(0x8D,OLED\_CMD);//set charge pump enable

OLED\_WR\_Byte(0x14,OLED\_CMD);//

OLED\_WR\_Byte(0xAF,OLED\_CMD);//--turn on oled panel

OLED\_Clear();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OLED程序——oled.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_\_OLED\_H

#define \_\_OLED\_H

#include "sys.h"

#include "stdlib.h"

//-----------------OLED IIC端口定义----------------

#define OLED\_SCLK\_Clr() GPIO\_ResetBits(GPIOB,GPIO\_Pin\_6 )//SCL

#define OLED\_SCLK\_Set() GPIO\_SetBits(GPIOB,GPIO\_Pin\_6)

#define OLED\_SDIN\_Clr() GPIO\_ResetBits(GPIOB,GPIO\_Pin\_7)//SDA

#define OLED\_SDIN\_Set() GPIO\_SetBits(GPIOB,GPIO\_Pin\_7)

#define OLED\_CMD 0 //写命令

#define OLED\_DATA 1 //写数据

//OLED控制用函数

void OLED\_WR\_Byte(unsigned dat,unsigned cmd);

void OLED\_Display\_On(void);

void OLED\_Display\_Off(void);

void OLED\_Init(void);

void OLED\_Clear(void);

void OLED\_DrawPoint(u8 x,u8 y,u8 t);

void OLED\_Fill(u8 x1,u8 y1,u8 x2,u8 y2,u8 dot);

void OLED\_ShowChar(u8 x,u8 y,u8 chr,u8 Char\_Size);

void OLED\_ShowNum(u8 x,u8 y,u32 num,u8 len,u8 size);

void OLED\_ShowString(u8 x,u8 y, u8 \*p,u8 Char\_Size);

void OLED\_Set\_Pos(unsigned char x, unsigned char y);

void OLED\_ShowCHinese(u8 x,u8 y,u8 no);

void OLED\_DrawBMP(unsigned char x0, unsigned char y0,unsigned char x1, unsigned char y1,unsigned char BMP[]);

void Delay\_50ms(unsigned int Del\_50ms);

void Delay\_1ms(unsigned int Del\_1ms);

void fill\_picture(unsigned char fill\_Data);

void Picture(u8 i);

void IIC\_Start(void);

void IIC\_Stop(void);

void Write\_IIC\_Command(unsigned char IIC\_Command);

void Write\_IIC\_Data(unsigned char IIC\_Data);

void Write\_IIC\_Byte(unsigned char IIC\_Byte);

void IIC\_Wait\_Ack(void);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ESP8266程序——gizwits\_product.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "stdio.h"

#include "string.h"

#include "gizwits\_product.h"

#include "common.h"

#include "usart3.h"

#include "led.h"

#include "beep.h"

#include "usart.h"

//定义变量

static uint32\_t timerMsCount;

uint8\_t aRxBuffer;

uint16\_t timeuser = 0;

extern dataPoint\_t currentDataPoint;

//在这里控制蜂鸣器等

int8\_t gizwitsEventProcess(eventInfo\_t \*info, uint8\_t \*gizdata, uint32\_t len)

{

uint8\_t i = 0;

dataPoint\_t \*dataPointPtr = (dataPoint\_t \*)gizdata;

moduleStatusInfo\_t \*wifiData = (moduleStatusInfo\_t \*)gizdata;

protocolTime\_t \*ptime = (protocolTime\_t \*)gizdata;

#if MODULE\_TYPE

gprsInfo\_t \*gprsInfoData = (gprsInfo\_t \*)gizdata;

#else

moduleInfo\_t \*ptModuleInfo = (moduleInfo\_t \*)gizdata;

#endif

if((NULL == info) || (NULL == gizdata))

{

return -1;

}

for(i=0; i<info->num; i++)

{

switch(info->event[i])

{

case EVENT\_LED:

currentDataPoint.valueLED = dataPointPtr->valueLED;

GIZWITS\_LOG("Evt: LED%d\n", currentDataPoint.valueLED);

if(0x01 == currentDataPoint.valueLED)

{

if(1 == currentDataPoint.valueLED1)

LED0 = 0;

}

else

{

if(1 == currentDataPoint.valueLED1)

LED0 = 1;

}

break;

case EVENT\_BEEP:

currentDataPoint.valueBEEP = dataPointPtr->valueBEEP;

GIZWITS\_LOG("Evt: BEEP %d \n", currentDataPoint.valueBEEP);

if(0x01 == currentDataPoint.valueBEEP)

{

if(1 == currentDataPoint.valueLED1)

BEEP = 1;

}

else

{

if(1 == currentDataPoint.valueLED1)

BEEP = 0;

}

break;

case EVENT\_LED1:

currentDataPoint.valueLED1 = dataPointPtr->valueLED1;

GIZWITS\_LOG("Evt: LED1 %d \n", currentDataPoint.valueLED1);

if(0x01 == currentDataPoint.valueLED1)

{

LED2 = 0;

}

else

{

LED2 = 1;

}

break;

case EVENT\_Temp\_AlarmScope:

currentDataPoint.valueTemp\_AlarmScope = dataPointPtr->valueTemp\_AlarmScope;

GIZWITS\_LOG("Evt:EVENT\_Temp\_AlarmScope %d\n",currentDataPoint.valueTemp\_AlarmScope);

break;

case EVENT\_Shidu\_AlarmScope:

currentDataPoint.valueShidu\_AlarmScope = dataPointPtr->valueShidu\_AlarmScope;

GIZWITS\_LOG("Evt:EVENT\_Shidu\_AlarmScope %d\n",currentDataPoint.valueShidu\_AlarmScope);

break;

case EVENT\_GQ\_AlarmScope:

currentDataPoint.valueGQ\_AlarmScope = dataPointPtr->valueGQ\_AlarmScope;

GIZWITS\_LOG("Evt:EVENT\_GQ\_AlarmScope %d\n",currentDataPoint.valueGQ\_AlarmScope);

break;

case WIFI\_SOFTAP:

break;

case WIFI\_AIRLINK:

break;

case WIFI\_STATION:

break;

case WIFI\_CON\_ROUTER:

break;

case WIFI\_DISCON\_ROUTER:

break;

case WIFI\_CON\_M2M:

break;

case WIFI\_DISCON\_M2M:

break;

case WIFI\_RSSI:

GIZWITS\_LOG("RSSI %d\n", wifiData->rssi);

break;

case TRANSPARENT\_DATA:

GIZWITS\_LOG("TRANSPARENT\_DATA \n");

break;

case WIFI\_NTP:

GIZWITS\_LOG("WIFI\_NTP : [%d-%d-%d %02d:%02d:%02d][%d] \n",ptime->year,ptime->month,ptime->day,ptime->hour,ptime->minute,ptime->second,ptime->ntp);

break;

case MODULE\_INFO:

GIZWITS\_LOG("MODULE INFO ...\n");

#if MODULE\_TYPE

GIZWITS\_LOG("GPRS MODULE ...\n");

#else

GIZWITS\_LOG("WIF MODULE ...\n");

GIZWITS\_LOG("Type : [%d] \n",ptModuleInfo->moduleType);

#endif

break;

default:

break;

}

}

return 0;

}

void userInit(void)

{

memset((uint8\_t\*)&currentDataPoint, 0, sizeof(dataPoint\_t));

}

void gizTimerMs(void)

{

timerMsCount++;

timeuser++;

}

uint32\_t gizGetTimerCount(void)

{

return timerMsCount;

}

void mcuRestart(void)

{

\_\_set\_FAULTMASK(1);

NVIC\_SystemReset();

}

//发送给云服务器

int32\_t uartWrite(uint8\_t \*buf, uint32\_t len)

{

uint32\_t i = 0;

if(NULL == buf)

{

return -1;

}

for(i=0; i<len; i++)

{

USART\_SendData(USART1,buf[i]);

//循环发送,直到发送完毕

while(USART\_GetFlagStatus(USART1,USART\_FLAG\_TC)==RESET);

if(i >=2 && buf[i] == 0xFF)

{

USART\_SendData(USART1,0x55);

//循环发送,直到发送完毕

while(USART\_GetFlagStatus(USART1,USART\_FLAG\_TC)==RESET);

}

}

#ifdef PROTOCOL\_DEBUG

GIZWITS\_LOG("MCU2WiFi[%4d:%4d]: ", gizGetTimerCount(), len);

for(i=0; i<len; i++)

{

GIZWITS\_LOG("%02x ", buf[i]);

if(i >=2 && buf[i] == 0xFF)

{

GIZWITS\_LOG("%02x ", 0x55);

}

}

GIZWITS\_LOG("\n");

#endif

return len;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ESP8266程序——gizwits\_product.h

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_GIZWITS\_PRODUCT\_H

#define \_GIZWITS\_PRODUCT\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <stdint.h>

#include <stm32f10x.h>

#include "gizwits\_protocol.h"

extern uint16\_t timeuser;

#define SOFTWARE\_VERSION "03030000"

#define HARDWARE\_VERSION "03010100"

#define MODULE\_TYPE 0 //0,WIFI ;1,GPRS

extern dataPoint\_t currentDataPoint;

void gizTimerMs(void);

void timerInit(void);

void uartInit(void);

void userInit(void);

void userHandle(void);

void mcuRestart(void);

uint32\_t gizGetTimerCount(void);

int32\_t uartWrite(uint8\_t \*buf, uint32\_t len);

int8\_t gizwitsEventProcess(eventInfo\_t \*info, uint8\_t \*data, uint32\_t len);

#ifdef \_\_cplusplus

}

#endif

#endif