



**本科毕业设计（论文）**

**基于W5500的远程数据采集监控系统**

学 院 **物理与光电工程学院**

专 业 **电子科学与技术**

年级班别 **15级1班**

学 号 **3215007630**

学生姓名 **于莹**

指导教师 **陈惠珊**

2019年 5 月

**摘 要**

本设计主要以STM32F103RCT6单片机作为主要控制芯片，使得DHT11温湿度传感器可以在远程环境中采集到现场的数据信息，并将数据信息通过单总线传输给STM32单片机系统进行处理。STM32单片机接收到数据信息后，通过SPI协议将该数据信息发送至W5500以太网控制芯片，该网络芯片通过RJ45连接器与路由器连接建立客户端，然后通过MQTT协议将数据信息上传至OneNET云平台。用户可以通过登录服务器来实时查看并监控远程环境数据。当数据超出设定阈值时，STM32单片机会开启报警系统提示用户，同时，OneNET云平台也会及时向用户发送提醒邮件。

该系统采用的是运用了嵌入式W5500全硬件TCP/IP协议栈的以太网芯片，实现了对远程环境的监控和查看，因此，这一系统可以适用于许多环境中，让环境恶劣并且难以手工采集与监控的工作现场可以通过芯片将设备和互联网之间进行连接。

**关键词**：单片机，W5500，物联网，MQTT协议，TCP/IP协议

**Abstract**

This design mainly uses STM32F103RCT6 single-chip microcomputer as the main control chip, which makes the DHT11 temperature and humidity sensor can collect the data information of the field in the remote environment, and transmit the data information to the STM32 single-chip system through a single bus for processing. After receiving the data information, the STM32 MCU sends the data information to the W5500 Ethernet control chip through the SPI protocol. The network chip connects to the router through the RJ45 connector to establish a client, and then uploads the data information to the OneNET cloud platform through the MQTT protocol. Users can view and monitor remote environment data in real time by logging into the server. When the data exceeds the set threshold, the STM32 MCU will open the alarm system to prompt the user, and the OneNET cloud platform will also send a reminder email to the user in time.

The system uses a built-in W5500 full hardware TCP / IP stack Ethernet chip to monitor and view the remote environment, so this system can be used in many environments, making the environment harsh and difficult to manually collect and monitor The work site can connect the device to the Internet via a chip.

**Key words**: single chip microcomputer, W5500, Internet of Things, MQTT protocol, TCP/IP protocol

**目 录**

[1.绪论](#_Toc24833_WPSOffice_Level1) [1](#_Toc24833_WPSOffice_Level1)

[1.1研究目的与意义](#_Toc2949_WPSOffice_Level2) [1](#_Toc2949_WPSOffice_Level2)

[1.2国内外发展现状 3](#_Toc14802_WPSOffice_Level2)

[1.3文章组织架构和主要内容 4](#_Toc28283_WPSOffice_Level2)

[2.课题相关技术和平台介绍 5](#_Toc2949_WPSOffice_Level1)

[2.1硬件TCP/IP协议栈应用技术 5](#_Toc29632_WPSOffice_Level2)

[2.1.1物理介质 5](#_Toc2949_WPSOffice_Level3)

[2.1.2链路层 6](#_Toc14802_WPSOffice_Level3)

[2.1.3网络层 7](#_Toc28283_WPSOffice_Level3)

[2.1.4传输层 9](#_Toc29632_WPSOffice_Level3)

[2.1.5应用层](#_Toc5216_WPSOffice_Level3) [10](#_Toc5216_WPSOffice_Level3)

[2.2 Onenet物联网云平台](#_Toc5216_WPSOffice_Level2) [10](#_Toc5216_WPSOffice_Level2)

[2.2.1 OneNET设备通信协议](#_Toc4776_WPSOffice_Level3) [10](#_Toc4776_WPSOffice_Level3)

[2.2.2接入流程](#_Toc25354_WPSOffice_Level3) [12](#_Toc25354_WPSOffice_Level3)

[2.3本章小结](#_Toc4776_WPSOffice_Level2) [12](#_Toc4776_WPSOffice_Level2)

[3.硬件电路](#_Toc14802_WPSOffice_Level1) [13](#_Toc14802_WPSOffice_Level1)

[3.1整体硬件电路总图](#_Toc25354_WPSOffice_Level2) [13](#_Toc25354_WPSOffice_Level2)

[3.1.1 STM32最小系统](#_Toc1602_WPSOffice_Level3) [13](#_Toc1602_WPSOffice_Level3)

[3.1.2以太网控制](#_Toc6415_WPSOffice_Level3) [13](#_Toc6415_WPSOffice_Level3)

[3.1.3传感器](#_Toc30955_WPSOffice_Level3) [14](#_Toc30955_WPSOffice_Level3)

[3.1.4蜂鸣器](#_Toc6274_WPSOffice_Level3) [14](#_Toc6274_WPSOffice_Level3)

[3.2各模块硬件电路设计](#_Toc3365_WPSOffice_Level2) [14](#_Toc3365_WPSOffice_Level2)

[3.2.1 STM32最小系统模块](#_Toc9320_WPSOffice_Level3) [14](#_Toc9320_WPSOffice_Level3)

[3.2.2以太网控制模块](#_Toc2085_WPSOffice_Level3) [15](#_Toc2085_WPSOffice_Level3)

[3.2.3传感器设计](#_Toc12370_WPSOffice_Level3) [17](#_Toc12370_WPSOffice_Level3)

[3.2.4蜂鸣器报警模块 1](#_Toc16528_WPSOffice_Level3)9

[3.3硬件测试 1](#_Toc1602_WPSOffice_Level2)9

[4.软件设计](#_Toc28283_WPSOffice_Level1) [23](#_Toc28283_WPSOffice_Level1)

[4.1主函数基本框架](#_Toc6415_WPSOffice_Level2) [23](#_Toc6415_WPSOffice_Level2)

[4.2 W5500以太网控制模块](#_Toc30955_WPSOffice_Level2) [23](#_Toc30955_WPSOffice_Level2)

[4.2.1初始化W5500](#_Toc11448_WPSOffice_Level3) [23](#_Toc11448_WPSOffice_Level3)

[4.2.2 初始化DHCP](#_Toc11270_WPSOffice_Level3) [24](#_Toc11270_WPSOffice_Level3)

[4.2.3 DNS域名解析](#_Toc21374_WPSOffice_Level3) [25](#_Toc21374_WPSOffice_Level3)

[4.2.4 TCP通信](#_Toc23261_WPSOffice_Level3) [25](#_Toc23261_WPSOffice_Level3)

[4.3传感器设计](#_Toc6274_WPSOffice_Level2) [26](#_Toc6274_WPSOffice_Level2)

[4.4本章小结](#_Toc9320_WPSOffice_Level2) [28](#_Toc9320_WPSOffice_Level2)

[总结 2](#_Toc29632_WPSOffice_Level1)9

[参考文献](#_Toc5216_WPSOffice_Level1) [31](#_Toc5216_WPSOffice_Level1)

[致谢](#_Toc4776_WPSOffice_Level1) [32](#_Toc4776_WPSOffice_Level1)

附录 A原理图 39

附录 B源程序 40

# 绪论

## 1.1研究目的与意义

本课题研究探讨的是一种基于硬件以太网接口芯片所构建的远程数据监控系统，并且通过利用微控制器连接到Internet。初步选用DHT11温度和湿度传感器，使得传感器数据上传至云服务器的功能得以实现。用户可以通过网络登录云服务器，随时随地查看数据变化。当数据超过设定的阈值时，MCU将自动处理相关情况。因而，该系统具备广泛的应用前景，如工业环境监测，农业蔬菜大棚监测，智能家居应用，智能车载监控等。还可以通过添加或替换传感器，实现远程和可视地监视各种数据信息。

物联网曾经是计算机、Internet和CMCC之后的第三次信息产业风暴，“智慧地球”的概念第一次被IBM公司提出之后，这一风暴也势必成为全球信息产业发展的必定趋向。与此同时，对于一个国家的Economic Development来说，物联网的存在更是一种及其强大的推动力，所以，各个国家纷纷制定出与之相关的物联网规划，并且都在积极促进其发展。在这股浪潮的推动下，各行各业纷纷开始应用这种信息技术设备，在实现万物联网的路途上，人们的生活质量也都得到了不同程度的提高，生产活动的管理也变得更加精细和便捷。

对于现阶段来说，物联网的运用已涵盖一切主要行业，物联网势必成为未来互联网行业的发展趋势。因而，物联网被普遍运用在人们的现实生活中，其中大到水质问题的安全监测，食品可追溯性，敌方侦察和情报收集；小到日常生活中的智能家居家电控制、老人病人康复护理、个人健康、农业大棚等多个领域都可以通过物联网来实现监测与控制[1]。

在这样的时代背景下，电子产品通过以太网接入互联网显得尤为重要。对于现阶段来说怎样能够在没有操作系统要求的情况下实现微控制器的Internet接入，目前行业内大概能够被分为两大类：一是使用传统的软件TCP/IP协定栈；二是最新的硬件TCP/IP协定栈。

现今行业内主要使用的是由MCU+MAC+PHY和网络接口构成的传统以太网接入模式，并在Master Chip中植入TCP/IP协议代码，以此来完成以太网的物理贯穿[2]和表层应用的通讯。软件TCP/IP协定栈的弊端主要有三个方面，首先，该方式须要主控MUC进行不停歇的响应中断，这就使得MCU的各种资源被大大的占用。如果开发者运用多个线程进行操作时，MCU的工作效率就会大大降低，即使是在使用单个线程操作的情况下MCU的运转速率以及数据的处置速度也仅仅能够满足须求，这一情况极其严重的影响到了通讯品质。第二，该方式所需要的代码量对于自身储存资源相对匮乏的单片机来说，是一种非常严重的负担，即使开发者选择LWIP协议这种代码量较小的协议，也会为Master Chip带来超出25-40KB的代码量负担。最后，如果以该方式将设备接入网络，一旦遭到恶意软件的攻击，单片机系统会有极大可能性进入瘫痪状态，这对于开发者的系统设备来说是极其致命的打击。所以当设备接入网络时，首先要考虑的就是网络安全问题。尽管现如今的网络技术依然在蓬勃发展，为了让网络通讯变得更加安全，诸多开发者们也都研究出了各式各样的全新的加密技术，然而还会出现许许多多的缺陷。

硬件协定栈芯片的处理方式是能够直接连接到网络接口的，主要通过MCU +硬件协议栈芯片的组合，单芯片组网能够被便捷地完成。硬件协议栈芯片能够解决TCP / IP协议的一切任务。目前行业内所使用的硬件协定栈是利用传统软件方式的硬件化的逻辑门电路来完成。因为在该方式中加入了硬件协议栈的使用，使得MCU基本可以完成全部的TCP / IP协议工作，因而不单大大降低了MCU的中断次数，使得单片机拥有更多资源来实现另外的功能，而且这一方式让系统处理协议的速度变得愈加敏捷与稳定。当开发者使用单线程操作时，这一方式的通讯速度能够比软件协议方式的加快十倍之多。随着线程增加，通讯速率被累积。另外，该方式所使用的芯片价钱与MAC + PHY的价格相差不多，在开发成本和性价比方面，对于开发者来说该方案也更加简单易用，开发过程也可以控制在较短的时间内实现。除此之外，官网例程库内容多样，主机程序也种类丰富，还可以做到测试过程收缩，后期维护也完全免费。在安全性方面，当有恶意软件想要对该系统进行攻击的时候，应用该方案的系统完全可以防御住这些恶意攻击，这得益于该方案使用了硬件的逻辑门来解决TCP/IP协定，完美的解决了网络协定安全性匮乏的弊端。基于这一决定性优势，硬件协定栈技术得以在物联网风暴中快速的发展，使人们在避免安全问题的同时享受现代技术的乐趣。

行业内现如今所使用的嵌入式以太网控制器主要以W5500芯片为核心，它不仅可以给单片机系统提供愈加稳固、快捷和安全的以太网接入方式，除此之外，该芯片内部还集成了MAC和PHY，可以进行主动协商、掉电模式和网络唤醒性能。总的来说，W5500芯片因为具有性能好，性价比高等优势，被广大开发者广泛应用，而它也正在急速攻克市场，赢得了行业用户的高度认可。

行业内所使用的远程数据采集监控同通常由监控中心、监控终端以及通讯网络形成。该系统大抵能够分为机械通讯和网络传输两种类型。随着物联网大浪潮的推动，计算机、Internet和自动化技术等诸多范畴的交流在通讯网络领域中也愈来愈广泛。现如今，作为用户的我们之所以可以经过计算机轻松访问到网络，能够在线查看各类信息，都得益于远程数据采集监控系统的通信模式，而这一方式也是基于以太网技术。以太网通讯模式已经成为远程数据采集系统发展过程当中最具有里程碑意义的存在。Internet技术一直在不断地提高并且在飞速发展的过程中，这时候，一种全新的信息平台出现了，那是一种基于Web的平台。Web服务器的优点之一是它让各种不同的平台软件能够在磁盘上进行相互通讯，因此，它成为了王伦讯息范畴中使用最为宽泛的交互式平台。监控设备能够获得自己独一无二的IP地址，这时，开发者或者管理者都可以通过浏览器来查看访问和管理监控该设备终端。诸多大型公司开始广泛的关注起这种基于Web服务器的新型系统，其中包括但不限于仪器行业和计算机行业。

## 1.2国内外发展现状

行业内所使用的传统数据采集监控系统采纳的是MCU+A/D的模式，并且以MCU作为核心。它的主要构成部分有MUC，传感器，可编程控制器等。这种由MUC控制的传统数据采集系统具备构成简略，灵敏便捷和稳定性强等长处。通常情况下，这种传统的采集系统往往支持一个独立的系统，虽然能够满足对现场环境的数据和存储等功能，但不可以进行远程的采集与监控。受这一现状的影响，开发者和管理者都迫切的需求更加先进的系统。

无论在国内还是在国外，远程监控这一课题都炙手可热，诸多研究者都开展了激烈的研究与讨论。1997年1月，Stanford和MIT共同赞助了第一个基于互联网的远程监测和诊断会议，本次集会主要探讨了远程监控系统的开放系统，诊疗信息协议，传输协议和用户法律限制，并期待未来的技术发展。与此同时，还对下一代基于互联网的远程监控和诊断演示系统进行了激烈的探讨，在最终决定由Stanford和MIT联合研发。这项工作得到了惠普，波音，英特尔和福特等12家大公司的大力支持与合作。

在国内，目前，哈工大，西安交大，华中大，南理工等诸多高校已经取得了非常先进的研究成果，例如XJTU研发的大型RMD，HUST研发的汽轮机KBGMD，HIT研发的MMMDES系统等。

现如今，伴随着物联网迅猛发展的大风暴和网络技术的快速展开，各种更加先进的数据采集监控系统已经越来越得到人们的认可，而它们的共同点都是基于Web服务器作为信息处理中心和Internet作为通信载体。我们现在可以通过在采集现场装置各种不同类型的传感器或者其信息获取设备来监测恶劣环境下的信息数据，这样，开发者和管理者都可以通过网页或者客户端来向远程的服务器发送请求，当监控中心收到来自客户端的申请时，会向监控终端发送一个申请数据包，这时候就能够通过控制现场设施来采集数据了。采集完成后，由于最后传输给服务器终端的信号必须是数字信号，在此之前监控终端会先把采集到的监控数据信息转化成数字信号。经过一系列的数据处理之后，服务器会将反馈信息提交给开发者或管理者。这种系统具备显著好处，它可以收集大量数据信息并拥有大量管制性能，具备良好的稳定性和集中管理性，在很大程度上亦能够完全完成真正意义上的无人监控。

目前，数据采集监测系统需要监测种类繁多的设备，复杂的模型，从而导致重叠的人力和物力的浪费，并有大量的保障后的工作是非常沉重的，所以数据采集和监控系统是实现统一的标准化必然趋势，让用户可以更方便地使用它，减少资源浪费。

## 1.3文章组织架构和主要内容

本文主要研究了一种能够采集现场温度湿度的远程数据采集监控系统，提出了具体的设计方案并研究了实现过程，全文一共有6个章节，具体概括如下：

第一章为绪论，主要介绍了课题的研究目的意义，国内外的发展现状以及未来发展趋势，并对本论文的结构和内容进行了设计。

第二章详细研究了构成本系统的关键技术及平台——硬件TCP/IP协议栈应用技术、Onenet物联网云平台等。

第三章为各模块的硬件电路分析。

第四章为各模块的软件编程设计与分析。

第五章为整体结论与展望。

最后是本文参考文献、致谢及附录。

# 课题相关技术和平台介绍

## 2.1硬件TCP/IP协议栈应用技术

构成网络通讯的核心框架就是TCP / IP协定栈，其中也包括了诸多网络协议的总和。该协议不单单对电子设施如何连接到Internet做了充分的定义，还规定了设备与Internet之间数据信息的传输方式。该协议具有四层构造，即应用层，传输层，网络层，以及链路层。每个层调用由下一层提供的方案，以满足其自身的需求[3]。因为我们大部分时间都在应层中工作而基本不会涉及到别的其他层，所以不必担心较低的层。最后不得不说的是，网络协定系统本身非常复杂，基本上是无法在短时间内就完全搞清楚TCP/IP的工作原理的，所以大概来说，TCP/IP的工作流程就是一个主机的数据经过一系列处理过程后被发送到另一方的主机。

### 2.1.1物理介质

连接设备与计算机时往往需要一些物理手段，这种物理手段就称之为物理介质。该介质分为三种形式，分别为FDDI、Twisted Pair和Airwave，这种介质不仅确认了电信号的传输模式，还确定电信号的传输带宽、速率、传输距离、抗干扰性等。TCP/IP协议栈分为四层，每层通过一个特定的协议与另一方通信，协议之间的通信最终转换为0和1的电信号，电信号可以通过物理介质传输到对方计算机，所以物理媒体是网络通信的基石。

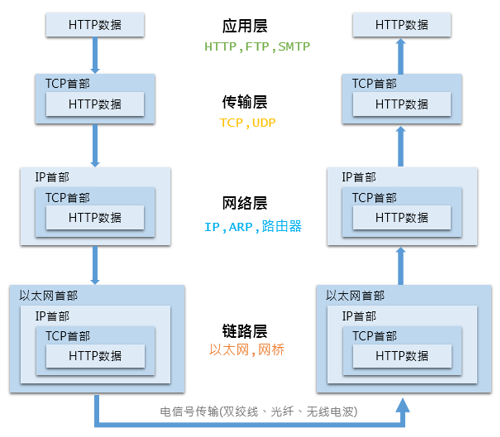


图 1 TCP/IP协议数据流示意图

当申请经过HTTP启动时，应用层，传输层，网络层，以及链路层的相关协议按此打包申请并执行相应的报头，最后生成的链路层的以太网分组，并且发送它通过物理介质的主机。其他主机接收数据分组后，将其解压缩它由层与相应的协议层，最后通过所述应用层数据以用于处理该应用程序。

完整的网络通信过程就像发送快递一样。信使外部的包裹层优于网络通信中的各种协议。协议包括产品信息，送货地址，发件人，收件人，联系方式等，还需要分发车辆，送货站和送货人员，最终产品可以到达客户。通常，快递员不能直接到达用户的手里，需要先将其转发到相应的发货站，然后发货站将发送件。物理介质是传送载体，网络管理是传送站，路由器是传送人员，IP地址是传送地址，MAC地址是联系信息。调度员负责将包裹转发到每个交付站，交付站根据收获地址和城区中的省和市确定是否有必要继续交付给其他交付站。当包裹成功到达目标递送站时，递送站然后找到接收者并根据联系信息发送一个片段。

### 2.1.2链路层

网络通信是指具有特定含义的数据通过物理介质传输给对方，但是只发送0和1是没有意义的。为了传输有意义的数据，必须按byte对0和1进行分组，并识别每组电信号的信息特征，然后按分组的顺序传输。以太网规定一组电信号是一个数据包，一个数据包称为帧，以太网协议拟定了这一标准规则。

一个完整的数据帧由三部分构成，分别为首部、数据和尾部。数据部分至少有46 byte长，最多1500 byte。如果开发者想要发送的数据是非常长，则必须被划分为多个很短的帧来用于传输。尾部部分固定为4个byte，除了能够指示数据查看序列，还可以判断这个数据包有没有在传输的过程当中损坏。因而，以太网协定组的数据信号，并构成一个数据帧，并随后发送经过物理介质中的数据帧到接收器。

根据以太网协议，连接到网络的设施必需安装网卡，也就是网络适配器。数据包必须在两个网卡之间相互传输。数据包的发送地址和接收地址统称为网卡地址，即帧首部中包含的MAC地址。MAC地址是每个网卡的标记，就好比一卡通上的卡号一样，是全球唯一的。传统的MAC地址采纳十六进制表示法，共6个byte，其中前3个byte为供应商编号，后3个byte为NIC序列号，例如，5b-0F-7E-14-d3-15。当成功获取到MAC地址至后，以太网会采用播送的方式将收到的数据包传递到子网内的一切主机。接收到数据分组后，子网中的每个主机都会读取在报头中的目的MAC地址，而后进行比拟，如果结果相同的，则下一步的处理；如果结果不同，则把该数据包抛弃。

综上所述，链路层拥有两个主要任务分别为，首先要能够将电信号进行分组，然后构成具备特殊含义的数据帧，最后还要能够把电信号以广播的方式经过物理介质发送给接收机。

### 2.1.3网络层

#### IP协议

MAC地址是唯一的和制造商有关的关系网络。因此，无法判断两台主机是否通过MAC地址属于同一子网。所以，网络层引入的IP协议和开发了新的组地址，以便它可以区分这两个主机是否属于同一网络。这套新的地址集就是我们所说的网络地址，也称之为IP地址[4]。

现阶段行业内使用的IP地址分为两个版本，分别是IPv4和IPv6。通常情况下IPv4由4个十进制数字表示，是一个拥有32 Bit的地址。IP协议规定32 Bit的地址都由两部分构成，其中，前部体现网络地址，后一部分体现局域网中主机的地址。由于不同类型地址的划分不同，以192.168.1.199为例，前24位是网络地址，后8位是主机地址。因而，假定两个IP地址的网络地址完全相同，那么它们必定在同一个子网中。IP协议中，还加入了子网掩码的概念来确认IP地址中的网络地址，其中，子网掩码和IP地址能够经过按位AND运算得到。

如果我们想要判断双方是不是都涵盖在相同的子网当中，我们只要经过子网掩码最两个IP地址进行AND操作，这是因为发送器和接收器的IP地址时经过上文所述应用曾协议来传输的。

#### APR协议

我们通常所说的地址解析协议也就是APR协议，该协议能够按照IP地址来获取MAC地址。首先，ARP会启动一个申请数据包，这个数据包的头部会包含目的主机的IP地址。然后，在链路层生成一个以太网数据包，并对其进行重新打包。最后，会由以太网将重新打包好的数据包经过播送的方式传递到子网。上述过程成成功完成后，所有主机都能够收到这个包，并而后在头部中取出IP地址，最后将其与本身拥有的IP地址进行比对。若比对结果相同，则将该IP地址返回到本身的MAC地址中。若比对结果不同，则抛弃这个数据包。在接收到返回的讯息之后，APR就可以确认目的设施的MAC地址了。同时，为了方便下次请求的时候能够直接查询，APR会把返回的MAC地址与对应的IP地址都暂时储存在本APR的缓存当中。

#### 路由器协议

由于ARP的MAC地址寻址过程依然是在一个相同的子网当中进行的，路由器协议被网络层加入了进来。首先，IP协议被用于确认两个主机是不是在一个相同的子网当中，若相同，则对应的MAC地址将会被相应的ARP协议查询出来，而后用广播的方法传送给子网上的主机；若不同，则该子网的网关会对这个数据包进行路由。网关会把处理好的数据包传输到目的IP所在的子网当中，而后经过ARP获取到目的机的MAC地址，最后进行广播[5]，在进行这一系列操作之前，作为Internet上子网与子网之间的连接桥梁，网关还会对数据包进行屡次转发。

#### IP数据包

所谓IP数据包，指的就是在网络层包装好的。IP数据包有两个部分构成，分别为报头和数据。报头是由20 byte构成，主要包含目的IP地址以及源IP地址，而网关路由的线索和基础都是目的IP地址，它的数据部分的最大长度为65515 byte。

综上所述，网络层的主要任务分为三个，首先需要路由不同子网的数据包，其次还要能够辨别子网中的网段和MAC地址，最后需要定义网络地址。

### 2.1.4传输层

MAC地址也就是主机的身份被链路层成功定制之后，网络层紧接着廓清了主机所在的网段，也就是成功定义了IP地址。成功完成这两个地址的定义之后，两个主机之间就可以进行数据包的传输了。不过现实情况却是，数据包是在两个主机上的应用程序中相互传输的。更糟糕的情况是，实际上每台计算机同时都有可能在运转着数个应用程序，所以当数据包到达主机之后，还是无法准确判断到底应该用哪一个应用程序接受这个数据包。

因而，为了处理这个问题UDP协议就出现了，它是被传输层引入的[6]。UDP协议为每一个应用程序都定义了独一无二的端口，以便于数据包来辨认各个应用程序的身份。网络数据包在进行传输之前都必需先对端口信息进行标记，然后再根据主机上的特定端口号来判断目的应用程序。有了这个过程，当数据包抵达主机时就可以按照对应的端口号发现对应的程序了。

当然，UDP协议除了相对简略并且易于完成的优点之外，还有一个很大的缺点，那就是没有确认机制，可靠性差。对方是不是已经收到这个数据包了呢？UDP无从得知。那么如何可以提高网络的可靠性呢？TCP协议应运而生。我们常说的传输控制协议也就是TCP协议，这种通信协议具有很多优质的性能，例如可靠性强等。通俗来说，TCP协议就是加强版的UDP协议。

有了这一加强功能，客户端再发送数据奥之前必需先和服务端建立牢靠的连接，而TCP协议正是再UDP协议的根基之上由确保了一次可靠性，那就是著名的三次握手过程。这一过程可以简单描述为：

主机A：我准备好给你发数据了，ok吗？

主机B：ok！什么时候发？

主机A：马上就发，接住！

三次握手全部成功之后，主机A才可以向主机B发送正式的数据包。但是UDP是没有这一确认过程的，因而，TCP能够保证数据包在传输过程中不会丢失，但好的东西必须付出代价，与UDP相比，TCP实现更复杂，连接资源消耗大，传输速度慢。

TCP数据包与UDP数据包惟一的区别是，TCP包没有长度方面的限制，然而，为了保障网络效率，TCP包正常情况下是不会超过IP数据包长度的，这样就避免了被拆分的情况。

综上所述，传输层的主要任务有三个，首先需要是定义端口，其次能够辨别应用程序标识，最后可以完成端口之间的通讯。

### 2.1.5应用层

理论上，通过上述三个层协议的操作过程，完全可以将数据从一个主机应用程序转移到另一个上，但此时传递的数据是一个byte流，而程序却无法认可其，可操作性差,因此，应用层定义不同的协议进行标准化的数据格式。



图 2 HTTP报文格式规范

有了这个规范，在接收到结果后，客户端将跟踪服务器，解析成服务端返回的格式。

综上所述，应用层的主要工作是定义数据格式，并根据相应的格式解释数据。

## 2.2 Onenet物联网云平台

### 2.2.1 OneNET设备通信协议

物联网的网络传输通常只有在第三方网络运营商的支持下才可以进行，目前，许多运营商都会为用户提供免费的物联网云服务，例如：亚马逊云平台、机智云、云智易和NLECloud等。

OneNET是一款中国移动开发的，支持第三方软件接入的专业物联网开放云平台。OneNET平台一经发布就收到了开发者们的广泛应用，通过把开发者们开发测试过的智能设备快速的接入物联网系统，OneNET能够为开发者提供有好的交互界面来显示监控数据，包括图表、文字等形式，能够直观的展现在控制界面，开发者可以非常方便的实时监测远程环境并加以控制。同时，OneNET还提供了诸多种类的瓦昂洛同通信协议和开发工具，使得无论开发者使用什么样的设备终端都可以方便快捷的接入网络，并进行远程数据传输和数据管理。

为了方便开发者使用，OneNET提供了7种常用的网络通信协议[7]，开发者可盈掬自身设备的应用要求来选择合适的同通信协议进行开发，其中包括：

1. HTTP协议：用来上报传稿器数据点
2. EDP协议：用于报告传感器数据点；接收实时指令或离线命令
3. MQTT协议：用来上报传感器数据点；订阅自定义TOPIC；接收实时消息或离线消息
4. Mod-bus协议：用来自定义采集命令与采集周期；接收实时命令或离线命令
5. JT/T808协议：用来上报传感器数据点；接收实时命令或离线命令
6. TCP透传协议：TCP透明传输协议：用于数据传输;无需开发的设备;命令发出
7. RTMP协议：用来RTMP实时流媒体推送；支持多协议（rtmp、hls）流媒体分发；本地视频远程查看

MQTT是消息队列遥测传输协议客户端或服务器来发布或订阅消息（Publish/Subscribe）[8]。MQTT协议是开放的，操作简单，且重量轻。开发人员可以实现具有的代码和有限的带宽少量设备的远程控制。它是沟通的一种非常有效的手段。同时，MQTT还具有开销低、带宽窄的即时通讯优势，这样就能够在远程控制、移动终端和物联网等场合得到广泛的应用。MQTT协议是在TCP/IP协议或者其他网络通讯协议中运行，可以提供有序、无损、并且基于byte流的双向传输。MQTT网络在进行用户数据传输时，会将相关的服务质量（QoS）和主题名（Topic）进行关联。

### 2.2.2接入流程

OneNET云平台支持三种不同的通信方式，分别为以太网通信、Wi-Fi通信和GPRS通信。以太网通信可以通过网线接入，Wi-Fi通信可以通过无线接入，GPRS通信可以通过SIM通信接入。由于本设计采用物联网解决方案和以太网技术，所以采用以太网通信方式进行数据传输。

云平台与终端进行介入的过程相对来说比较容易，用户首先在OneNET云平台上注册个人或者企业账号并创建自己专属的“开发者中心”，经过账号建立属于本人的项目并为设施请求ID。OneNET云平台会自动根据用户的申请为相应的智能终端硬件分配虚拟的ID和APIkey。完成虚拟ID和APIkey的建立后，OneNET云平台会通过相应的协议与终端智能硬件建立TCP连接，完成之后就可以实现数据的采集和控制信号的发出。

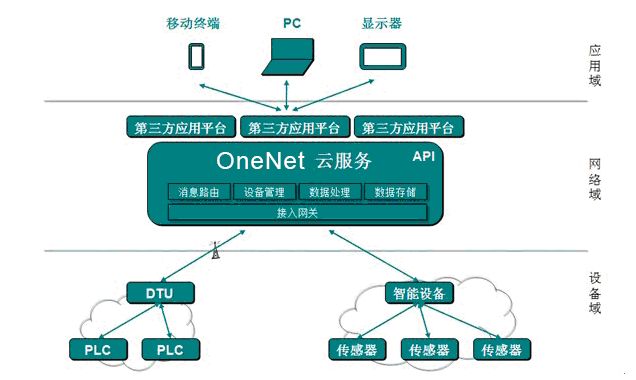


图 3 OneNET云平台接入流程

## 2.3本章小结

本章详细的研究了本次设计所涉及到的各种协议、技术以及平台。首先，详细分析了全硬件TCP/IP协议栈的主要构成，具体讲述了每一层的功能目的以及在本次设计中的使用情况，介绍了与软件TCP/IP协议栈相比硬件TCP/IP协议栈的优点，并且详细分析了在建立网络过程中的三次握手经过。最后，介绍了相关网络云平台的基本信息以及接入设备的简要流程。

# 硬件电路

## 3.1整体硬件电路总图

W5500以太网模块

DHT11传感器模块

蜂鸣器报警模块

STM32最小系统

图 4 整体硬件框图

### 3.1.1 STM32最小系统

本文设计的远程数据采集系统采用的是STM32F103RCT6单片机，由于设计功能没有过于复杂，本设计使用了该单片机的最小系统来完成所需的功能。当相应的数据信息被远程环境中的传感器采集到后，传输给单片机模块，单片机会对传感器采集到的数据进行接纳和处置。处理成功后，单片机模块会将处理好的数据信息传输至以太网模块，并通过以太网模块上的控制芯片将数据信息传输至服务器。除此之外，该单片机模块还承担着数据信息监测与控制功能。当从传感器传输过来的数据信息超出阈值范围，单片机模将控制报警模块进行报警。

### 3.1.2以太网

本模块是由W5500控制芯片及其外围电路和RJ45连接器组成，其中W5500控制芯片负责将从单片机传输过来的数据信息上传至服务器，已达到物联网的功能。RJ45连接器则是整个数据采集监控系统与互联网进行联通的接口，通过物理介质——网线，来连接智能终端与互联网。W5500控制芯片通过SPI总线与单片机模块进行连接，RJ45连接器通过网线与路由器进行连接。该模块是整个远程数据采集监控系统的关键模块之一，有了这个网络控制模块，才能实现设备联网，从而可以实时的远程查看以及监测数据。

### 3.1.3传感器

本模块是由DHT11温湿度传感器及其核心电路构成。在远程环境中，DHT11传感器连续地采集环境的温度和湿度，并发送所收集的数据信息到MCU模块，然后执行一系列由MCU操作控制的。DHT11温湿度传感器的数据传输端与单片机模块的一个IO口进行连接，以达到数据传输的功能。

### 3.1.4蜂鸣器

本模块是由一个有源蜂鸣器及其外围电路和一个电源指示灯组成。通过单片机模块IO口的高低电平来控制蜂鸣器的开关，当单片机IO口输出高电平时，蜂鸣器关闭；当单片机IO口输出低电平时，蜂鸣器进行报警。

## 3.2各模块硬件电路设计

### 3.2.1 STM32最小系统模块

本设计采纳的STM32单片机芯片为STM32F103RCT6，该芯片的基本资源包括有：256K的Flash、内部SRAM 48K、SPI串口3个、5个IO口等。STM32F103RCT6芯片内核采用的是32位的高性能ARM Cortex-M3处理器[9]，始终可以高达72M，不过在实际中也可以超频至80M，但不建议这么做。同时，该芯片还支持单周期的乘法以及硬件除法。该芯片一共拥有64个引脚，在51个IO口中，除了模拟通道不耐5V，其他所有IO口都耐5V。电源支持2.0-3.6V，通常情况选择3.3V电源，此时IO口输出高电平3.3V，低电0V。

同时，该芯片还具有强大的时钟系统，首先支持外部接4-16M的外部高速晶振，通常情况下使用8M；内部含有8MHz的高速RC振荡器，当外部晶振失效后可自动切换至内部晶振以保证系统正常运行；内部含有低速RC振荡器，可以为看门狗提供时钟；外部可以外接低速晶振做RTC时钟源。STM32F103RCT6还具备三种低功耗形式，分别为寝息、停止和待机，也可以用后备电池给备份寄存器以及RTC供电。

STM32F103RCT6还具有3个12位AD，最多可达32个外部测量通道，转换范围为0-3.6V，内部通道可以用于内部温度测量，但该芯片只具有2个12位DA。

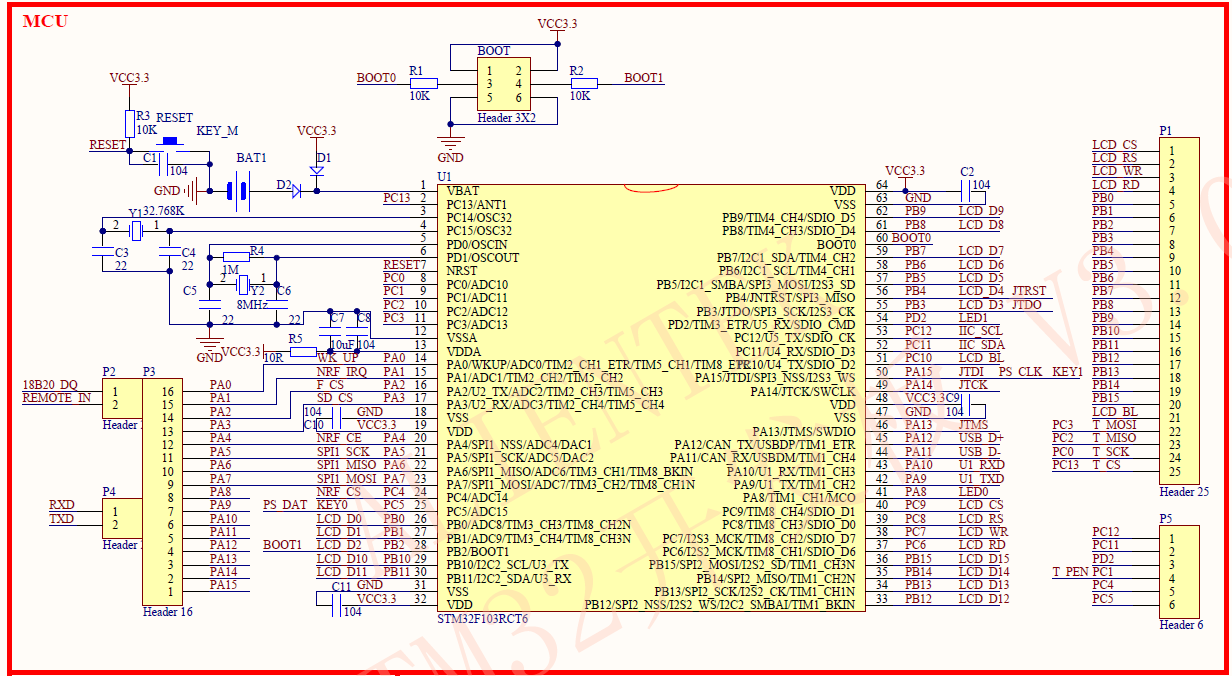


图 5 STM32F103RCT6最小系统原理图

### 3.2.2以太网控制模块

W5500以太网模块是一款基于WIZnet W5500芯片的以太网控制模块。该模块具备出色的功能和超高的性价比，能够满足开发者的基本功能需求。与此同时，该模块还集成了硬件TCP / IP协议，内部TX / RX缓存由32 Kbyte的存储器构成。除了支持10/100 Mbps网络的传输速率，该模块还支持同时运行的8个独立的端口。此外，该模块支持3.3V或5V电源。为了便于在不同的微控制器系统中使用的开发，当模块被5V供电，它也可以输出3.3V。该模块通过一个简单方便的SPI总线与所述微控制器系统进行通信。

W5500以太网模块具有10个排针，具体功能如下：

**表 1 W5500模块排针功能说明**

|  |  |  |  |
| --- | --- | --- | --- |
| 排针标识 | 功能说明 | 排针标识 | 功能说明 |
| 3.3V | 3.3V电源输入引脚 | 5V | 5V电源输入引脚 |
| MISO | SPI主机输入从机输出引脚 | GND | 电源地引脚 |
| MOSI | SPI主机输出从机输入引脚 | RST | W5500硬件初始化引脚（低电平有效） |
| SCS | SPI SLAVE选择引脚（低电平有效） | INT | W5500中断引脚（低电平有效） |
| SCLK | SPI时钟引脚 | NC | 悬空 |

W5500模块具备两种供电模式，也就是3.3V供电或者5V供电。

以太网控制模块与STM32开发板的接线方式如下：

PC5->W5500\_RST

PC4->W5500\_INT

PA4->W5500\_SCS

PA5->W5500\_SCK

PA6->W5500\_MISO

PA7->W5500\_MOSI

网线一端接RJ45连接器，一端接路由器，当然也可以直接接电脑。

W5500芯片作为一款嵌入式以太网控制器，之所以能够使得嵌入式系统可以通过SPI（串行外设接口）接口方便快捷的连接到互联网，都得益于这其中集成的全硬件TCP/IP协议栈。

W5500芯片是一款嵌入式以太网控制器，其中集成了TCP/IP协议栈 。而这个全硬件TCP/IP协议栈能够做到全程支持TCP、ICMP、 PPPoE、ARP、UDP、IGMP 和 IPv4协议，在连续多年的实际使用过程中也得到了很好的印证。有了W5500，开发者不必再处理复杂而庞大的以太网控制器，只需要通过简单的socket程序便可以实现以太网的应用。

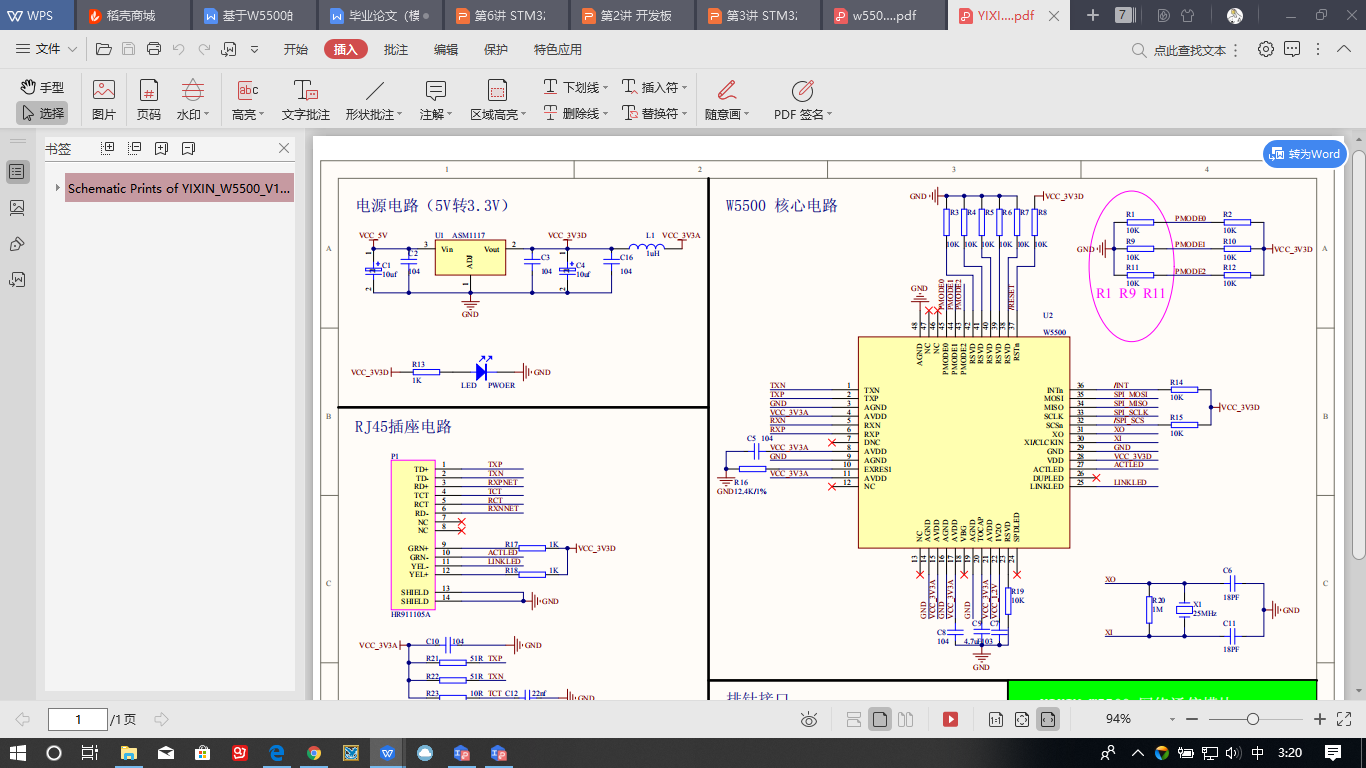


图 6 W5500原理图

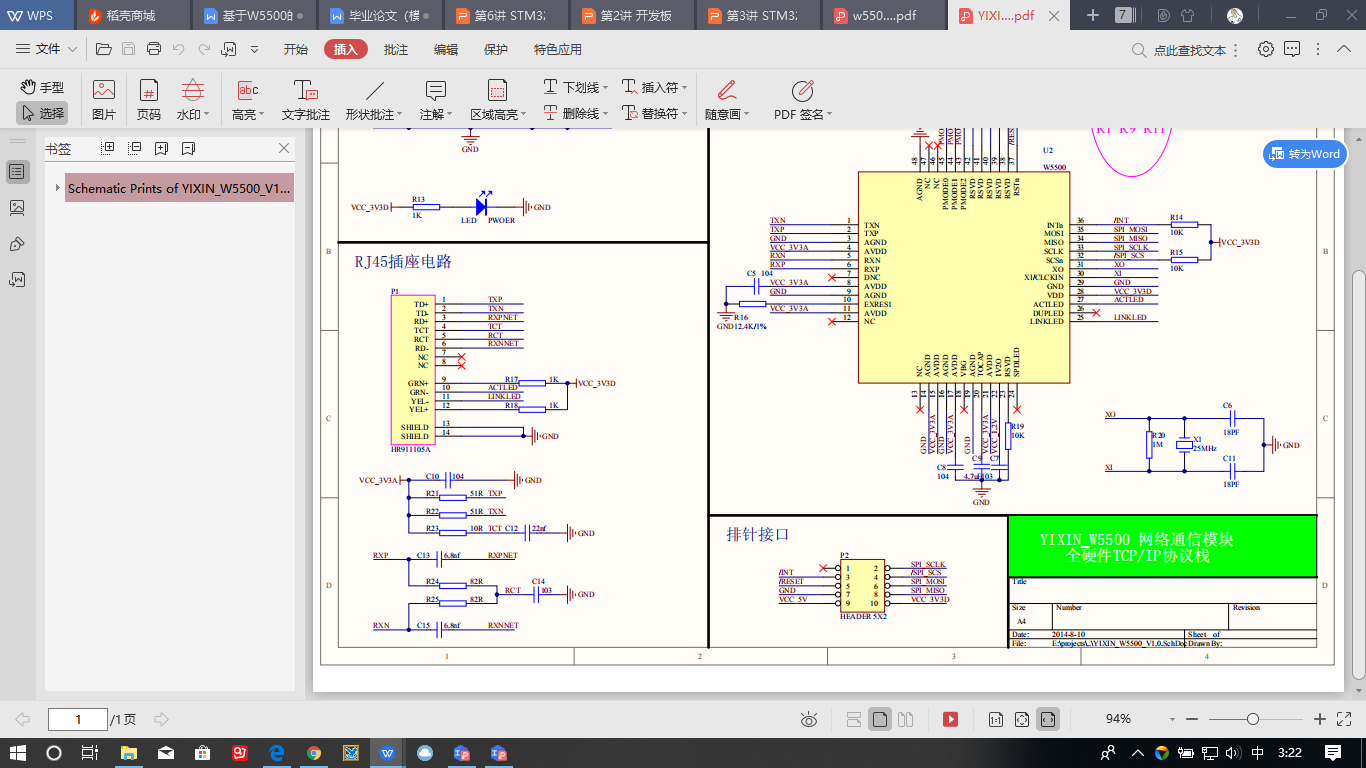


图7 RJ45连接器原理图

### 3.2.3传感器模块

传感器模块是由DHT11温湿度传感器、电源指示灯及其外围电路组成。其中，DHT11的工作电压范围在3.3V-5V，对于3.3V或者5V的单片机系统都可以驱动。工作电流很低，平均0.5mA，采取的是单总线数字信号输出的形式。它的测量范围是温度0-50℃，湿度20-90%RH，实际上DHT11大部分时间多用于测量湿度。湿度的精确度为±5％，这是因为湿度是难以准确判断的范围内，所以它是在±2℃的温度的近似测量。分辨率湿度1％，温度1℃。

所述DHT11数字湿温度传感器之所以能够使用单个数据管脚端口输入和输出的双向传输，是因为它应用了单一总线的数据格式。DHT11的数据分组由5Byte（40位），其中，所述数据被划分成一个小数部分和整数部分，和完整的数据传输是40位，并且高阶先出。DHT11的数据格式是：8位整数湿度数据+ 8位湿度十进制数据+ 8位整数温度数据+ 8位温度十进制数据+ 8位校验和。例如，8位湿度的整数是52，和8位湿度的小数是6。那么它的湿度就是52.6；8bit温度的整数是28，8bit温度的小数是9，温度就是28.9。在这之中，校验以及数据为前四个byte相加。未编码的二进制数据被传感器输出。数据与数据之间。无论是湿度、温度、整数还是小数，都需要分开来处置。例如，某次从DHT11读到的数据如图所示：

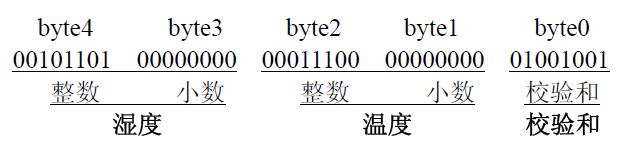


图 8 DHT11读取数据

由以上数据就可得到湿度和温度的值，计算方法：

湿度= byte4 . byte3=45.0 (％RH)

温度= byte2 . byte1=28.0 ( ℃)

校验= byte4+ byte3+ byte2+ byte1=73(校验正确)

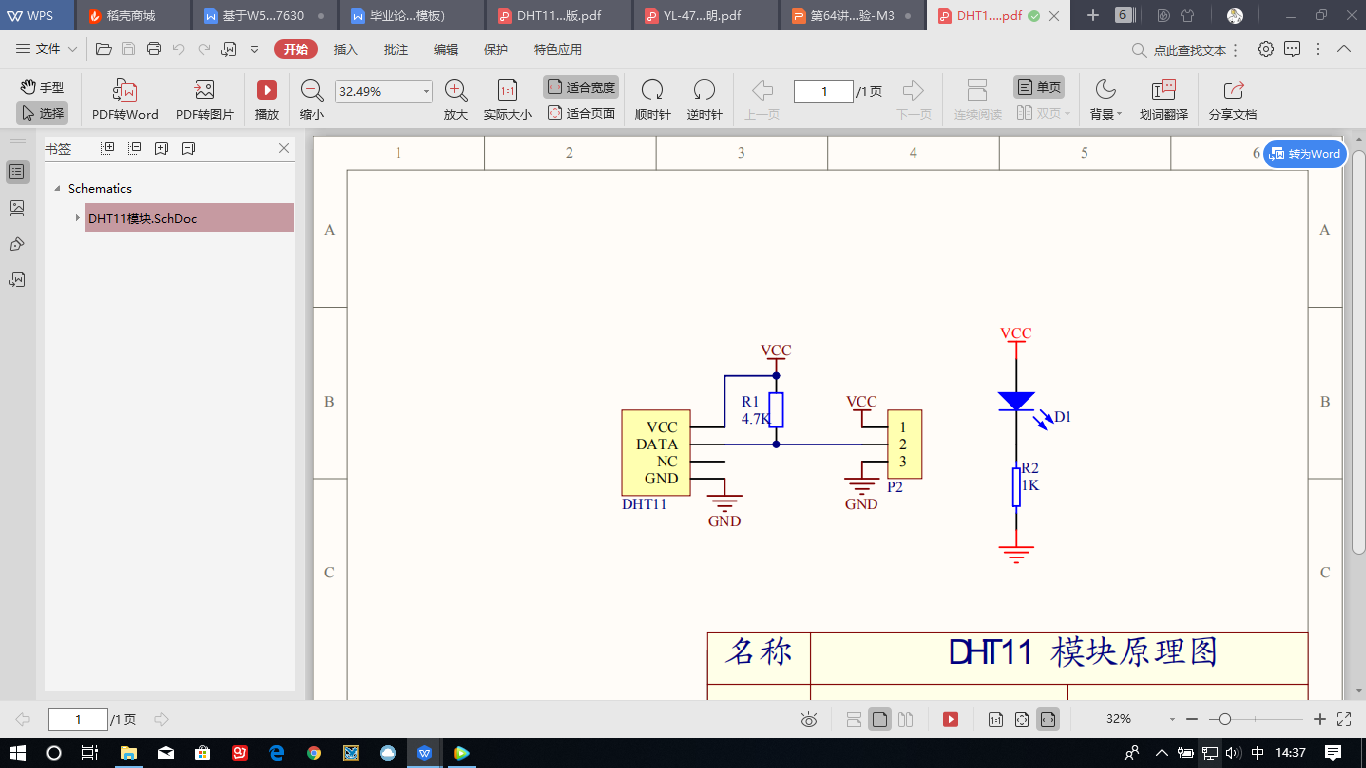


图 9 DHT11原理图

### 3.2.4蜂鸣器报警模块

本模块采用了一个低电平触发的有源蜂鸣器及其外围电路组成，与单片机之间通过一个IO口连接，当单片机收到的来自传感器的数据信息超出阈值范围时，会向该报警模块发送一个低电平信号，蜂鸣器模块接收到这个低电平信号会被触发，发出报警。介绍了各模块的具体功能作用，以及模块之间如何连接并进行数据传输。介绍了主要相关原件的基本信息以及引脚情况。

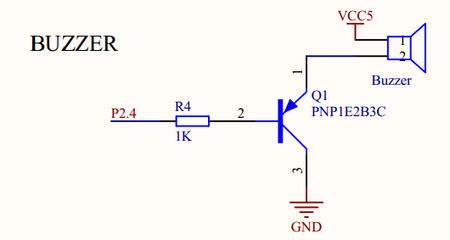


图 10 蜂鸣器原理图

## 3.3硬件测试

1. 测试STM32单片机系统是否可以正常工作。本次设计所有成勋代码都是通过串口烧录到单片机中运行，连接电脑与单片机串口USB，按下电源按键，电源指示灯亮，说明上电成功。然后，通过串口工具将建议跑马灯程序带烧录到系统当中并运行，看到左侧LED分别亮起并不停闪烁，测试成功，单片机系统可以正常工作。

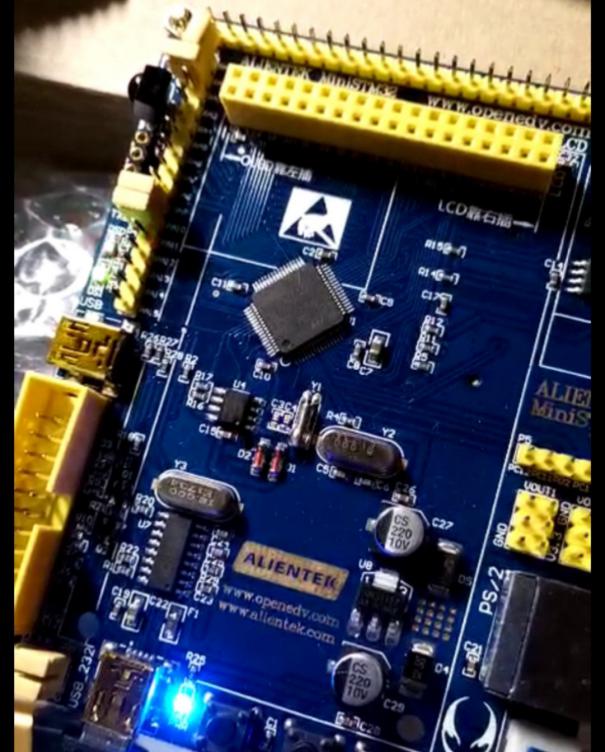


图 11 跑马灯测试结果a

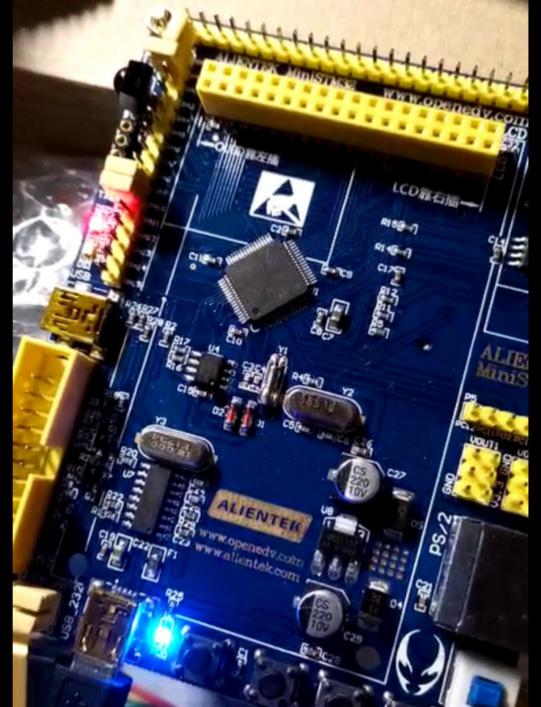


图 12 跑马灯测试结果b

1. 测试W5500模块是否可以正常工作。首先按照数据手册将单片机与W5500各个引脚连接，再通过RJ45连接器连接路由器和以太网模块，板子上电，看到W5500模块上电源指示灯亮起，绿色网络连接指示灯开始闪烁。然后对Load\_Net\_Parameters()函数进行修改，修改成功后将代码烧录进单片机，打开测试工具，客户端测试成功（服务端测试同理）。



图 13 函数修改过程

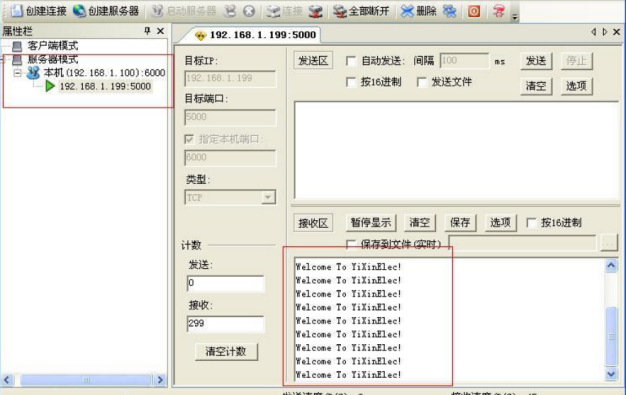


图 14 测试工具测试结果

1. 测试传感器模块是否能够正常工作。首先根据数据手册将传感器与单片机引脚相连，开发板上电，电源指示灯亮起。将简易传感器例程烧录至开发板，开发板能够接受到数据信息并显示在显示屏上，传感器测试成功。



图 15 传感器测试a

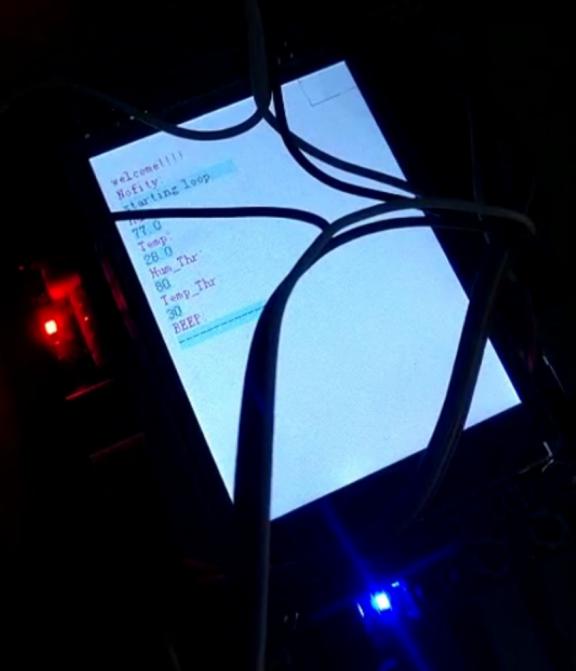


图 16 传感器显示测试b

1. 测试蜂鸣器是否可以正常工作。首先根据数据手册连接单片机引脚与蜂鸣器，下载一个简易蜂鸣器程序代码到单片机中并运行，蜂鸣器开始每隔300ms报警一次，测试成功。

# 软件设计

## 4.1主函数基本框架

首先，主函数会对W5500以太网控制模块、DHT11传感器模块和蜂鸣器报警模块等各个电路进行初始化；然后，在主循环中获取传感器传输过来的数据信息并进行判断，当数据信息在设定阈值范围内时，断开蜂鸣器，当数据信息在超出设定阈值时，接通蜂鸣器；最后，每次获取传感器数据信息后都会将数据上传至云服务器。

主程序流程图如下：

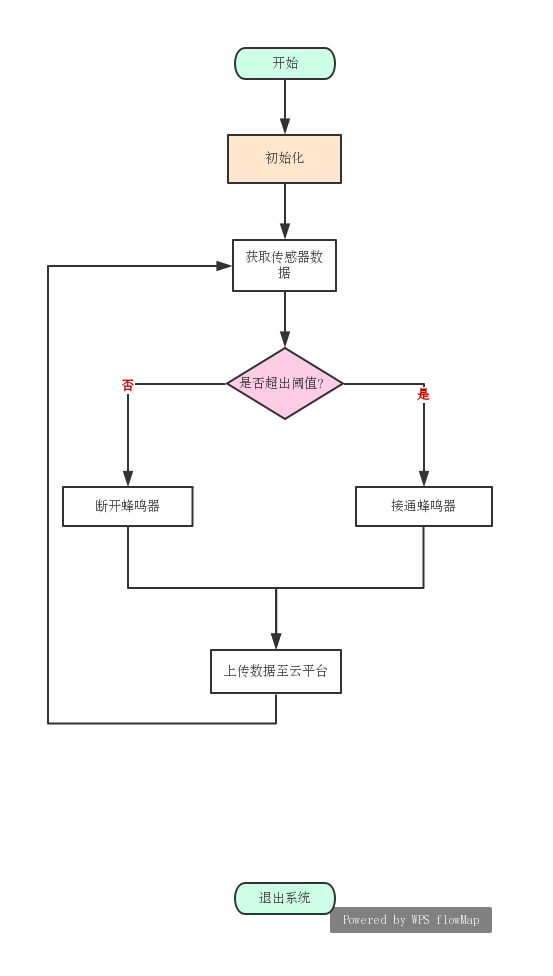


图 17 主函数流程图

## 4.2W5500以太网控制模块

### 4.2.1初始化W5500

首先初始化Systick工作时钟

第二初始化串口通信

第三初始化eeprom

第四初始化W5500

第五初始化MCU相关引脚

第六复位W5500

最后初始化8个socket的发送接收缓存大小

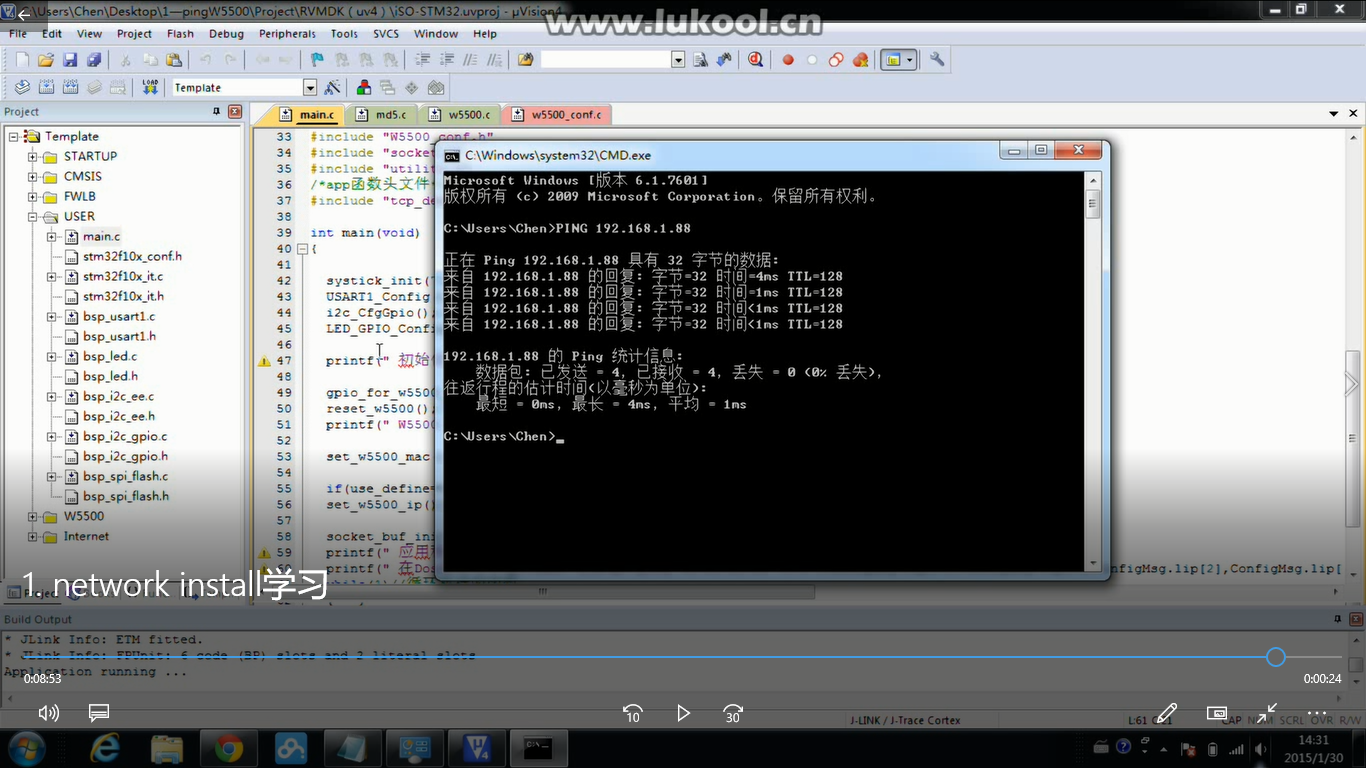


图 18 pin通测试结果

### 4.2.2 初始化DHCP

如果本地没有设定IP地址，那么DHCP在第一次登陆网络的时候，客户端就会发现，这时候就会向网络发出一个广播包，而这个广播包只有DHCP服务器会响应。DHCP服务器会提供IP地址。在接收到客户端的广播分组后，服务器会从IP地址池的IP地址中选择一个还未分配的IP地址，并将其分配给客户端。其次，在选择阶段，选择阶段实际上是在DHCP客户端选择IP地址的，也就是说，如果有多个DHCP服务器，DHCP报文发送给客户端，客户端随机选择他们中的一个，而后进行广播的形式响应DHCP服务器，并正式请求DHCP服务器分配一个地址。最后，正式请求之后DHCP服务器就确认了IP地址。

首先初始化DHCP客户端

然后获取DHCP服务器状态

其中switch状态机模型范围内为四种状态，分别为：IP地址获取不成功；IP地址获取超时；成功获取到IP地址；IP地址获取冲突

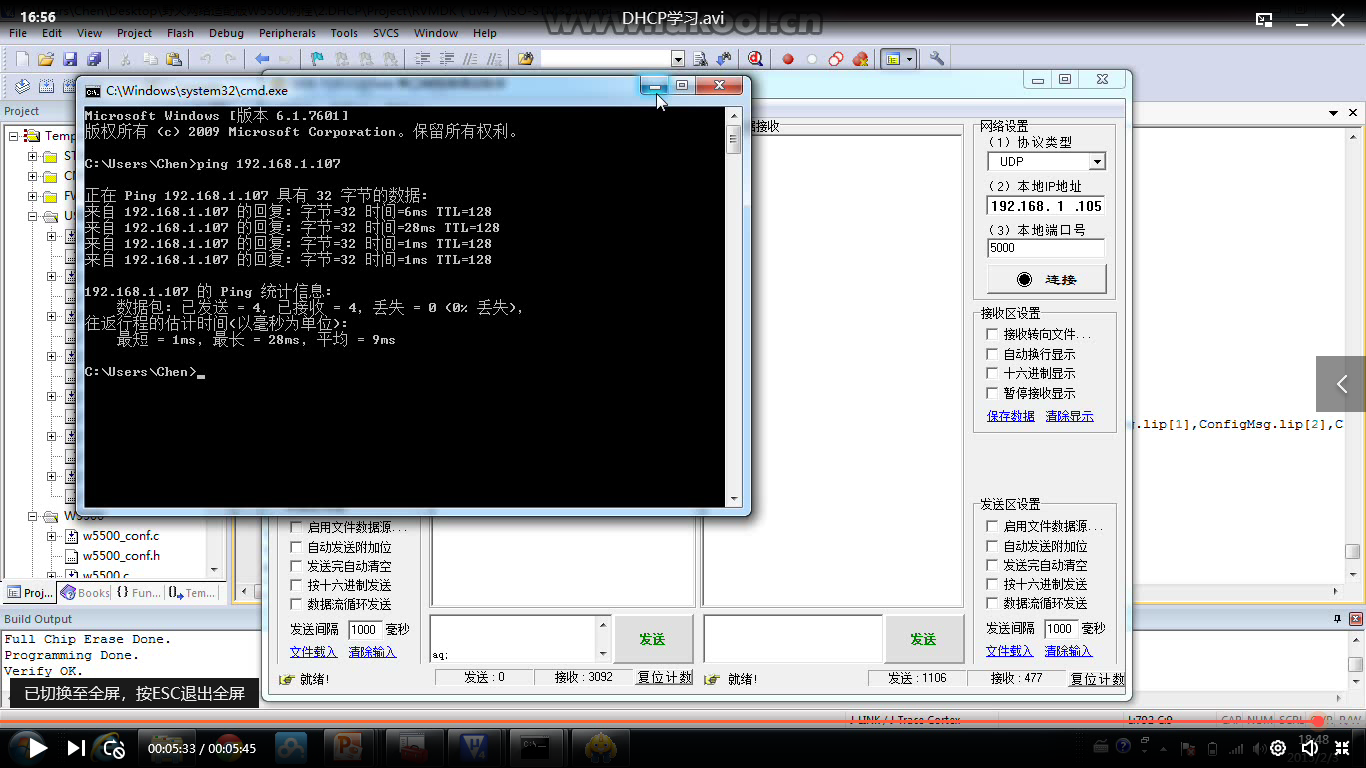


图 19 验证动态IP是否冲突结果示意图

### 4.2.3 DNS域名解析

DNS是域名服务器的简称，用于解析域名，也就是说DNS可以用一串字母来代替复杂难记的数字IP地址。在平时访问网站的过程中，我们可以直接在地址栏输入网站域名来实现直接访问，但实际上访问域名的过程就是域名解析的过程。当我们想要访问open.iot.10086.cn这个网站的时候，发送访问请求给DNS，这时DNS会将网络域名解析成数字IP地址返回给客户端，客户端就会收到219.137.170.227的IP地址，然后就可以直接的访问到OneNET云平台首页。

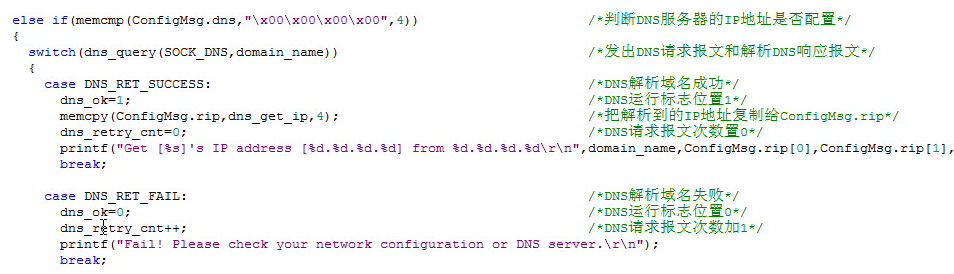


图 20 DNS主函数部分代码

首先switch定义的dns\_query是解析过程中状态的获取，是发送DNS报文和解析DNS响应报文的一个函数。需要定义一个结构体用来包括报文头信息，这个结构体就是用报文头信息来获取每一步进行的状态，然后做出反馈。开一个socket并定义一个端口。域名解析成功DNS运行标志位置为1，域名解析不成功DNS运行标志位置为0。

### 4.2.4 TCP通信

TCP通信过程中需要严格区分客户端和服务器端，因为在扮演不同角色时需要做不同的事情。在双方进行数据传输之前，需要先建立一个连接，服务器端需要先进入监听状态然后，客户机发出建立请求，这是三次握手过程。完成建立连接之后许哟断开连接，也就是四次握手过程。

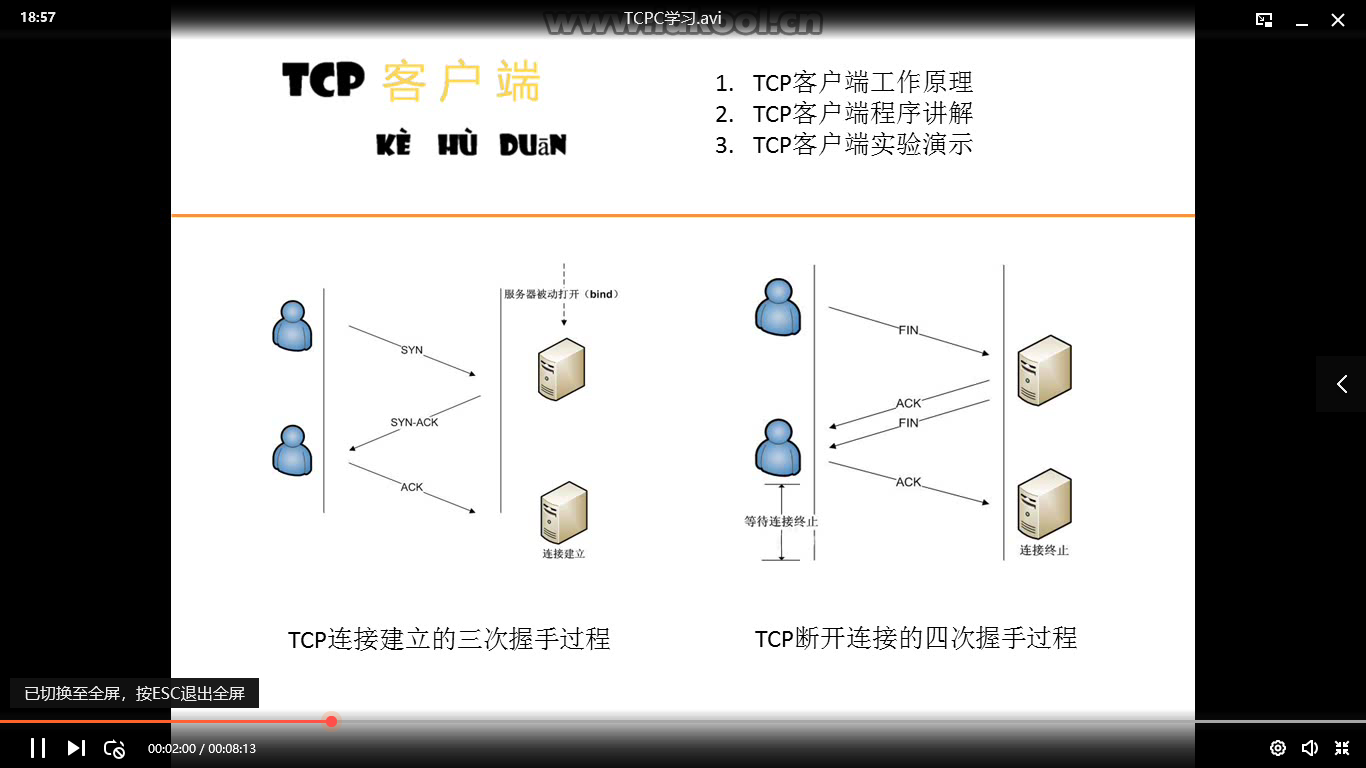


图 21 TCP连接与断开连接过程

执行TCP\_Client，主循环中主要涉及socket状态变迁，以及各个状态下所需要完成的任务。使用switch来实现socket四个状态，分别为：初始化、连接、等待关闭和关闭。

## 4.3传感器设计

DHT11温湿度传感器的数据发送流程，首先主机须要拉低数据线，也就是要发送开始信号，并且坚持t1至少18ms的时间，而后拉高数据线t2，最后就能够读取到DHT11的响应。通常情况下，DHT11作为响应信号会拉低数据线，并坚持t3 40-50us的时间，而后DHT11须要拉高数据线，保持t4 40-50us的时间后，最后开始输出数据信息。

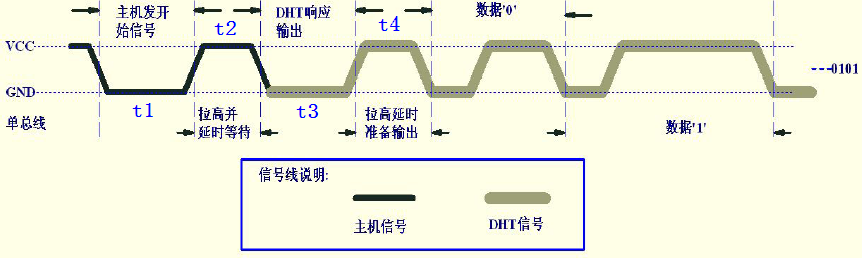


图 22 DHT11数据发送流程

对于数据“0”，是DHT11输出一个低电平，例如12-14us，然后输出一个26-18us的高电平，就代表低电平标识位0的输出。

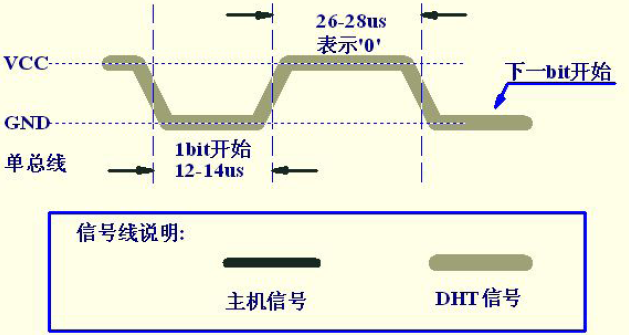


图 23 DHT1输出数字“0”的时序图

对于位1，同样输出一个12-14的低电平，然后输出一个更长的116-118us的高电平。对于0来说，一个低一个高，都比较窄，对于1来说，是一个同样宽度的低，但有一个更长的高，所以检测的时候检测到高电平后，延迟一段时间再去检测。

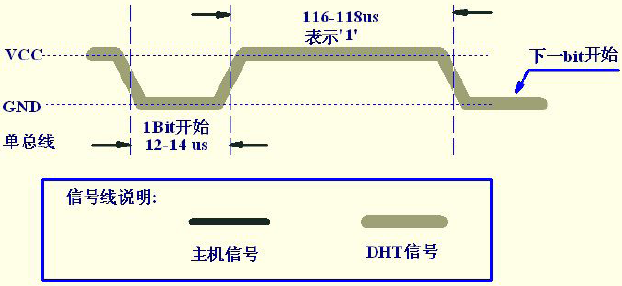


图 24 DHT1输出数字“1”的时序图

在代码中首先把主机设置为输出，然后拉低总线，再拉高

//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

}

等待回应过程中信号由DHT11输出，所以首先把IO口设置为输入，然后判断，如果是高电平则一直循环，等待低电平[10]

//等待DHT11的回应

//返回1:未检测到DHT11的存在

//返回0:存在

u8 DHT11\_Check(void)

{

u8 retry=0;

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

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

{

retry++; delay\_us(1);

};

if(retry>=100)return 1;

else retry=0;

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

{

retry++; delay\_us(1);

};

if(retry>=100)return 1;

return 0;

}

## 4.4本章小结

本章首先对整个系统的主函数框架进行了介绍和概括，明确了整体软件设计框架，对整个系统的软件设计框架进行了梳理。接着分析了各模块所需要实现的软件模块功能以及程序设计，在W55OO以太网控制模块逐步分析了如何实现数据连接，在传感器模块分析了数据如何进行联通和传输，进行了代码分析。

# 总 结

为期一个月的毕业设计进入尾声，本次毕业设计对于我本人来说是一个巨大的挑战，由于对网络通信方面知识的空白，在设计初期我基本都在了解、熟悉并且自学各种网络通信协议，为了能够顺利完成本次设计，我阅读了大量相关资料，逐渐在迷茫中找到一些头绪。最后，在不懈的努力下基本完成了本次设计。

在接触设计题目之前，我甚至不知道W5500是什么，对stm32单片机的使用也仅限于入门级了解。于是我在网上不断寻找相关教学视频，了解本次设计的基础和必备技术，渐渐对网络协议和STM32开发有了一定的基础。我跟着网络教学视频一步步学习，一步步学会如何调通W5500和互联网云平台，如何让W5500与单片机进行数据联通。除此之外，我还巩固了传感器的相关知识，学习到了如何使用传感器进行数据传输。在这个逐步学习的过程中我发现，单单只学习独立的模块如何使用是不行的，作为一个完整的系统，需要的是各个模块之间相互连接，相互作用，相互合作来实现，所以需要把之前学习到的模块知识进行交互与融合，这一过程也锻炼了我的整体思维能力。在设计中后期，各模块的硬件电路已经成功搭建后，软件程序部分的设计对我来说也具有一定的困难，对于大学四年主修硬件课程的我来说，完成这样一个庞大的系统程序的设计是一个非常大的挑战。但是通过查找资料，并且向老师同学寻求帮助，软件设计部分最终也能够成功的实现功能。在这一过程中，大大提高了我之前较为匮乏的软件方面知识。

在本次的设计中，由于不能很好的实现远程控制功能，只能由单片机控制，我放弃了只能单方面向OneNET云平台上传数据点，而不能从云平台直接接受命令的HTTP协议，选择了可以实现远程控制，通过OneNET云平台可以直接控制的MQTT协议。但是，我的设计中也还有许多不足之处，例如该系统的报警模块较为单一，只能实现单方面的报警功能，后续过程中可以考虑加入消除警报功能。并且可以通过更换不同类型的传感器，从而把该系统应用到不同的远程环境中。

毕业设计是一个能够极大程度提高自身各方面能力的过程，在这个过程当中我受益匪浅，也学习到了很多，比如收集资料的能力，阅读外文文献的能力以及文字撰写能力等，可以说是对我本人全方面能力的提升。从设计初期的查找文献，设计中期的学习新知识新技术，到设计后期的不断调试，不断的寻找问题并且解决问题，在这整个过程当中我完整的经历了一个大的系统从雏形到成型的过程，倍感欣慰。相信以后无论是否走技术路线，这都是我人生当中的一次宝贵经历。

# 参 考 文 献

1. 孙胜齐.下一代网络与物联网发展的有机结合[J].中国新通信,2018,20(20):160.
2. 杨红波,朱磊,张博,鲁露.STM32的W5500以太网通信模块快速实现方法[J].单片机与嵌入式系统应用,2018,18(08):11-16.
3. 汪海兵,董天宝.基于TCP/IP硬件协议栈的嵌入式HTTP服务器设计与实现[J].电气自动化,2018,40(05):8-9+29.
4. Qinyun Luo, Jinguo Zhao, Min Chen. Research of TCP/IP protocol stack based on embedded system[P]. Computer Research and Development (ICCRD), 2011 3rd International Conference on,2011.
5. 段晓忠.TCP/IP协议栈浅析[J].经贸实践,2017(19):280.
6. Shang, Bo,Wu, Chengdong,Meng, Tingting,Gao, Chengxi,Zhang, Yunzhou. A Data/Image Transmission Device Based on TCP/IP Protocol[P]. ,2012.
7. 公茂法, 公鑫, 张敏, 等.基于OneNet平台的电力负荷监测系统的研究[J].电测与仪表, 2017, 54, (15) :65-68.
8. 马跃, 孙翱, 贾军营, 等. MQTT协议在移动互联网即时通信中的应用[J].计算机系统应用, 2016, 25 (3) :170-176.
9. 杨余柳.基于ARM Cortex-M3的STM32微控制器实战教程(第2版)[M].电子工业出版社，2017.
10. 谭浩强著. C程序设计（第三版）[M].北京：清华大学出版社，2005.7.

# 致 谢

首先，感谢陈惠珊导师的悉心指导，没有陈老师的辅导与支持，我无法顺利完成本次远程数据采集监控系统。无论是毕业课题的选择，还是任务书的确立，陈老师都秉承着严谨治学的态度，以及在整个设计过程当中对我的耐心指导，指出我设计内容的问题所在，在我遇到问题时悉心解答，陈老师还教会了我诚恳做人的态度。

同时，感谢我的家人和同学，是他们的热心和支持，让我在设计路上能够不断前行。由衷的感谢我的同学们，如果没有他们帮助我学习网络协议相关知识，为我解答各类问题，本次设计也不会进行的如此顺利。还要感谢我的家人，是他们让我没有后顾之忧，能够全身心的投入到毕业设计当中，并且一直在支持我，鼓励我。最后，感谢本次毕业设计评审的老师和专家。

向大家表示由衷的感谢！

# 附录A 原理图

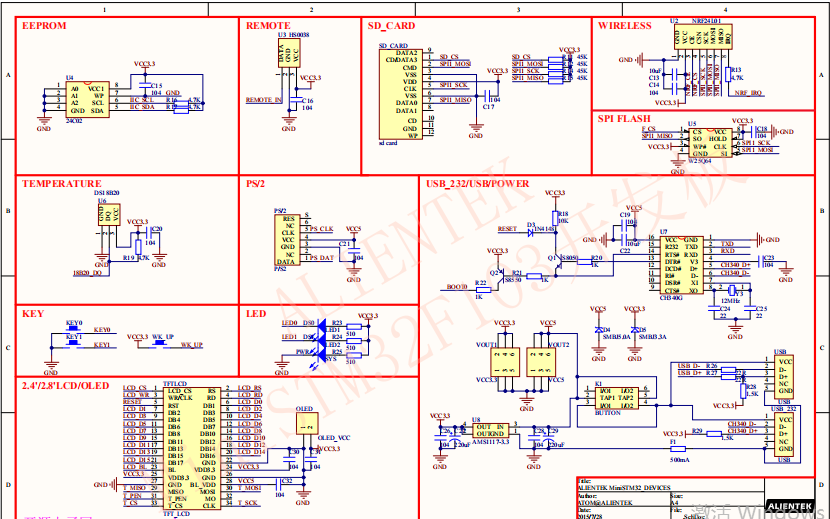


图 A1 STM32原理图

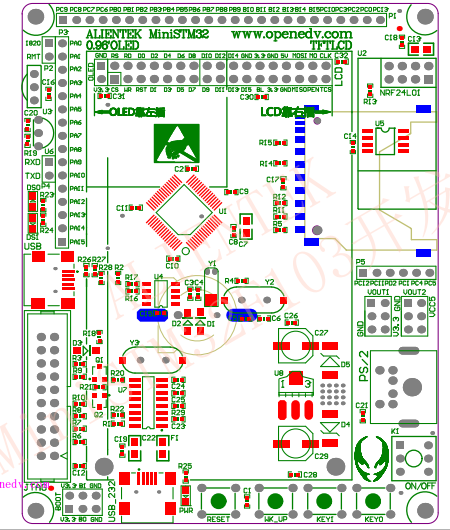


图 A2 STM32PCB

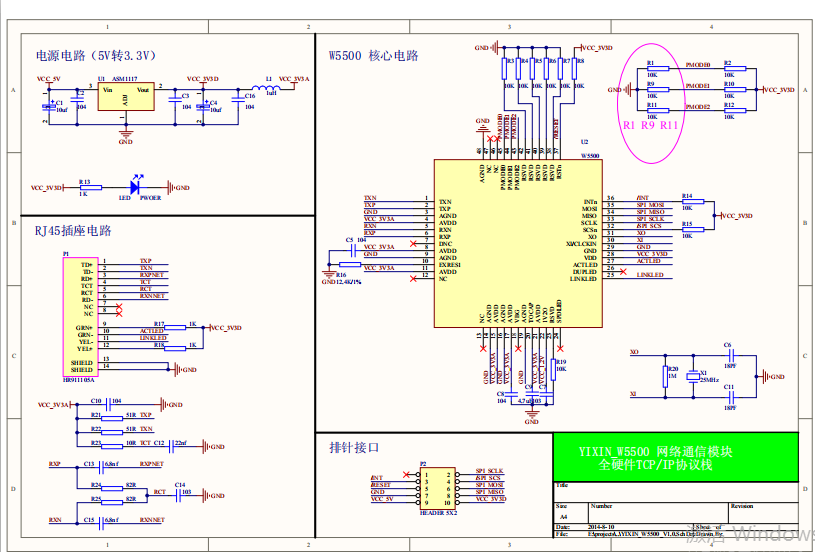


图 A3 W5500原理图

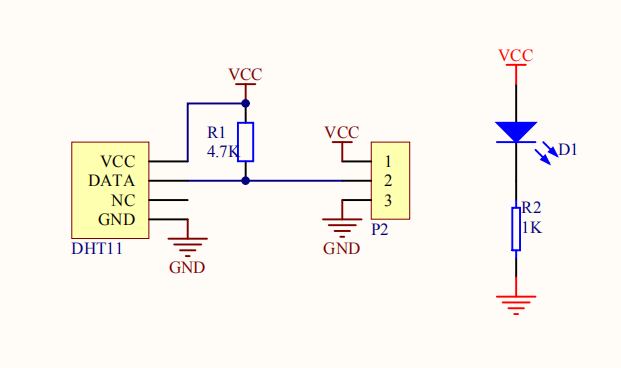


图 A4 DHT11原理图

# 附录B 源程序

## 主函数

#include "stm32f10x.h" //包含需要的头文件

#include "main.h" //包含需要的头文件

#include "delay.h" //包含需要的头文件

#include "usart1.h" //包含需要的头文件

#include "iic.h" //包含需要的头文件

#include "24c02.h" //包含需要的头文件

#include "timer1.h" //包含需要的头文件

#include "timer3.h" //包含需要的头文件

#include "led.h" //包含需要的头文件

#include "mqtt.h" //包含需要的头文件

#include "key.h" //包含需要的头文件

#include "wizchip\_conf.h" //包含需要的头文件

#include "socket.h" //包含需要的头文件

#include "dhcp.h" //包含需要的头文件

#include "spi.h" //包含需要的头文件

#include "dns.h" //包含需要的头文件

#include "networking.h"

#include "mqtt\_message.h"

#include "lcd.h"

#include "nofity.h"

#include "dht22.h"

#include "beep.h"

#include "key.h"

unsigned char gDATABUF[DATA\_BUF\_SIZE]; //数据缓冲区

int tcp\_state; //用于保存TCP返回的状态

int temp\_state; //用于临时保存TCP返回的状态

char my\_dhcp\_retry = 0; //DHCP当前共计重试的次数

char SOCK\_flag = 0; //端口链接标志 0：没有链接服务器 1：已经链接服务器，等待服务器响应

//事件计数器

unsigned long int ev\_re\_dhcp = 0;

unsigned long int ev\_get\_temp = 0;

unsigned long int ev\_update\_lcd = 0;

unsigned long int ev\_tempture\_controller = 0;

//设定

int GET\_TEMP\_DUR = 5000; //15秒上报一次温度

int UPDATE\_LCD\_DUR = 300; //刷新lcd

int TEMP\_CONTROLLER\_DUR = 200; //刷新温度控制器

int main(void)

{

int ret; //用于保存函数返回值

int dhcpstat = 0;

int dht22\_status = 0;

int key\_status = 0;

Delay\_Init(); //延时功能初始化

MILLIS\_Init(); //开机时间计时器

Usart1\_Init(9600); //串口1功能初始化，波特率9600

//LED\_Init(); //LED初始化

LCD\_Init();

InitNofity();

dht22\_status = DHT11\_Init();

u1\_printf("dht status:%d\r\n",dht22\_status);

makenofity("Initing mqtt ");

MQTT\_Buff\_Init(); //初始化接收,发送,命令数据的 缓冲区 以及各状态参数

makenofity("mqtt success ");

makenofity("Initing OneNetIot");

OneNetIoT\_Parameter\_Init(); //初始化连接OneNet云IoT平台MQTT服务器的参数

makenofity("OneNetIot success");

makenofity("Initing w5500 ");

W5500\_init(); //初始化W5500

makenofity("w5500 success ");

BEEP\_Init(); //启动蜂鸣器

Delay\_Ms(2000);

makenofity("starting loop ");

EXTIX\_Init();

//dhcpstat = DHCP\_run();

while(1) //主循环

{

//dhcpstat = DHCP\_run();

if(millis()-ev\_re\_dhcp>=2000){

dhcpstat = DHCP\_run();

ev\_re\_dhcp = millis();

}

if(millis()-ev\_get\_temp>=GET\_TEMP\_DUR){

printDht11Data();

ev\_get\_temp = millis();

}

if(millis()-ev\_update\_lcd>=UPDATE\_LCD\_DUR){

UpdateLcd();

ev\_update\_lcd = millis();

}

if(millis()-ev\_tempture\_controller>=TEMP\_CONTROLLER\_DUR){

temp\_controller();

ev\_tempture\_controller = millis();

}

switch(dhcpstat) //判断DHCP执行在哪个过程

{

case DHCP\_IP\_ASSIGN: //该状态表示，路由器分配给开发板ip了

my\_ip\_assign(); //调用IP参数获取函数，记录各个参数

break; //跳出

case DHCP\_IP\_CHANGED: //该状态表示，路由器改变了分配给开发板的ip

my\_ip\_assign(); //调用IP参数获取函数，记录各个参数

break; //跳出

case DHCP\_IP\_LEASED: //该状态表示，路由器分配给的开发板ip，正式租用了，表示可以联网通信了

tcp\_state=getSn\_SR(SOCK\_TCPS); //获取TCP链接端口的状态

if(tcp\_state!=temp\_state){ //如果和上次的状态不一样

temp\_state = tcp\_state; //保存本次状态

u1\_printf("状态编码:0x%x\r\n",tcp\_state); //串口输出信息

}

if(ctlwizchip(CW\_GET\_PHYLINK, (void\*)&ret) == -1){ //如果if成立，表示未知错误

u1\_printf("未知错误，准备重启\r\n"); //提示信息

NVIC\_SystemReset(); //重启

}

if(ret == PHY\_LINK\_OFF){ //判断网线是否断开

u1\_printf("网线断开\r\n"); //如果检测到，网线断开，串口提示信息

MQTT\_Buff\_ReInit();

Delay\_Ms(1000); //延时1s

}

switch(tcp\_state){ //switch语句，判断当前TCP链接的状态

case SOCK\_INIT:

if(Connect\_flag==0){ //如果还没有链接服务器，进入if

u1\_printf("准备连接服务器\r\n"); //串口输出信息

ret = connect(SOCK\_TCPS,ServerIP,ServerPort); //链接服务器

u1\_printf("连接服务器返回码：%d\r\n",ret); //串口输出信息

if(ret==SOCKERR\_NOPEN){ //服务器未开启进入if

Delay\_Ms(3000); //延时3s

}

}

break; //跳出

//链接状态的事件

case SOCK\_ESTABLISHED:

if((Connect\_flag==0)&&(getSn\_IR(SOCK\_TCPS)==Sn\_IR\_CON)){ //判断链接是否建立

u1\_printf("连接已建立\r\n"); //串口输出信息

MQTT\_Buff\_ReInit(); //重新初始化发送缓冲区

Connect\_flag = 1; //链接标志=1

}

ret = recv(SOCK\_TCPS,gDATABUF,DATA\_BUF\_SIZE); //接收数据

if(ret > 0){ //如果ret大于0，表示有数据来

memcpy(&MQTT\_RxDataInPtr[1],gDATABUF,ret); //拷贝数据到接收缓冲区

MQTT\_RxDataInPtr[0] = ret; //记录数据长度

MQTT\_RxDataInPtr+=BUFF\_UNIT; //指针下移

if(MQTT\_RxDataInPtr==MQTT\_RxDataEndPtr) //如果指针到缓冲区尾部了

MQTT\_RxDataInPtr = MQTT\_RxDataBuf[0]; //指针归位到缓冲区开头

TIM\_SetCounter(TIM3, 0); //清零定时器3计数器，重新计时ping包发送时间

}

/\*--------------------------------------------------------------------\*/

/\* Connect\_flag=1同服务器建立了连接,我们可以发布数据和接收推送了 \*/

/\*--------------------------------------------------------------------\*/

if(Connect\_flag==1){

mqtt\_message\_processor(); //mqtt处理函数

}

break; //跳出

case SOCK\_CLOSE\_WAIT: u1\_printf("等待关闭连接\r\n"); //串口输出信息

if((ret=disconnect(SOCK\_TCPS)) != SOCK\_OK) //端口关闭

{

u1\_printf("连接关闭失败，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

}

u1\_printf("连接关闭成功\r\n"); //串口输出信息

Connect\_flag = 0; //链接标志=0

break;

case SOCK\_CLOSED: u1\_printf("准备打开W5500端口\r\n"); //串口输出信息

Connect\_flag = 0; //链接标志=0

ret = socket(SOCK\_TCPS,Sn\_MR\_TCP,5050,Sn\_MR\_ND); //打开W5500的端口，用于建立TCP链接，本地TCP端口5050

if(ret != SOCK\_TCPS){ //如果打开失败，进入if

u1\_printf("端口打开错误，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

}else u1\_printf("打开5050端口成功\r\n"); //串口输出信息

break; //跳出

}

break; //跳出

case DHCP\_FAILED: //该状态表示DHCP获取IP失败

my\_dhcp\_retry++; //失败次数+1

if(my\_dhcp\_retry > MY\_MAX\_DHCP\_RETRY){ //如果失败次数大于最大次数，进入if

u1\_printf("DHCP失败，准备重启\r\n");//串口提示信息

NVIC\_SystemReset(); //重启

}

break; //跳出

}

}

}

/\*-------------------------------------------------\*/

/\*函数名：判断开关和倒计时状态，并发布给服务器 \*/

/\*参 数：无 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

//void LED\_CD\_State(void)

//{

// char temp[20]; //定义一个临时缓冲区

//

// memset(temp,0,20); //清空临时缓冲区

// sprintf(temp,"+OneSW:"); //构建回复数据

//

// if(LED1\_IN\_STA) temp[7] = '0'; //如果LED1是高电平，说明是熄灭状态，开关1状态位置0

// else temp[7] = '1'; //反之，说明是点亮状态，开关1状态位置1

// temp[8] = CDTime; //加入剩余倒计时时间

// MQTT\_PublishQs0(P\_TOPIC\_NAME,temp,9); //添加数据，发布给服务器

//}

//温度报警控制器

void temp\_controller(void){

//转换

int temp = 0;

int hum = 0;

temp = atoi(Nofity\_temp);

hum = atoi(Nofity\_hum);

if(hum>=hum\_thr&&temp>=temp\_thr){

setbeep(3);

return;

}

if(temp>=temp\_thr){

setbeep(1);

return ;

}else if(hum>=hum\_thr){

setbeep(2);

return ;

}else{

unsetbeep(1);

}

}

## 以太网模块

### MQTT

#include "stm32f10x.h" //包含需要的头文件

#include "mqtt\_message.h"

#include "mqtt.h"

#include "stdio.h"

#include "stdlib.h"

#include "nofity.h"

char cmd\_buf[32] = {0};

void mqtt\_message\_processor(void){

/\*-------------------------------------------------------------\*/

/\* 处理发送缓冲区数据 \*/

/\*-------------------------------------------------------------\*/

if(MQTT\_TxDataOutPtr != MQTT\_TxDataInPtr){ //if成立的话，说明发送缓冲区有数据了

//3种情况可进入if

//第1种：0x10 连接报文

//第2种：0x82 订阅报文，且ConnectPack\_flag置位，表示连接报文成功

//第3种：SubcribePack\_flag置位，说明连接和订阅均成功，其他报文可发

if((MQTT\_TxDataOutPtr[1]==0x10)||((MQTT\_TxDataOutPtr[1]==0x82)&&(ConnectPack\_flag==1))||(SubcribePack\_flag==1)){

u1\_printf("发送数据:0x%x\r\n",MQTT\_TxDataOutPtr[1]); //串口提示信息

MQTT\_TxData(MQTT\_TxDataOutPtr); //发送数据

MQTT\_TxDataOutPtr += BUFF\_UNIT; //指针下移

if(MQTT\_TxDataOutPtr==MQTT\_TxDataEndPtr) //如果指针到缓冲区尾部了

MQTT\_TxDataOutPtr = MQTT\_TxDataBuf[0]; //指针归位到缓冲区开头

}

}//处理发送缓冲区数据的else if分支结尾

/\*-------------------------------------------------------------\*/

/\* 处理接收缓冲区数据 \*/

/\*-------------------------------------------------------------\*/

if(MQTT\_RxDataOutPtr != MQTT\_RxDataInPtr){ //if成立的话，说明接收缓冲区有数据了

u1\_printf("接收到数据:");

/\*-----------------------------------------------------\*/

/\* 处理CONNACK报文 \*/

/\*-----------------------------------------------------\*/

//if判断，如果一共接收了4个字节，第一个字节是0x20，表示收到的是CONNACK报文

//接着我们要判断第4个字节，看看CONNECT报文是否成功

if((MQTT\_RxDataOutPtr[0]==4)&&(MQTT\_RxDataOutPtr[1]==0x20)){

switch(MQTT\_RxDataOutPtr[4]){

case 0x00 : u1\_printf("CONNECT报文成功\r\n"); //串口输出信息

ConnectPack\_flag = 1; //CONNECT报文成功，订阅报文可发

break; //跳出分支case 0x00

case 0x01 : u1\_printf("连接已拒绝，不支持的协议版本，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支case 0x01

case 0x02 : u1\_printf("连接已拒绝，不合格的客户端标识符，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支case 0x02

case 0x03 : u1\_printf("连接已拒绝，服务端不可用，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支case 0x03

case 0x04 : u1\_printf("连接已拒绝，无效的用户名或密码，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支case 0x04

case 0x05 : u1\_printf("连接已拒绝，未授权，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支case 0x05

default : u1\_printf("连接已拒绝，未知状态，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支case default

}

}

//if判断，如果一共接收了5个字节，第一个字节是0x90，表示收到的是SUBACK报文

//接着我们要判断订阅回复，看看是不是成功

else if((MQTT\_RxDataOutPtr[0]==5)&&(MQTT\_RxDataOutPtr[1]==0x90)){

switch(MQTT\_RxDataOutPtr[5]){

case 0x00 :

case 0x01 : u1\_printf("订阅成功\r\n"); //串口输出信息

SubcribePack\_flag = 1; //SubcribePack\_flag置1，表示订阅报文成功，其他报文可发送

TIM3\_ENABLE\_30S(); //启动30s的PING定时器

break; //跳出分支

default : u1\_printf("订阅失败，准备重启\r\n"); //串口输出信息

NVIC\_SystemReset(); //重启

break; //跳出分支

}

}

//if判断，如果一共接收了2个字节，第一个字节是0xD0，表示收到的是PINGRESP报文

else if((MQTT\_RxDataOutPtr[0]==2)&&(MQTT\_RxDataOutPtr[1]==0xD0)){

u1\_printf("PING报文回复\r\n"); //串口输出信息

}

//if判断，如果第一个字节是0x30，表示收到的是服务器发来的推送数据

//我们要提取控制命令

else if((MQTT\_RxDataOutPtr[1]==0x30)){

u1\_printf("服务器等级0推送\r\n"); //串口输出信息

MQTT\_DealPushdata\_Qs0(MQTT\_RxDataOutPtr); //处理等级0推送数据

}

MQTT\_RxDataOutPtr += BUFF\_UNIT; //指针下移

if(MQTT\_RxDataOutPtr==MQTT\_RxDataEndPtr) //如果指针到缓冲区尾部了

MQTT\_RxDataOutPtr = MQTT\_RxDataBuf[0]; //指针归位到缓冲区开头

}//处理接收缓冲区数据的else if分支结尾

/\*-------------------------------------------------------------\*/

/\* 处理命令缓冲区数据 \*/

/\*-------------------------------------------------------------\*/

if(MQTT\_CMDOutPtr != MQTT\_CMDInPtr){ //if成立的话，说明命令缓冲区有数据了

MQTT\_CMDOutPtr[MQTT\_CMDOutPtr[0]+1] = '\0'; //加入字符串结束符

u1\_printf("命令:%s\r\n",&MQTT\_CMDOutPtr[1]); //串口输出信息

sprintf(cmd\_buf,"%s",&MQTT\_CMDOutPtr[1]);

handler\_cmd();

MQTT\_CMDOutPtr += BUFF\_UNIT; //指针下移

if(MQTT\_CMDOutPtr==MQTT\_CMDEndPtr){ //如果指针到缓冲区尾部了

MQTT\_CMDOutPtr = MQTT\_CMDBuf[0]; //指针归位到缓冲区开头

}

}

}

void handler\_cmd(){

if(cmd\_buf[0]=='T'){

temp\_thr=atoi(cmd\_buf+1);

}else if(cmd\_buf[0]=='H'){

hum\_thr=atoi(cmd\_buf+1);

}else if(cmd\_buf[0]=='C'){

u1\_printf("close buzzer\r\n");

beeper\_enable = 0;

}else if(cmd\_buf[0]=='O'){

u1\_printf("enable buzzer\r\n");

beeper\_enable = 1;

}

//刷新nofity

sprintf(Nofity\_hum\_thr,"%d",hum\_thr);

sprintf(Nofity\_temp\_thr,"%d",temp\_thr);

u1\_printf("change Nofity\_hum\_thr:%d,Nofity\_temp\_thr:%d\r\n",hum\_thr,temp\_thr);

}

int onenet\_new\_json\_str(char \*key,char \*value,char \*payload){

int i = 0;

char str\_json[34] = {0};

int str\_json\_length = 0;

sprintf(str\_json,"{\"%s\":%s}\r\n",key,value);

for(i=0;i<35;i++){

if(str\_json[i]=='\r'&&str\_json[i+1]=='\n'){

str\_json\_length = i;

}

}

u1\_printf("raw: %s",str\_json);

u1\_printf("length: %d\r\n",str\_json\_length);

payload[0] = 3;

payload[1] = str\_json\_length/256;

payload[2] = str\_json\_length%256;

memcpy(payload+3,str\_json,str\_json\_length);

return str\_json\_length+3;

}

void publishTempAndHumMsg(u8\* hum,u8\* temp){

int loop=0;

int pub\_length = 0;

char payload\_temp[37] = {0};

char payload\_hum[37] = {0};

char str\_temp[8] = {0};

char str\_hum[8] = {0};

sprintf(str\_temp,"%d.%d",temp[0],temp[1]);

sprintf(str\_hum,"%d.%d",hum[0],hum[1]);

strcpy(Nofity\_temp,str\_temp);

strcpy(Nofity\_hum,str\_hum);

if(Connect\_flag==1){

pub\_length = onenet\_new\_json\_str("temp",str\_temp,payload\_temp);

MQTT\_PublishQs0(P\_TOPIC\_SYS\_DATAPOINT,payload\_temp,pub\_length);

pub\_length = onenet\_new\_json\_str("hum",str\_hum,payload\_hum);

MQTT\_PublishQs0(P\_TOPIC\_SYS\_DATAPOINT,payload\_hum,pub\_length);

}

//MQTT\_PublishQs0(P\_TOPIC\_TEMP\_NAME,str\_temp,8);

}

void publishBeepTypeMsg(int beep\_type){

int pub\_length = 0;

char payload\_beeptype[37] = {0};

char str\_beeptype[8] = {0};

sprintf(str\_beeptype,"%d",beep\_type);

pub\_length = onenet\_new\_json\_str("beep\_type",str\_beeptype,payload\_beeptype);

if(Connect\_flag==1)

MQTT\_PublishQs0(P\_TOPIC\_SYS\_DATAPOINT,payload\_beeptype,pub\_length);

}

### 网络连接

#include "stm32f10x.h" //包含需要的头文件

#include "networking.h"

#include "nofity.h"

unsigned char gDNSBUF[DNS\_DATA\_BUF\_SIZE]; //数据缓冲区

char DNS\_flag = 0; //DNS解析标志 0:未成功 1：成功

wiz\_NetInfo gWIZNETINFO = //MAC地址自己写一个，不要和路由器下其他的设备一样即可

{

0x00, 0x08, 0xdc,0x00, 0xab, 0xce,

};

/\*-------------------------------------------------\*/

/\*函数名：DNS域名解析 \*/

/\*参 数：DomainName:域名 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

void W5500\_DNS(unsigned char \*DomainName)

{

u1\_printf("开始解析域名\r\n"); //串口输出信息

DNS\_init(SOCKET\_DNS, gDNSBUF); //DNS初始化

if(DNS\_run(gWIZNETINFO.dns,DomainName,ServerIP)==1){ //开始解析域名

u1\_printf("DNS解析成功：%d.%d.%d.%d\r\n",ServerIP[0],ServerIP[1],ServerIP[2],ServerIP[3]);//串口输出信息

DNS\_flag = 1; //DNS解析成功，标志=1

}

}

/\*-------------------------------------------------\*/

/\*函数名：获取到IP时的回调函数 \*/

/\*参 数：无 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

void my\_ip\_assign(void)

{

getIPfromDHCP(gWIZNETINFO.ip); //把获取到的ip参数，记录到机构体中

getGWfromDHCP(gWIZNETINFO.gw); //把获取到的网关参数，记录到机构体中

getSNfromDHCP(gWIZNETINFO.sn); //把获取到的子网掩码参数，记录到机构体中

getDNSfromDHCP(gWIZNETINFO.dns); //把获取到的DNS服务器参数，记录到机构体中

gWIZNETINFO.dhcp = NETINFO\_DHCP; //标记使用的是DHCP方式

network\_init(); //初始化网络

u1\_printf("DHCP租期 : %d 秒\r\n", getDHCPLeasetime());

}

/\*-------------------------------------------------\*/

/\*函数名：获取IP的失败函数 \*/

/\*参 数：无 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

void my\_ip\_conflict(void)

{

u1\_printf("获取IP失败，准备重启\r\n"); //提示获取IP失败

NVIC\_SystemReset(); //重启

}

/\*-------------------------------------------------\*/

/\*函数名：初始化网络函数 \*/

/\*参 数：无 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

void network\_init(void)

{

char tmpstr[6] = {0};

wiz\_NetInfo netinfo;

ctlnetwork(CN\_SET\_NETINFO, (void\*)&gWIZNETINFO);//设置网络参数

ctlnetwork(CN\_GET\_NETINFO, (void\*)&netinfo); //读取网络参数

ctlwizchip(CW\_GET\_ID,(void\*)tmpstr); //读取芯片ID

//打印网络参数

if(netinfo.dhcp == NETINFO\_DHCP) u1\_printf("\r\n=== %s NET CONF : DHCP ===\r\n",(char\*)tmpstr);

else u1\_printf("\r\n=== %s NET CONF : Static ===\r\n",(char\*)tmpstr);

u1\_printf("===========================\r\n");

u1\_printf("MAC地址: %02X:%02X:%02X:%02X:%02X:%02X\r\n",netinfo.mac[0],netinfo.mac[1],netinfo.mac[2],netinfo.mac[3],netinfo.mac[4],netinfo.mac[5]);

u1\_printf("IP地址: %d.%d.%d.%d\r\n", netinfo.ip[0],netinfo.ip[1],netinfo.ip[2],netinfo.ip[3]);

u1\_printf("网关地址: %d.%d.%d.%d\r\n", netinfo.gw[0],netinfo.gw[1],netinfo.gw[2],netinfo.gw[3]);

u1\_printf("子网掩码: %d.%d.%d.%d\r\n", netinfo.sn[0],netinfo.sn[1],netinfo.sn[2],netinfo.sn[3]);

u1\_printf("DNS服务器: %d.%d.%d.%d\r\n", netinfo.dns[0],netinfo.dns[1],netinfo.dns[2],netinfo.dns[3]);

u1\_printf("===========================\r\n");

sprintf(Nofity\_ipv4address,"%d.%d.%d.%d\r\n", netinfo.ip[0],netinfo.ip[1],netinfo.ip[2],netinfo.ip[3]);

}

/\*-------------------------------------------------\*/

/\*函数名：初始化W5500 \*/

/\*参 数：无 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

void W5500\_init(void)

{

//W5500收发内存分区，收发缓冲区各自总的空间是16K，（0-7）每个端口的收发缓冲区我们分配 2K

char memsize[2][8] = {{2,2,2,2,2,2,2,2},{2,2,2,2,2,2,2,2}};

char tmp;

SPI\_Configuration(); //初始化SPI接口

W5500\_Reset(); //W5500复位

reg\_wizchip\_cris\_cbfunc(SPI\_CrisEnter, SPI\_CrisExit); //注册临界区函数

reg\_wizchip\_cs\_cbfunc(SPI\_CS\_Select, SPI\_CS\_Deselect); //注册SPI片选信号函数

reg\_wizchip\_spi\_cbfunc(SPI\_ReadByte, SPI\_WriteByte); //注册读写函数

if(ctlwizchip(CW\_INIT\_WIZCHIP,(void\*)memsize) == -1){ //如果if成立，表示收发内存分区失败

u1\_printf("初始化收发分区失败,准备重启\r\n"); //提示信息

NVIC\_SystemReset(); //重启

}

do{ //检查连接状态

if(ctlwizchip(CW\_GET\_PHYLINK, (void\*)&tmp) == -1){ //如果if成立，表示未知错误

u1\_printf("未知错误，准备重启\r\n"); //提示信息

NVIC\_SystemReset(); //重启

}

if(tmp == PHY\_LINK\_OFF){

u1\_printf("网线未连接\r\n");//如果检测到，网线没连接，提示连接网线

Delay\_Ms(2000); //延时2s

}

}while(tmp == PHY\_LINK\_OFF); //循环执行，直到连接上网线

setSHAR(gWIZNETINFO.mac); //设置MAC地址

DHCP\_init(SOCK\_DHCP, gDNSBUF); //初始化DHCP

reg\_dhcp\_cbfunc(my\_ip\_assign, my\_ip\_assign, my\_ip\_conflict);//注册DHCP回调函数

DNS\_flag = 0; //DNS解析标志=0

my\_dhcp\_retry = 0; //DHCP重试次数=0

tcp\_state = 0; //TCP状态=0

temp\_state = -1; //上一次TCP状态=-1

}

/\*-------------------------------------------------\*/

/\*函数名：W5500发送数据 \*/

/\*参 数：data:数据 \*/

/\*返回值：无 \*/

/\*-------------------------------------------------\*/

void W5500\_TxData(unsigned char \*data)

{

send(SOCK\_TCPS,&data[1],data[0]); //W5500发送数据

}

## 蜂鸣器

#include "stm32f10x.h" //包含需要的头文件

#include "nofity.h"

#include "lcd.h"

#include "usart1.h"

#include "beep.h"

#include "mqtt\_message.h"

#include "mqtt.h"

char Nofity\_msg[24]={0};

char Nofity\_hum[24]={0};

char Nofity\_temp[24]={0};

char Nofity\_hum\_thr[24]={0};

char Nofity\_temp\_thr[24]={0};

char Nofity\_beep[24]={0};

char Nofity\_buzzerstatus[16]={0};

char Nofity\_ipv4address[24]={0};

char Nofity\_onenetstatus[16]={0};

int temp\_thr = 32;

int hum\_thr = 80;

int beep\_nofity = 0;

int beeper\_enable = 1; //是否许可蜂鸣器

void InitNofity(void){

POINT\_COLOR=RED;

LCD\_Clear(WHITE);

LCD\_ShowString(10,20,200,24,12,"welcome!!!! ");

LCD\_ShowString(10,33,200,24,12,"Nofity: ");

LCD\_ShowString(10,57,200,24,12,"Hum:");

LCD\_ShowString(10,81,200,24,12,"Temp:");

LCD\_ShowString(10,105,200,24,12,"Hum\_Thr:");

LCD\_ShowString(10,129,200,24,12,"Temp\_Thr:");

LCD\_ShowString(10,153,200,24,12,"BEEP:");

LCD\_ShowString(10,177,200,24,12,"Buzzer Status:");

LCD\_ShowString(10,201,200,24,12,"IPV4 Address:");

LCD\_ShowString(10,225,200,24,12,"OneNet");

//初始化参数

sprintf(Nofity\_hum\_thr,"%d",hum\_thr);

sprintf(Nofity\_temp\_thr,"%d",temp\_thr);

sprintf(Nofity\_beep,"-------------");

}

void set\_buzzer(){

if(beeper\_enable==1){

sprintf(Nofity\_buzzerstatus,"Enable ");

}else if(beeper\_enable==0){

sprintf(Nofity\_buzzerstatus,"Disable");

}

}

void setbeep(int type){

//重置蜂鸣器

switch(type){

case 1:sprintf(Nofity\_beep,"Temp Beep!!!!");if(beeper\_enable==1)Sound();else UnSound(); if(beep\_nofity!=1)publishBeepTypeMsg(1); beep\_nofity=1; break;

case 2:sprintf(Nofity\_beep,"Hum Beep!!!!!");if(beeper\_enable==1)Sound();else UnSound(); if(beep\_nofity!=2)publishBeepTypeMsg(2); beep\_nofity=2; break;

case 3:sprintf(Nofity\_beep,"Temp&Hum Beep");if(beeper\_enable==1)Sound();else UnSound(); if(beep\_nofity!=3)publishBeepTypeMsg(3); beep\_nofity=3; break;

default:sprintf(Nofity\_beep,"-------------");UnSound(); beep\_nofity=0; break;

}

}

void setonenetstatus(){

if(Connect\_flag==1){

sprintf(Nofity\_onenetstatus,"Online ");

}else{

sprintf(Nofity\_onenetstatus,"Offline");

}

}

void unsetbeep(int type){

if(beep\_nofity!=0){

beeper\_enable = 1;

}

sprintf(Nofity\_beep,"-------------");UnSound();

beep\_nofity = 0;

}

void makenofity(u8 \*p){

u1\_printf("%s\r\n",p);

sprintf(Nofity\_msg,"%s",p);

LCD\_ShowString(10,45,200,48,12,Nofity\_msg);

}

void UpdateLcd(){

setonenetstatus();

set\_buzzer();

LCD\_ShowString(10,45,200,48,12,Nofity\_msg);

LCD\_ShowString(10,69,200,24,12,Nofity\_hum);

LCD\_ShowString(10,93,200,24,12,Nofity\_temp);

LCD\_ShowString(10,117,200,24,12,Nofity\_hum\_thr);

LCD\_ShowString(10,141,200,24,12,Nofity\_temp\_thr);

LCD\_ShowString(10,165,200,24,12,Nofity\_beep);

LCD\_ShowString(10,189,200,24,12,Nofity\_buzzerstatus);

LCD\_ShowString(10,213,200,24,12,Nofity\_ipv4address);

LCD\_ShowString(10,237,200,24,12,Nofity\_onenetstatus);

}