



黄淮学院

本科毕业设计

基于物联网的多场景预警系统设计

院 系：智能制造学院

姓 名：吴松

学 号：1534130208

专 业：电子信息工程

年 级：2015 级

指导教师：姚巧鸽

职 称：讲师

完成日期：2019 年 5 月

摘 要

本文介绍了一种以 ESP8266 芯片为处理核心,以 python 语言为程序语言,通过 GPIO 读取传感器信息,使用网络连接发送数据,在后端服务器进行处理分析返回交互的预警系统。

本设计选用了 ESP8266 芯片,具有低功耗,高集成度的特点。内部集成了 32 位处理器,标准数字外设接口,功率放大器和电源管理模块等元件,CPU 时钟速度最高可达 160MH 并且原生支持完整的 Wi-Fi 协议栈。使用了红外热释传感器,火焰传感器,烟雾气体传感器,温度传感器,超声波传距离传感器等元件用来检测环境信息,ESP8266 系统获取传感器信息后通过 WIFI 网络发送数据至后端服务器,再由服务器进行数据的获取,处理分析与返回交互等任务。本次设计中完成了系统的硬件连接,ESP8266 系统上运行环境的部署与程序的编写,后端服务器程序的编写与交互页面的编写等任务。

关键词: 物联网; Python; ESP8266; 传感器

Abstract

This paper introduces an early warning system that uses ESP8266 chip as the core, python language as the programming language, reads sensor information through GPIO, sends data through network, and analyses and returns interactive in back-end server.

This design uses the ESP8266 chip , which has low power consumption and high integration. It integrates 32-bit cpu, standard digital peripheral interface, power amplifier and power management module, CPU clock speed up to 160MH, and supports the complete Wi-Fi protocol stack. It uses infrared pyroelectric sensors, flame sensors, smoke gas sensors, temperature sensors, ultrasonic sensors and other components to detect environmental information. After obtaining the sensor information, the ESP8266 system sends the data to the back-end server through the WIFI network, and then the server obtains the data, processes the analysis and returns the interactive task. In this design, the hardware connection of the system, the deployment of the running environment and the writing of the program on the ESP8266 system, the writing of the back-end server program and the preparation of the interactive page are completed.

Key words: Internet of Things; Python;ESP8266;sensor

独 创 性 声 明

本人郑重声明：所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得黄淮学院或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：

年 月 日

学位论文版权使用授权书

本学位论文作者完全了解黄淮学院有关保留、使用学位论文的规定。特授权黄淮学院可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校按规定向国家有关部门或机构送交论文和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：

年 月 日

导师签名：

年 月 日

目 录

| | |
|------------------------|----|
| 1 绪论..... | 1 |
| 1.1 课题研究的背景..... | 1 |
| 1.2 国内外研究现状..... | 1 |
| 1.3 本文所做的工作..... | 2 |
| 2 系统总体设计..... | 3 |
| 2.1 任务分析与实现方式..... | 3 |
| 2.2 硬件系统总体设计..... | 3 |
| 2.2.1 处理芯片..... | 3 |
| 2.2.2 传感器模块..... | 4 |
| 2.3 软件系统总体设计..... | 4 |
| 2.3.1 编程语言..... | 4 |
| 2.3.2 软件系统组成..... | 4 |
| 3 系统硬件设计..... | 6 |
| 3.1 概述..... | 6 |
| 3.2 ESP8266 芯片系统 | 6 |
| 3.2.1 USB 转串口 | 6 |
| 3.2.2 稳压电路..... | 7 |
| 3.2.3 按键电路..... | 7 |
| 3.3 传感器..... | 8 |
| 3.3.1 红外感应..... | 8 |
| 3.3.2 火焰检测..... | 8 |
| 3.3.3 气体检测..... | 9 |
| 3.3.4 温度检测..... | 9 |
| 3.3.5 距离检测..... | 10 |
| 3.5.6 AD/DA 转换 | 10 |

| | |
|----------------------------------|----|
| 4 系统软件设计..... | 11 |
| 4.1 软件任务分析..... | 11 |
| 4.2 ESP8266 的固件烧录 | 11 |
| 4.3 部分传感器驱动程序 | 13 |
| 4.3.1 DS18B20 的单总线协议驱动程序 | 13 |
| 4.3.2 PCF8591 的 I2C 总线驱动程序 | 14 |
| 4.3.3 HC-SR04P 的驱动程序设计 | 16 |
| 5 系统测试..... | 18 |
| 5.1 硬件线路连接..... | 18 |
| 5.2 局域网环境下的系统测试..... | 18 |
| 5.3 广域网环境下的系统测试..... | 20 |
| 5.4 交互页面测试..... | 21 |
| 6 总结 | 23 |
| 6.1 感想..... | 23 |
| 6.2 遇到的问题与不足..... | 23 |
| 6.3 扩展与展望..... | 24 |
| 参考文献..... | 25 |
| 致 谢..... | 26 |
| 附 录..... | 27 |

1 绪论

1.1 课题研究的背景

中国是世界上发展最快的大国，在改革开放后的几十年里已经超过了日本的两倍GDP 成为世界第二大经济体。而且不仅仅是在经济上面发展迅速，在军事，科技等方面也飞速发展，更在一些方面成为世界上的领导者。

近年来，中国互联网以惊人的速度发展。在国家政策的大力支持下，运营商不断进行设备升级，提速降费。从无到有，到现在中国互联网现在已经成为全世界第一大网，用户最多，区域最大。而以互联网为载体的物联网也得到急速发展，在各方各面推进着社会的发展进步。

随着科学技术的创新，安防预警技术也出现了时代性的特征。包括网络化、集成化、智能化。智能化安防预警是在新时代安防的必然趋势^[1]。

随着行业的发展与相关产业的成熟，智能安防预警系统的设备成本已经不在如之前那样高昂，但是当前市场上已存在各种各样的安防产品，都有各自的缺陷诸如成本，场合，交互操作困难等问题，无法有效的适应多种场景。所以需要一种价格低廉，交互简单，组成灵活，模块化配置以适应不同场景的设计。

1.2 国内外研究现状

随着物联网，云计算，大数据，人工智能等技术的发展，智能化安防预警已经成为新时代安防技术的发展方向，行业需求稳定增长。在 2018 年里，我国该行业总产值以接近七千五百亿元，还在保持高速增长了，许多大型企业都纷纷选择进入这个领域来提升自己的价值和影响力。

在过去的几年里，亚马逊已经收购许多智能安防相关的企业，谷歌也是如此。亚马逊旗下的 Ring 和谷歌旗下的 Nest 都发布了 DIY 安防系统。但还是有着众多的业务专注的小型企业取得一定的市场份额。我国的许多企业也都开始部署自己的 IOT 生态，以海康、大华、宇视、东方网力等大量传统安防厂商进入智能安防领域，以通信设备起家的华为和视频存储硬件起家的浪潮等企业也纷纷加入到这一行列，形成百花齐放的局面。

1.3 本文所做的工作

本文主要工作是利用 ESP8266 芯片和传感器组合，完成环境信息采集和通信，在服务端进行处理分析最终于用户完成交互。主要介绍了系统的设计思想，设计方案，元器件选择，传感器的使用分析，通信方式，程序编写等。整体上分为硬件部分的设计，ESP8266 设备上的信息采集及通信的程序设计和位于服务端上的程序设计。

硬件部分主要为器件的选择如处理芯片的选择，传感器的选择，电路的设计等。

软件部分由 ESP8266 设备上的程序和后端服务器上运行的程序组成。ESP8266 设备上的程序需要完成的功能有系统的初始化，将设备连接入网络，采集传感器数据，数据转码，建立 SOCKET 通信，发送数据到服务器。运行在服务端的程序需要完成的功能有监听通信，获取数据，分析处理，存入数据库等。还需要有负责交互的程序在收到请求后提取数据库中的相关数据分析处理后响应交互。

2 系统总体设计

2.1 任务分析与实现方式

本此设计任务是：以 ESP8266 芯片为核心，读取多种传感器数据并通过建立 SOCKET 通信与服务端完成信息交互，在服务端进行数据处理分析和用户交互任务。

本系统总体思路如下：将 ESP8266 烧入 MicroPython 的系统固件，建立 python 语言在嵌入式和微控制器上的运行环境，使用 python 程序语言编写程序获取传感器信息，通过引出 GPIO 引脚完成与传感器进行简单的数字信息获取和相对复杂的串行总线协议的实现。并建立 SOCKET 通信即使的将信息发送到后端服务器，在后端服务器进行数据处理与储存并进一步分析处理后响应用户交互。系统总体框图如图 1-1 所示。

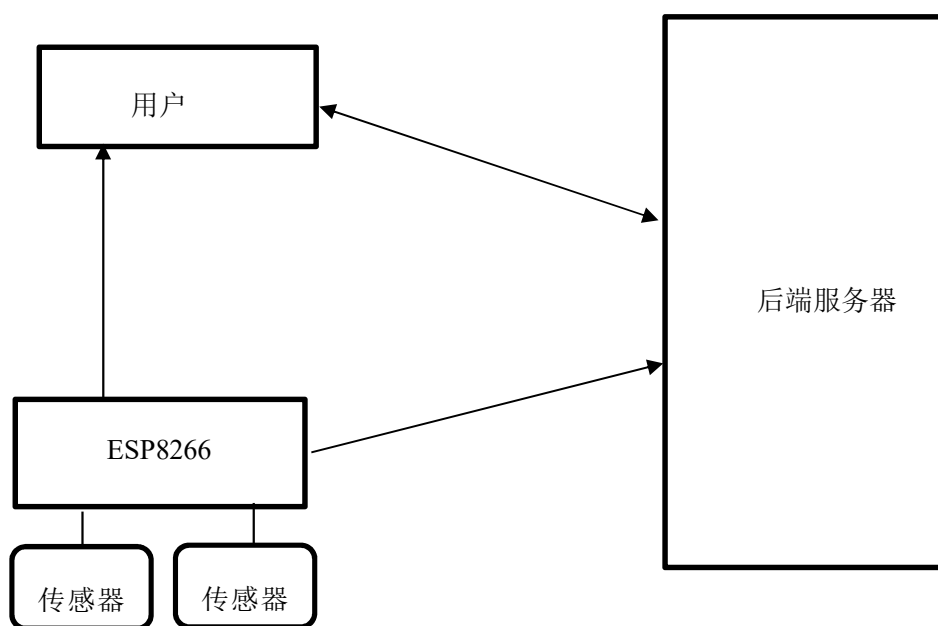


图 1-1 总体框图

2.2 硬件系统总体设计

2.2.1 处理芯片

此设计选择以 ESP8266 芯片作为传感器信息采集核心处理器。

ESP8266 是乐鑫公司出品的一款物联网芯片，价格低廉，工作稳定，功耗理想，而且集成了完整的 Wi-Fi 协议栈和 32 位的超低功耗处理器，主频最高可达 160MHz。官方完

全开源提供了基于 AT 指令的 SDK 工具包，也可以根据自己需要编译自己需要的具有独特功能的固件。

此设计中使用了 NodeMCU 封装的 ESP8266 串口模块。实物如图 2-1 所示。



图 2-1 NodeMCU

2.2.2 传感器模块

为了适应多种场景,此设计中使用了红外感应,火焰,温度,烟雾气敏,超声波多种用于检测环境的传感器以及用于处理模拟信号的 AD/DA 转换模块。

2.3 软件系统总体设计

2.3.1 编程语言

由于乐鑫公司对 ESP8266 系列开源的大力支持,除了官方提供的基于 AT 指令的 Non-OS SDK,还有基于 FreeRTOS 的 RTOS SDK,原生的 c 语言开发,基于 MircoPython 的 Python 语言开发,基于 Arduino 固件的语言,NodeMCU 提供的 Lua 语言固件,甚至支持 Espruino 固件的 JavaScript 语言开发。此次设计中考虑到 ESP8266 的性能和资源,任务的需求及效率,选择使用 MircoPython 固件,使用 python 作为程序语言。

2.3.2 软件系统组成

此设计的软件系统由 ESP8266 芯片上的程序与后端服务器上的服务端程序共同组成,ESP8266 芯片上的程序负责处理传感器数据和与服务端建立 SOCKET 通信进行数据发送,服务端上的程序负责监听通信,获取数据,对数据进行储存分析。ESP8266 芯片及

传感器作为客户机的模式，后端服务器作为服务器与客户机共同组成 c/s 结构。系统总体流程图如图 2-2。

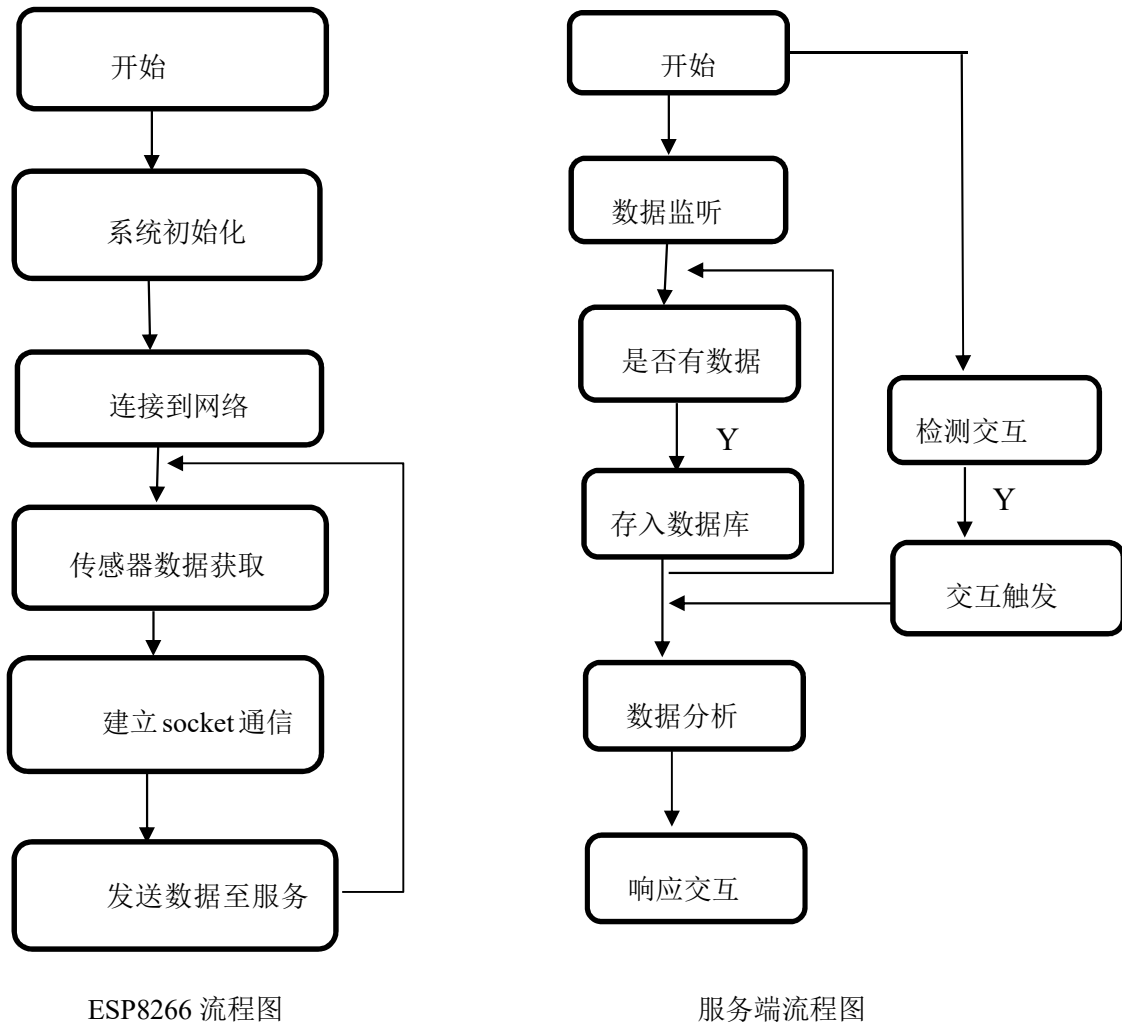


图 2-2 软件系统流程图

3 系统硬件设计

3.1 概述

此设计采用 NODEMCU 的 ESP8266 芯片对传感器进行信息采集。

3.2 ESP8266 芯片系统

3.2.1 USB 转串口

ESP8266 刷入 MicroPython 的固件后是通过访问 REPL 交互式解析器来进行运行命令和代码测试的。在 ESP8266 设备上，可以通过使用有限连接的方式听通过 UART 串口协议来访问交互式解析器，也可以将 ESP8266 设备通过 WIFI 连接入网络，通过 TCP/IP 协议来进行交互式解析器的访问。但是由于安全问题，MicroPython 上的 WebREPL-WiFi 功能是默认关闭的，即使是要使用 WebREPL-WiFi，也需要先通过有线连接的方式通过交互式解析器来启动 WebREPL-WiFi 环境与设置安全密码。由于现在的计算机和笔记本电脑设备都不在配置串口，为了在 pc 端上实现与 ESP8266 设备的通信，需要进行 USB 转串口来实现计算机的 USB 接口到通用串口之间的转换。此设计使用的 NODEMCU 采用了 CH340G 来进行 USB 转串口。如图 3-1 所示。

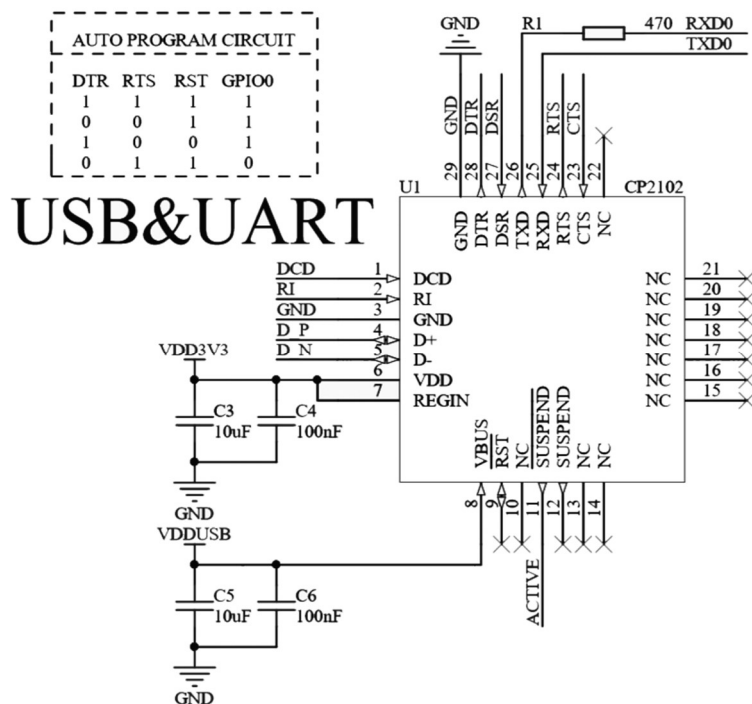


图 3-1 CH340G USB 转串口

3.2.2 稳压电路

SPX3819M5-L-3-3 稳压器，具有高精度，低噪声，低压降等特点的线性稳压器。NODEMCU 采用此稳压器来给 ESP8266 系统提供稳定的供电。避免因为电源的不稳定对系统产生影响。如图 3-2 所示。

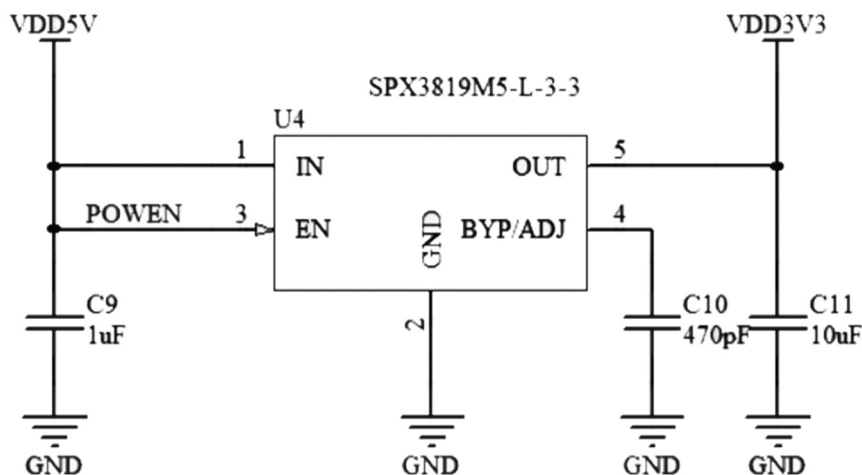


图 3-2 稳压电路

3.2.3 按键电路

NODEMCU 上具有两个按键，分别控制 ESP8266 的复位与运行状态。如图 3-3 所示。

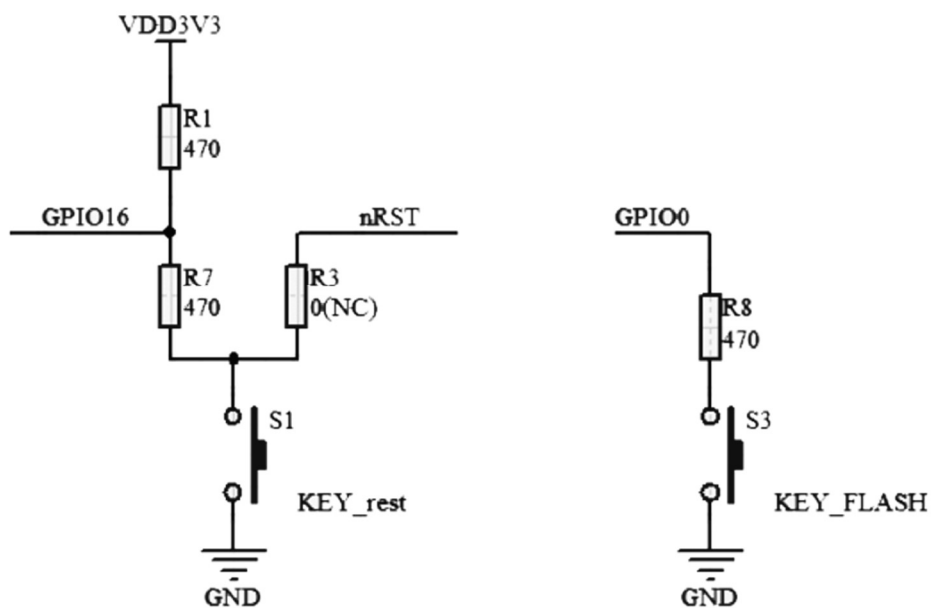


图 3-3 按键电路

3.3 传感器

3.3.1 红外感应

HC-SR501 通过热释电效应来检测人体发出的特定波长的红外线^[2]，此处传感器信号输出端口接 ESP8266 的 D0 口 GPIO16。ESP8266 检测 D0 引脚来检测传感器状态。如图 3-4 所示。

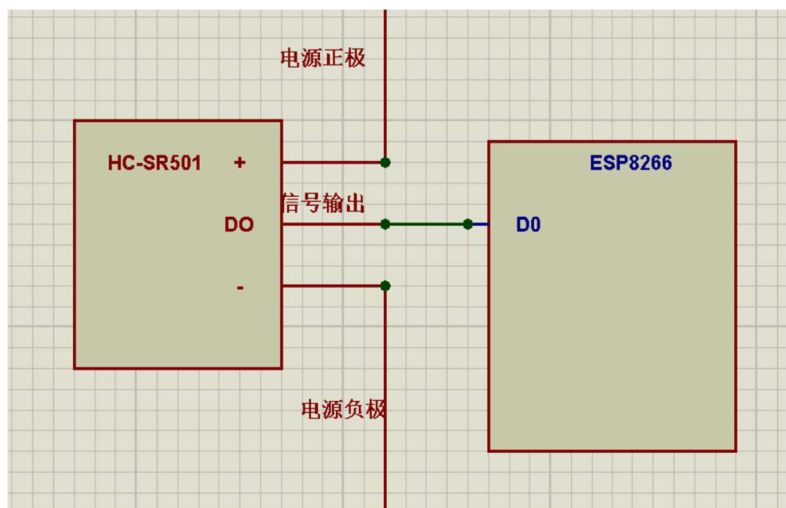


图 3-4 红外感应

3.3.2 火焰检测

此传感器可以检测火焰或相似的波长的光源，具有数字开关输出量和模拟电压输出。可以通过电位器来进行灵敏度调节以适应不同的环境。模拟电压输出可以通过 AD 转换得到更高的精度。此处数字量输出接入 ESP8266 的 D1 口，GPIO5。如图 3-5 所示。

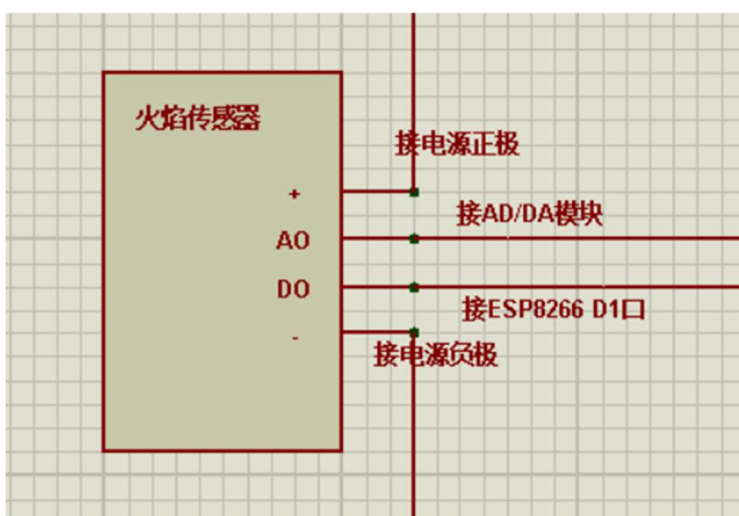


图 3-5 火焰检测

3.3.3 气体检测

此传感器可以检测空气中的可燃气体与烟雾，具有数字开关输出量和模拟电压输出。连接 ESP8266GPIP 与 AD/DA 转换模块后可以进行气体检测预警。此处数字输出接入 ESP8266 的 D3 口，模拟电压输出接 AD/DA 转换模块。模拟量电压越高则空气中气体浓度越大。通过 AD/DA 转化获取更高的精度，GPIO0。如图 3-6 所示。

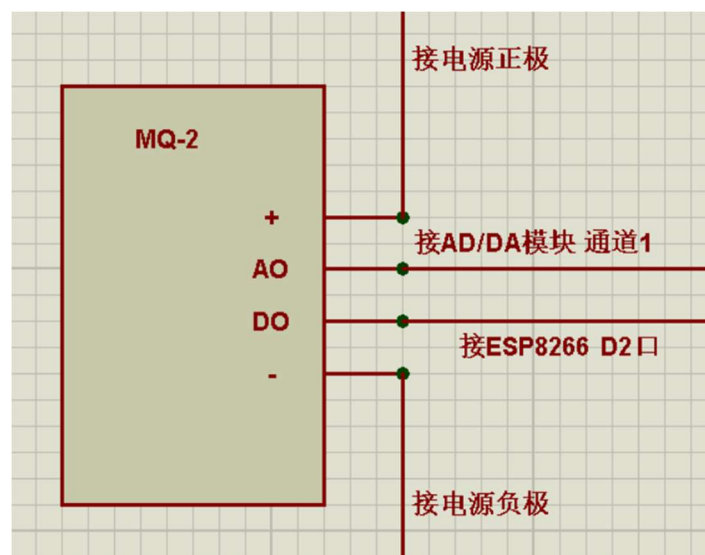


图 3-6 气体检测

3.3.4 温度检测

温度检测采用 DS18B20 传感器。通过单总线协议与 ESP8266 通信，信号输出端口接 ESP8266 的 D4 口，GPIO2。如图 3-7。

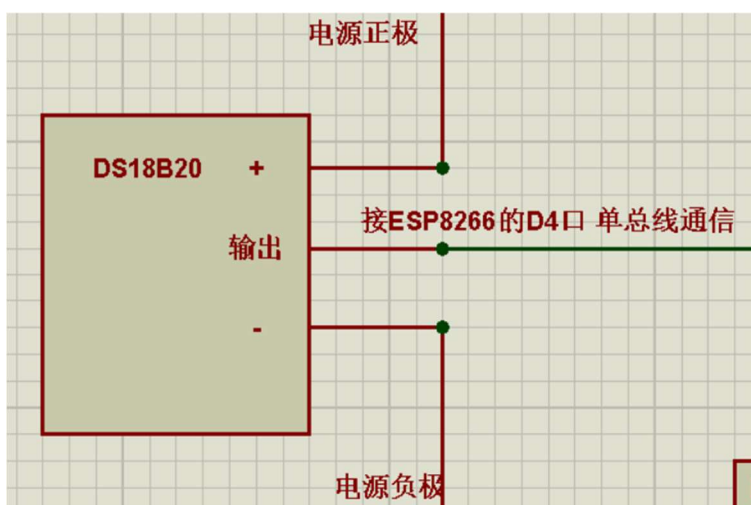


图 3-7 温度检测

3.3.5 距离检测

此处采用 HC-SR04P，HC-SR04P 的 TRIG 控制端口接 ESP8266 的 D6 口，ECHO 接收端口接 ESP8266 的 D5 口，GPIO 分别是 GPIO12 和 GPIO14。如图 3-8 所示。

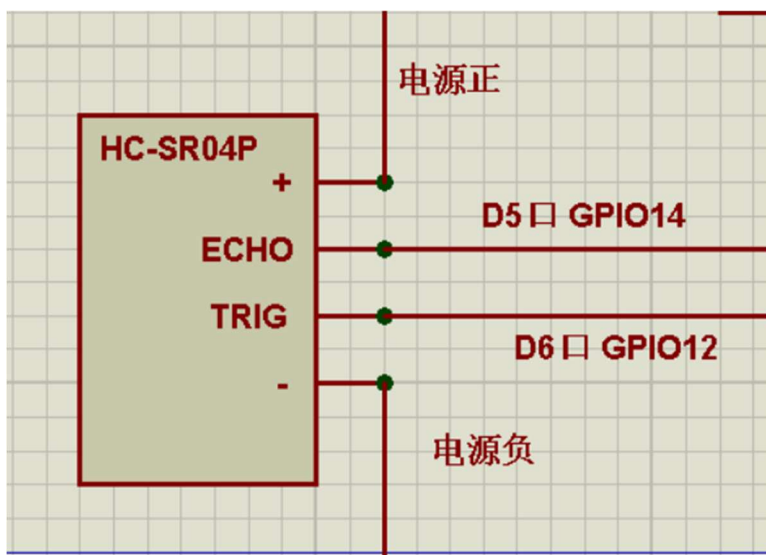


图 3-8 距离检查

3.5.6 AD/DA 转换

此处使用了 PCF8591 模块，PCF8591 具有四路 8-bit 模数转换，一路 8-bit 数模转换。通过 I2C 总线输出经过转换后的模拟输入值。如图 3-9 所示。

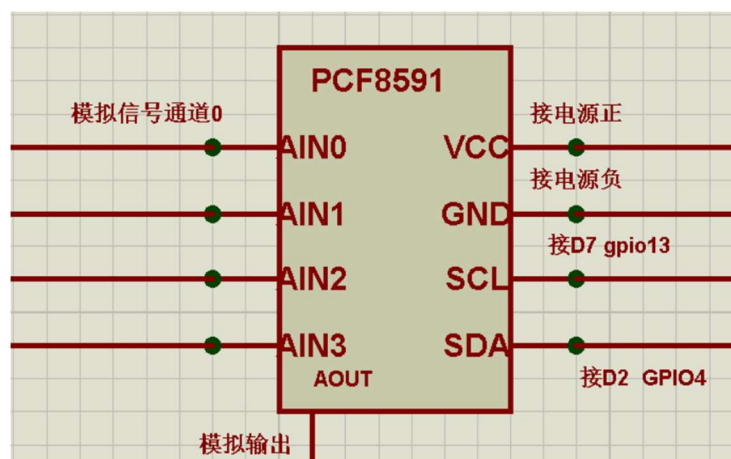


图 3-9 AD/DA 转换

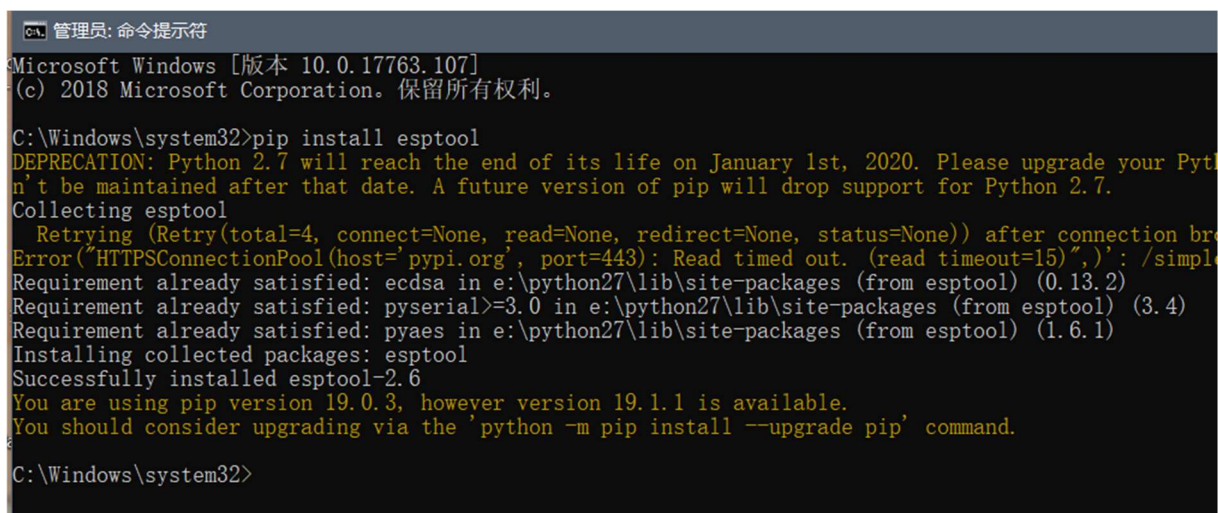
4 系统软件设计

4.1 软件任务分析

在此设计中，系统判断环境信息首先需要对环境信息进行收集，而信息的收集是通过传感器的操作进行的，就需要有处理各种传感器的窗处理程序。信息收集完成之后需要传送信息到后端服务器，需要与服务器建立通信，所以需与后端服务器建立通信交互的通信程序。在服务器端需要接收客户端传送的数据故需要服务端的数据监听程序，服务端接收数据之后需要对数据进行保存，存入数据库，所以需要对数据库进行操作的程序。^[3]

4.2 ESP8266 的固件烧录

要在 ESP8266 上运行进行 python 语言开发，需要建立 python 的运行环境。根据 MicroPython 官方文档上的固件部署指南，首先要下载 ESP8266 的固件，然后通过 pip 工具安装 esptool 的 python 烧录程序。通过在 windows 上下载安装 python 并添加环境变量后就可以使用 pip 包管理器，在 Windows 命令提示符输入以下指令通过 pip 包管理器安装 esptool。如图 4-1 所示。



```
管理员: 命令提示符
Microsoft Windows [版本 10.0.17763.107]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Windows\system32>pip install esptool
DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Please upgrade your Python to Python 3.x to avoid this deprecation warning.
Collecting esptool
  Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by HTTPConnectionPool(host='pypi.org', port=443): Read timed out. (read timeout=15)
Requirement already satisfied: ecdsa in e:\python27\lib\site-packages (from esptool) (0.13.2)
Requirement already satisfied: pyserial>=3.0 in e:\python27\lib\site-packages (from esptool) (3.4)
Requirement already satisfied: pyaes in e:\python27\lib\site-packages (from esptool) (1.6.1)
Installing collected packages: esptool
Successfully installed esptool-2.6
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Windows\system32>
```

图 4-1 pip 包管理器安装 esptool

esptool 安装完成后就可以通过 esptool 对 ESP8266 固件进行擦除与烧录。

首先将 ESP8266 通过 usb 连接到 Windows，在 windows 上安装 ch340G 驱动程序后就可 Windows 上识别 esp8266 的端口了。通过 windows 设备管理器得知 esp8266 的端口：com32。如图 4-2 所示。

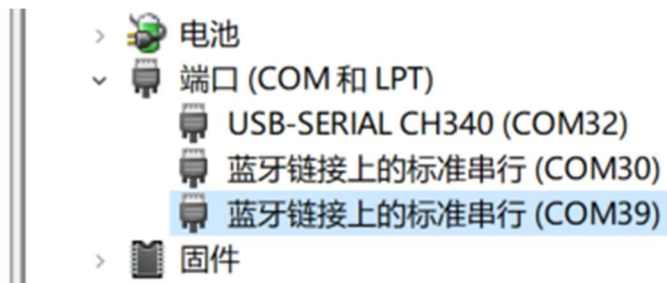


图 4-2 esp8266 端口

在 windows 终端中输入命令擦除 ESP8266 闪存中原本的固件：如图 4-3 所示。

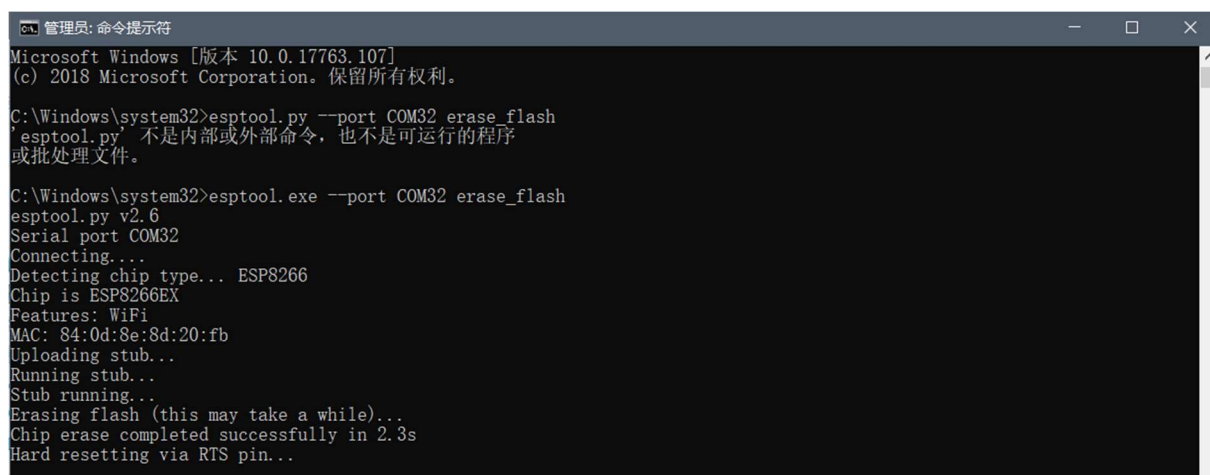


图 4-3 擦除 esp8266 固件

输入指令烧入 MicroPython 的固件：如图 4-4 所示。

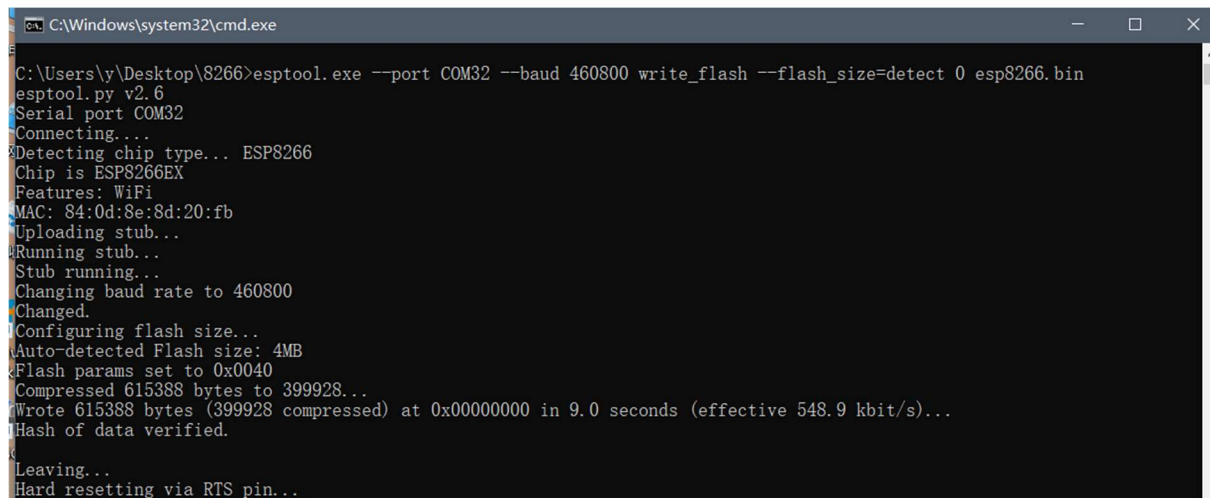


图 4-4 esp8266 固件烧录

固件烧录成功之后就可以通过 Xshell 或者 putty 等软件通过串口进入 ESP8266 的交互式解析环境，对设备进行进一步的配置与测试。如图，通过波特率 115200 成功连接。

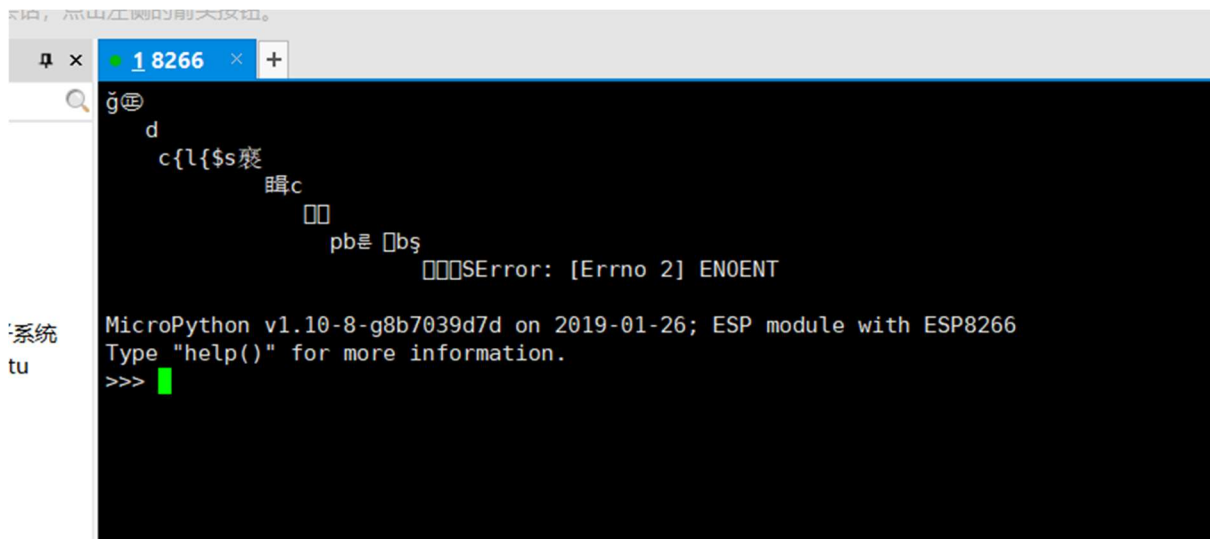


图 4-5 串口连接

4.3 部分传感器驱动程序

系统是通过驱动程序来控制传感器进行数据收集工作的，部分简单的传感器只需要通过读取数据接口引脚状态就可以得到需要的环境信息，但是有的传感器发送的信息不叫复杂，无法通过简单的引脚状态来得到想要的信息，就需要专门的驱动程序来实现高级的通信协议。

4.3.1 DS18B20 的单总线协议驱动程序

根据 DS18B20 的数据手册，得到以下信息

(1) 在 DS18B20 的内部有一个制只读 64 位的光刻 rom，为此设备的硬件地址，而且每个设备的地址都是不重复的，这样可以通过对硬件地址的寻找选择响应的设备从而实现多设备被挂载在同一跟总线下。

(2) DS18B20 的重要功能指令

CCH 指令：来跳过 rom 选择。44H 指令：来启动设备的温度转换。BEH： 允许读取暂存器。

(3) 驱动程序设计

数据手册中提供了传感器的控制流程。对传感器的控制首先要对传感器进行复位操作。复位后发送控制指令寻找指定设备，根据传感器内部光刻 rom 地址选择器件，当总

线上只存在一个设备时,可以通过指令跳过 rom 地址选择,之后发送传感器操作功能指令,完成温度测试。

MicroPython 实质上是由 c 语言编写实现的 python 解释器,所以执行速度比原生 c 语言低。虽然在绝大部分任务中不会因为性能的原因产生影响,但是在通信协议上无法精准实现微秒级的时隙控制,如果触发垃圾回收机制也可能会对通信产生干扰,所以通过软件延时的方式来实现所需要的通信协议可能会收到干扰。但是固件本身提供了位于硬件底层相关通信协议的驱动程序比如单总线, SPI 总线, I2C 总线等。可以通过调用相关的底层驱动,无需再进行软件延时,避免干扰。

在此驱动程序例程中,引入 giop 和单总线底层驱动的库后,在 GPIO2 引脚上生成一个单总线驱动程序对象。调用.reset()方法进行初始化总线,在此单总线系统中只挂载一个 DS18B20 设备时,无需进行设备选择,所以发送 CCH 指令来跳过 rom 选择。然后通过发送 44H 指令来启动设备的温度转换,接着再次进行初始化总线,发送 CCH 指令来跳过 rom 选择,发送读取暂存器指令 BEH 后,开始从寄存器中读取温度数据,进行转后后输出温度。如图 4-6,成功输出当前温度值。

```
=== def wendu():
===     ow = onewire.OneWire(Pin(2))
===     ow.reset()
===     ow.writebyte(0xCC)
===     ow.writebyte(0x44)
===     ow.reset()
===     ow.writebyte(0xCC)
===     ow.writebyte(0xBE)
===     d1=ow.readbyte()
===     d2=ow.readbyte()
===     a=(d1+d2*256)/16
===     print('温度值:'+str(a))
>>> wendu()
温度值:20.375
>>> █
```

图 4-6 驱动程序测试

4.3.2 PCF8591 的 I2C 总线驱动程序

根据器件数据手册得知: I2C 总线系统通过发送 3 个控制字到传感器来进行设备的控制。在总线初始化后, PCF8591 的硬件地址作为第一个字节发送到 PCF8591。第二个字节用于控制 PCF8591 的功能^[4], 如通道选择, 自动增量是否开启和模拟量输入方式等。第三个控制字用于控制 DAC 转换。此设计中无需进行 DAC 转换。PCF8591 具有 3 个地址引脚用于进行硬件地址编程。图 4-7 为 PCF8591 原理图。

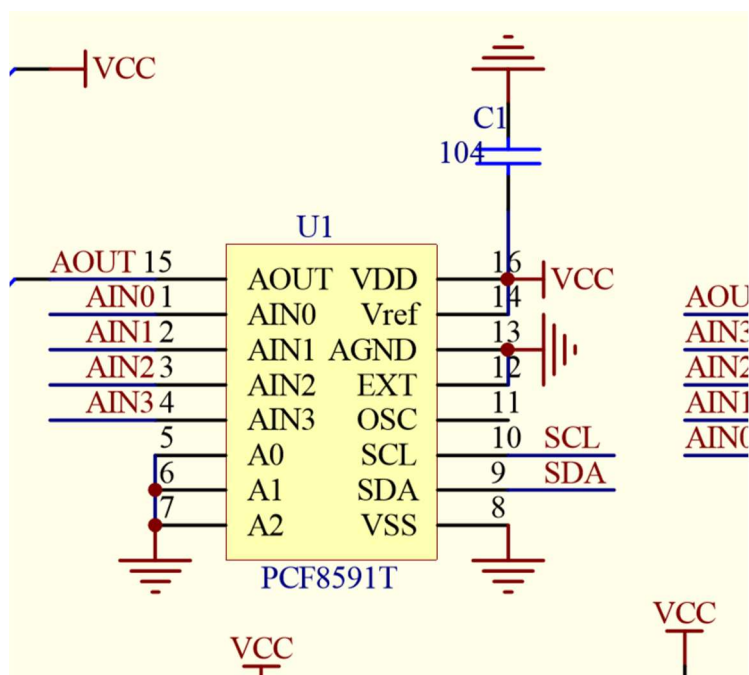


图 4-7 PCF8591 原理图

根据图 4-7 PCF8591 原理图 A2, A1, A0 接地可知 A2, A1, A0 处于低电平状态, 所以硬件地址为 1001000, 0x48。

首先通过 I2C 总线协议, 发送地址选择字节, 选择设备后向选择的设备写入控制字节, 选择需要进行工作的通道, 启动 pcf8591 工作, 之后再次选择改地址, 从设备读取字节, 对字节进行转码后输出。如图 4-8 所示。

```
>>> from machine import Pin, I2C
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== def run():
===     PCF8591 = 0x48 # I2C bus address
===     PCF8591_ADC_CH0 = '\x00' # thermistor
===     PCF8591_ADC_CH1 = '\x01' # photo-voltaic cell
===     PCF8591_ADC_CH2 = '\x02'
===     PCF8591_ADC_CH3 = '\x03' # potentiometer
===     gpio_scl = Pin(13)
===     gpio_sda = Pin(4)
===     i2c = I2C(scl=gpio_scl, sda=gpio_sda, freq=100000)
===     i2c.writeto(PCF8591, PCF8591_ADC_CH0)
===     i2c.readfrom(PCF8591, 1)
===     data = i2c.readfrom(PCF8591, 1)
===     a=str(ord(chr(data[0])))
===     print('adc 0: ' + a)
>>> run()
adc 0: 10
>>>
```

图 4-8 驱动程序测试

4.3.3 HC-SR04P 的驱动程序设计

HC-SR04P 的控制流程：通过给控制端 TRIG 端口一个 10US 以上的高电平电平信号来触发传感器启动，传感器启动之后会发送 8 个 40KHZ 的方波声波信号，并在信号接受端口 ECHO 端口触发高电平信号。^[5]如果检测到当超声波模块接收到返回的声波信号时，ECHO 端口从高电平转换到低电平。ECHO 端口触发高电平的时间就是超声波信号发送到返回的时间。这个时间乘以声速除以 2 所得就是传感器所测量的距离。下图 4-9 为控制流程。

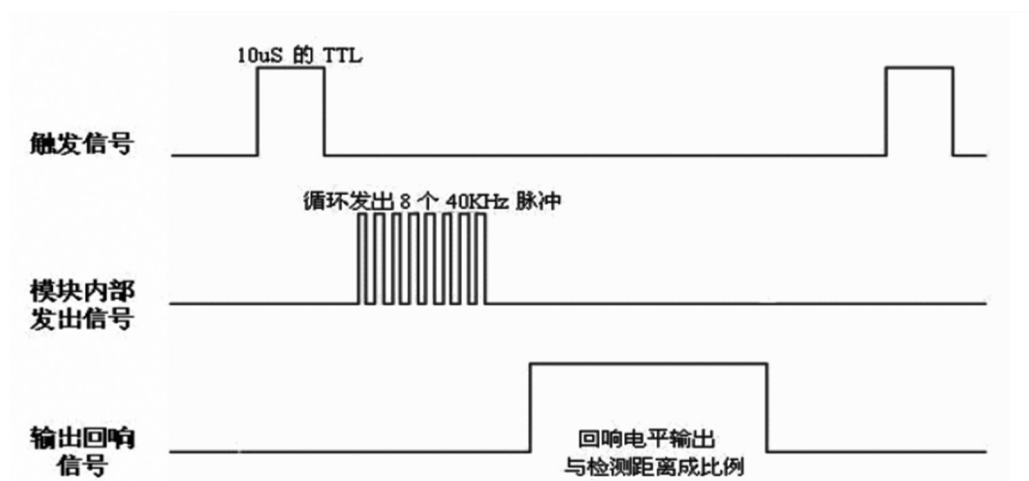


图 4-9 HC-SR04P 控制流程

驱动程序例程如图 4-10 所示，输出超声波传感器所测量得的距离。

```
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== import utime
=== from machine import Pin
=== def juli():
===     trig=Pin(12, Pin.OUT, value=0)
===     echo=Pin(14, Pin.IN)
===     trig.on()
===     utime.sleep_us(20)
===     trig.off()
===     while echo.value()==0:
===         pass
===     a=utime.ticks_us()
===
===     while echo.value()==1:
===         pass
===     b=utime.ticks_us()
===     c=str((b-a)*344*0.5*0.000001)
===     print('距离 :'+str(c)+'m')
===
>>> juli()
距离 :0.40764m
>>>
```

图 4-10 超声波传感器驱动程序例程

4.4 与服务端建立 SOCKET 通信程序

通过使用 MicroPython 固件的 `usocket` 库可以简单的实现发送数据到服务端。先生成一个 `cosket` 实例，通过此实例连接向服务端，完成数据发送后关闭实例，进行内存回收。在此设计中，通过循环读取传感器信息，将相关信息按顺序存入一个列表，所有传感器信息收集完毕后对此列表进行格式转换，通过 SOCKET 通信发送给服务端进行处理。

4.5 服务端程序与交互页面

服务端程序通过监听约定网络端口来获取客户端发送的数据进行初步的处理并将数据打上时间标签储存入数据库，通过对数据库的 SELECT 操作获取需要的数据进行分析处理完成交互。这里编写了一个简单的 http 服务，收到 http 请求后从返回数据库中查询到的最新的数据。

5 系统测试

5.1 硬件线路连接

各传感器与 ESP8266 的引脚连接

表 5-1 引脚连接

| 传感器模块 | 封装引脚 | GPIO 引脚 |
|----------------------------|------|---------|
| HC-SR501 红外感应模块 | D0 | GPIO16 |
| 4 针 火焰传感器 | D1 | GPIO 5 |
| ZYMQ-2 烟雾气敏传感器 | D3 | GPIO 0 |
| DS18B20 模块 单总线 | D4 | GPIO 2 |
| HC-SR04P 超声波测距 ECHO 接收端 | D5 | GPIO 14 |
| HC-SR04P 超声波测距 trig 控制端 | D6 | GPIO 12 |
| PCF8591 AD/DA 转换 scl(蓝) | D7 | GPIO 13 |
| PCF8591 AD/DA 转换 sda | D2 | GPIO 4 |

5.2 局域网环境下的系统测试

在服务端与 ESP8266 同于局域网环境下的测试。通过开启笔记本电脑的移动热点网络共享，配置 ESP8266 连接笔记本电脑共享的 WIFI 网络。在笔记本设备上开启服务端程序进行测试。

```
以太网适配器 以太网:
    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址. . . . . : fe80::65df:b54b:af0e:944e%26
    IPv4 地址 . . . . . : 10.134.0.55
    子网掩码 . . . . . : 255.254.0.0
    默认网关. . . . . : fe80::6415:cdb:37e9:640d%26
                        10.135.255.254
E:\BY\8266\NODEMCU资料\chengxu>python s.py
```

图 5-2 局域网环境下的服务端程序测试

上图 5-2 为局域网环境下的 Windows 用命令提示符运行服务端程序，此时 ESP8266 还未开始工作，服务端未收到消息，程序保持监听状态。

通过命令提示符运行 ipconfig 命令获取电脑的内网 IP 地址，在 ESP8266 程序上修改相应地址并运行。在终端上进行程序测试。如下图 5-3 所示。

```

循环计数2
红外值0
火焰值1
气体值1
温度值:23.8125
adc 0: 8
adc 1: 199
adc 2: 125
adc 3: 0
距离 :1.10596m
['2', 0, 1, 1, 23.8125, '8', '199', '125', '0', '1.10596']
str :2 0 1 1 23.8125 8 199 125 0 1.10596

循环计数3
红外值0
火焰值1
气体值1
温度值:23.75
adc 0: 8
adc 1: 200

```

图 5-3 局域网环境下的 ESP8266 串口调试图

在 ESP8266 上运行程序之后，服务端程序收到 ESP8266 设备发送的数据

```

C:\Windows\system32\cmd.exe - python s.py

[C:\Users\y\Desktop\8266]>python s.py
Accept new connection from 10.134.0.55:61661...
<type 'str'>
1 0 1 1 24.6875 8 200 125 0 1.1039
Connection from 10.134.0.55:61661 closed.
Accept new connection from 10.134.0.55:61662...
<type 'str'>
2 0 1 1 23.8125 8 199 125 0 1.10596
Connection from 10.134.0.55:61662 closed.
Accept new connection from 10.134.0.55:61663...
<type 'str'>
3 0 1 1 23.75 8 200 125 0 1.00482
Connection from 10.134.0.55:61663 closed.
Accept new connection from 10.134.0.55:61664...
<type 'str'>
4 0 1 1 23.8125 8 199 125 0 1.01652
Connection from 10.134.0.55:61664 closed.
Accept new connection from 10.134.0.55:61665...

```

图 5-4 局域网环境下的服务端程序测试

QliteSpy 是 sqlite 数据库可视化管理工具，通过此工具来查看服务端程序是否成功储存数据到数据库，通过 QliteSpy 的 File, OpenDatabase 选择数据库文件，打开查看在 user 表中由服务端程序所储存的数据，如下图 5-5 所示。

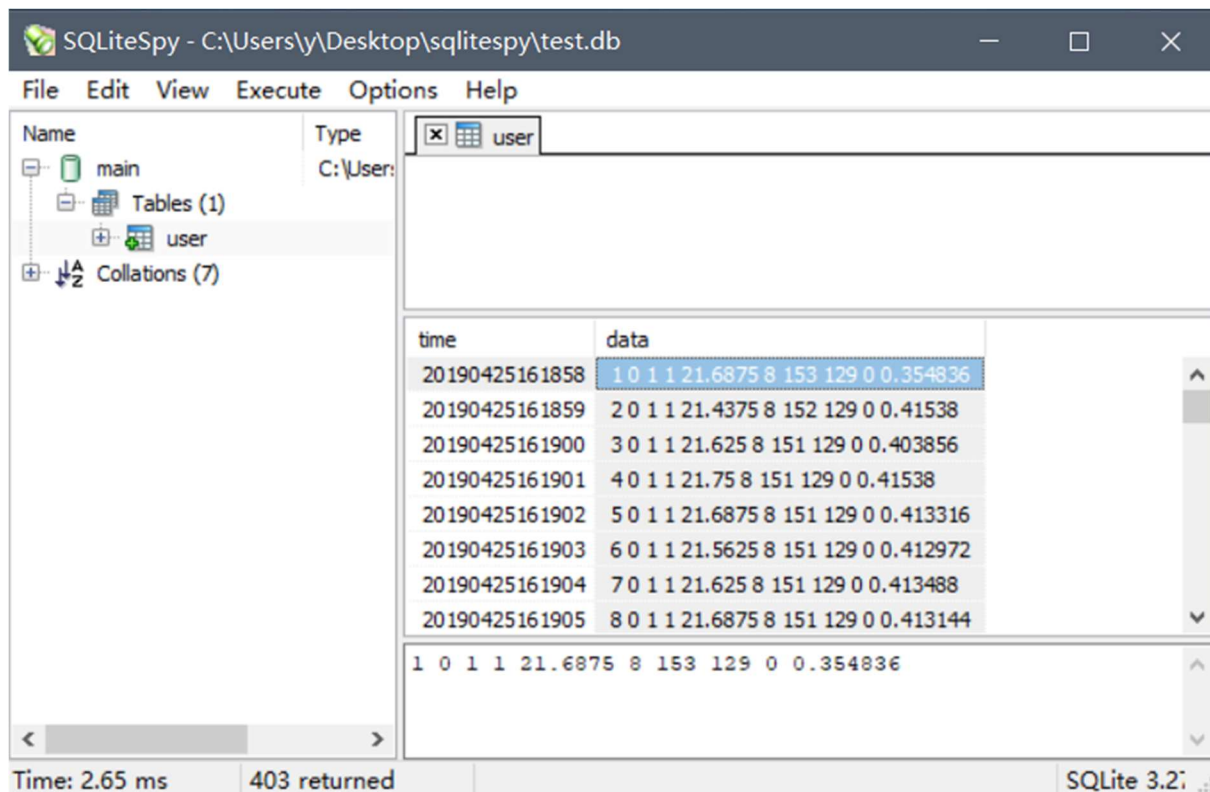


图 5-5 数据库可视化

5.3 广域网环境下的系统测试

此处将服务端程序部署于亚马逊云服务器上。此时 ESP8266 未启动程序。如图 5-6 所示。

图 5-6 广域网环境下的服务端程序测试

```

ubuntu@ip-172-31-30-207:~/8266$
ubuntu@ip-172-31-30-207:~/8266$ python s.py
Traceback (most recent call last):
  File "s.py", line 5, in <module>
    s.bind(('0.0.0.0',235))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 13] Permission denied
ubuntu@ip-172-31-30-207:~/8266$ sudo python s.py

```

ESP8266 设备上的程序启动之后,部署在云服务器上的服务端程序收到发送的数据。如下图 5-7 所示。

```
ubuntu@ip-172-31-30-207:~/8266$ sudo python s.py
Accept new connection from 223.89.150.242:38912...
<type 'str'>

1 0 1 1 23.9375 9 220 126 0 0.41366

Connection from 223.89.150.242:38912 closed.
Accept new connection from 223.89.150.242:38913...
<type 'str'>

2 0 1 1 23.25 9 220 126 0 0.415552

Connection from 223.89.150.242:38913 closed.
Accept new connection from 223.89.150.242:38914...
<type 'str'>

3 0 1 1 23.5 9 220 126 0 0.493984

Connection from 223.89.150.242:38914 closed.
Accept new connection from 223.89.150.242:38915...
<type 'str'>
```

图 5-7 广域网环境下的服务端程序测试

广域网环境下工作正常。

5.4 交互页面测试

将编写的简易交互界面部署于服务器的 80 端口,在浏览器输入服务器的 ip 地址或域名获取数据库数据,刷新后获取新的数据,如图 5-8 所示。

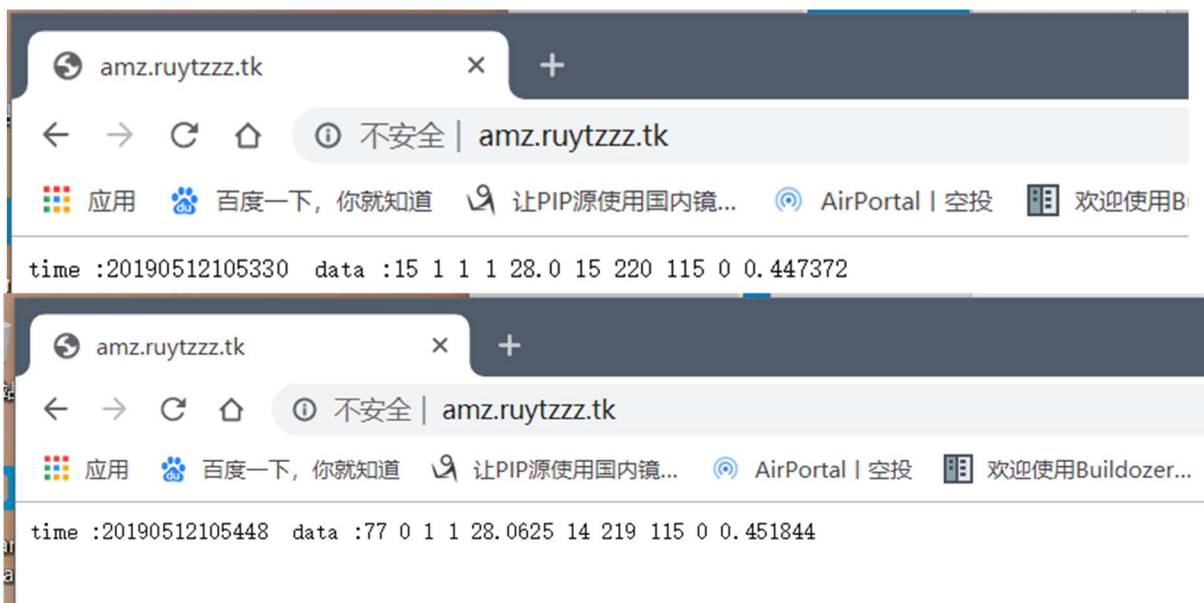


图 5-8 web 交互测试

这个页面还是太过于简单了，在这个页面上加入了一点 html 元素添加了标题，背景色和换行后以显示更加直观，具体的信息。如图 5-9 所示。



图 5-9 web 交互测试

6 总结

6.1 感想

此次的毕业设计任务，涉及到了许多方面的知识，比如单片机，数电模电，通信协议，数据结构，网络通信，环境部署与服务器维护等知识，此次任务，使我认识到了平时所学基础课程的重要性，使我明白了平时所学到的知识都将会在哪些方面发挥着什么样的作用，也使我认识到了要想完成一个项目，只有书本上的知识是远远不够的。有太多问题需要去书本外的地方去寻找答案，需要去搜集资料，去查找文献，需要去了解前沿技术和行业的发展方向。

6.2 遇到的问题与不足

此次设计中还存在着许多不足的地方，没有完善的错误处理。目前只在 SOCKET 通信程序部分做了简单的错误处理，系统不够健壮，可能会因为某个部分的意外对整个系统产生影响。部分程序没有采用模块化设计，可能使之后的维护升级工作变的复杂。

在此次任务中，遇到了许多问题。有时查找到的资料也不一定是正确的，需要自己判断和尝试不同的方案。

在进行 MicroPython 固件烧录的时候，根据官网上面的文档并不能正确烧录固件，在命令提示符中输入文档中的指令出现找不到程序的错误，在将程序名后缀改写为 .exe 后成功擦除和烧录固件，猜测可能是由运行环境的不同而引起的。

在进行广域网上的通信程序测试时，测试服务端程序是在百度云服务器上进行的。在云服务去放行相关端口的安全组后发现在建立 SOCKET 实例向服务端发起连接的时经过一小段时间的无响应后抛出 ECONNABORTED 错误。这个错误一出现在客户与服务完成了三次握手之后客户发送了一个 RST 分节（可能时由于 ESP8266 尝试连接超时后抛出 RST 分节），按照 POSIX 规定此时应抛出 ECONNABORTED 错误。在 Windows 与 linux 环境下构建套接字连接向百度云服务器上的程序均不能成功，抛出 Errno 10060。此错误一般出现在访问无法到达的情况下。但是在百度云服务器上通过对相应端口进行抓包发现有 tcp 的三次握手认证。似乎数据被拦截后丢弃了。猜测可能时因为国内环境下的云服务器与域名的备案问题导致的，没有经过备案被服务提供商屏蔽了连接。后来使用无需进行备案的亚马国际区的云服务器进行测试相同的程序工作良好。抓包测试如下图 6-1

所示。

```
rtt min/avg/max/mdev = 0.843/0.843/0.843/0.000 ms
root@instance-6bvf7527:~# tcpdump tcp port 235
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
17:33:05.605573 IP 223.89.150.108.50089 > instance-6bvf7527.235: Flags [S], seq 4282341156, win 64240, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0
17:33:08.605416 IP 223.89.150.108.50089 > instance-6bvf7527.235: Flags [S], seq 4282341156, win 64240, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0
17:33:14.643715 IP 223.89.150.108.50089 > instance-6bvf7527.235: Flags [S], seq 4282341156, win 64240, options [mss 1440,nop,wscale 8,nop,nop,sackOK], length 0
```

图 6-1 抓包检测

服务端监听到数据后会将数据打上时间标签存入数据库，时间是通过 python 的 `time.time()` 方法返回时间戳后使用 `time.localtime()` 的方法转换为本地的时间，再使用 `time.strftime()` 方法格式化为所需要的形式。但是测试时发现时间并不是北京时间。位于新加坡的这台亚马逊云服务器系统时区并不是新加坡时间，通过使用 `tzselect` 命令重新设置系统本地时间为北京时间使测试结果正常。

6.3 扩展与展望

在交互上面只使用简单的 web 页面交互是远远不够的，需要更多灵活的交互方式。但是由于 ESP8266 设备的限制，无法接入视频设备，也进行视频监控与图像处理。在以后的工作中可以更换平台添加视频设备的支持。也可以接入腾讯，阿里等平台使用多种途径更加灵活的交互，如短信，电话，app 推送和小程序等。或者借助 python 诸多开如 Face Recognition, ImageAI 等项目进行人脸识别，目标跟踪检测等高级应用。

参考文献

- [1]高亚敏. 安防行业中智能化技术的应用现状与前景研究[J]. 企业科技与发展, 2018(09):101-102.
- [2]王松德, 梁会琴, 张须欣, 朱小龙. 热释电红外传感器在无线遥控报警系统中的应用[J]. 安防科技, 2008(03):50-52.
- [3]宋潇. 清华同方企业物流管理系统设计与实现[D]. 北京工业大学, 2012.
- [4]张林. PCF8591 芯片与 MCS-51 单片机通信的探讨[J]. 电子制作, 2013(17):291.
- [5]陈军俊. 基于 Arduino 技术的六足机器人声呐测距功能设计与实现[J]. 电脑知识与技术, 2018, 14(14):174-177.
- [6]蔡业飞. 智能安防——给城市一个智慧的“大脑”[D]. 中国公共安全(综合版), 2012.
- [7]Sanner MF. Python: a programming language for software integration and development. [J]. Journal of Molecular Graphics & Modelling, 1999, 17(1):57-61.
- [8]Bell C. How to Program in MicroPython[M]// MicroPython for the Internet of Things. 2017.
- [9]冯立晖. 网络化多智能体系统的一致性分析[D]. 浙江理工大学, 2015.
- [10]林贤炼, 方遁. MicroPython 语言在物联网中的运用[J]. 闽江学院学报, 2017(2).
- [11]Kodali RK, Mahesh K S. Low cost ambient monitoring using ESP8266[C]// International Conference on Contemporary Computing & Informatics. IEEE, 2017.
- [12]Gay W. PCF8591 ADC[M]// Custom Raspberry Pi Interfaces. Apress, 2017.
- [13]周剑利, 郭建波, 崔涛. 具有 I2C 总线接口的 A/D 芯片 PCF8591 及其应用[J]. 微计算机信息, 2005(7).
- [14]陈涛. DS18B20 芯片与单片微控制器的接口设计与应用[J]. 山东煤炭科技, 2002(3):57-59.
- [15]甘勇, 宋春来, 宋寅卯. 数字温度传感器 DS18B20 在多点测温系统中的应用[J]. 河南农业大学学报, 2001, 35(4):391-393.
- [16]柴晓路. Web 服务架构与开放互操作技术[M]. 清华大学出版社, 2002.
- [17]SartajSahni, 萨尼, 汪诗林, et al. 数据结构、算法与应用[M]. 机械工业出版社, 2005.
- [18]Jianfeng H, Lin L, Wei Z, et al. The design of embedded SQLite3 and application of Commodity Rapid Positioning System based on Wi-Fi technology[J]. IEEE, 2011, 2:48 - 51.
- [19]张青, 田跃欣. 嵌入式 SQLite 的家居服务器设计[J]. 微计算机信息, 2009, 25(29):70-71.
- [20]Niharika Mehta, Shikhar Verma, Shivani Bhattacharjee. Automatic Garbage Collector Bot Using Arduino and GPS[M]. Springer Singapore:2018-05-16.
- [21]Xing Xiaojiao Wang Jianli Li. Services and Key Technologies of the Internet of Things[D]. ZTE Corporation, 2010.
- [22]张林. PCF8591 芯片与 MCS51 单片机通信的探讨[D]. 电子制作, 2013.

致 谢

本次的毕业设计任务持续了近三个月的时间，现在终于即将结束了。在此我要感谢老师们给予的细心指导和不懈支持，感谢所有为此文提供帮助的人们。

附 录

ESP8266 系统程序代码:

```
from machine import Pin, I2C
import onewire
import utime
import time
import network
import usocket

# 计数 红外 火焰 气体 温度 ad0 ad1 ad2 ad3 距离
def wlan():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('ruytzzz', '123456789')
        while not wlan.isconnected():
            pass
    print('network config:', wlan.ifconfig())
def send(a,b):
    try:
        s = usocket.socket(usocket.AF_INET,usocket.SOCK_STREAM)
        s.connect(("10.134.0.55",b))
        s.send(a)
        s.close()
    except:
        print('发送错误')
def juli():
    trig=Pin(12, Pin.OUT, value=0)
    echo=Pin(14, Pin.IN)
```

```
    trig.on()
    utime.sleep_us(20)
    trig.off()
    while echo.value()==0:
        pass
    a=utime.ticks_us()
    while echo.value()==1:
        pass
    b=utime.ticks_us()
    c=str((b-a)*344*0.5*0.000001)
    tcp[9]=c
    print(' 距离  :'+str(c)+'m')
def wendu():
    ow = onewire.OneWire(Pin(2))
    ow.reset()
    ow.writebyte(0xCC)
    ow.writebyte(0x44)
    ow.reset()
    ow.writebyte(0xCC)
    ow.writebyte(0xBE)
    d1=ow.readbyte()
    d2=ow.readbyte()
    a=(d1+d2*256)/16
    tcp[4]=a
    print(' 温度值:'+str(a))
def run():
    PCF8591 = 0x48 # I2C bus address
    PCF8591_ADC_CH0 = '\x00' # thermistor
    PCF8591_ADC_CH1 = '\x01' # photo-voltaic cell
    PCF8591_ADC_CH2 = '\x02'
```

```
PCF8591_ADC_CH3 = '\x03' # potentiometer
gpio_scl = Pin(13)
gpio_sda = Pin(4)
i2c = I2C(scl=gpio_scl, sda=gpio_sda, freq=100000)
i2c.writeto(PCF8591, PCF8591_ADC_CH0)
i2c.readfrom(PCF8591, 1)
data = i2c.readfrom(PCF8591, 1)
a=str(ord(chr(data[0])))
tcp[5]=a
print('adc 0: ' + a)
i2c.writeto(PCF8591, PCF8591_ADC_CH1)
i2c.readfrom(PCF8591, 1)
data = i2c.readfrom(PCF8591, 1)
a=str(ord(chr(data[0])))
tcp[6]=a
print('adc 1: ' + a)
i2c.writeto(PCF8591, PCF8591_ADC_CH2)
i2c.readfrom(PCF8591, 1)
data = i2c.readfrom(PCF8591, 1)
a=str(ord(chr(data[0])))
tcp[7]=a
print('adc 2: ' + a)
i2c.writeto(PCF8591, PCF8591_ADC_CH3)
i2c.readfrom(PCF8591, 1)
data = i2c.readfrom(PCF8591, 1)
a=str(ord(chr(data[0])))
tcp[8]=a
print('adc 3: ' + a)
n=0
while 1:
```

```
tcp=['',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']  
n=n+1  
tcp[0]=str(n)  
print('  ')  
print('  ')  
print('  ')  
print(' 循环计数'+str(n))  
hongwai=Pin(16, Pin.IN)  
print(' 红外值'+str(hongwai.value()))  
tcp[1]=hongwai.value()  
if hongwai.value():  
    print('          红外检测  ')  
huoyan=Pin(5, Pin.IN)  
print(' 火焰值'+str(huoyan.value()))  
tcp[2]=huoyan.value()  
if huoyan.value()!=1:  
    print('          火焰检测')  
qiti=Pin(0, Pin.IN)  
print(' 气体值'+str(qiti.value()))  
tcp[3]=qiti.value()  
if qiti.value()!=1:  
    print('          气体检测')  
wendu()  
run()  
juli()  
print(tcp)  
strtcp=''  
for i in tcp:  
    strtcp=strtcp+str(i)+' '  
print(' str :'+strtcp)
```

```
    send(strtcp, 235)
    time.sleep(1)
服务端程序:
import socket
import time, threading
import sqlite3
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('0.0.0.0', 235))
s.listen(10)
def tcplink(sock, addr):
    print('Accept new connection from %s:%s...' % addr)
    data = sock.recv(64)
    print(type(data))
    print('')
    print(data)
    print('')
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    a = time.strftime('%Y%m%d%H%M%S', time.localtime(time.time()))
    cursor.execute("INSERT INTO user VALUES (?,?)", (a, data))
    cursor.close()
    conn.commit()
    conn.close()
    sock.close()
    print('Connection from %s:%s closed.' % addr)
while True:
    sock, addr = s.accept()
    t = threading.Thread(target=tcplink, args=(sock, addr))
    t.start()
```

交互页面程序:

```
#coding=utf-8
#!/usr/bin/python
import sqlite3

def sql():
    conn = sqlite3.connect('test.db')
    c = conn.cursor()
    print "Opened database successfully";
    cursor = c.execute("SELECT * from user order by time desc LIMIT
1")
    data=cursor.fetchone()
    t=str(data[0])
    d=str(data[1])
    dlist=d.split()

    ret='<title>ESP866交互页面</title>'+<p style="background-
color:#888888">'+time :'+t+' data :'+d+'<br />'+循环:
'+dlist[0]+'<br />'+红外（触发时高电平）: '+dlist[1]+'<br />'+火焰
（触发时低电平）: '+dlist[2]+'<br />'+气体（触发时低电平）:
'+dlist[3]+'<br />'+温度: '+dlist[4]+'<br />'+ad0: '+dlist[5]+'<br
/>'+ad1: '+dlist[6]+'<br />'+ad2: '+dlist[7]+'<br />'+ad3:
'+dlist[8]+'<br />'+距离: '+dlist[9]+'</p>'
    print(ret)
    print "Operation done successfully";
    c.close()
    conn.close()
    return ret

import web
urls = (
    '/', 'index'
)
class index:
    def GET(self):
        web.header('Content-Type', 'text/html; charset=UTF-8')
        return sql()
if __name__ == "__main__":
    app = web.application(urls, globals())
    app.run()
```