长沙理工大学

《嵌入式系统设计综合实训》报告

基于 STM32 与 IMX6ULL 的智能家居系统
-驱动控制模块

学　　院：	计算机与通信工程	专　　业：	计算机科学与技术
班　　级：	计算机 19-1 班	学　　号：	201908010117
学生姓名：	谭润清	指导教师：	郑斌老师
课程成绩：		完成日期：	2022.07.01

实训成绩评定标准

实训目标/毕业要求	考核与评价方式及成绩比例（%）				目标点比例（%）
	日常考核	日志汇报	答辩验收	实训报告	
实训目标 1/**工程知识**	/	/	30	/	30
实训目标 2/**设计/开发解决方案**	/	/	10	20	30
实训目标 3/**个人和团队**	/	10	/	10	20
实训目标 4/**项目管理 11.1**	/	/	10	/	10
实训目标 5/**项目管理 11.2**	10	/	/	/	10
各考核项比例	10	10	50	30	100

实训成绩评定表

实训目标/毕业要求	考核与评价成绩（%）				目标点得分（%）
	日常考核	日志汇报	答辩验收	实训报告	
实训目标 1/**工程知识**	/	/		/	
实训目标 2/**设计/开发解决方案**	/	/			
实训目标 3/**个人和团队**	/		/		
实训目标 4/**项目管理 11.1**	/	/		/	
实训目标 5/**项目管理 11.2**		/	/	/	
各考核项得分					

指导教师对实训的评定意见

综合成绩 _____指导教师签字_____ 年 月 日

基于 STM32 与 IMX6ULL 的智能家居系统

摘要

随着知识经济和网络经济时代的到来，住宅智能化已经成为新世纪全新生活方式的根本保证，在社会信息化飞速发展的今天，人们对住宅的关注已经不再局限于居住面积、周边环境、交通等方面，而是吧更多的注意力放在信息服务与住房安全等方面，因此，智能家居应运而生，智能家居系统是近些年来的热词之一，智能家居系统的实现能够大大便利人们的日常居家生活，保证居家安全。本系统在当今智能家居开发热潮的背景下，基于 STM32F407 开发平台与 IMX6ULL 开发平台设计出了一套能够通过超声波对人体距离的感应控制窗帘与小灯的打开与闭合、检测房间温湿度以及烟雾报警功能的家居系统。同时，用户还可以通过 IMX6ULL 端通过 WIFI 模块对家居进行控制，并且可以通过 QT 界面查询房间内温湿度等信息。

关键词： 智能家居；STM32；WIFI；IMX6ULL；人体感应模块；烟雾报警

SMART HOME SYSTEM BASED ON STM32 AND IMX6ULL

ABSTRACT

With the advent of the era of knowledge economy and network economy, intelligent housing has become the fundamental guarantee for a new way of life in the new century. Today, with the rapid development of social informatization, people's attention to housing is no longer limited to the living area, surrounding environment, transportation and other aspects, but more attention is paid to information services and housing safety. Therefore, smart home came into being, Smart home system is one of the hot words in recent years. The implementation of smart home system can greatly facilitate people's daily home life and ensure home safety. Under the background of current smart home development upsurge, this system has designed a set of home system based on stm32f407 development platform and imx6ull development platform, which can control the opening and closing of curtains and small lights, detect room temperature and humidity and smoke alarm functions through ultrasonic induction of human distance. At the same time, the user can also control the home through the WiFi module through the imx6ull terminal, and query the temperature and humidity in the room through the QT interface.

Key words: Smart home; STM32; WIFI；IMX6ULL；Human body sensing module; Smoke alarm

目　录

目　录 5
1.绪论 7
1.1 课题背景 7
1.1.1 智能家居的由来 7
1.1.2 智能家居的发展 8
1.2 设计目的及意义 9
1.2.1 设计的目的 9
1.2.2 设计的意义 9
1.3 项目的主要任务 11
2. 可行性分析 12
2.1 项目简介 12
2.2 技术可行性分析 13
2.2.1 STM32F407 平台介绍 13
2.2.2 IMX6ULL 开发平台介绍 14
2.2.3 42 步进电机介绍 14
2.2.4 HC-SRO4 超声波感应模块介绍 16
2.2.5 MQ2 烟雾传感模块介绍 17
2.2.6 ESP8266 以及 RTL8189WIFI 模块介绍 18
2.2.7 DHT11 温湿度传感器 19
2.2.8 QT 界面设计 20
2.3 经济可行性分析 21
3. 系统总体设计 22
3.1 下位机模块介绍 22
3.2 上位机介绍 24
4.系统详细设计 25
4.1 步进电机驱动模块 25

4.1.1 电机工作的基本原理 25
4.1.2 电机及驱动的连接 27
4.1.3 代码驱动实现 29
4.2 HC-SR04 超声波模块 33
4.2.1 HC-SR04 超声波工作基本原理 33
4.2.2 超声波代码实现 34
4.3 MQ2 烟雾感应模块 37
4.3.1 MQ2 烟雾模块感应原理 37
4.3.2 MQ2 代码实现 38
4.4 DHT11 温湿度感应模块 39
4.4.1 DHT11 温湿度模块的通信协议 39
4.4.2 DHT11 模块代码实现 40
4.5 QT 模块 42
4.5.1 信号与槽 42
4.5.2 socket 通信 42
4.5.3 点灯程序实现 43
4.5.4 Socet 通信（数值的获取） 45
4.6 WIFI 模块（IMX6ULL） 46
5、系统演示 48
6. 总结与心得体会 53
参考文献 55
附录 56
附录 A：硬件原理图 56
附录 B：软件源代码 60

1. 绪论

1.1 课题背景

随着物联网、以及嵌入式等相关技术应用的不断普及，智能家居技术日益成熟。通过将智能语音技术融入终端，能够对家中的各个电器进行网络链接，通过语音控制、多点触控、手势识别等技术，实现对家居设备的智能化控制。智能家居产品设计充分关注用户的体验和情感融合，更好地实现家居产品的互联互通、跨界合作和第三方服务拓展，满足用户的个性化需求。

互联网的快速发展对智能家居市场提供了新的思路和方向，开启了"物联网+家居产品”的智能新时代。随着人们对家居生活质量要求的不断提升，智能家居产品应运而生，通过对用户多元化需求的深入挖掘，利用物联网现有技术与家居产品相互链接与融合，能够使智能家居产品更加智能化、人性化和便捷化，满足人们对智能家居产品的体验和情感需求，提高人们的居住品质。

1.1.1 智能家居的由来

智能家居是以住宅为平台，利用综合布线技术、网络通信技术、安全防范技术、自动控制技术、音视频技术等当今主流的技术将家居生活有关的设施集成，构建高效的住宅设施与家庭日程事务的管理系统，提升家居安全性、便利性、舒适性、艺术性并实现环保节能的居住环境。它是互联网技术和物联网技术的集中体现。

智能家居的概念很早就提出了，但直到 1984 年，美国联合科技公司成功制造了一栋智能建筑后，智能家居才正式出现，并得到大家的密切关注。20 世纪 80 年代，随着电子科技行业的不断发展，越来越多的电子设备被应用到智能家居，最早用于智能家居的是安全设备，如电子锁等。

现在，随着人工智能的发展、电力电子材料的日益更新、物联网的壮大，智能家居不再局限于某一个或者某一类家居，现在智能家居是一个利用电脑和网络布局而形成的智能化系统。随着城市化、智能化现代家居的推进，智能家居基于逐渐成熟的技术条件，正不断完善发展并真真切切地改变着人们的生活。

1.1.2 智能家居的发展

国外智能家居的发展概况。国外智能家居起步比较早，发展也比较迅速。早在 1995 年，美国和新加坡就已经大量推广智能家居，其平均安装一家智能家居消费高达 7000 美元，但是使用率仅占 0.3%。从目前来看，美国是全世界智能家居使用最多的国家，其次是日本、德国等国家。随着智能家居的推进，美国的智能家居市场也在不断增长和扩大，2016 年，美国的家居市场容量已经达到 97 亿美元，并且以平均每年 30 亿美元左右的增长速度迅速增长，市场的总收入已经达到 9912 万美元，家庭普及率为 5.3%。以现在的发展态势估计，2020 年北美地区的智能家居家庭普及率将达 17.23%。

国内智能家居的发展概况。相较于国外的智能家居，国内的智能家居发展比较缓慢，主要分为 4 个时期。1994—1999 年，智能家居还处于萌芽时期，国内关于其的概念比较少，局势不太明朗。2000—2005 年，处于开创期的智能家居已经在国内得到重视，在长三角和珠三角已经有企业开始研究智能家居并希望得到应用，此时国外的智能家居技术还没有引入国内。2006—2010 年是国内智能家居的徘徊期。由于之前国内智能家居处于摸索阶段，很多企业夸大其特点和作用，导致宣传效果和使用效果相差较多，一部分国内智能家居企业在这个阶段倒闭，但仍然有一部分国内企业熬过了徘徊期。

此时，很多国外企业正在我国布局增加其规模。到了 2014 年，智能家居迎来爆发期，各种智能家居的需求和认可度逐年提高，市场规模和销售额呈现指数型增长。目前国内做得比较好的企业有家电企业海尔提出的 U-HOME，以及小米。虽然小米的起步较晚，但是由于其掌握手机、电视、路由器等设备，最近几年依旧发展迅猛。

1.2 设计目的及意义

1.2.1 设计的目的

社会科技的进涉往往带动着许多行业的发展。近年科技、经济突飞猛进，人民的生活水平也得到了提升,对生活的质星要求也相应的越来越高，而生活的智能就是一个突出表现的方面，智能家居。智能家用机照人等科技产品也慢慢的被人们关注起来。在智能家庭中,智能家居可以更好地为我们提供便利的服务，在生活中给我们提供重要的信息，能对我们的日常生活进行合理的安排，充分的利用我们的时间和资源。但现在的智能家居的发展还有许多需要注意的问题值得探讨和研究。作为计算机科学与专业的学生，在充分了解当今智能家居的发展状况以后，结合所学习的相关嵌入式的理论与实验知识，通过设计一款智能家居系统不仅仅能够提高我们的工程思想，提高自身自己发现问题解决问题的能力。

同时，通过对该系统的设计，其目的不仅仅是通过此次的系统开发使得我们能够有机会运用到自身所学习到的 STM32 开发板与 IMX6ULL 开发板的相关知识于实践之中，培养自主学习能力、动手能力、团队合作能力以及知识整理能力，培养嵌入式开发人才，使得计算机学科学生能够接触到当今计算机领域的相关前沿技术，更多的是为了使得我们能够接触到在当今全球智能化、信息化的浪潮之下的先进系统的实现过程，为今后的学习或是工作提供了开发经验、并为智能化家居开发贡献出了自己的力量。

通过本课题的学习可以让我们从动手实践中深入理解嵌入式的知识，对嵌入式的整体概念有新的理解，并且从完成的项目中逐渐产生对于嵌入式学习的兴趣。

1.2.2 设计的意义

由摘要概述可知，本系统实现了人体感应、家居自动化、温湿度传感、智能控制以及烟雾检测等功能，通过人体感应与家居自动化，能够解放户主的“双手”，使得家居在特定的情况下做出正确的，有助于户主体验的决策。例如当用户靠近窗户时，窗帘会自动感应用户的位置，并为用户打开窗帘，同时，当用户离开窗户后，本系统能够及时关闭窗帘以防止紫外线从窗户射入房间内。通过温湿度传感器，用户能够及时通过用户界面（QT）查询当前室内环境，并根据室内环境对室内的环境进行调节。近年来家庭火灾频发,危险性也在不断提升,特别是"日渐拔高"的高层、超高层建筑带来了火灾扑救难度的越来越大:一方面,面对中国庞大的家庭数量和严重紧缺的消防警力资源,家庭防火不可能单单依靠消防部门去防治,另一方面,人“百密一疏”的弱点也放大了家居火灾的突发性和危险性,如何能

够将家庭火灾"扼杀在摇篮里"成为了城市消防又一难题。本系统带有烟雾感应功能与房间内温度感应功能，当检测到房间的烟雾浓度超过 1500ppm 时，将及时通过蜂鸣器进行报警提醒用户此时房间内的烟雾浓度过高可能引起火灾，能够充分的警示户主及时进行撤离或进行消防报警。

总的来说，设计本系统的目的，不仅仅能够提高计算机学科人才有关物联网以及嵌入式方面的知识的学习，此外，设置智能家居系统，满足了当今人们对于家庭智能化的追求以及能够提高户主居家安全。把智能家居系统作为初期火灾的最直接工具，无异于是解决家居火灾扑救难题的最佳办法之一。

1.3 项目的主要任务

本实验基于 STM32F407 开发板以及 IMX6ULL 开发平台，在智能控制部分，通过 HCSR04 超声波模块，实时感应用户与窗户之间的距离，并反馈给 STM32F407 开发平台，当人体靠近窗台时（距离<50cm）时，即判断此时用户位于窗台胖，此时开发板将实现通过定时器产生 PWM 波输出来控制 42 步进电机进行转动，从而控制窗帘使窗帘向上卷动从而打开窗帘并改变 LED 灯的状态，当人体远离后，开发板将会控制电机朝相反的方向转动，从而关闭窗帘并打开 LED 灯实现室内照明的效果。此外本系统还要求通过有已校准数字信号输出的温湿度传感器 DHT11 进行室内温度和湿度的实时感应，同时能够通过 MQ2 烟雾传感器进行室内空气中颗粒的检测，MQ-2 型传感器对天然气、液化石油气等烟雾有很高的灵敏度，尤其对烷类烟雾更为敏感具有良好的抗干扰性，可准确排除有刺激性非可燃性烟雾的干扰信息，能够较好的保证室内安全同时又避免了误警的情况。当 STM32F407 开发板通过这两个模块检测到的相关数值，通过 WIFI 模块（STM32 开发板中使用的是 ESP8266，，IMX6ULL 中使用的是 RTL8189）将数据传输到 IMX6ULL 开发板的 QT 界面上供用户实时查看当前房间内的各项数值，同时，用户还可以通过 QT 界面进行家具的控制等

2. 可行性分析

2.1 项目简介

项目“基于 STM32F407 与 IMX6ULL 智能家居系统”是本小组结合当今智能化发展趋势，充分了解当今智能家居系统所要求实现的功能以及了解相关市场需求后所决定进行的开发项目。旨在响应当今智能化号召，充分利用在嵌入式课程上所学习到的相关知识与实验知识，开展嵌入式项目的开发。本项目主要基于 STM32F407 开发板与 IMX6ULL 开发平台，同时利用到了 HCSR04 超声波感应模块、MQ2 烟雾传感模块、ESP8266WIFI 模块以及温湿度模块等来模拟实时检测当前环境的相关信息，并根据检测到的情况进行对应的操作（例如关灯等），本系能够用于企业和家庭的中、低端无线网络产品如无线接入点、无线网关、无线网桥、无线路由器以及无线网卡等设备进行近距离的传输，甚至还可以使用蓝牙等设备，可以在近距离对家居产品进行控制，能够广泛用于各种办公场所。

在本系统的构建中，主要的技术点以及创新点如下：

（1）通过 STM32F407 开发板的定时器产生 PWM 波进行对 42 步进电机的控制。

（2）通过 HCSR04 超声波感应模块实时检测用户与模块之间的距离，根据距离来判断是否要开关窗帘。

（3）通过 MQ2 烟雾模块实时检测房间内烟雾浓度，当浓度大于 1500ppm 时，判断此时房间内烟雾浓度过高，并使用蜂鸣器进行报警。

（4）使用 ESP8266 与 RTL8189 模块（WIFI 模块）实现 STM32F407 与 IMX6ULL 开发板之间的数据传输。

（5）构造 QT 界面，使得用户能通过 IMX6ULL 上的图形化界面实时查看当前环境的相关情况，并能通过界面对 LED 灯等家具进行控制。

（6）能够通过 DHT11 模块实现对房间温湿度的实时检测等。

2.2 技术可行性分析

2.2.1 STM32F407 平台介绍

ALIENTEK 探索者 STM32F4 开发板，资源十分丰富，并把 STM32F407 的内部资源发挥到了极致，基本所有 STM32F407 的内部资源，都可以在此开发板上验证，同时扩充丰富的接口和功能模块，整个开发板显得十分大气。开发板的外形尺寸为 121mm*160mm 大小，板子的设计充分考虑了人性化设计，并结合 ALIENTEK 多年的 STM32 开发板设计经验，同时听取了很多网友以及客户的建议，经过多次改进（面市之前，硬件改版超过 5 次，目前面市版本为 V2.2），最终确定了这样的设计。

ALIENTEK 探索者 STM32F4 开发板的特点包括：

（1） 接口丰富。板子提供十来种标准接口，可以方便的进行各种外设的实验和开发。

（2） 设计灵活。板上很多资源都可以灵活配置，以满足不同条件下的使用。我们引出了除晶振占用的 IO 口外的所有 IO 口，可以极大的方便大家扩展及使用。另外板载一键下载功能，可避免频繁设置 B0、B1 的麻烦，仅通过 1 根 USB 线即可实现 STM32 的开发。

（3） 资源充足。主芯片采用自带 1M 字节 FLASH 的 STM32F407ZGT6，并外扩 1M 字节 SRAM 和 16M 字节 FLASH，满足大内存需求和大数据存储。板载高性能音频编解码芯片、六轴传感器、百兆网卡、光敏传感器以及各种接口芯片，满足各种应用需求。

（4） 人性化设计。各个接口都有丝印标注，且用方框框出，使用起来一目了然；部分常用外设大丝印标出，方便查找；接口位置设计合理，方便顺手。资源搭配合理，物尽其用。

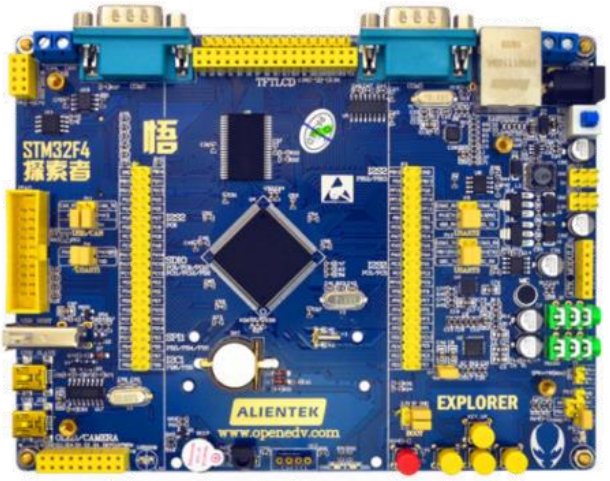


图 2.1 STM32F407 探索者开发板

STM32F407 开发板中使用定时器产生 PWM 波，同时提供了相应的中断源以及串口，

同时还提供了 WIFI 接口与 LCD 显示屏，因此，我们可以充分利用此开发板中所提供的功能完成下位机对设备的控制，通过 STM32F407 开发板，我们可以将各个模块检测到的数据进行汇总并传输至 WIFI 模块，通过 WIFI 模块传输给上位机。同时 STM32F407 所提供的丰富的 GPIO 口（共 144 个）为下位机的驱动控制提供了可能，本项目采用此开发板进行底层应用的开发技术上是可行的。

2.2.2 IMX6ULL 开发平台介绍

阿尔法 imx6ull 开发板是正点原子与 2019 年 10 月发布的一款 Linux 开发板，其主控使用恩智浦公司的 imx6ull 芯片。如下图所示这块为 EMMC BTB 接口核心板，该核心板采用 6 层板设计，其 CPU 型号为 MCIMX6Y2CVM08AB，主频为 800MHz(实际为 792MHz)， BGA289 封装外扩 ddr3L 型号为 NT5CC256M16EP-EK，256MB 字节，工业级，其提供了包括核心的 NAND FLASH 模块、主控芯片 MCIMX6Y2CVM08AB、8GB 的 EMMC 和供以扩展的摄像头模块、以太网接口等，硬件资源非常丰富。

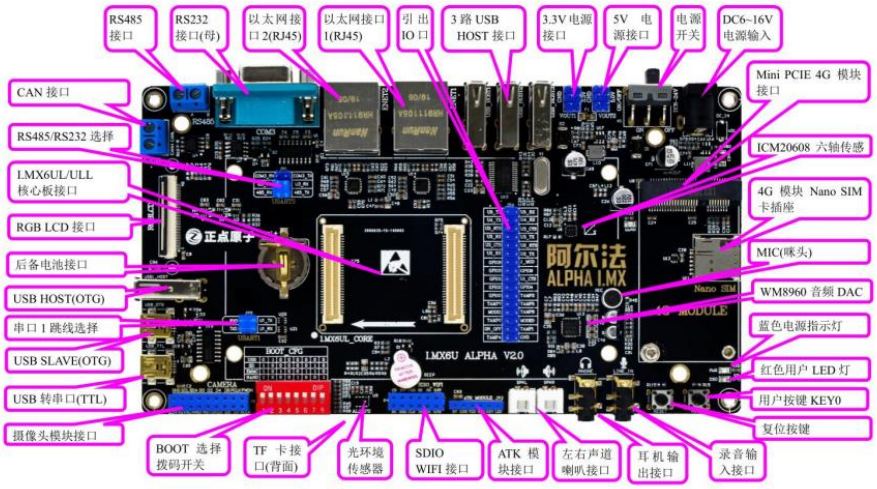


图 2.2 IMX6ULL 开发板

该平台搭载着 Linux 操作系统，在本系统中可以用作 QT 开发界面的搭载平台，经过上课所学习的相关 QT 理论知识以及交叉编译环境的搭载，使用 IMX6ULL 开发平台搭载本系统的人机交互界面是可行的

2.2.3 42 步进电机介绍

在家居系统中，我们要利用驱动来进行窗帘等家居的开关控制，在本系统中，我们可以通过使用 42 步进电机来实现这一功能。所谓步进电机又称为脉冲电机，基于最基本的电磁铁原理，它是一种可以自由回转的电磁铁，其动作原理是依靠气隙磁导的变化来产生电

磁转矩。它接收数字控制信号（电脉冲信号）并转化成与之相对应的角位移或直线位移，它本身就是一个完成数字模式转化的执行元件。而且它可开环位置控制，输入一个脉冲信号就得到一个规定的位置增量，这样的所谓增量位置控制系统与传统的直流控制系统相比，其成本明显减低，几乎不必进行系统调整。所谓 42 步进电机，所谓的 42 步进电机就是指的安装机座尺寸是 42mm 的电机，其最大输出力矩是 0.5NM;57 电机是指安装机座尺寸是 57mm 的电机，其最大输出力矩是 3.0NM。

图 2.3 42 步进电机

步进电机作为一种控制用的特种电机，步进电机无法直接接到直流或交流电源上工作，必须使用专用的驱动电源（步进电机驱动器）。在微电子技术，特别计算机技术发展以前，控制器（脉冲信号发生器）完全由硬件实现，控制系统采用单独的元件或者集成电路组成控制回路，但步进电机在同等价位下相比于舵机，其扭力更大，更适合完成控制家居等产品的功能，这也是我们采用步进电机的原因之一。

图 2.4 42 步进电机驱动器

步进电机具有输出转矩大 ，高转速。电机发热小，噪音低，效率高。高速停止平稳快速，无零速振荡运行平稳，振动噪声小。响应速度快等特点，适合于本系统的窗帘驱动开发。

2.2.4 HC-SRO4 超声波感应模块介绍

在本次系统设计中，我们需要实时的感应人体与“窗户”直接的距离，并实时的反馈给 STM32 开发板，在经过小组的商讨下，我们认为使用 HC-SR04 超声波模块进行人体距离的测量是可行的，其具体工作方法以及相关内容如下：HC-SR04 超声波测距模块可提供 2cm-400cm 的非接触式距离感测功能，测 距精度可达高到 3mm；模块包括超声波发射器、接收器与控制电路。

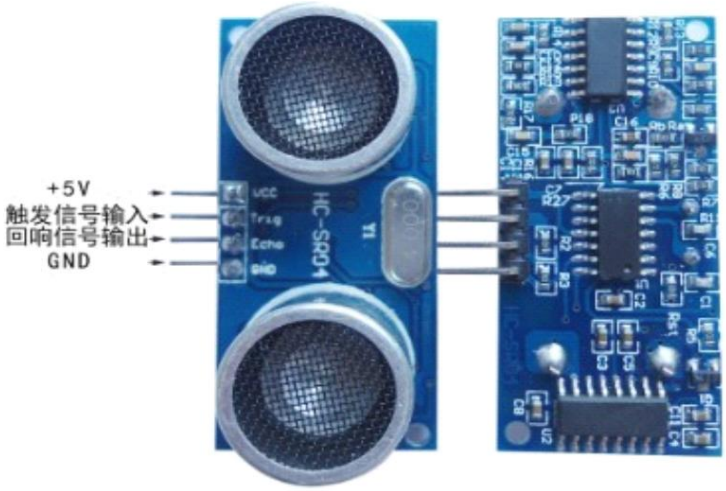


图 2.5 HC-SR04 超声波模块

其基本的工作原理为：

(1)采用 IO 口 TRIG 触发测距，给最少 10us 的高电平信呈。

(2)模块自动发送 8 个 40khz 的方波，自动检测是否有信号返回；

(3)有信号返回，通过 IO 口 ECHO 输出一个高电平，高电平持续的时间就是超声波从发射到返回的时间。测试距离=(高电平时间*声速(340M/S))/2;

表 2-1 HC-SR04 电平参数

电气参数	HC-SR04 模块
工作电压	DC 5V
工作电流	15mA
工作频率	40kHz
最远射程	4m
最近射程	2cm
测量角度	15 度
输入触发信号	10uS 的 TTL 脉冲
输出回响信号	输出 TTL 电平信号，与射程成比例
规格尺寸	45*20*15mm

因此，采用超声波模块进行人体距离的测量是可行的，能充分的满足本系统的需求，较为精准的感应人体距离。

2.2.5 MQ2 烟雾传感模块介绍

本系统设计中，我们需要对室内的空气环境进行实时的监控，以模拟防止家庭火灾给用户带来生命与财产的损失，在本小组查阅相关资料以后，决定采用 MQ2 烟雾传感模块进行空气中颗粒物的实时检测。MQ2 对液化气、天然气以及城市煤气有较为良好的灵敏度，具有长期的使用寿命和良好的稳定性，且其响应速度较为灵敏，通常适用于家庭或工厂的气体泄漏监测装置，适宜于液化气、丁烷、丙烷、甲烷、酒精、氢气、烟雾等监测装置。

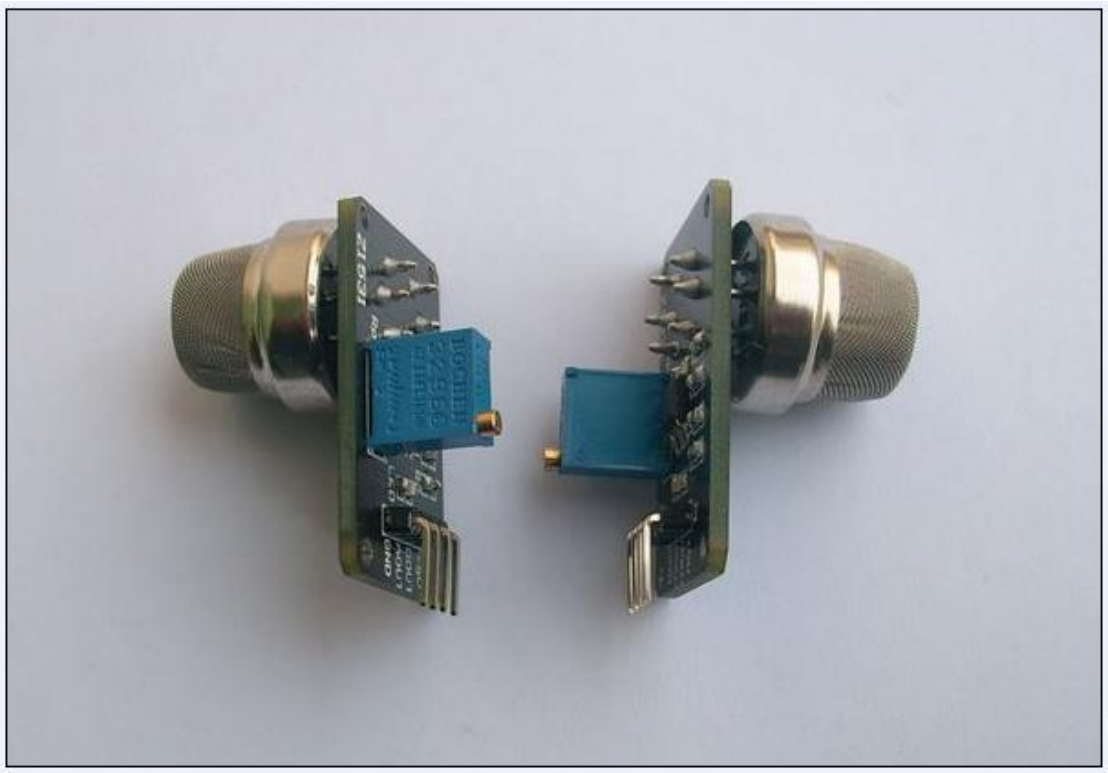


图 2.6 MQ2 烟雾传感器

2.2.6 ESP8266 以及 RTL8189WIFI 模块介绍

在本系统中，为了使得房间数据可视化同时为了方便用户能够实时的查询当前环境的相关数值，团队需要将 STM32 下位机上检测到的有关烟雾、温度、房间湿度等相关数据反馈给 IMX6ULL 上位机程序（QT），在项目之初，本团队尝试使用了蓝牙模块进行两个平台之前的数据通信，但因为在上位机 IMX6ULL 平台中对蓝牙模块具有较高的要求，需要在 LINUX 环境下的免驱动蓝牙，而与 STM32 平台上使用的蓝牙存在蓝牙协议栈不匹配等问题，因此，在团队的讨论下，最终决定采用 WIFI 模块进行数据的传输。

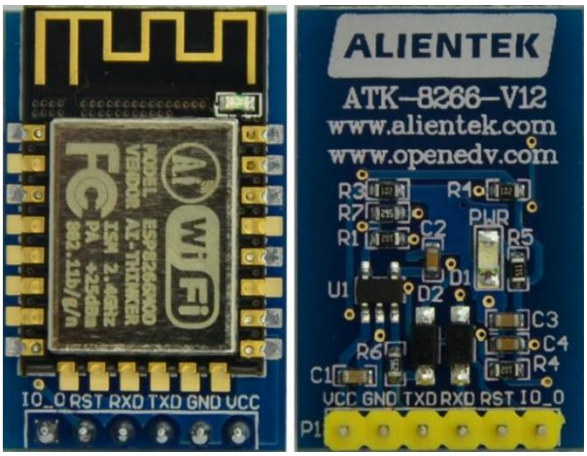


图 2.7 ESP8266WIFI 模块

ESP8266 是一个完整且自成体系的 Wi-Fi 网络解决方案，能够搭载软件应用，或通过另一个应用处理器卸载所有 Wi-Fi 网络功能。因此，在使用 WIFI 模块进行数据传输时，我们只需要保证此时两个开发平台处于同一网关内，再利用我们在嵌入式实验课程上所学习的 socket 通信相关的编程知识，就能够绕开复杂的协议栈，从而实现数据的传输。

这里值得一提的是，在 IMX6ULL 上我们采用的是不同的 WIFI 模块，我们在此平台上所采用的是 RTL8189WIFI 模块，虽然采用不同的硬件模块，但是经过测试，发现两者之间能够成功进行数据的通信。因此，采用 WIFI 代替蓝牙模块来实现开发平台之间的通信是可行的。小组成员采用 WIFI 模块成功完成本次系统的数据传输服务。

图 2.8 RTL8189WIFI 模块

2.2.7 DHT11 温湿度传感器

由概述可知，我们要监测房间内的温湿度并及时的反馈给用户，在温湿度模块我们可以采用 DHT11 温湿度传感器模块。CJSLDHT11 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。传感器包括一个电阻式感湿元件和一个 NTC 测温元件，并与一个高性能 8 位单片机相连接。因此该产品具有品质卓越、超快响应、抗干扰能力强、性价比极高等优点。每个 CJSLDHT11 传感器都在极为精确的湿度校验室中进行校准。校准系数以程序的形式储存在 OTP 内存中，传感器内部在检测信号的处理过程中要调 用这些校准系数。单线制串行接口，使系统集成变得简易快捷。超小的体积、极低的

功耗，且可长距离通讯，使其成为各类应用甚至最为苛刻的应用场合的最佳选则。产品为 4 针单排引脚封装。连接方便，特殊封装形式可根据用户需求而提供。

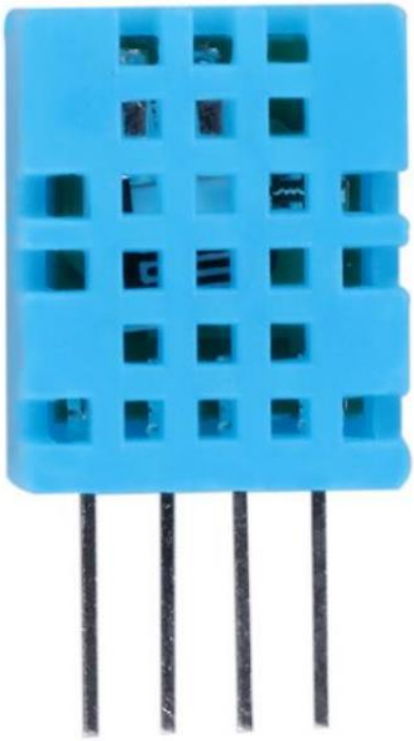


图 2.9 DHT11 模块

该器件成本低、长期稳定、品质卓越、超快响应、抗干扰能力强、超长的信号传输距离、数字信号输出、精确校准。一般用于暖通空调、除湿器、测试及检测设备、消费品、汽车、自动控制、数据记录器、气象站、家电、湿度调节器、 医疗、其他相关湿度检测控制。

2.2.8 QT 界面设计

为了满足并提高人机交互功能，提高用户体验，本设计组在结合在日常所学习的知识后，决定采用 QT 界面来实时反馈给用户房间内当前数值，并提供给用户操作的接口，QT 界面设计如下：

图 2.10 QT 界面设计

图 2.11 QT 界面设计

2.3 经济可行性分析

在本次项目运行之前，团队成员还对本次项目开发所需要的模块进行了经济评估，在查阅了有关商品的价格以后以及不同模块之间性能的比较之后，确定使用以上介绍的模块，预算总共控制在百元以内，符合本系统以低成本开发的初衷，并产生了高于成本的开发价值，符合项目开发的要求与目的，达到了实训的基本要求。

3. 系统总体设计

本系统总共可分为上位机、下位机两个模块。下位机模块主要以 STM32 为核心模块，作为其他各个模块之间的调度以及数据的接受，以及发送有关控制命令等等。而上位机模块以 IMX6ULL 开发板为核心，通过接收由下位机传送过来的数据并通过 QT 人机交互界面进行显示，提高本系统的人机交互功能，同时，用户还可以同 QT 界面发送有关命令，从而控制下位机发送相关命令，从而控制各个功能模块。

3.1 下位机模块介绍

下位机模块以 32 为核心，同时还连接了本系统中所有的感应模块，我们使用 32 开发板作为下位机的核心，接受来自各个模块的数据，STM32F407 会对来自超声波模块的数据进行监测，当超声波模块通过代码实现计算出有物体与模块之间的距离小于 15cm 时，我们判断此时需要通过 STM32F407 开发板通过定时器 14 产生 PWM 波，发送脉冲控制步进电机驱动器从而进一步控制电机的转动，同时，还将通过 DIR 信号通过电机的驱动器从而控制电机转动的方向，从而实现窗帘的打开与关闭。

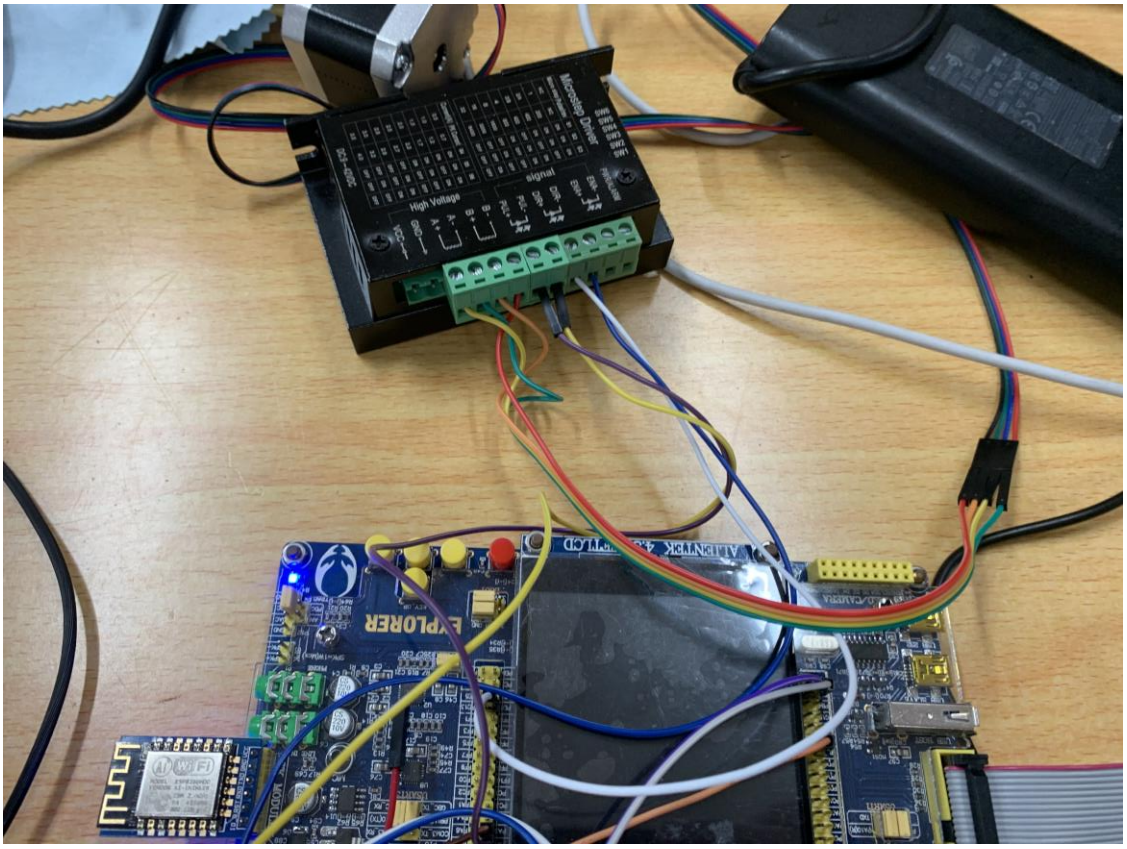


图 3.1 42 步进电机与开发板的连接

除此以外，下位机还具有烟雾报警系统，由图可知，我们通过 MQ2 烟雾感应模块与 STM32F407 的连接，当烟雾感应模块通过模数转换后反馈回来的值此时就是目前空气中颗粒的浓度，当颗粒浓度大于 1500ppm 时，我们通过 STM32F407 判断此时房间内的烟雾或颗粒浓度过高，并通过使能蜂鸣器进行持续的报警，从而实现本系统在消防安全方面的报警功能。

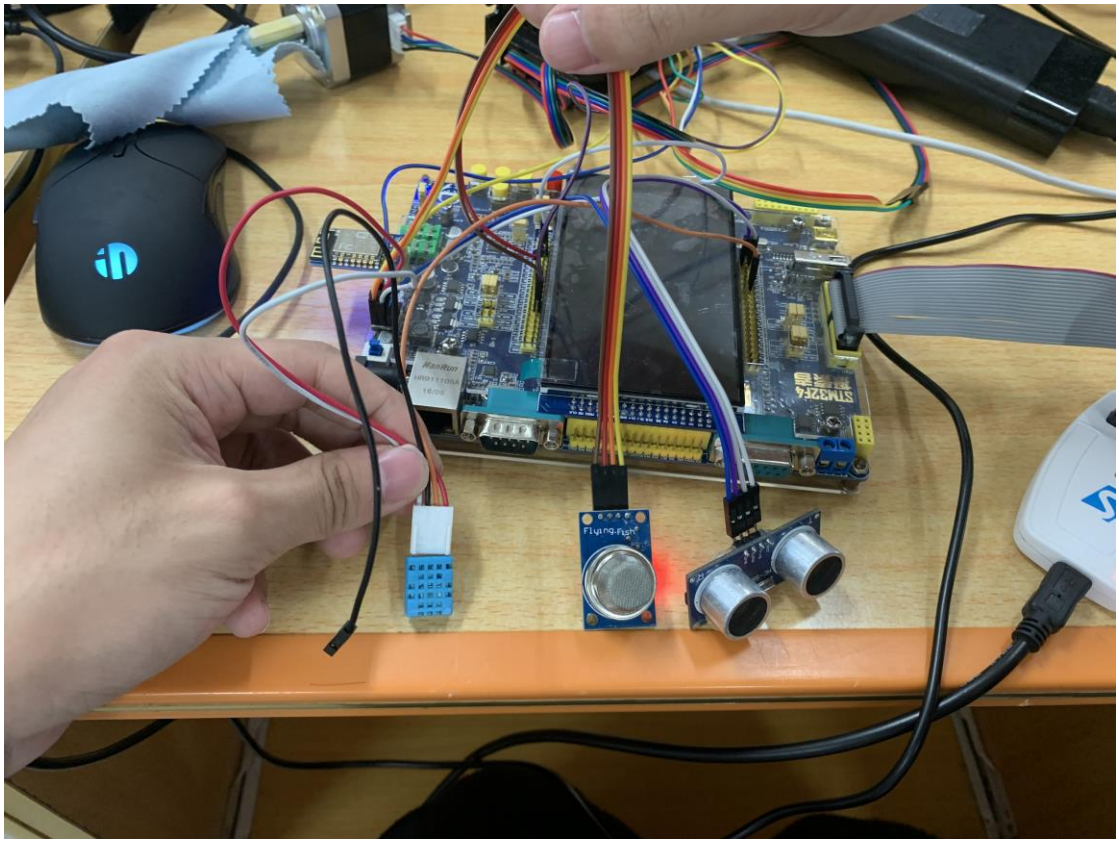


图 3.2 烟雾模块与开发板的连接

下位机还设有温湿度传感器，该传感器能够实时监测当前室内环境的温度以及湿度，我们通过将数据传输给下位机核心板后通过 WIFI 的传输模块，将数据反馈给上位机即可，总的来说，下位机以 STM32F407 为核心，进行数据或控制命令的接受与发送，并对家居的各个方向进行控制，并把数据反馈给上位机使得用户能够直观的查询到有关的数据，甚至通过接受上位机所发送过来的字符命令，识别后对相对应的模块进行控制等等。

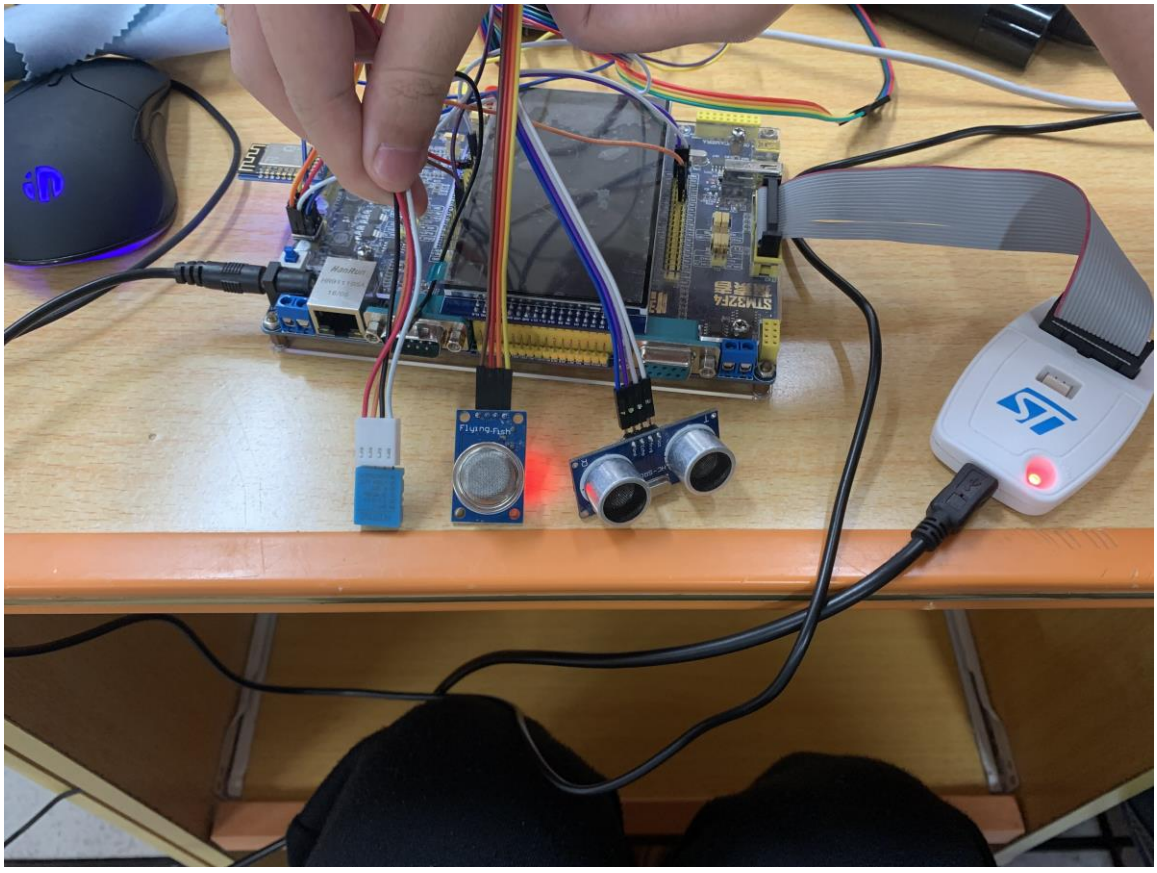


图 3.3 温湿度模块与 WIFI 模块

3.2 上位机介绍

上位机模块主要用于构建良好的用户交互界面，在上位机模块中，我们以 IMX6ULL 模块为核心，在 Ubuntu 上构造交叉编译环境，编写 Qt 代码，为用户创建一个良好的人机交互界面，在本 Qt 界面中，用户不仅可以查阅当前时间等，还可以实时查阅当前环境下的温湿度模块以及当前环境的空气质量。不仅如此，我们还增设了控制功能，用户能够通过 Qt 界面上的点灯程序，进行家中开灯与关灯。因为实训时间较为紧迫，我们在一期的开发中，基本已经实现所有的基本功能点，但我们从 Qt 界面可以看出，此系统还可以进行更多模式的开发，相信在今后的开发中，我们将不断的完善我们的系统。

图 3.4 Qt 界面

不仅如此，本团队还在系统的开发过程中，尝试通过搭建安卓的编译环境，并成功在手机上进行了移植，同样也能实现有关功能，但是为了演示时更为直观与清晰，我们最终还是采用了 IMX6ULL 作为 QT 界面的搭建板。

4. 系统详细设计

4.1 步进电机驱动模块

4.1.1 电机工作的基本原理

步进电机驱动模块是本次系统开发中较为重要的一个模块之一，步进电机的驱动，涉及到定时器中断、PWM 脉冲输出以及控制信号的输出等，因此，我们首先应该简单的了解一下步进电机的工作原理：42 步进电机为一四相步进电机，采用单极性直流电源供电。只要对步进电机的各相绕组按合适的时序通电，就能使步进电机步进转动。下图是该四相反应式步进电机工作原理示意图。

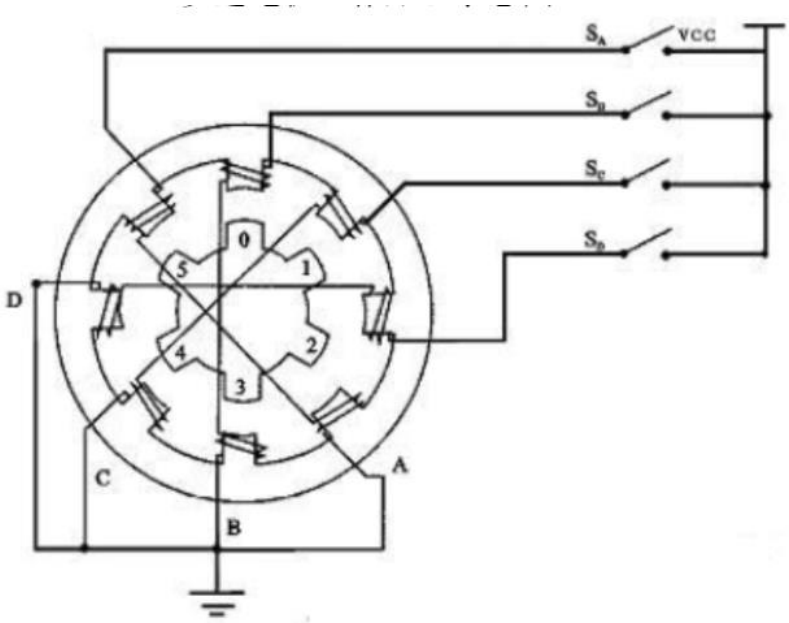


图 4.1 步进电机工作原理图

开始时，开关 SB 接通电源，SA、SC、SD 断开，B 相磁极和转子 0、3 号齿对齐，同时，转子的 1、4 号齿就和 C、D 相绕组磁极产生错齿，2、5 号齿就和 D、A 相绕组磁极产生错齿。

当开关 SC 接通电源，SB、SA、SD 断开时，由于 C 相绕组的磁力线和 1、4 号齿之间磁力线的作用，使转子转动，1、4 号齿和 C 相绕组的磁极对齐。而 0、3 号齿和 A、B 相绕组产生错齿，2、5 号齿就和 A、D 相绕组磁极产生错齿。依次类推，A、B、C、D 四相绕组轮流供电，则转子会沿着 A、B、C、D 方向转动。

四相步进电机按照通电顺序的不同，可分为单四拍、双四拍、八拍三种工作方式。单

四拍与双四拍的步距角相等，但单四拍的转动力矩小。八拍工作方式的步距角是单四拍与双四拍的一半，因此，八拍工作方式既可以保持较高的转动力矩又可以提高控制精度。

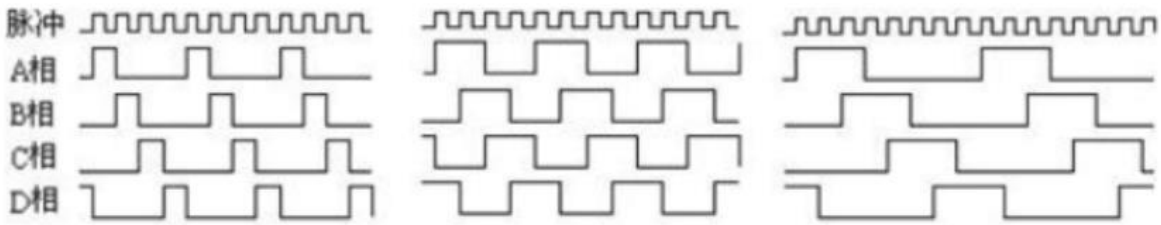


图 4.2 不同拍的工作脉冲

步进电机必须有驱动器与控制器才能够进行正产的工作，驱动器的作用是对脉冲进行控制，在我们 STM32F407 的开发板上，单个 IO 口的输出或输入电平一般不超过 5V，这对于想要控制电机的转动是远远不够的，驱动器能够将控制脉冲进行环形的分配并放大功率，使得步进电机绕组按照一定的顺序进行通电，从而控制电机进行转动。其中，电机与电机的驱动器的连接原理图如下图所示：

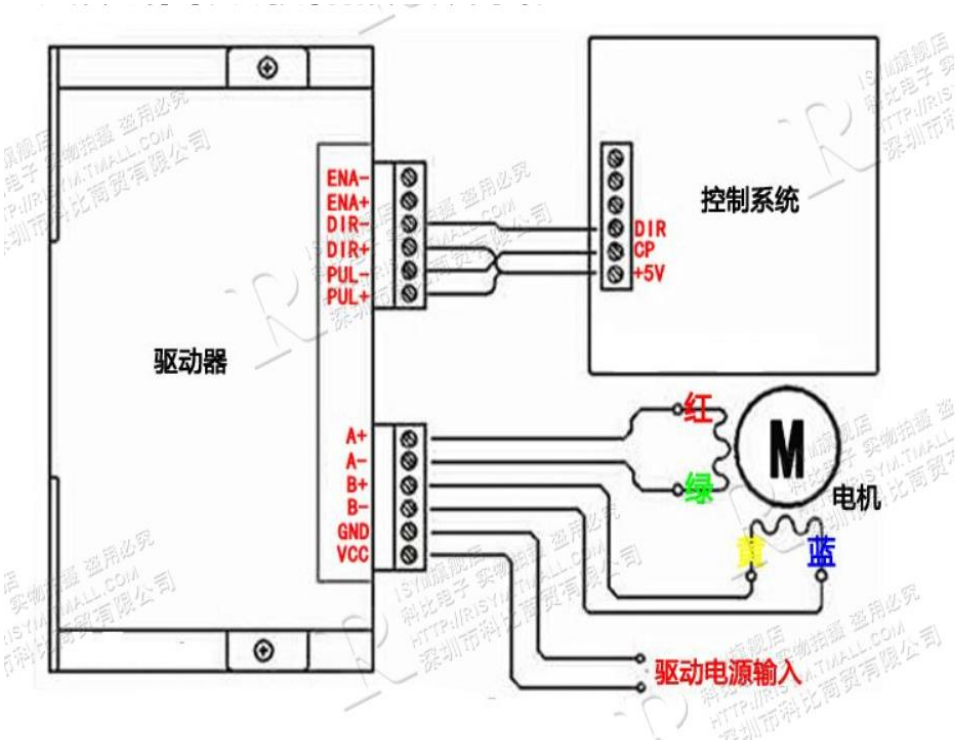


图 4.3 电机及驱动器连接原理图

在我们使用电机时，我们要充分了解相关术语所包含的意思，这样才能充分的利用好电机的开发

（1）相数:产生不同对极 N、S 磁场的励磁线圈对数。

（2）步距角:对应一个脉冲信号，电机转子转过的角位移。一般二相电机的步距角为

1.8 度，即电机运动 200 步为一周。

（3）静力矩（(HOLDING TORQUE):是指步进电机通电但没有转动时，定子锁住转子的力矩。它是步进电机最重要的参数之一，通常步进电机在低速时的力矩接近于静力矩。

（4）定位力矩（DETENT TORQUE) :是指步进电机没有通电的情况下，定子锁住转子的力矩。步距角精度:步进电机每转过一个步距角的实际值与理论值的误差。用百分比表示:(误差 l 步距角）×100%。步进角的误差不累积。

（5）最大空载起动频率:是指电机在某种驱动形式，电压以及额定电流下，电机不带负载的最高的运行频率

4.1.2 电机及驱动的连接

电机与驱动器的连接较为复杂，两者之间需要通过复杂的线路连接才能实现电机的正常工作，因此，在电机与驱动连接之前，我们需要对电机驱动器上的不同的开关引脚有一定的了解：

（1）信号的输入端：

表 4-1 电机驱动器输入端

PUL+：脉冲信号输入正。	PUL-：脉冲信号输入负。
DIR+ ：电机正、反转控制正。	DIR- ：电机正、反转控制负。
EN+ ：电机脱机控制正。	EN-：电机脱机控制负。

（2）电机线连接段：

表 4-2 电机线连接段

A+：连接电机绕组 A+相	A-：连接电机绕组 A-相。
B+：连接电机绕组 B+相。	B-：连接电机绕组 B-相。

（3）电源电压连接：

表 4-3 电源电压连接

VCC：直流电源正端“+”	GND：直流电源负端“-”

通过上述表格中的各个接口的介绍，我们便可以进行驱动器与电机之间的连接了，在本次系统的设计中，我们采用的是共阴极的接法，其接法如下：分别将 PUL-，DIR-，EN-

连接到控制系统的地端； 脉冲输入信号通过 PUL+接入，方向信号通过 DIR+接入，使能信号通过 EN+接 入。若需限流电阻，限流电阻 R 的接法取值与共阳极接法相同。如下图：

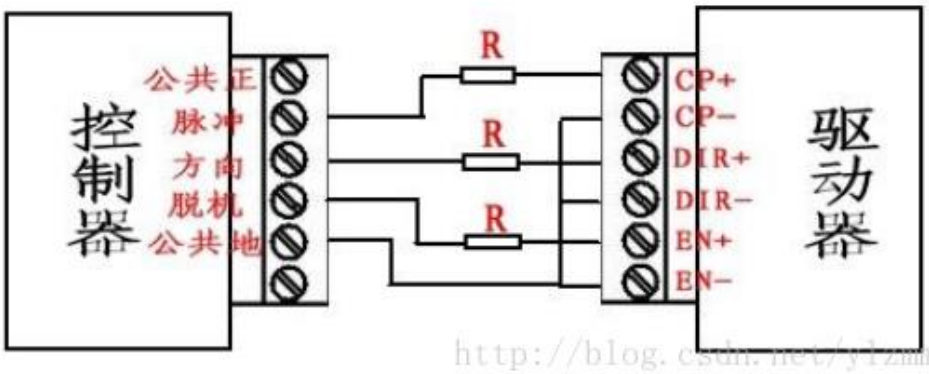


图 4.4 控制信号共阴极接法

随后，当我们再连接上电源以后，通过 STM32F407 开发板时钟发送的 PWM 波脉冲，就可以看到电机在步进电机驱动器的驱动下，实现旋转的效果，同时我们还能够通过控制 DIR 接口端来控制电机的正转与反转的效果。

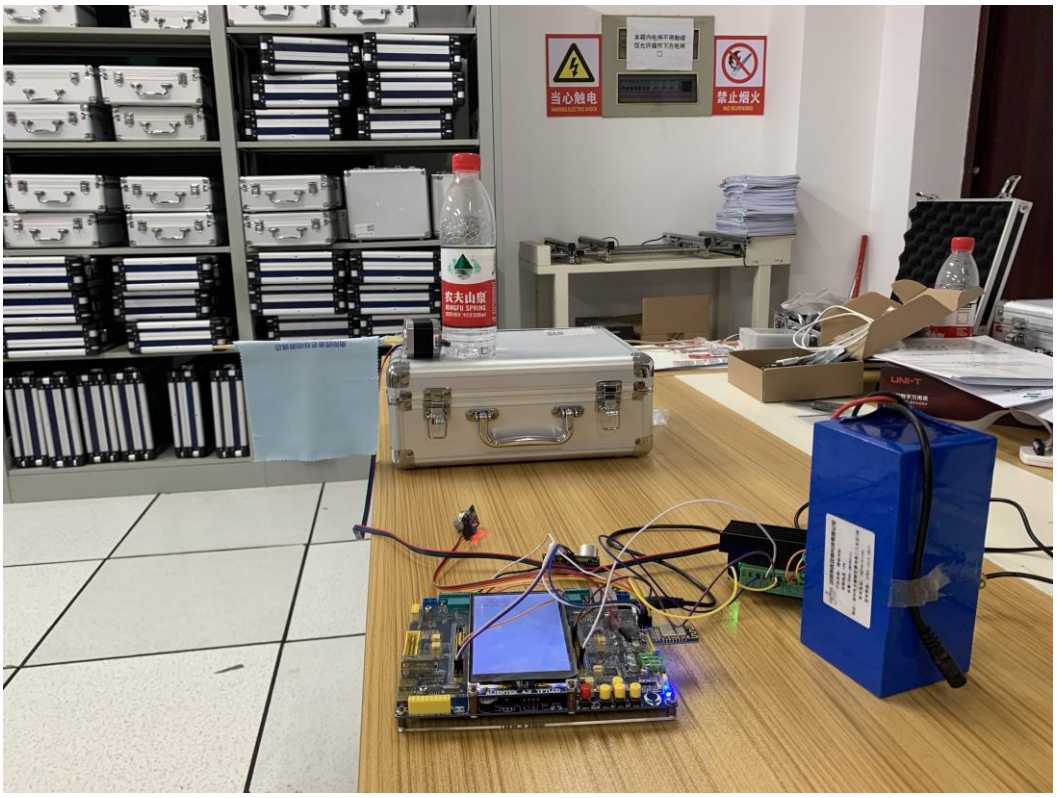


图 4.5 电机转动效果图

4.1.3 代码驱动实现

在此之前，我们首先要对 STM32F407 开发板中的 PWM 的工作原理有一个简单的了解，佑图可知，当我们的定时器在计数的过程中，不断与 CCRX 进行比较，当小于 CCRX 所定的值时，产生低电平的脉冲，仅当其超过该计数值时，才产生高电平，因此，我们便可以通过改变 CCRx 的值来控制脉冲的占空比，当有效电平的占空比越高，电机的转速就会越快，当有效占空比越小，则电机的转动速度将会越慢。

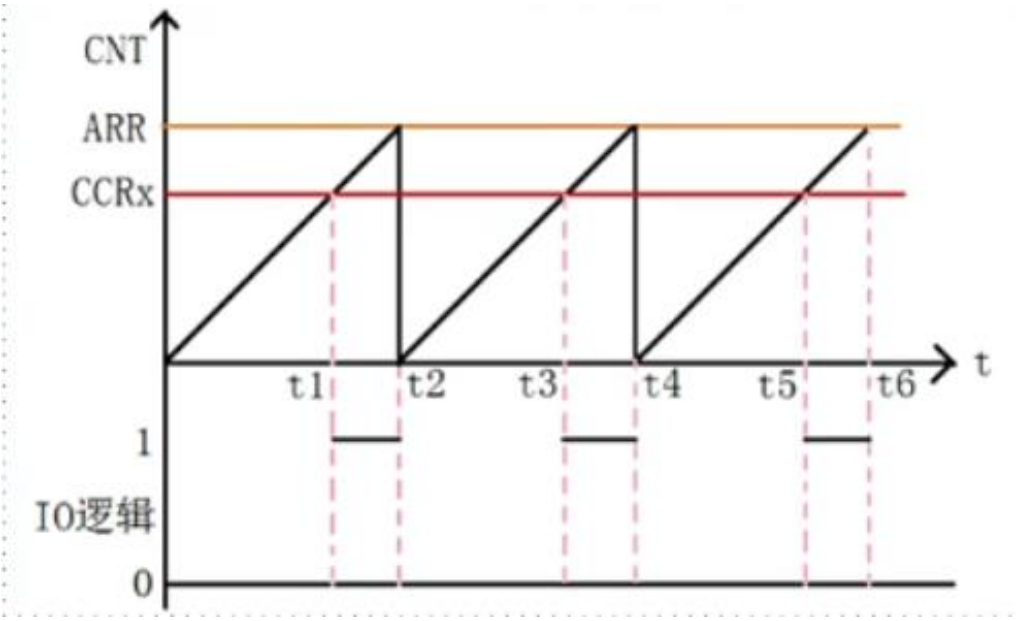


图 4.6 STM32 PWM 的工作过程

那 PWM 的脉冲输出又是如何实现的呢？由下图可知，此处的核心为输出模式控制器，根据上述的分析产生不同的电平信号，同时可以通过 OCCM[2:0]中的 0-2 位进行 PWM 模式的选定，从而发出有效脉冲发送给使能电路。

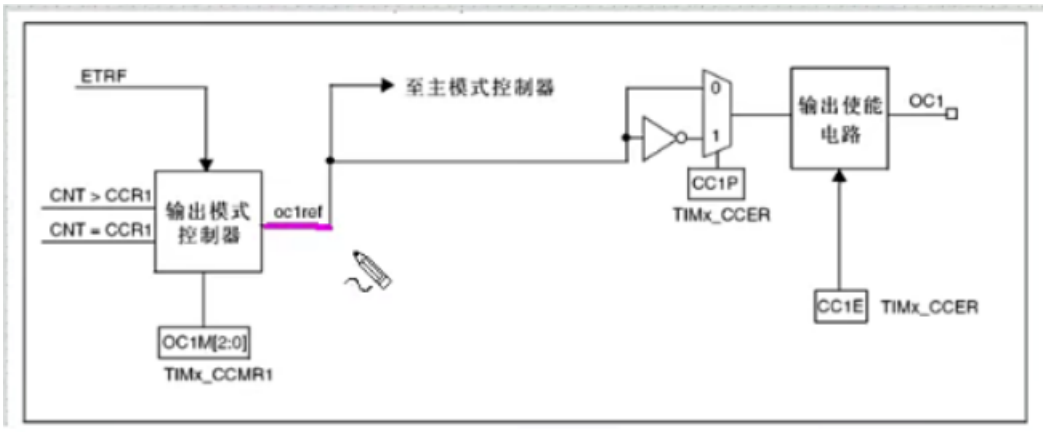


图 4.7 STM32 PWM 的产生过程

随后，我们将对脉冲的产生进行初始化操作，查阅 STM32F407 中文参考手册可知，此

处我们采用 A7 口复用为定时器 14，采用定时器的通道 1 来进行 PWM 波的输出，将 PA7 口模式设置成复用功能，输入与输出方式为推挽复用输出，同时对定时器 TIME14 进行初始化设置定时器分频，同时采用向上计数的模式，并设置重装载值为 arr（用户设置），并进行定时器的初始化。

```
GPIO_PinAFConfig(GPIOA,GPIO_PinSource7,GPIO_AF_TIM14); //GPIOA7复用为定时器14

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;           //GPIOF9
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;        //复用功能
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  //速度100MHz
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      //推挽复用输出
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;        //上拉
GPIO_Init(GPIOA,&GPIO_InitStructure);              //初始化PF9

TIM_TimeBaseStructure.TIM_Prescaler=psc;  //定时器分频
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数模式
TIM_TimeBaseStructure.TIM_Period=arr;   //自动重装载值
TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;

TIM_TimeBaseInit(TIM14,&TIM_TimeBaseStructure);//初始化定时器14
```

图 4.8 定时器及 IO 口初始化编程

进行 IO 口以及定时器的初始化后，我们就要对产生的 PWM 脉冲进行初始化的配置，首先我们将定时器的模式设置成 PWM 模式 1，并使能比较输出，输出有效电平设置成低极性，并通过 TIM_OC1PreloadConfig 使能 TIM14 在 CCR1 上的预装载进村器。

```
//初始化TIM14 Channel1 PWM模式
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //选择定时器模式:TIM脉冲宽度调制模式2
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较输出使能
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low; //输出极性:TIM输出比较极性低
TIM_OC1Init(TIM14, &TIM_OCInitStructure);  //根据T指定的参数初始化外设TIM1 4OC1 通道1

TIM_OC1PreloadConfig(TIM14, TIM_OCPreload_Enable);  //使能TIM14在CCR1上的预装载寄存器

TIM_ARRPreloadConfig(TIM14,ENABLE);//ARPE使能

TIM_Cmd(TIM14, ENABLE);  //使能TIM14
```

图 4.9 PWM 初始化编程

在主函数中，我们要对重装载值进行初始化，由下图所示，在嵌入式的学习中我们可知：STM32 中的是中的频率为 84M，因此 84M/84=1Mhz 的计数频率，同时因为技术周期为频率倒数为 1us，因此这里定义的重装载值的时间长度为 2ms

```
TIM14_PWM_Init(2000-1,84-1);
```

图 4.10 ARR 初始化

此外，我们还定义了一个 DIR 也就是步进电机旋转方向的控制端，其初始化操作较为简单，仅需要把 PE4IO 口初始化为输出模式。

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);//使能G

//GPIOF9,F10初始化设置
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;//普通输出模式
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;//推挽输出
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;//100MHz
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;//上拉
GPIO_Init(GPIOA, &GPIO_InitStructure);//初始化
```

图 4.11 DIR 控制信号

下图是主函数逻辑实现，本函数实现的主要功能为，当距离小于 15cm，即 a<15 时，电机将会通过函数 TIM_SetCompare1(TIM14,led1pwmval),设置 CCR 的值同时使能 TIM14，设置占空比，从而使的能够产生 PWM 的脉冲，从而控制电机进行转动，进而控制窗帘的转动，同时，在转动一定的时间以后，将会把 CCR 的值设置为 0，也就是不产生脉冲，此时，电机转动将会停止。同时将 flag 为置 1，防止窗帘在已经打开的情况下再次进入该状态，导致窗帘进行空转的情况。

```
if(flag1==0&&a<15)
{
  while(1){
  flag2=0;
  delay_ms(10);
  LED0=0;
  LED1=0;
  delay_ms(num);
  DIR=1;
  flag++;
    number=18000/num;
  if(flag>=number){
    flag1=1;
    delay_ms(num);
  TIM_SetCompare1(TIM14,0);
    break;
  }
else
  {
    TIM_SetCompare1(TIM14,led1pwmval);
    delay_ms(10);
  }
}
}
```

图 4.11 开窗控制函数

下图是关窗函数逻辑实现，本函数实现的主要功能为，当距离大于 15cm，即 a>15 时，电机将会通过函数 TIM_SetCompare1(TIM14,led1pwmval),设置 CCR 的值同时使能 TIM14，设置占空比，从而使的能够产生 PWM 的脉冲，从而控制电机进行转动，进而控制窗帘的转动，同时，在转动一定的时间以后，将会把 CCR 的值设置为 0，也就是不产生脉冲，此时，电机转动将会停止。同时将 flag 为置 0，防止窗帘在已经关闭的情况下再次进入该状态，导致窗帘进行空转的情况。由两段程序可以看出，其 DIR 的值不同，此时通过 DIR 的值进行电机转动方向的控制。

```
if(flag1==1&&a>15)
  {
    while(1)
    {
      flag=0;
      delay_ms(1000);
      LED0=1;
      LED1=1;
      DIR=0;
      flag2++;
      if(flag2>=19){
        flag1=0;
        TIM_SetCompare1(TIM14,0);
        break;
      }
      else
      {
        TIM_SetCompare1(TIM14,led1pwmval);
        delay_ms(10);

      }

  }
```

图 4.11 关窗控制函数

当我们编译通过后，通过 STlink 将程序烧录到 STM32F407 开发板中，同时连接好驱动器与开发板、驱动器与电机之后，我们就可以发现电机在 DIR 以及脉冲信号的控制下就可以实现电机的转动了。

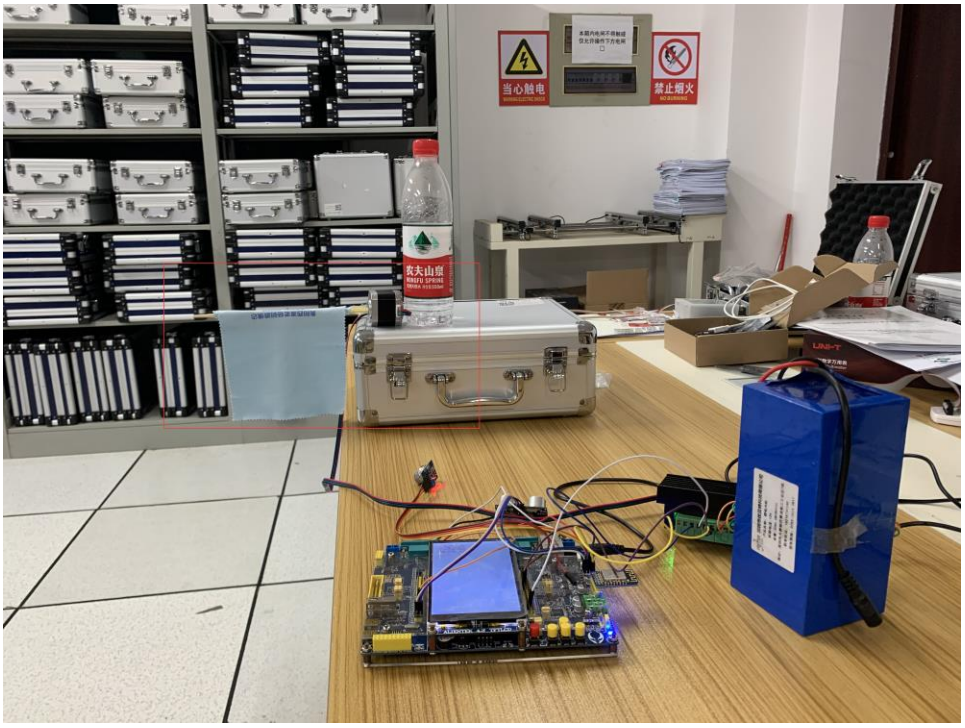


图 4.12 电机转动

总的来说，电机驱动模块在第一次接触时可能会有一定的难度，但是通过 STM32 官方的文档以及正点原子官方视频的学习，同时通过自己在网上所查阅的相关资料，在充分了解电机驱动模块原理以后，我们就很容易进行电机模块的开发了。

4.2 HC-SR04 超声波模块

4.2.1 HC-SR04 超声波工作基本原理

HC-SR04 超声波测距模块可提供 2cm-400cm 的非接触式距离感测功能，测距精度可达高到 3mm；模块包括超声波发射器、接收器与控制电路。其基本的工作原理如下：采用 IO 口 TRIG 触发测距，给最少 10us 的高电平信呈，同时，模块自动发送 8 个 40khz 的方波，自动检测是否有信号返回；当检测到有信号进行返回时，该模块将通过 IO 口想 ECHO 输出一个高电平，在此期间会启动定时器时钟进行定时，高电平持续的时间就是超声波从发射到返回的时间，我们可以通过测试距离计算公式测试距离=(高电平时间*声速(340M/S))/2 从而计算出物体距离该模块的距离。

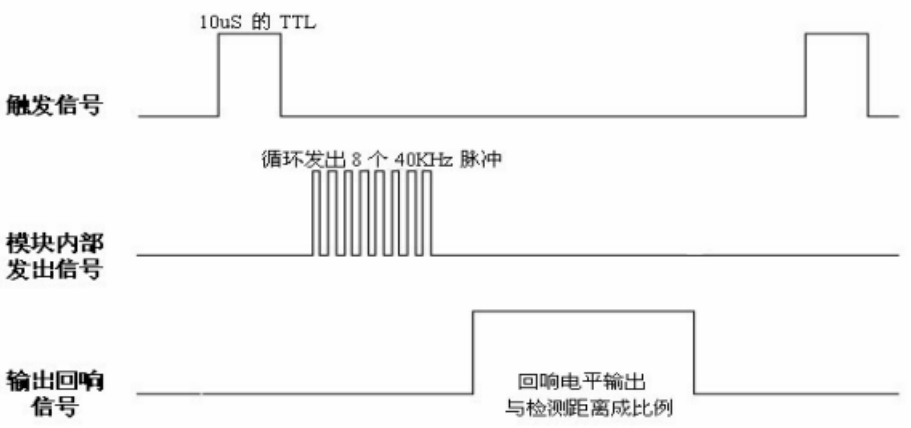


图 4.13 超声波时序图

以上时序图表明你只需要提供一个 10uS 以上脉冲触发信号，该模块内部将发出 8 个 40kHz 周期电平并检测回波。一旦检测到有回波信号则输出回响信号。回响信号的脉冲宽度与所测的距离成正比。由此通过发射信号到收到的回响信号 时间间隔可以计算得到距离。公式：uS/58=厘米或者 uS/148=英寸；或是：距离=高电平时间*声速（340M/S）/2；建议测量周期为 60ms 以上，以防止发射信号对回响信号的影响。

4.2.2 超声波代码实现

在超声波模块代码初始化过程中，由 4.1.1 节超声波实现原理可知，当我们使用超声波进行测距时，需要测量高电平所持续的时间，因此，在程序中我们要使用到定时器来进行计时，因此，我们不仅仅要对对应的 GPIO 口进行初始化，更多的我们需要对相应的定时器进行初始化。

```
//超声波模块初始化
void HCSR04Init(){
  //设置gpio端口
  GPIO_InitTypeDef GPIO_InitStructure;
  RCC_AHB1PeriphClockCmd(HCSR04_CLK,ENABLE);
  GPIO_InitStructure.GPIO_Pin=HCSR04_TRIG;
  GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
  GPIO_Init(GPIOB,&GPIO_InitStructure);
  GPIO_ResetBits(GPIOB,HCSR04_TRIG);

  GPIO_InitStructure.GPIO_Pin=HCSR04_ECHO;
  GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
  GPIO_Init(GPIOB,&GPIO_InitStructure);
  GPIO_ResetBits(GPIOB,HCSR04_ECHO);
  //定时器配置，使用通用定时器3通道3，，引脚PB0
  TIM3_Int_Init();
```

图 4.14 超声波初始化代码

此处为了防止不同之间的 IO 进行重复定义，因此，在这里我们将 IO 口赋值在了变量之中，在这里我们初始化的 IO 口是 PB6 与 PB7 口，由原理分析可知，我们将 PB6 口作为脉冲发送的函数，因此在此处函数的工作模式为输出模式，PB7 口作为接受返回的电磁脉冲的接收口，因此此时设置为输入模式。

```
1  #ifndef __HCSR04_H
2  #define __HCSR04_H
3  #include "sys.h"
4
5
6  #define HCSR04_PORT GPIOB
7  #define HCSR04_CLK RCC_AHB1Periph_GPIOB
8  #define HCSR04_TRIG GPIO_Pin_6
9  #define HCSR04_ECHO GPIO_Pin_7
10
11 #define TRIG_Send PBout(6)
12 #define ECHO_Reci PBin(7)
13
14 double GetHCSR04_Distance(void);
15 void HCSR04Init(void);
16 void TIM3_Int_Init(void);
17
18 #endif
19
```

图 4.15 超声波初始化所对应的引脚

随后我们要对定时器进行初始化配置，此处我们进行定时器 3 的初始化，同时将定时

器 3 的重装载值设置成 1000 并允许定时器产生中断。

```
void TIM3_Int_Init()
{
  TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);  ///使能TIM3时钟

  TIM_DeInit(TIM3);
  TIM_TimeBaseInitStructure.TIM_Period = (1000-1);  //自动重装载值
  TIM_TimeBaseInitStructure.TIM_Prescaler=(84-1);  //定时器分频      计数频率1MHz,,计数一次为1us,,计数1000次溢出，计数时间1ms
  TIM_TimeBaseInitStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数模式
  TIM_TimeBaseInitStructure.TIM_ClockDivision=TIM_CKD_DIV1;
  TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStructure);//初始化TIM3

  TIM_ClearFlag(TIM3,TIM_FLAG_Update);
  TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE); //允许定时器3更新中断
  TIM_Cmd(TIM3,DISABLE); //使能定时器3

  NVIC_InitStructure.NVIC_IRQChannel=TIM3_IRQn; //定时器3中断
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0x01; //抢占优先级1
  NVIC_InitStructure.NVIC_IRQChannelSubPriority=0x03; //子优先级3
  NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

图 4.16 定时器初始化

在完成了定时器的基本初始化步骤以后，我们根据 HC-SR04 超声波模块的基本工作原理就可以进行获取距离函数的编写，由原理分析部分我们你可以知道，超声波测距的基本原理是通过脉冲的往返时间来计算出与物体或人体之间的距离，因此，利用这一原理我们可以较为轻松的编写出测量距离的代码，同时，我们也可以参考该模块所提供的距离测量代码实现距离的测量。

```
//获取当前的距离
double GetHCSR04_Distance(){
  int i=0;
  uint32_t t=0;
  double lengthtemp=0;
  double sum=0;
  while(i!=5){
    TRIG_Send=1;
    delay_us(20);
    TRIG_Send=0;
    while(ECHO_Reci==0);
    OpenTimerForHC();
    i++;
    while(ECHO_Reci==1);
    CloseTimerForHC();
    t=GetHCSR04Timer();
    lengthtemp=((double)t/58.0);//得到距离预设值
    sum=sum+lengthtemp;
  }
  lengthtemp=sum/5;
  return lengthtemp;
}
```

图 4.16 HC-SR04 超声波模块测距

结合源码与 HC-SR04 超声波模块测距的原理，我们能够较快的理解这份源码所实现功能的步骤，首先将 TRIG_Send 设置成状态 1，使得模块发出电磁波，在延时 20us 后设置成

状态 0，此时表明发出的电磁波持续了 20us，表明此时不再进行发送，此时接收处将持续高电平，此时开启接受检查，并开启定时器，接收处 IO 口变为低电平后定时器不再计时，此时通过 GetHCSR04Timer 函数获取到时间间隔，并通过公式(高电平时间*声速(340M/S))/2 从而计算出物体距离该模块的距离。超声波检测到人体距离后反馈在串口上的距离如下图所示：

图 4.17 HC-SR04 超声波模块测量到的距离

4.3 MQ2 烟雾感应模块

4.3.1 MQ2 烟雾模块感应原理

MQ-2 型烟雾传感器属于二氧化锡半导体气敏材料 ,属于表面离子式 N 型半导体。处于 200~300 摄氏度时,二氧化锡吸附空气中的氧,形成氧的负离子吸附,使半导体中的电子密度减少,从而使其电阻值增加。当与烟雾接触时,如果晶粒间界处的势垒收到烟雾的调至而变化,就会引起表面导电率的变化。利用这一点就可以获得这种烟雾存在的信息,烟雾的浓度越大,

导电率越大,输出电阻越低,则输出的模拟信号就越大。

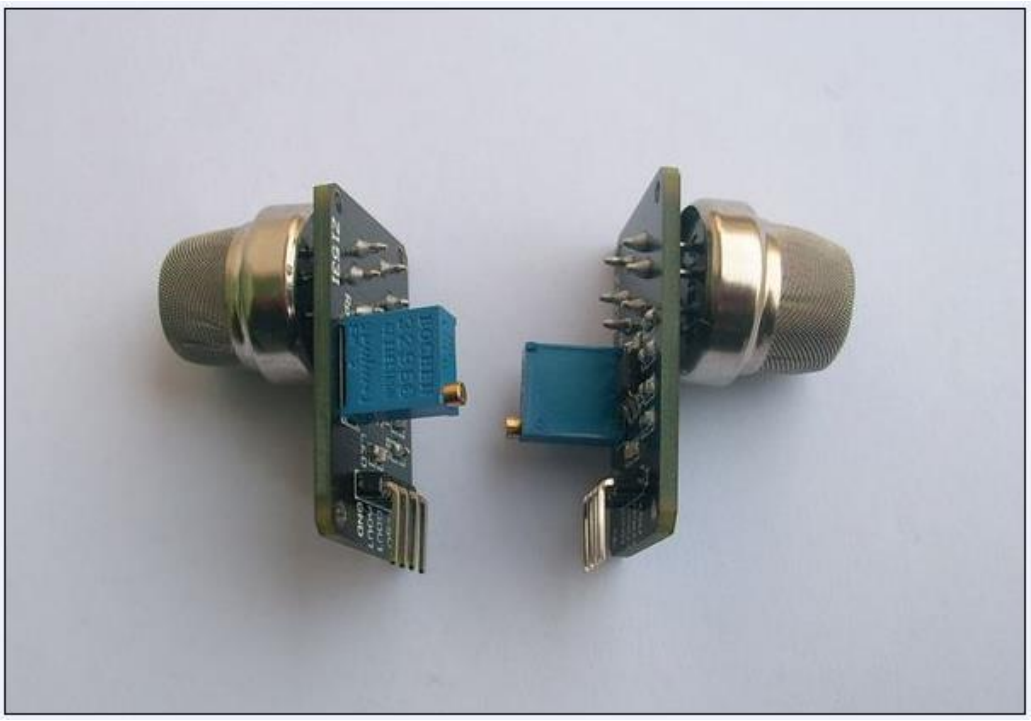


图 4.18 MQ2 烟雾传感器

4.3.2 MQ2 代码实现

在 MQ2 进行烟雾浓度的检测时，我们会通过制定的数模转换的制定通道进行模拟量与数据量的转换，并将转换后的结果取 20 次，并在此 20 此的取值中取出平均值，以平均值的值作为每一次监测后的值。

```
u16 Get_Adc_Average(u8 ch,u8 times)
{
  u32 temp_val=0;
  u8 t;
  for(t=0;t<times;t++)
  {
    temp_val+=Get_Adc(ch);
    delay_ms(5);
  }
  return temp_val/times;
}
```

图 4.19 MQ2 烟雾传感器代码实现

在取值函数 Get_Adc 中，我们通过设置指定的规则通道，序列以及采样时间，并将结果进行转化成数值，并将其返回给调用此函数的变量。

```
u16 Get_Adc(u8 ch)
{
    //设置指定ADC的规则组通道，一个序列，采样时间
  ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_480Cycles ); //ADC1,ADC通道,480个周期,提高采样时间可以提高精确度

  ADC_SoftwareStartConv(ADC1);    //使能指定的ADC1的软件转换启动功能

  while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束

  return ADC_GetConversionValue(ADC1);  //返回最近一次ADC1规则组的转换结果
}
//获取通道ch的转换值，取times次,然后平均
```

图 4.20 MQ2 烟雾传感器代码实现

最终我们就可以在串口调试助手上查询到当前环境下的颗粒值（即烟雾浓度）单位为 PPM，经过测试，通过 MQ2 模块所采集到的数值基本上较为准确，能够满足本次系统开发的需求。

4.4 DHT11 温湿度感应模块

DHT11 温湿度传感模块对于温度与湿度的获取原理以及转换原理较为复杂，在实际的开发过程中我们只需要关注其传输过来的温度与湿度的数据组成，充分了解其通信协议并对数据进行合理的转换即可，而不需要过分的去深究其数模转化的原理。在实训过程中，我也尝试过在网络上查找相关资料,但是关于 DHT11 模块的具体数模转化原理的解释较少，因此，在这里我们只要充分了解其通信协议即可。

4.4.1 DHT11 温湿度模块的通信协议

（1）通信接口（单线双向）：

CJSLDHT11 器件采用简化的单总线通信。单总线即只有一根数据线，系统中的数据交换、控制均由单总线完成。设备（主机或从机）通过一个漏极开路或三态端口连至该数据线，以允许设备在不发送数据时能够释放总线，而让其它设备使用总线；单总线通常要求外接一个约 5.1kΩ的上拉电阻，这样，当总线闲置时，其状态为高电平。由于它们是主从结构，只有主机呼叫从机时，从机才能应答，因此主机访问器件都必须严格遵循单总线序列，如果出现序列混乱，器件将不响应主机。

（2）单总线传送数据位定义

DATA 用于微处理器与 SDHT11 之间的通讯和同步,采用单总线数据格式，一次传送 40 位数据，高位先出。数 据 格 式 :8bit 湿度整数数据+8bit 湿度小数数据+8bit 温度整数数据+8bit 温度小数数据+8bit 校验位。注：其中湿度小数部分为 0。

（3）校验位数据定义

8bit 湿度整数数据+8bi 湿度小数数据+8bit 温度整数数据+8bit 温度小数数据”8bit 校验位等于所得结果的末 8 位。

表 4-4 DHT11 数据报文

名 称	单总线格式定义
起始信号	主机向数据总线(DATA)输出低电平至少 200us 唤醒 CJSLDHT11 模块且让模块稳定，然后 切换成输入(上拉或浮空),释放数据总线。
响应信号	CJSLDHT11 检测到主机释放数据总线(DATA),切换成输出，向数据总线输出低电平 83μs，再输出高 电平 87μs， 以响应主机的起始信号。
传 输数 据	接着 CJSLDHT11 通过数据总线(DATA)向主机发送 40-bit 温湿度数据,然后发送 1 个结束信号，完成一次数据发送，然后切换成输入，释放数据总线。
数据格式	1byte 湿度整数数据+1byte 湿度小数数据+1byte 温度整数数据+1byte 温度小数数据 +1byte 校验位，字节高位在前
湿度	湿度高字节为湿度整数部分数据，湿度低字节为湿度小数部分数据
温度	温度高字节为温度整数数据，温度低字节为温度小数数据，若低字节的 bit7 为 0 表示正温度，为 1 表示负温度
校验位	校验位＝(湿度高位+湿度低位+温度高位+温度低位)取结果的低 8 位

4.4.2 DHT11 模块代码实现

由下图的读取数值函数可知，我们每次通过 DHT11_Read_Bit()函数读取一个 bit，重复八次，通过此函数，我们每次就可以读出一个字节的数据，由通信协议的原理分析可知，DHT11 每 8 位数据或是温度的高、低八位或是湿度的高、低八位，或是校验位，当获取到一个字节后返回给变量。

```
u8 DHT11_Read_Byte(void)
{
    u8 i,dat;
    dat=0;
  for (i=0;i<8;i++)
   {
      dat<<=1;
      dat|=DHT11_Read_Bit();
    }
    return dat;
}
```

图 4.21 DHT11 模块实现

随后，将读取后的数值进行处理，由下图中的程序可知：通过读取 DHT11 模块中的 40 位数据，也就是一条报文后，首先进行判断当前传输过来的报文是否有传输错误的情况，当正确传输时，我们将报文中所对应的温度的高八位与湿度的高八位分别赋值给 temp 与 humi 变量，此时取出的就是温湿度的整数为的数值，将这两个值返回给调用函数，我们就可以查询到温度与湿度的整数位的数值了。

```
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;
}
```

图 4.22 DHT11 模块代码实现

此时，将模块与下位机核心 STM32F407 开发板进行连接以后，我们通过串口调试助手就可以实时的监控到此时房间内的温湿度的数值大小。

图 4.22 温湿度数值显示

4.5 QT 模块

在本项目中，QT 界面设计是本系统开发的核心点之一，在 QT 的开发中，最为重要的就是信号与槽以及 sokcet 通信，通过信号与槽的连接，我们可以实现人与界面的交互，用户就可以通过 QT 界面发送相关指令，对下位机进行相应的控制，通过 soket 的通信，下位机的数据才能够通过 WIFI 模块传输到 IMX6ULL 平台上并在 QT 界面进行显示，用户才得以在上位机（客户端）查看当前环境的相关数值。

4.5.1 信号与槽

想要利用信号与槽的机制进行 QT 可操作化界面的开发，我们首先要充分了解信号与槽的机制，通过查阅正点原子的相关资料以及结合在嵌入式课程中所学习到的相关知识，我们对信号与槽的机制有一个较为清楚的认识。

信号与槽（Signal & Slot）是 Qt 编程的基础，也是 Qt 的一大创新。因为有了信号与槽的编程机制，在 Qt 中处理界面各个组件的交互操作时变得更加直观和简单。信号（Signal）就是在特定情况下被发射的事件，例如 PushButton 最常见的信号就是鼠标单击时发射的 clicked() 信号，一个 ComboBox 最常见的信号是选择的列表项变化时发射的 CurrentIndexChanged() 信号， GUI 程序设计的主要内容就是对界面上各组件的信号的响应，只需要知道什么情况下发射 哪些信号，合理地去响应和处理这些信号就可以了。槽（Slot）就是对信号响应的函数。槽就是一个函数，与一般的 C++函数是一样的，可以定义在类的任何部分（public、private 或 protected），可以具有任何参数，也可以被直接调用。槽函数与一般的函数不同的是：槽函数可以与一个信号关联，当信号被发射时，关联的槽函数被自动执行。

在 Qt 中我们采用 connect 函数实现信号与槽函数之间的连接。

QObject::connect(sender, SIGNAL(signal()), receiver, SLOT(slot()));

4.5.2 socket 通信

在嵌入式实验的学习中，我们已经对于 Socket 的通信已经有了一定的了解，因此，下面将对 Socket 通信做一个简单的介绍，Socket，即套接字。就是两台主机之间逻辑连接的端点。（通俗来说：网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个 socket）。

TCP/IP 协议是传输层协议，主要解决数据如何在网络中传输。HTTP 是应用层协议，主要解决如何包装数据。Socket 是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议、本地主机的 IP 地址、本地进程的协议端口、远程主机的 IP 地址、远程进程的协议端口。

Socket 编程主要涉及到客户端和服务端两个方面，首先是在服务器端创建一个服务器套接字（ServerSocket），并把它附加到一个端口上，服务器从这个端口监听连接。端口号的范围是 0 到 65536，但是 0 到 1024 是为特权服务保留的端口号，可以选择任意一个当前没有被其他进程使用的端口。

客户端(Socket)请求与服务器进行连接的时候，根据服务器的域名或者 IP 地址，加上端口号，打开一个套接字(也就是连接一个服务端)。当服务器接受连接后，服务器和客户端之间的通信就像输入输出流一样进行操作。

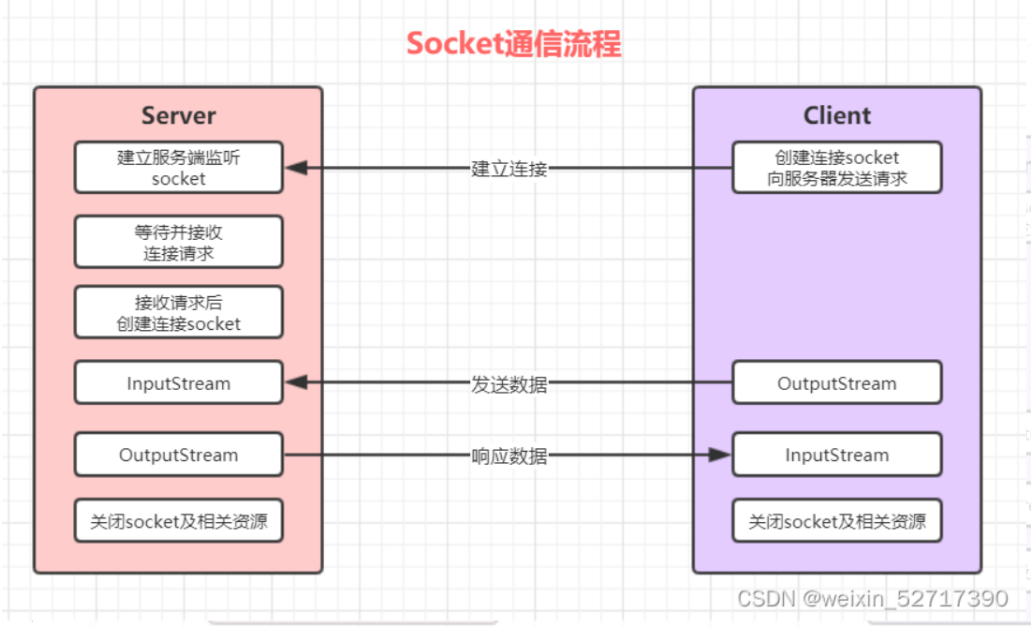


图 4.23 socket 通信流程

4.5.3 点灯程序实现

在本系统设计中，我们在进行充分的讨论后，决定以通过 QT 界面点亮 LED 灯来模拟用户通过 QT 界面对家居进行控制的过程，因此，我们在 Qt 中编写了点灯的槽函数（执行函数）如下图所示：

```
168 void tcpclient::slotlighton()
169 {
170     m_client->write(lighton.toUtf8());
171 }
172 void tcpclient::slotlightoff()
173 {
174     m_client->write(lightoff.toUtf8());
175 }
```

图 4.24 点灯与关灯槽函数

在函数 slotlighton 函数中，我们通过发送 lighton 变量中的值，该值中存放的是字符串“1”到 STM32 开发板中，STM32F407 开发板在接受到数据以后，会对传输过来的字符串进行识别，并控制 LED 灯的亮灭情况。

```
//卧室灯
connect(ui->lighton, &QPushButton::clicked, this, &tcpclient::slotlighton);
//关卧室灯
connect(ui->lightoff,&QPushButton::clicked, this, &tcpclient::slotlightoff);
```

图 4.25 连接点击事件与槽函数

由信号与槽的原理我们可知，我们能够通过 connect 连接信号与槽函数，在本次的 Qt 界面中，我们希望能够为用户提供进行灯光亮灭控制的开关，因此在这里，我们连接了点击事件与开关灯的槽函数，当我们触发点击事件时，将会引发所对应的槽函数，从而执行槽函数中的功能，最终使得 STM32F407 开发板控制 LED 灯进行开关，下图是对 LED 灯的演示。

图 4.26 LED 灯的亮灭

4.5.4 Socet 通信（数值的获取）

在 Qt 的界面中，我们还要对相关的参数进行获取，例如房间温度、房间湿度以及房间内的颗粒浓度等等，这样用户就能够在 Qt 界面上实时获取对应的数值，在具体的实现代码中，因为 Socket 通信的基本原理我们已经在之前有所介绍与了解，因此在这里我们主要对数据的处理进行解释。

```
void tcpclient::slotReadyRead()
{
    //接收数据
    QString first;
    QString humid;
    QString smoke;
    array = m_client->readAll();
    first=array.at(0);
    first.append(array.at(1));
    first.append('C');
    humid=array.at(2);
    humid.append(array.at(3));
    humid.append('%');
    smoke=array.at(4);
    smoke.append(array.at(5));
    smoke.append(array.at(6));
    smoke.append(array.at(7));
    smoke.append("ppm");
    state1=array.at(8);
    state2=array.at(9);

    ui->textEdit_t->setText(first);
    ui->textEdit_h->setText(humid);
    ui->textEdit_s->setText(smoke);


    //居中对齐
    ui->textEdit_t->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
    ui->textEdit_h->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
    ui->textEdit_s->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter);
}
```

图 4.27 slotReadyRead 函数

因为我们从 STM32F407 所传输过来的是一段字符串，其中包含了温度、湿度以及烟雾浓度等等数值，因此，我们要对字符串进行相应的处理，在发送过来的字符串中，第 1 位（下标为 0），与第 2 位传输过来的数据为温度的数值，因此，我们取出第 1 位与第 2 位的数据并将其显示在温度的文本框中。第 3 位与第 4 位所显示的数值为当前空气中的湿度占比，并将其显示到文本框中，剩余位数为空气中颗粒浓度以及当前 LED 灯的状态，我们能够根据所对应的数值来进行相应的处理。

以下是 Qt 界面的显示效果：

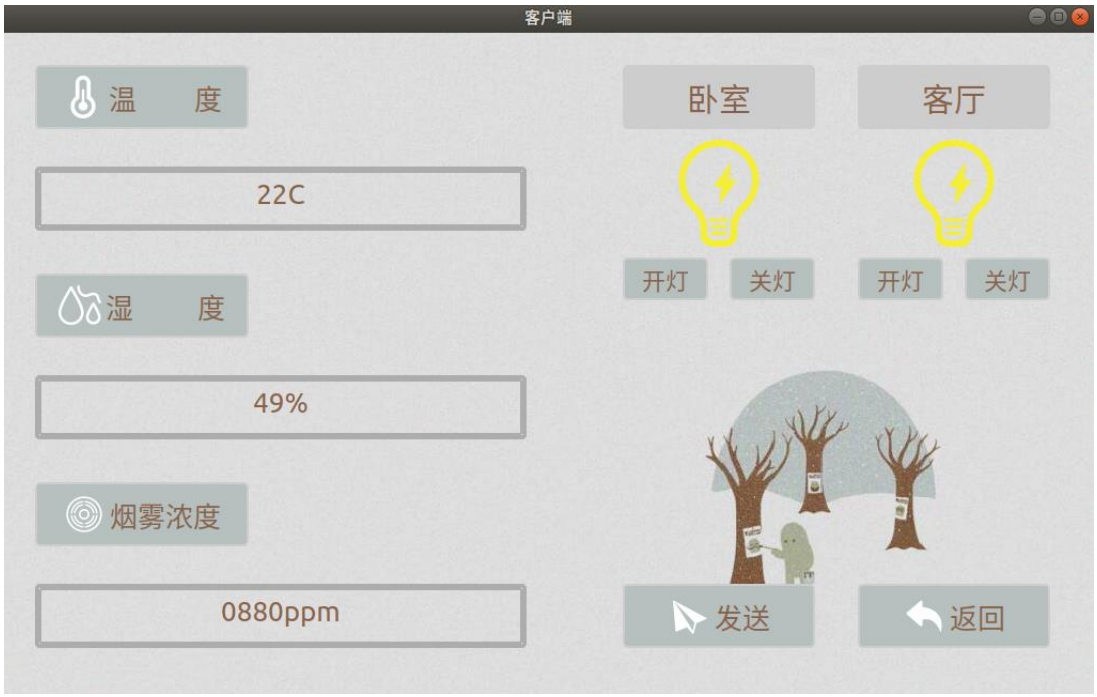


图 4.28 Qt 界面显示效果

4.6 WIFI 模块（IMX6ULL）

在本次项目中，WIFI 模块是我们进行通信的较为重要的模块，只有通过 WIFI 模块，使得下位机与上位机位于同一网关内，从而通过 Socket 进行数据的传输，在本次实验中，我完成了对于 IMX6ULL 的 WIFI 模块的实现，现将实现步骤介绍如下：

实验前准备正点原子 SDIO WIFI 模块（RTL8189）,由于底板 SDIO WIFI 接口与 TF 卡槽接口共用了大部分管脚，所以在使用 SDIO WIFI 时不 能使用 TF 卡。也就是说，我们测试 SDIO WIFI 时不能从 TF 卡启动系统。我们应该从 eMMC 或者 Nand Flash 启动系统。

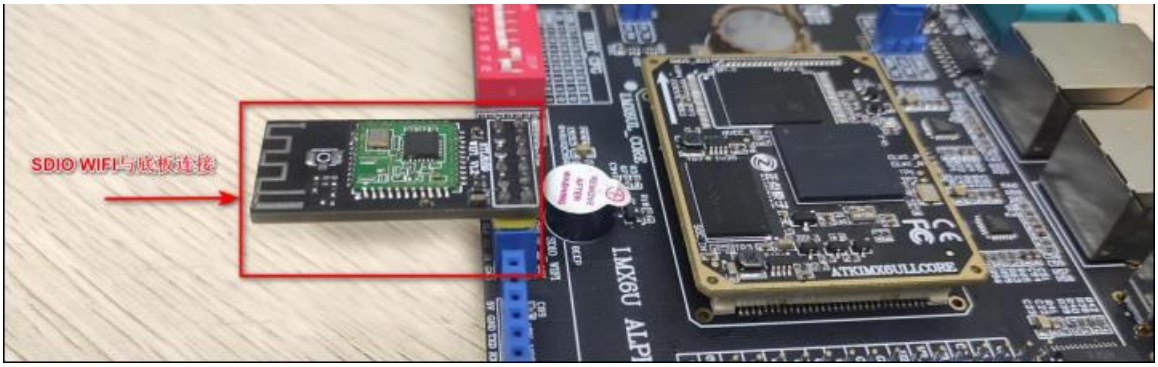


图 4.29 Qt 插入 RTL8189

随后我们将通过以下命令查询 RTL8189 的驱动位置：cd /home/root/driver/rtl8189/

```
root@ATK-IMX6U:~# cd /home/root/driver/rtl8189/
root@ATK-IMX6U:~/driver/rtl8189# ls
8189fs.ko
root@ATK-IMX6U:~/driver/rtl8189#
```

图 4.30 驱动模块所在的位置

随后，我们在前/home/root/目录下编辑成一个 wpa_supplicant.conf 配置文件，该配置文件用于配置模块需要连接的 WIFI 的相关数据如 WIFI 名称与 WIFI 密码等，其具体的配置格式如下（其中 ssid 为无线网络名称 psk 为无线网络密码 ）：

```
network={
        ssid="ALIENTEK-YF"
        psk="1590202****"
}
```

图 4.31 WIFI 配置

随后，我们能够使用下面的指令来连接配置的无线网络，当看到“wlan0:CTRL-EVENT-CONNECTED-Connection to ec:88:8f:77:d6:da completed [id=0 id_str=]”这句话表明连接成功。如果连接不成功会提示“wlan0: WPA: 4-Way Handshake failed - pre-shared key may be incorrect”这样的话。

```
root@ATK-IMX6U:~/driver/rtl8189# Successfully initialized wpa_supplicant
wlan0: Trying to associate with e[ 1126.711798] RTL871X: rtw_set_802_11_connect(wlan0)  fw_state=0x00000008
4:0e:ee:f2:11:14 (SSID='ALIENTEK-YF' freq=2437 MHz)
[ 1126.873215] RTL871X: start auth
[ 1126.880934] RTL871X: auth success, start assoc
[ 1126.895123] RTL871X: assoc success
[ 1126.898975] RTL871X: recv eapol packet
wlan0: Associated with e4:0e:ee:f2:11:14
[ 1126.912104] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[ 1127.886023] RTL871X: recv eapol packet
[ 1127.890421] RTL871X: send eapol packet
[ 1127.916252] RTL871X: recv eapol packet
[ 1127.924132] RTL871X: send eapol packet
[ 1127.928706] RTL871X: set pairwise key camid:4, addr:e4:0e:ee:f2:11:14, kid:0, type:AES
wlan0: WPA: Key negotiation completed with e4:0e:ee:f2:11:14 [PTK[ 1127.941820] RTL871X: set group key camid:5, addr:e4:0e:ee:f2:11:14, kid:1, type:AES
=CCMP GTK=CCMP]
wlan0: CTRL-EVENT-CONNECTED - Connection to e4:0e:ee:f2:11:14 completed [id=0 id_str=]

root@ATK-IMX6U:~/driver/rtl8189#
```

图 4.32 WIFI 连接成功显示

同时，我们如果要利用到命令查询当前的 IP 地址，从而能够进行 Socket 之间的通信 udhcpc -i wlan0

```
root@ATK-IMX6U:~/driver/rtl8189# udhcpc -i wlan0
udhcpc (v1.24.1) started
Sending discover...
Sending select for 192.168.101.177...
Lease of 192.168.101.177 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.101.1
root@ATK-IMX6U:~/driver/rtl8189#
```

图 4.33 IP 地址查询

5、系统演示

在本模块中，我将对系统的全开发流程以及演示效果进行说明，使老师能够更加系统的了解到我们此次系统的基本功能于过程，如图 5.1 所示，该图为我们系统的总体框架图，其中包含了上下位机、窗帘、LED 以及烟雾、温湿度等各个模块的设计。

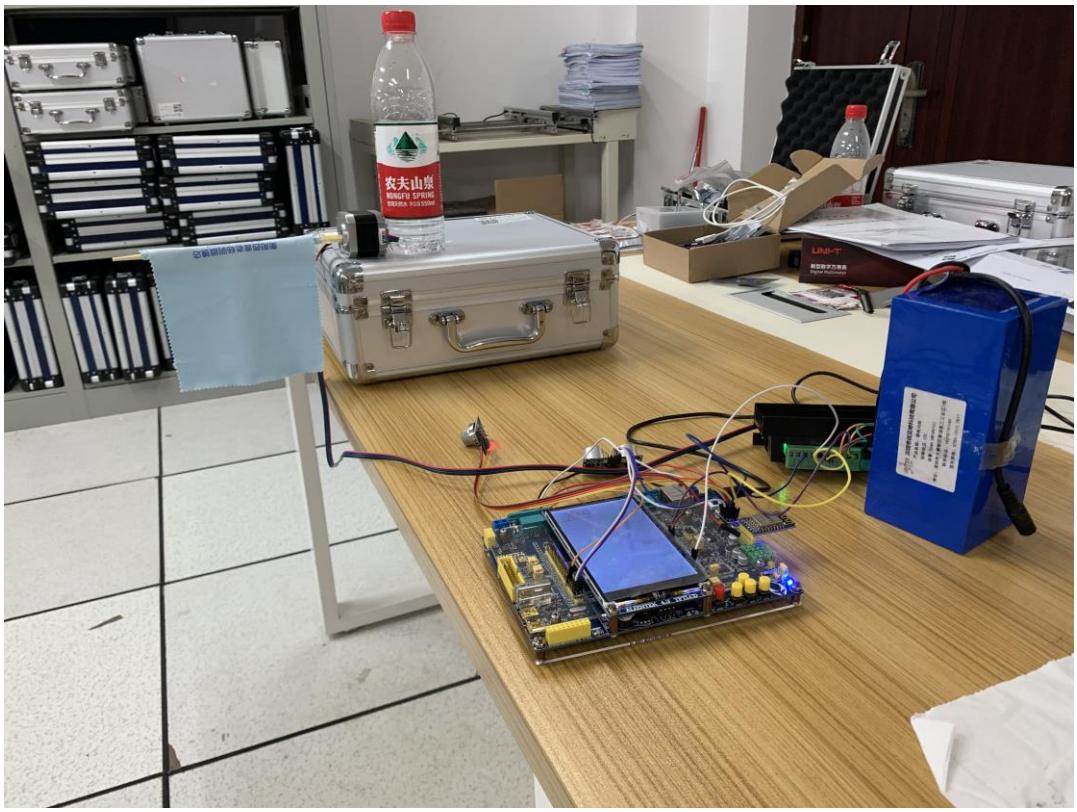


图 5.1 总系统框架图

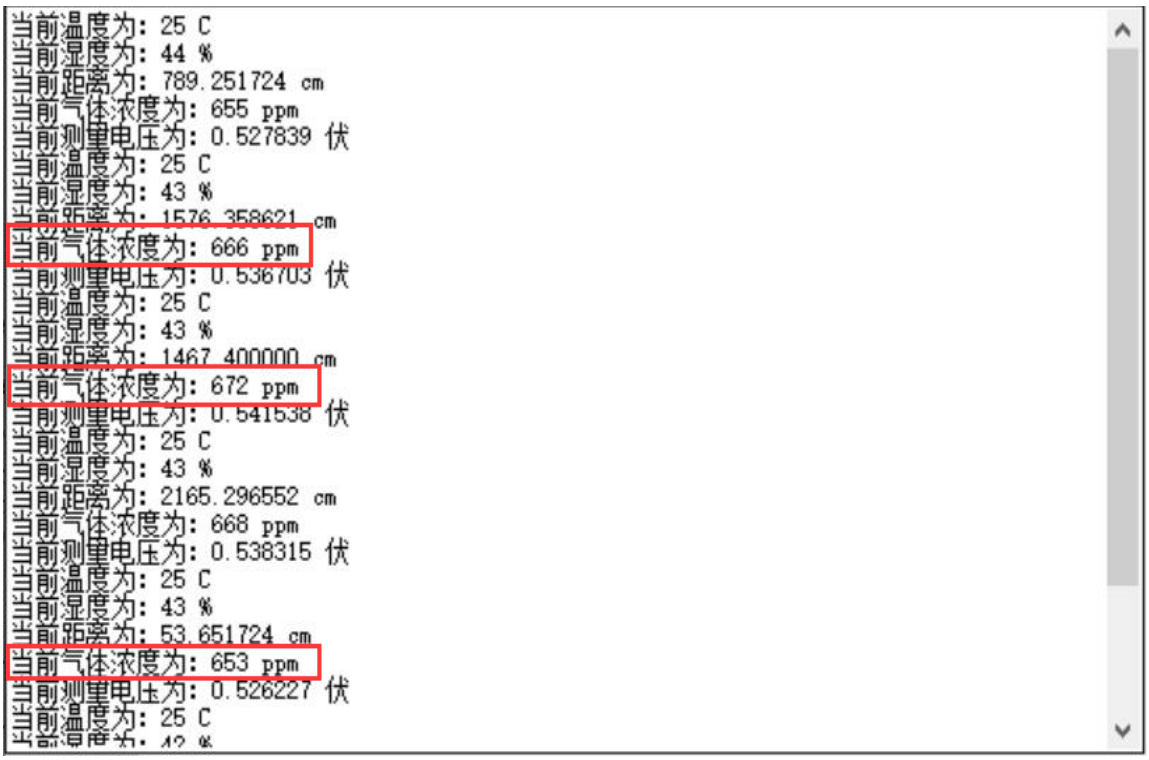
由摘要的介绍可知，我们的系统能够完成自主开关窗帘以及检测当前环境的相关数值的功能，在我们进行开发的过程中，我们首先完成了传感器模块的调试，首先对 HC-SR04 模块进行了调试，并将打印信息输出至串口，经过测试，发现通过超声波模块感应人体的距离测试结果较为准确，且取样时间较短，能够满足我们此次系统开发的要求。

5.2 串口输出信息

随后，我们对于 DHT11 温湿度感应模块进行了测量，进行实时监测当前环境下的空气中温度与湿度的检测，并将其打印在串口调试助手中，通过检测，发现该模块能够较为准确的检测当前环境下的温湿度的值。

5.3 串口输出信息

最后我们进行了烟雾模块的测试，并在串口打印出当前环境下的颗粒浓度

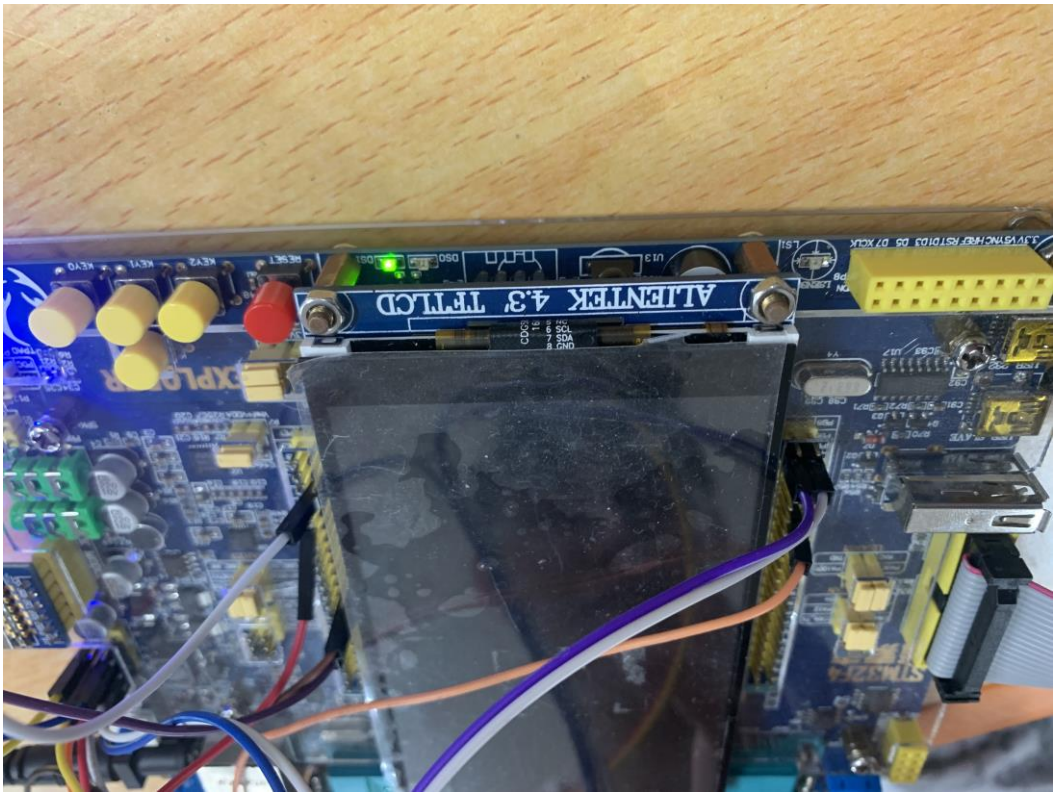


5.4 串口输出信息

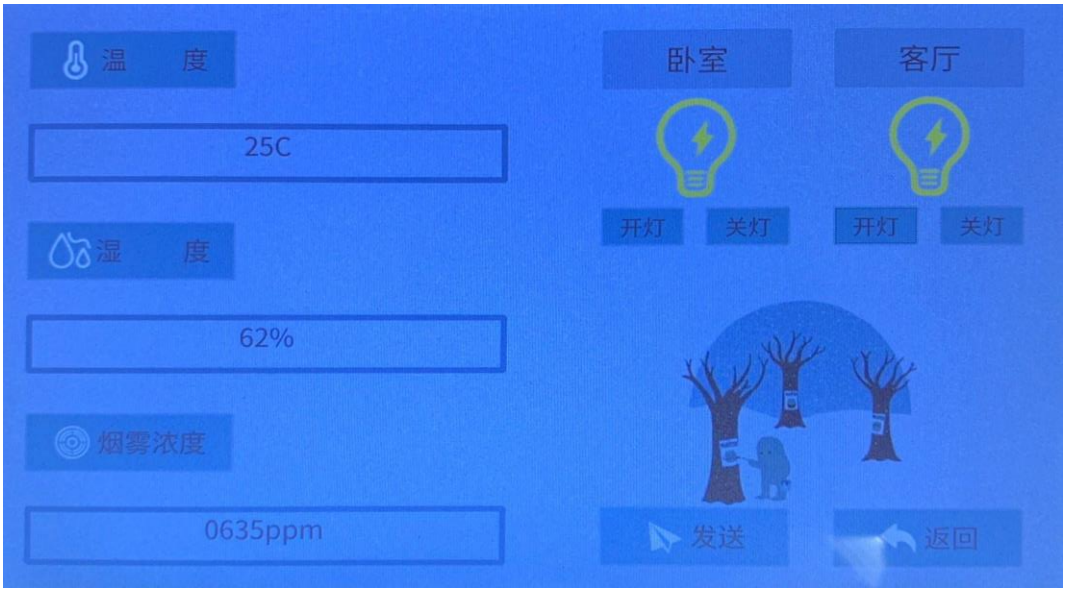
随后，我们进行了电机模块的测试，电机在脉冲信号与控制信号的作用下，能够进行正反方向的运转，结合上述传感器模块的介绍，我们的系统设定为：当人体靠近超声波传感器时，电机将会进行转动，在 DIR 方向脉冲的控制下进行正反转，从而实现窗帘的开关功能，由图 5.5 我们能够看出，在电机的控制下，窗帘能够模拟被打开或是关闭

5.5 电机驱动模块

最后，是通信模块，在通信模块的设计中，我们采用的是 WIFI 来实现上、下位机之间的通信，并能够通过 Qt 上的点灯与关灯按键实时的控制 LED 灯的开关（此处模拟的是家居控制），同时，还能够在 Qt 界面上实时的显示当前环境下的相关环境参数，如温湿度以及烟雾浓度等



5.5 led 灯被点亮



5.5 实时数据

6. 总结与心得体会

本次嵌入式综合实训，是我们在团队协作的情况下完整的开发了我们的小项目，在选题之前，我也曾经犹豫要选怎样的题才能综合的运用到我们所学习到的嵌入式方面的知识，使我们的理论所学能够真正的运用到我们的实际开发中去，为此，我也曾查阅过相关论文，并翻阅正点原子所提供更多一些官方文档，了解了 STM32F407 开发板与 IMX6ULL 开发板中所能实现的功能。在查阅资料的过程中，我发现对于目前对于智能家居小项目的开发的热度很高，同时在进行可行性分析之后，发现此次项目不仅仅充分用到了我们所学习的相关知识，许多模块都包含了嵌入式开发以及物联网的思想，同时，智能家居项目的开发我认为很具有挑战性与新颖性，这也是我们最终决定进行智能家居项目开发的原因。最后的结果也证明，此次系统的开发，对我们的专业素养的提升十分巨大。

嵌入式的开发不可能是一帆风顺的，在本次系统的开发中，我们也遇到了很多的困难，在这些困难中，不仅让我学习到了面对困难时所要具备的心态以及如何去采取方法解决困难，更多的让我学习到作为一名项目负责人在项目开发中遭遇挫折时，应该如何去引导项目的开发与实施。本次项目的开发过程中，小组所遇到的两个最为重大的技术点就是 Qt 的界面设计以及通信的实现。在界面设计方面，因为小组内的所有成员之前均无 C++Qt 开发的经验，对于我们来说，能够算作是一个完全陌生的领域，但是界面的设计，是我们此次项目开发中不可或缺的重要部分，因此，当我们完成基本的开发模块以后，我们开始致力于对于 Qt 界面设计的学习以及开发中，在做 Qt 界面时给我最大的教诲是：在面对自己不熟悉的领域时，我们首先要克服的就是自身的畏难情绪，在刚接触 Qt 的过程中，我自身的畏难情绪较为强烈，以至于导致本应该在两天内所完成的工作量却整整延期到了四天，因此，对于畏难情绪的克服，是作为一名嵌入式开发工程师所首先应该具备的素养。其次，在通信模块中，项目规划之初，团队本打算采用蓝牙模块来实现上、下位机之间的数据传输，但是在实际的操作中却发现，因为 IMX6ULL 开发板中搭载的 LINUX 操作系统对于蓝牙模块的要求较为苛刻，存在两个蓝牙模块之间协议栈不兼容的情况，因此，大大的延缓了我们项目开发的进度，最终小组内及时开展小组会议，决定采用我们更为熟悉的 WIFI 模块来实现数据通路的构建。但总的来说，虽然在我们尝试蓝牙的过程中遇到了挫折，但也让我们借此机会，学习了有关蓝牙的相关知识，嵌入式开发就是在这一个又一个的困难之中，最终达到系统设计的目的。

最后，作为本次项目开发小组的小组长，也让我学习到了很多项目管理方面的知识，我曾经一度认为作为一名项目组的组长，其能力与学习能力都是要远远高于其他组员的，且许多事情一定要自己亲力亲为，为小组的其他组员起到模范作用。但是经过这次系统的开发，我逐渐意识到：作为一名项目负责人，拥有较强的专业能力固然重要，但更多的是要培养自己作为一名“船长”的能力，在项目开发的过程中，要能够及时的根据组内开发进度进行组内分工的调整，自己也要在组员开发遇到困难的时候充当助力器的作用，例如在本次开发中，我本来是负责 QT 与电机驱动的开发，但当小组成员完成项目初期阶段的开发以后，发现本项目中通信部分遇到了较大的困难，因此，我及时的进行项目调整，发现传感器部分的进度较快，于是，我将 Qt 界面的设计交于了传感器设计部分的成员，自己与通信小组进行困难的攻坚，也正是这样及时的调整，才让我们的项目不至于放弃上位机模块的开发。

总的来说，作为项目开发者之一，本次项目让我学习到了许多的专业知识，提升了自己的专业能力，作为小组组长，让我学习到了作为项目负责人是组内良心开发以及项目开发方向的“引舵人”。对我各方面的能力都有较大的提升。最后也十分感谢指导老师郑老师在我们项目开发中给予我们在项目开发上的指导与帮助，小组在老师的指导以及团队的通力合作之下，顺利的完成了本次项目的开发，并取得了不错的学习效果！

参考文献

[1] 杨勇.物联网智能家居发展探究[J].网络安全技术与应用,2022(05):133-134.

[2] 房雅珉.关于物联网在智能家居中的应用与发展[J].居舍,2022(13):32-35.

[3]郭铭,王佳佳.基于STM32的智能家居湿度控制系统[J].内蒙古济,2021(24):81-83.

[4]田启松,江祖旺,叶鹏.基于 STM32 的物联网智能家居环境监控系统[J].科技创新与应用,2019(35):29-30.

[5]张琳,郭雄,姚咏涛.基于 STM32 智能家居开关控制系统设计[J].信息系统工程,2017(05):104-107.

[6]连丽红.基于 Qt 的嵌入式实验平台开发[J].中国集成电路,2018,27(08):59-62.

[7]刘星,刘天缘.智能家居在住宅空间设计中的应用研究[J].房地产世界,2022(05):37-39.

附录

附录 A：硬件原理图

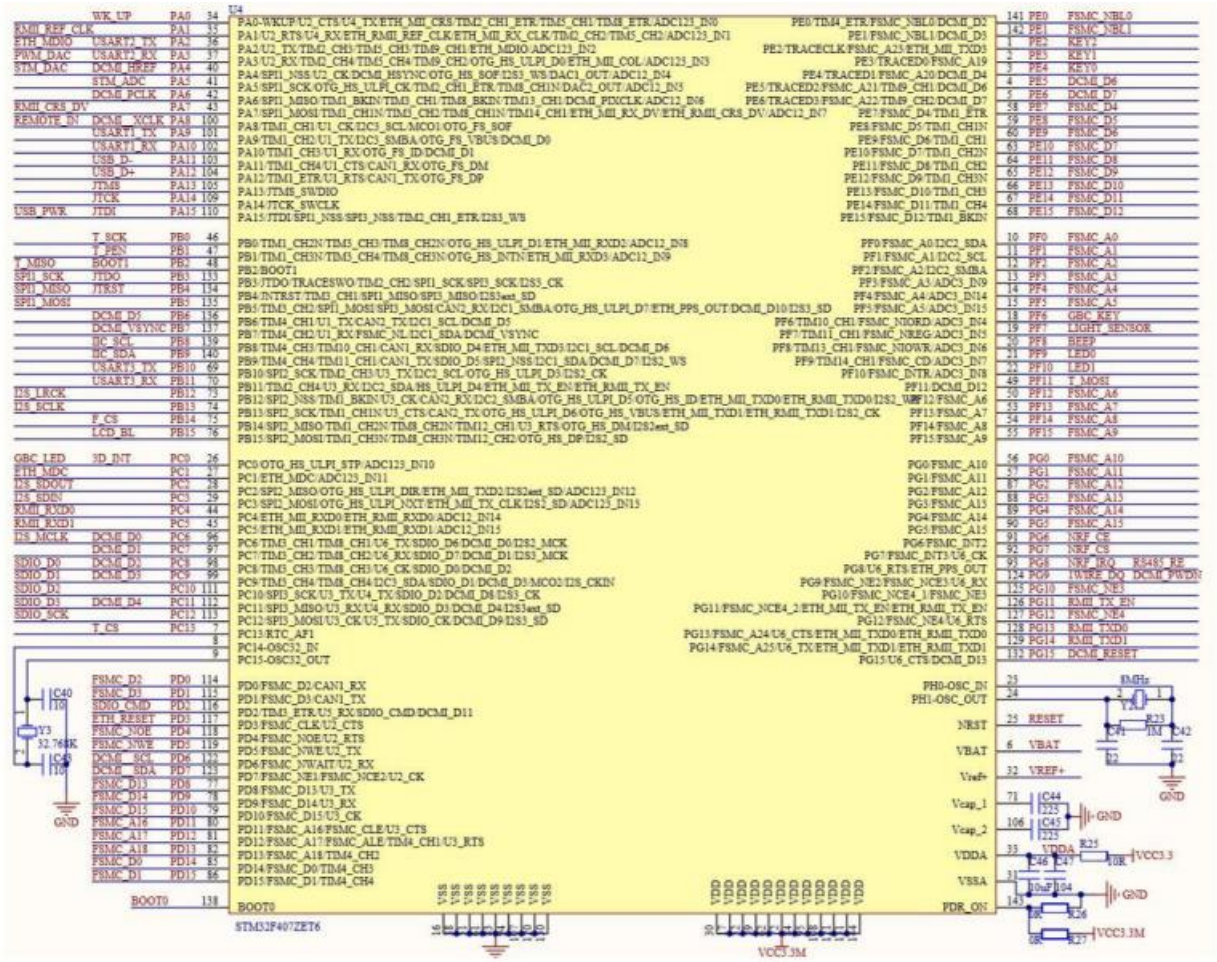


图 A1 STM32F407 原理图

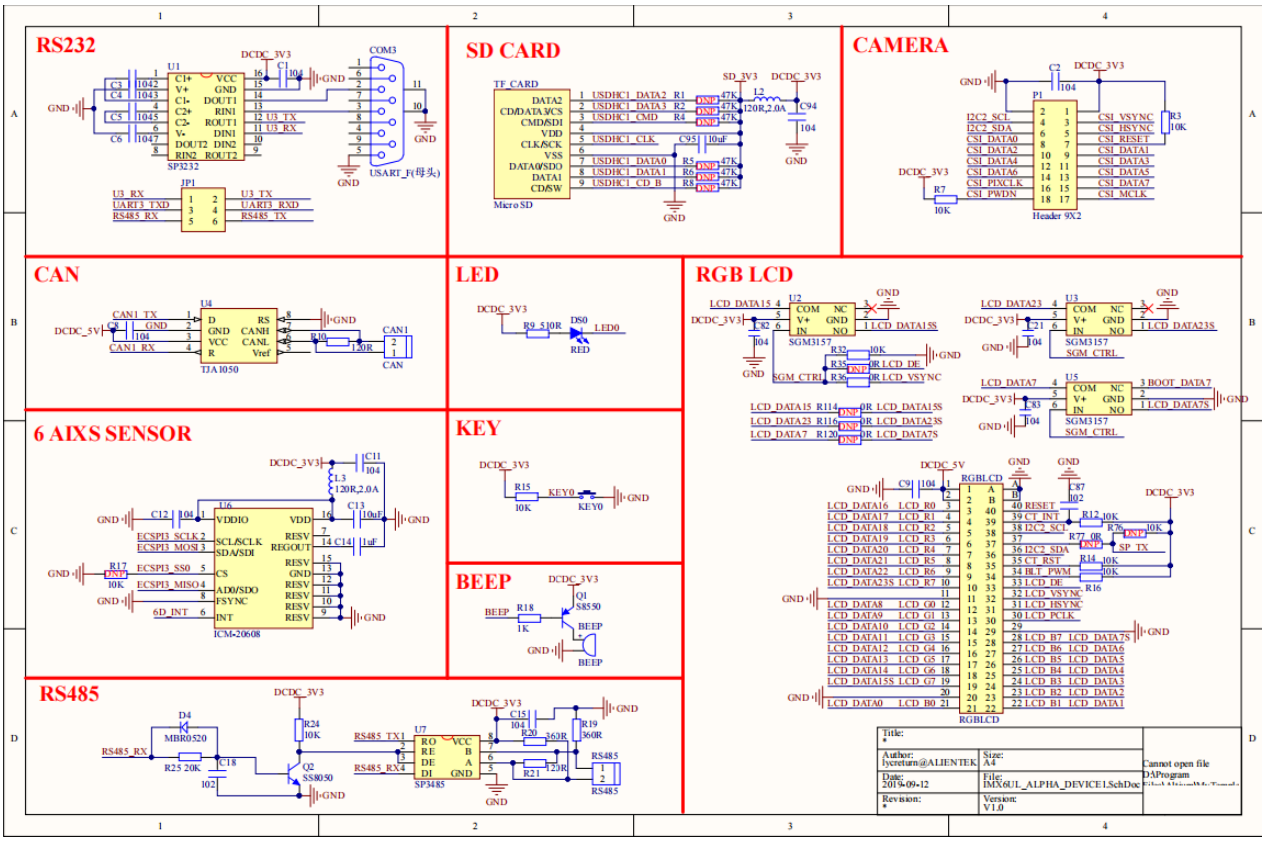


图 A2 IMX6ULL 原理图

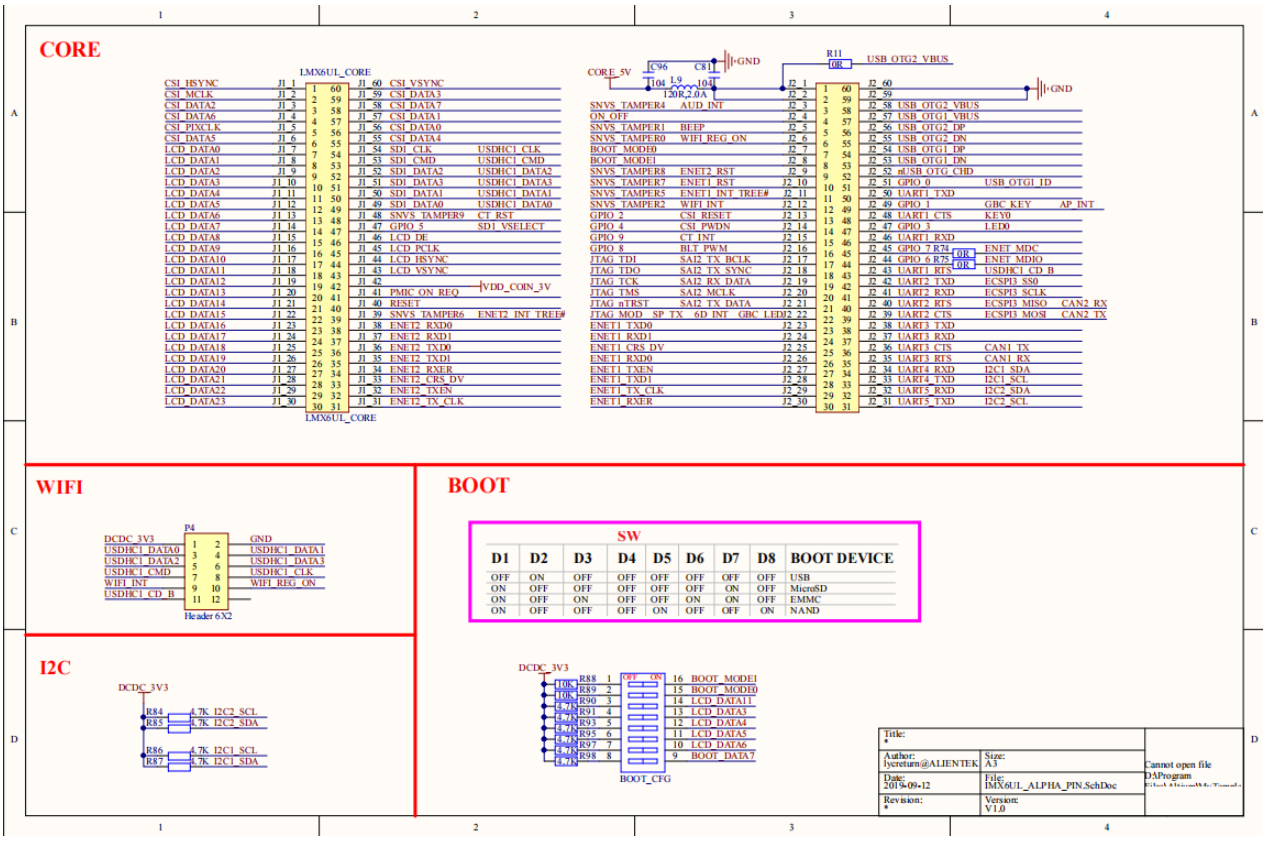


图 A3 IMX6ULL 原理图

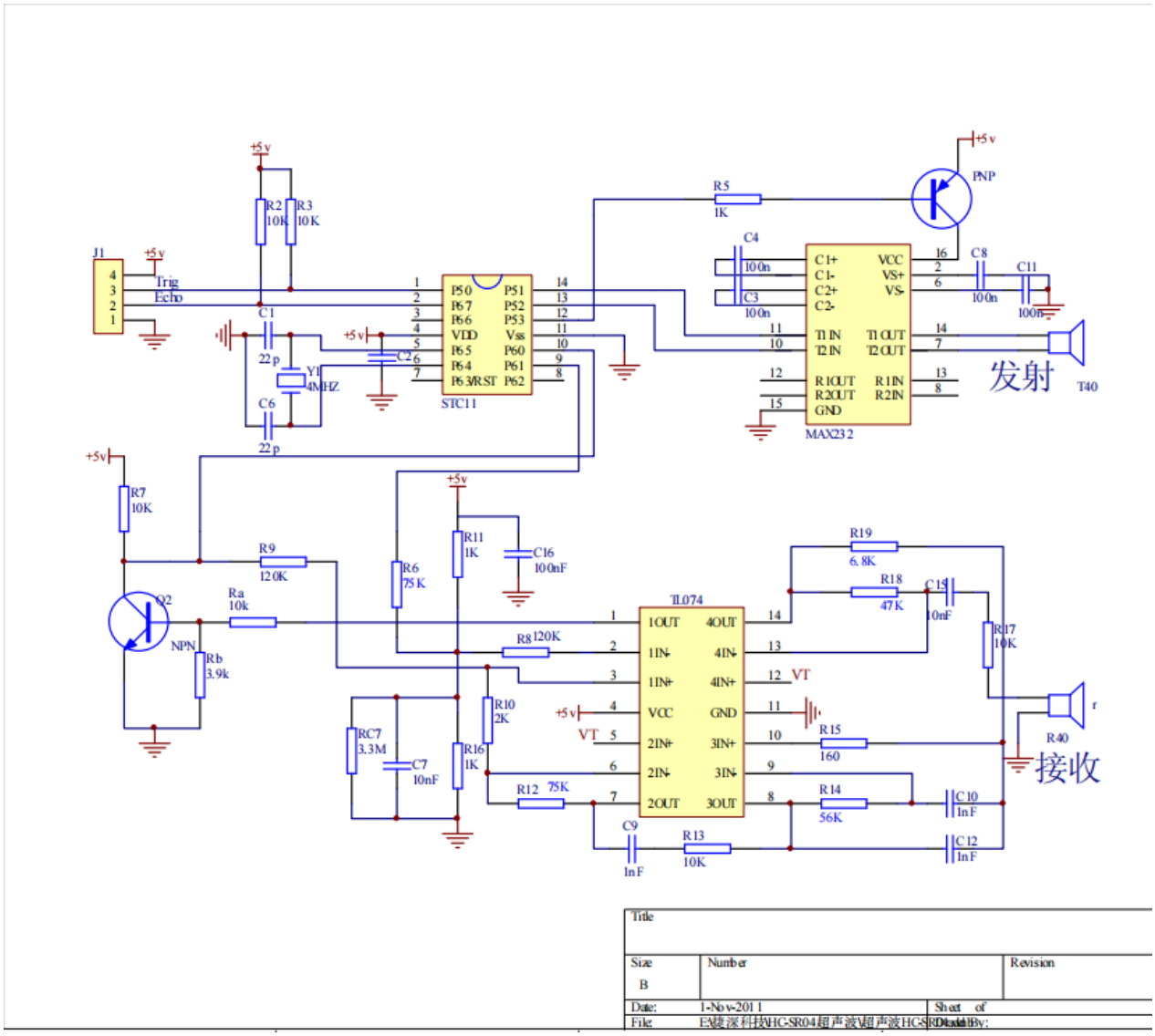


图 A4 HC-SR04 原理图

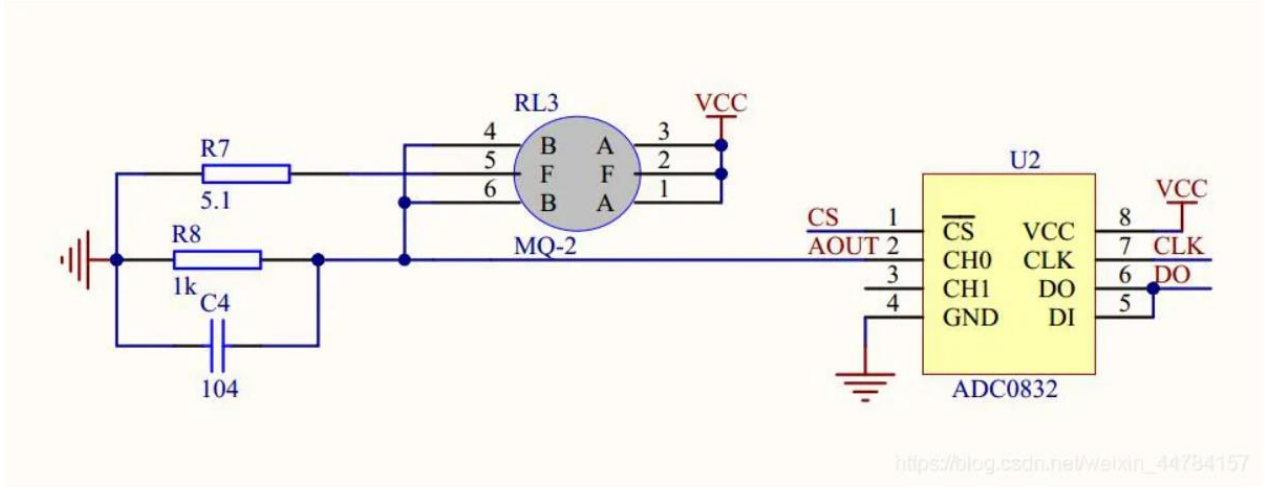


图 A5 MQ2 原理图

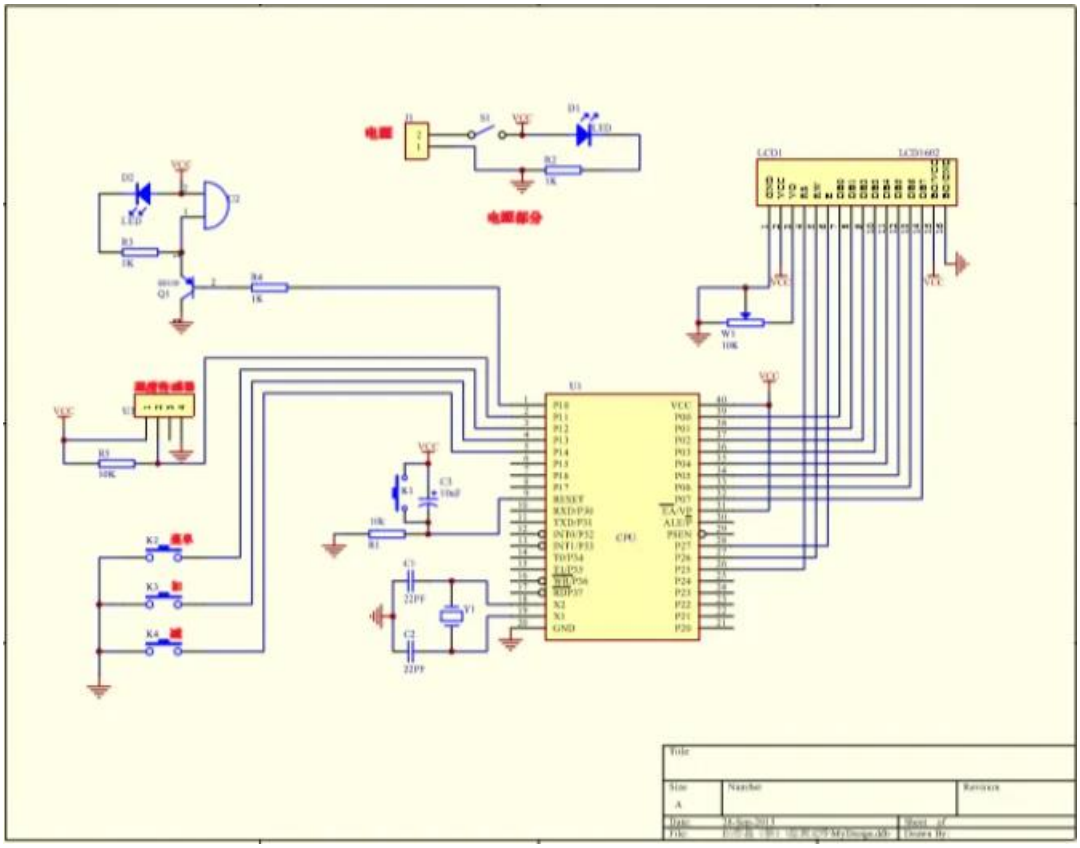


图 A6 DHT11 原理图

附录 B：软件源代码

Main.c

```
#include "sys.h"
#include "delay.h"
#include "usart.h"
#include "led.h"
#include "lcd.h"
#include "key.h"
#include "usmart.h"
#include "sram.h"
#include "malloc.h"
#include "w25qxx.h"
#include "sdio_sdcard.h"
#include "ff.h"
#include "exfuns.h"
#include "fontupd.h"
#include "text.h"
#include "usmart.h"
#include "touch.h"
#include "usart3.h"
#include "common.h"
#include "stm32f4xx.h"
#include "beep.h"
#include "pwm.h"
#include "HCSR04.h"
#include "dht11.h"
#include "MQ2.h"
#include "math.h"


//ALIENTEK 探索者 STM32F407 开发板 扩展实验 5
//ATK-RM04 WIFI 模块测试实验     -库函数版本
//技术支持：www.openedv.com
//淘宝店铺：http://eboard.taobao.com
//广州市星翼电子科技有限公司
//作者：正点原子 @ALIENTEK

void Initialization(){
    BEEP_Init();
    HCSR04Init();
    DHT11_Init();
    Adc_Init();         //初始化模数转换器
}
```

```
int main(void)
{
    u8 key,fontok=0;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置系统中断优先级分组 2
    delay_init(168);  //初始化延时函数
    uart_init(115200);      //初始化串口波特率为 115200
    usart3_init(115200);  //初始化串口 3 波特率为 115200
    Initialization();
    LED_Init();                     //初始化 LED
    LCD_Init();                     //LCD 初始化
    KEY_Init();                     //按键初始化
    W25QXX_Init();                  //初始化 W25Q128
    tp_dev.init();                  //初始化触摸屏
    usmart_dev.init(168);           //初始化 USMART
    my_mem_init(SRAMIN);            //初始化内部内存池
    my_mem_init(SRAMCCM);           //初始化 CCM 内存池
    exfuns_init();                  //为 fatfs 相关变量申请内存
    f_mount(fs[0],"0:",1);          //挂载 SD 卡
    f_mount(fs[1],"1:",1);          //挂载 FLASH.
    key=KEY_Scan(0);
    if(key==KEY0_PRES)              //强制校准
    {
        LCD_Clear(WHITE);   //清屏
        TP_Adjust();            //屏幕校准
        TP_Save_Adjdata();
        LCD_Clear(WHITE);   //清屏
    }
    fontok=font_init();         //检查字库是否 OK
    if(fontok||key==KEY1_PRES)//需要更新字库
    {
        LCD_Clear(WHITE);               //清屏
        POINT_COLOR=RED;                //设置字体为红色
        LCD_ShowString(60,50,200,16,16,"ALIENTEK STM32");
        while(SD_Init())                //检测 SD 卡
        {
            LCD_ShowString(60,70,200,16,16,"SD Card Failed!");
            delay_ms(200);
            LCD_Fill(60,70,200+60,70+16,WHITE);
            delay_ms(200);
        }
        LCD_ShowString(60,70,200,16,16,"SD Card OK");
        LCD_ShowString(60,90,200,16,16,"Font Updating...");
        key=update_font(20,110,16,"0:");//从 SD 卡更新
        while(key)//更新失败
```

```
            {
                LCD_ShowString(60,110,200,16,16,"Font Update Failed!");
                delay_ms(200);
                LCD_Fill(20,110,200+20,110+16,WHITE);
                delay_ms(200);
            }
            LCD_ShowString(60,110,200,16,16,"Font Update Success!");
            delay_ms(1500);
            LCD_Clear(WHITE);//清屏
        }
        atk_8266_test();            //进入 ATK_ESP8266 测试
    }
```

Common.c:

```
    #include "common.h"

    ////////////////////////////////////////////////////////////////////////////////////////////////////
    //本程序只供学习使用，未经作者许可，不得用于其它任何用途
    //ALIENTEK STM32 开发板
    //ATK-ESP8266 WIFI 模块  公用驱动代码
    //正点原子@ALIENTEK
    //技术论坛:www.openedv.com
    //修改日期:2014/4/3
    //版本：V1.0
    //版权所有，盗版必究。
    //Copyright(C) 广州市星翼电子科技有限公司  2009-2019
    //All rights reserved
    ////////////////////////////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////////////////////////////
    //用户配置区

    //连接端口号:8086,可自行修改为其他端口.
    const u8* portnum="8086";

    //WIFI STA 模式,设置要去连接的路由器无线参数,请根据你自己的路由器设置,自行修改.
    const u8* wifista_ssid="REDMIK20";              //路由器 SSID 号
    const u8* wifista_encryption="wpawpa2_aes";    //wpa/wpa2 aes 加密方式
    const u8* wifista_password="12345678";    //连接密码

    //WIFI AP 模式,模块对外的无线参数,可自行修改.
    const u8* wifiap_ssid="ATK-ESP8266";                //对外 SSID 号
    const u8* wifiap_encryption="wpawpa2_aes";    //wpa/wpa2 aes 加密方式
    const u8* wifiap_password="12345678";          //连接密码
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////
//4 个网络模式
const u8 *ATK_ESP8266_CWMODE_TBL[3]={"STA 模式 ","AP 模式 ","AP&STA 模式 "};
//ATK-ESP8266,3 种网络模式,默认为路由器(ROUTER)模式
//4 种工作模式
const u8 *ATK_ESP8266_WORKMODE_TBL[3]={"TCP 服务器","TCP 客户端"," UDP 模式"};
//ATK-ESP8266,4 种工作模式
//5 种加密方式
const u8
*ATK_ESP8266_ECN_TBL[5]={"OPEN","WEP","WPA_PSK","WPA2_PSK","WPA_WAP2_PSK"};
//////////////////////////////////////////////////////////////////////////////////////////////////////////

//usmart 支持部分
//将收到的 AT 指令应答数据返回给电脑串口
//mode:0,不清零 USART3_RX_STA;
//      1,清零 USART3_RX_STA;
void atk_8266_at_response(u8 mode)
{
    if(USART3_RX_STA&0X8000)        //接收到一次数据了
    {
        USART3_RX_BUF[USART3_RX_STA&0X7FFF]=0;//添加结束符
        printf("%s",USART3_RX_BUF);  //发送到串口
        if(mode)USART3_RX_STA=0;
    }
}
//ATK-ESP8266 发送命令后,检测接收到的应答
//str:期待的应答结果
//返回值:0,没有得到期待的应答结果
//      其他,期待应答结果的位置(str 的位置)
u8* atk_8266_check_cmd(u8 *str)
{

    char *strx=0;
    if(USART3_RX_STA&0X8000)        //接收到一次数据了
    {
        USART3_RX_BUF[USART3_RX_STA&0X7FFF]=0;//添加结束符
        strx=strstr((const char*)USART3_RX_BUF,(const char*)str);
    }
    return (u8*)strx;
}
//向 ATK-ESP8266 发送命令
//cmd:发送的命令字符串
//ack:期待的应答结果,如果为空,则表示不需要等待应答
//waittime:等待时间(单位:10ms)
```

```
//返回值:0,发送成功(得到了期待的应答结果)
//        1,发送失败
u8 atk_8266_send_cmd(u8 *cmd,u8 *ack,u16 waittime)
{
    u8 res=0;
    USART3_RX_STA=0;
    u3_printf("%s\r\n",cmd);     //发送命令
    if(ack&&waittime)       //需要等待应答
    {
        while(--waittime)   //等待倒计时
        {
            delay_ms(10);
            if(USART3_RX_STA&0X8000)//接收到期待的应答结果
            {
                if(atk_8266_check_cmd(ack))
                {
                    printf("ack:%s\r\n",(u8*)ack);
                    break;//得到有效数据
                }
                    USART3_RX_STA=0;
            }
        }
        if(waittime==0)res=1;
    }
    return res;
}
//向 ATK-ESP8266 发送指定数据
//data:发送的数据(不需要添加回车了)
//ack:期待的应答结果,如果为空,则表示不需要等待应答
//waittime:等待时间(单位:10ms)
//返回值:0,发送成功(得到了期待的应答结果)luojian
u8 atk_8266_send_data(u8 *data,u8 *ack,u16 waittime)
{
    u8 res=0;
    USART3_RX_STA=0;
    u3_printf("%s",data);    //发送命令
    if(ack&&waittime)       //需要等待应答
    {
        while(--waittime)   //等待倒计时
        {
            delay_ms(10);
            if(USART3_RX_STA&0X8000)//接收到期待的应答结果
            {
                if(atk_8266_check_cmd(ack))break;//得到有效数据
```

```
                    USART3_RX_STA=0;
                }
            }
            if(waittime==0)res=1;
        }
        return res;
}
//ATK-ESP8266 退出透传模式
//返回值:0,退出成功;
//          1,退出失败
u8 atk_8266_quit_trans(void)
{
        while((USART3->SR&0X40)==0);    //等待发送空
        USART3->DR='+';
        delay_ms(15);                   //大于串口组帧时间(10ms)
        while((USART3->SR&0X40)==0);    //等待发送空
        USART3->DR='+';
        delay_ms(15);                   //大于串口组帧时间(10ms)
        while((USART3->SR&0X40)==0);    //等待发送空
        USART3->DR='+';
        delay_ms(500);                  //等待 500ms
        return atk_8266_send_cmd("AT","OK",20);//退出透传判断.
}
//获取 ATK-ESP8266 模块的 AP+STA 连接状态
//返回值:0，未连接;1,连接成功
u8 atk_8266_apsta_check(void)
{
        if(atk_8266_quit_trans())return 0;          //退出透传
        atk_8266_send_cmd("AT+CIPSTATUS",":",50); //发送 AT+CIPSTATUS 指令,查询连接状态
        if(atk_8266_check_cmd("+CIPSTATUS:0")&&
            atk_8266_check_cmd("+CIPSTATUS:1")&&
            atk_8266_check_cmd("+CIPSTATUS:2")&&
            atk_8266_check_cmd("+CIPSTATUS:4"))
            return 0;
        else return 1;
}
//获取 ATK-ESP8266 模块的连接状态
//返回值:0,未连接;1,连接成功.
u8 atk_8266_consta_check(void)
{
        u8 *p;
        u8 res;
        if(atk_8266_quit_trans())return 0;          //退出透传
        atk_8266_send_cmd("AT+CIPSTATUS",":",50); //发送 AT+CIPSTATUS 指令,查询连接状态
```

```
    p=atk_8266_check_cmd("+CIPSTATUS:");
    res=*p;                                          //得到连接状态
    return res;
}
//键盘码表
const u8* kbd_tbl[13]={"1","2","3","4","5","6","7","8","9",".","0","#","DEL"};
u8* kbd_fn_tbl[2];
//加载键盘界面（尺寸为 240*140）
//x,y:界面起始坐标（320*240 分辨率的时候，x 必须为 0）
void atk_8266_load_keyboard(u16 x,u16 y)
{
    u16 i;
    POINT_COLOR=RED;
    LCD_Fill(x,y,x+240,y+140,WHITE);
    LCD_DrawRectangle(x,y,x+240,y+140);
    LCD_DrawRectangle(x+80,y,x+160,y+140);
    LCD_DrawRectangle(x,y+28,x+240,y+56);
    LCD_DrawRectangle(x,y+84,x+240,y+112);
    POINT_COLOR=BLUE;
    for(i=0;i<15;i++)
    {
        if(i<13)Show_Str_Mid(x+(i%3)*80,y+6+28*(i/3),(u8*)kbd_tbl[i],16,80);
        else Show_Str_Mid(x+(i%3)*80,y+6+28*(i/3),kbd_fn_tbl[i-13],16,80);
    }
}
//按键状态设置
//x,y:键盘坐标
//key:键值（0~8）
//sta:状态，0，松开；1，按下；
void atk_8266_key_staset(u16 x,u16 y,u8 keyx,u8 sta)
{
    u16 i=keyx/3,j=keyx%3;
    if(keyx>15)return;
    if(sta)LCD_Fill(x+j*80+1,y+i*28+1,x+j*80+78,y+i*28+26,GREEN);
    else LCD_Fill(x+j*80+1,y+i*28+1,x+j*80+78,y+i*28+26,WHITE);
    if(j&&(i>3))Show_Str_Mid(x+j*80,y+6+28*i,(u8*)kbd_fn_tbl[keyx-13],16,80);
    else Show_Str_Mid(x+j*80,y+6+28*i,(u8*)kbd_tbl[keyx],16,80);
}
//得到触摸屏的输入
//x,y:键盘坐标
//返回值：按键键值（1~15 有效；0,无效）
u8 atk_8266_get_keynum(u16 x,u16 y)
{
    u16 i,j;
```

```
        static u8 key_x=0;//0,没有任何按键按下；1~15，1~15 号按键按下
        u8 key=0;
        tp_dev.scan(0);
        if(tp_dev.sta&TP_PRES_DOWN)            //触摸屏被按下
        {
            for(i=0;i<5;i++)
            {
                for(j=0;j<3;j++)
                {

    if(tp_dev.x[0]<(x+j*80+80)&&tp_dev.x[0]>(x+j*80)&&tp_dev.y[0]<(y+i*28+28)&&tp_dev.y[0]>(y+i*2
8))
                    {
                        key=i*3+j+1;
                        break;
                    }
                }
                if(key)
                {
                    if(key_x==key)key=0;
                    else
                    {
                        atk_8266_key_staset(x,y,key_x-1,0);
                        key_x=key;
                        atk_8266_key_staset(x,y,key_x-1,1);
                    }
                    break;
                }
            }
        }else if(key_x)
        {
            atk_8266_key_staset(x,y,key_x-1,0);
            key_x=0;
        }
        return key;
    }
    //获取 Client ip 地址
    //ipbuf:ip 地址输出缓存区
    void atk_8266_get_wanip(u8* ipbuf)
    {
        u8 *p,*p1;
            if(atk_8266_send_cmd("AT+CIFSR","OK",50))//获取 WAN IP 地址失败
            {
                ipbuf[0]=0;
```

```
            return;
        }
        p=atk_8266_check_cmd("\"");
        p1=(u8*)strstr((const char*)(p+1),"\"");
        *p1=0;
        sprintf((char*)ipbuf,"%s",p+1);
}

//获取 AP+STA ip 地址并在指定位置显示
//ipbuf:ip 地址输出缓存区
void atk_8266_get_ip(u8 x,u8 y)
{
        u8 *p;
        u8 *p1;
        u8 *p2;
        u8 *ipbuf;
        u8 *buf;
        p=mymalloc(SRAMIN,32);                          //申请 32 字节内存
        p1=mymalloc(SRAMIN,32);                         //申请 32 字节内存
        p2=mymalloc(SRAMIN,32);                         //申请 32 字节内存
      ipbuf=mymalloc(SRAMIN,32);                        //申请 32 字节内存
        buf=mymalloc(SRAMIN,32);                        //申请 32 字节内存
        if(atk_8266_send_cmd("AT+CIFSR","OK",50))//获取 WAN IP 地址失败
        {
            *ipbuf=0;
        }
        else
        {
            p=atk_8266_check_cmd("APIP,\"");
            p1=(u8*)strstr((const char*)(p+6),"\"");
            p2=p1;
            *p1=0;
            ipbuf=p+6;
            sprintf((char*)buf,"AP IP:%s  端口:%s",ipbuf,(u8*)portnum);
            Show_Str(x,y,200,12,buf,12,0);              //显示 AP 模式的 IP 地址和端口
            p=(u8*)strstr((const char*)(p2+1),"STAIP,\"");
            p1=(u8*)strstr((const char*)(p+7),"\"");
            *p1=0;
            ipbuf=p+7;
            sprintf((char*)buf,"STA IP:%s  端口:%s",ipbuf,(u8*)portnum);
            Show_Str(x,y+15,200,12,buf,12,0);           //显示 STA 模式的 IP 地址和端口
            myfree(SRAMIN,p);       //释放内存
            myfree(SRAMIN,p1);      //释放内存
            myfree(SRAMIN,p2);      //释放内存
```

```
                myfree(SRAMIN,ipbuf);           //释放内存
                myfree(SRAMIN,buf);         //释放内存
            }
    }

    //ATK-ESP8266 模块信息显示
    //x,y:显示信息的起始坐标.
    //wanip:0,全部更新显示;1,仅更新 wanip.
    void atk_8266_msg_show(u16 x,u16 y,u8 wanip)
    {
        u8 *p,*p1,*p2;
        p=mymalloc(SRAMIN,32);                          //申请 32 字节内存
        p1=mymalloc(SRAMIN,32);                         //申请 32 字节内存
        p2=mymalloc(SRAMIN,32);                         //申请 32 字节内存
        POINT_COLOR=BLUE;
        atk_8266_send_cmd("AT+CWMODE=2","OK",20);
        atk_8266_send_cmd("AT+RST","OK",20);
        delay_ms(1000);//延时 2s 等待模块重启
        delay_ms(1000);//
        delay_ms(1000);
        delay_ms(1000);
        sprintf((char*)p,"AT+CWSAP=\"%s\",\"%s\",1,4",wifiap_ssid,wifiap_password);      //配置模块 AP
模式无线参数
        atk_8266_send_cmd(p,"OK",1000);
        if(wanip==0)//全更新
        {
            atk_8266_send_cmd("AT+GMR","OK",20);       //获取固件版本号
            p=atk_8266_check_cmd("SDK version:");
            Show_Str(x,y,240,16,"固件版本:",16,0);Show_Str(x+72,y,240,16,p,16,0);
            atk_8266_send_cmd("AT+CWMODE?","+CWMODE:",20);    //获取网络模式
            p=atk_8266_check_cmd(":");
            Show_Str(x,y+16,240,16,"网络模
式:",16,0);Show_Str(x+72,y+16,240,16,(u8*)ATK_ESP8266_CWMODE_TBL[*(p+1)-'1'],16,0);
        atk_8266_send_cmd("AT+CWSAP?","+CWSAP:",20);     //获取 wifi 配置
            p=atk_8266_check_cmd("\"");
            p1=(u8*)strstr((const char*)(p+1),"\"");
            p2=p1;
            *p1=0;
            Show_Str(x,y+32,240,16,"SSID 号:",16,0);Show_Str(x+56,y+32,240,16,p+1,16,0);
            p=(u8*)strstr((const char*)(p2+1),"\"");
            p1=(u8*)strstr((const char*)(p+1),"\"");
            p2=p1;
            *p1=0;
            Show_Str(x,y+48,240,16,"密码:",16,0);Show_Str(x+40,y+48,240,16,p+1,16,0);
```

```
            p=(u8*)strstr((const char*)(p2+1),",");
            p1=(u8*)strstr((const char*)(p+1),",");
            *p1=0;
            Show_Str(x,y+64,240,16,"通道号:",16,0);Show_Str(x+56,y+64,240,16,p+1,16,0);
            Show_Str(x,y+80,240,16,"加密方
式:",16,0);Show_Str(x+72,y+80,240,16,(u8*)ATK_ESP8266_ECN_TBL[*(p1+1)-'0'],16,0);
        }
        myfree(SRAMIN,p);          //释放内存
        myfree(SRAMIN,p1);         //释放内存
        myfree(SRAMIN,p2);         //释放内存
    }
    //ATK-ESP8266 模块 WIFI 配置参数显示(仅 WIFI STA/WIFI AP 模式测试时使用)
    //x,y:显示信息的起始坐标.
    //rmd:提示信息
    //ssid,encryption,password:无线网络的 SSID,加密方式,密码
    void atk_8266_wificonf_show(u16 x,u16 y,u8* rmd,u8* ssid,u8* encryption,u8* password)
    {
        POINT_COLOR=RED;
        Show_Str(x,y,240,12,rmd,12,0);//显示提示信息
        Show_Str(x,y+20,240,12,"SSID:",12,0);
        Show_Str(x,y+36,240,12,"加密方式:",12,0);
        Show_Str(x,y+52,240,12,"密码:",12,0);
        POINT_COLOR=BLUE;
        Show_Str(x+30,y+20,240,12,ssid,12,0);
        Show_Str(x+54,y+36,240,12,encryption,12,0);
        Show_Str(x+30,y+52,240,12,password,12,0);
    }
    //工作模式选择
    //返回值:
    //0,TCP 服务器
    //1,TCP 客户端
    //2,UDP 模式
    u8 atk_8266_netpro_sel(u16 x,u16 y,u8* name)
    {
        u8 key=0,t=0,*p;
        u8 netpro=0;
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
        p=mymalloc(SRAMIN,50);//申请 50 个字节的内存
        sprintf((char*)p,"%s  工作模式选择",name);
        Show_Str_Mid(0,y,p,16,240);
        Show_Str(x,y+30,200,16,"KEY0:下一个",16,0);
        Show_Str(x,y+50,200,16,"KEY1:上一个",16,0);
        Show_Str(x,y+70,200,16,"WK_UP:确定",16,0);
```

```
    Show_Str(x,y+100,200,16,"请选择:",16,0);
    POINT_COLOR=BLUE;
    Show_Str(x+16,y+120,200,16,"TCP 服务器",16,0);
    Show_Str(x+16,y+140,200,16,"TCP 客户端",16,0);
    Show_Str(x+16,y+160,200,16,"UDP 模式",16,0);
    POINT_COLOR=RED;
    Show_Str(x,y+120,200,16,"→",16,0);
    while(1)
    {
        key=KEY_Scan(0);
        if(key)
        {
            if(key==WKUP_PRES)break;            //WK_UP 按下
            Show_Str(x,y+120+netpro*20,200,16,"   ",16,0);//清空之前的显示
            if(key==KEY0_PRES)//KEY0 按下
            {
                if(netpro<2)netpro++;
                else netpro=0;
            }else if(key==KEY1_PRES)//KEY1 按下
            {
                if(netpro>0)netpro--;
                else netpro=2;
            }
            Show_Str(x,y+120+netpro*20,200,16,"→",16,0);//指向新条目

        }
        delay_ms(10);
        atk_8266_at_response(1);
        if((t++)==20){t=0;LED0=!LED0;}//LED 闪烁
    }
    myfree(SRAMIN,p);
    return netpro;//返回网络模式选择值
}

//STA 模式下的 AP 的 TCP、UDP 工作模式配置
u8 atk_8266_mode_cofig(u8 netpro)
{
    //u8 netpro;
    u8 ipbuf[16];  //IP 缓存
    u8 *p;
    u8 key;
    p=mymalloc(SRAMIN,32);//申请 32 个字节的内存
PRESTA:
    netpro|=(atk_8266_netpro_sel(50,30,(u8*)ATK_ESP8266_CWMODE_TBL[1]))<<4;    //网络模式
```

```
选择
        if(netpro&0X20)
        {
            LCD_Clear(WHITE);
            if(atk_8266_ip_set("WIFI-AP 远端 UDP IP 设置","UDP 模式",(u8*)portnum,ipbuf))goto
PRESTA;//IP 输入
            if(netpro&0X03)sprintf((char*)p,"AT+CIPSTART=1,\"UDP\",\"%s\",%s",ipbuf,(u8*)portnum);
//配置目标 UDP 服务器,及 ID 号，STA 模式下为 0
            else sprintf((char*)p,"AT+CIPSTART=0,\"UDP\",\"%s\",%s",ipbuf,(u8*)portnum);      //配置目
标 UDP 服务器,及 ID 号，STA 模式下为 0
            delay_ms(200);
            LCD_Clear(WHITE);
            atk_8266_send_cmd(p,"OK",200);
        }
        else if(netpro&0X10)        //AP TCP Client       透传模式测试
        {
            LCD_Clear(WHITE);
            POINT_COLOR=RED;
            Show_Str_Mid(0,30,"ATK-ESP WIFI-STA  测试",16,240);
            Show_Str(30,50,200,16,"正在配置 ATK-ESP 模块,请稍等...",12,0);
            if(atk_8266_ip_set("WIFI-AP  远端 IP 设置","TCP Client",(u8*)portnum,ipbuf))goto PRESTA;
    //IP 输入
            if(netpro&0X03)sprintf((char*)p,"AT+CIPSTART=1,\"TCP\",\"%s\",%s",ipbuf,(u8*)portnum);
//配置目标 TCP 服务器,及 ID 号，STA 模式为 client 时，为 1
            else sprintf((char*)p,"AT+CIPSTART=0,\"TCP\",\"%s\",%s",ipbuf,(u8*)portnum);      //配置目
标 TCP 服务器,及 ID 号，STA 模式为 server 时，为 0
            while(atk_8266_send_cmd(p,"OK",200))
                {
                    LCD_Clear(WHITE);
                    POINT_COLOR=RED;
                    Show_Str_Mid(0,40,"WK_UP:返回重选",16,240);
                    Show_Str(30,80,200,12,"ATK-ESP  连接 TCP SERVER 失败",12,0); //连接失败
                    key=KEY_Scan(0);
                    if(key==WKUP_PRES)goto PRESTA;
                }
        }
        else;    //服务器模式不用配置
        myfree(SRAMIN,p);
        return netpro;
    }
```

```
//IP 设置
//title:ip 设置标题
//mode:工作模式
//port:端口号
//*ip:ip 缓存区(返回 IP 给上层函数)
//返回值:0,确认连接;1,取消连接.
u8 atk_8266_ip_set(u8* title,u8* mode,u8* port,u8* ip)
{
    u8 res=0;
    u8 key;
    u8 timex=0;
    u8 iplen=0;                 //IP 长度
    LCD_Clear(WHITE);
    POINT_COLOR=RED;
    Show_Str_Mid(0,30,title,16,240);        //显示标题
    Show_Str(30,90,200,16,"工作模式:",16,0);  //工作模式显示
    Show_Str(30,110,200,16,"IP 地址:",16,0);   //IP 地址可以键盘设置
    Show_Str(30,130,200,16,"端口:",16,0);//端口号
    kbd_fn_tbl[0]="连接";
    kbd_fn_tbl[1]="返回";
    atk_8266_load_keyboard(0,180);              //显示键盘
    POINT_COLOR=BLUE;
    Show_Str(30+72,90,200,16,mode,16,0);      //显示工作模式
    Show_Str(30+40,130,200,16,port,16,0);      //显示端口
    ip[0]=0;
    while(1)
    {
        key=atk_8266_get_keynum(0,180);
        if(key)
        {
            if(key<12)
            {
                if(iplen<15)
                {
                    ip[iplen++]=kbd_tbl[key-1][0];
                }
            }else
            {
                if(key==13)if(iplen)iplen--;   //删除
                if(key==14&&iplen)break;          //确认连接
                if(key==15){res=1;break;}         //取消连接
            }
            ip[iplen]=0;
```

```
                LCD_Fill(30+56,110,239,110+16,WHITE);
                Show_Str(30+56,110,200,16,ip,16,0);              //显示 IP 地址
            }
            timex++;
            if(timex==20)
            {
                timex=0;
                LED0=!LED0;
            }
            delay_ms(10);
            atk_8266_at_response(1);//WIFI 模块发过来的数据,及时上传给电脑
        }
        return res;
}
//测试界面主 UI
void atk_8266_mtest_ui(u16 x,u16 y)
{
        LCD_Clear(WHITE);
        POINT_COLOR=RED;
        Show_Str_Mid(0,y,"ATK_ESP8266 WIFI 模块测试",16,240);
        Show_Str(x,y+25,200,16,"请选择:",16,0);
        Show_Str(x,y+45,200,16,"KEY0:WIFI STA+AP",16,0);
        Show_Str(x,y+65,200,16,"KEY1:WIFI STA",16,0);
        Show_Str(x,y+85,200,16,"WK_UP:WIFI AP",16,0);
        atk_8266_msg_show(x,y+125,0);
}



//ATK-ESP8266 模块测试主函数
void atk_8266_test(void)
{
//      u16 rlen=0;

        u8 key;

        POINT_COLOR=RED;
            //初始化函数

        Show_Str_Mid(0,30,"ATK-ESP8266 WIFI 模块测试",16,240);
        while(atk_8266_send_cmd("AT","OK",20))//检查 WIFI 模块是否在线
        {
            atk_8266_quit_trans();//退出透传
            atk_8266_send_cmd("AT+CIPMODE=0","OK",200);   //关闭透传模式
```

```
        Show_Str(40,55,200,16,"未检测到模块!!!",16,0);
        delay_ms(800);
        LCD_Fill(40,55,200,55+16,WHITE);
        Show_Str(40,55,200,16,"尝试连接模块...",16,0);
    }
        while(atk_8266_send_cmd("ATE0","OK",20));//关闭回显
        atk_8266_mtest_ui(32,30);
    while(1)
    {
        delay_ms(10);
        atk_8266_at_response(1);//检查 ATK-ESP8266 模块发送过来的数据,及时上传给电脑
        key=KEY_Scan(0);
        if(key)
        {
            LCD_Clear(WHITE);
            POINT_COLOR=RED;
            switch(key)
            {
                case 1://KEY0
                    Show_Str_Mid(0,30,"ATK-ESP WIFI-AP+STA 测试",16,240);
                    Show_Str_Mid(0,50,"正在配置 ATK-ESP8266 模块，请稍等...",12,240);
                    atk_8266_apsta_test();  //串口以太网测试
                    break;
                case 2://KEY1
                    Show_Str_Mid(0,30,"ATK-ESP WIFI-STA 测试",16,240);
                    Show_Str_Mid(0,50,"正在配置 ATK-ESP8266 模块，请稍等...",12,240);
                    atk_8266_wifista_test();//WIFI STA 测试
                    break;
                case 4://WK_UP
                    atk_8266_wifiap_test(); //WIFI AP 测试
                    break;
            }
            atk_8266_mtest_ui(32,30);
        }

    }
```

Wisita.c:

```
#include "common.h"
#include "stdlib.h"
#include "beep.h"
#include "led.h"
#include "key.h"
#include "pwm.h"
```

```
#include "delay.h"
#include "sys.h"
#include "usart.h"
#include "HCSR04.h"
#include "dht11.h"
#include "MQ2.h"
#include "math.h"
#include "stm32f4xx.h"
//////////////////////////////////////////////////////////////////////////////////////////////////////
//本程序只供学习使用，未经作者许可，不得用于其它任何用途
//ALIENTEK STM32 开发板
//ATK-ESP8266 WIFI 模块  WIFI STA 驱动代码
//正点原子@ALIENTEK
//技术论坛:www.openedv.com
//修改日期:2015/4/3
//版本：V1.0
//版权所有，盗版必究。
//Copyright(C) 广州市星翼电子科技有限公司  2009-2019
//All rights reserved
//////////////////////////////////////////////////////////////////////////////////////////////////////

//ATK-ESP8266 WIFI STA 测试
//用于测试 TCP/UDP 连接
//返回值:0,正常
//       其他,错误代码
u8 netpro=0;//网络模式

u8 atk_8266_wifista_test(void)
{
    //u8 netpro=0;    //网络模式
    double length=0;
    u8 temperature;
    u8 humidity;
    u16 adcx;
    double a=0.0;
    u8 flag=0;
    u8 flag2=0;
    u8 flag1=0;
    u8 key;
    u8 ledstate=0;
    u8 ledstate1=0;
    u8 timex=0;
    u8 ipbuf[16];     //IP 缓存
    u8 *p;
```

```
    u16 t=999;          //加速第一次获取链接状态
    u8 res=0;
    u16 rlen=0;
    u8 constate=0;    //连接状态
    u16 number;
    u16 num=1000;
    u16 led1pwmval=1500;
    //char state;
    char formerstate='0';
    TIM14_PWM_Init(2000-1,84-1);
    LED0=1;
    LED1=1;
    DIR=0;

    p=mymalloc(SRAMIN,32);                                  //申请 32 字节内存
    atk_8266_send_cmd("AT+CWMODE=1","OK",50);           //设置 WIFI STA 模式
    atk_8266_send_cmd("AT+RST","OK",20);       //DHCP 服务器关闭(仅 AP 模式有效)
    delay_ms(1000);                 //延时 3S 等待重启成功
    delay_ms(1000);
    delay_ms(1000);
    delay_ms(1000);
    //设置连接到的 WIFI 网络名称/加密方式/密码,这几个参数需要根据您自己的路由器设置进行修改!!
    sprintf((char*)p,"AT+CWJAP=\"%s\",\"%s\"",wifista_ssid,wifista_password);//设置无线参数:ssid,密码
    while(atk_8266_send_cmd(p,"WIFI GOT IP",300));                  //连接目标路由器,并且获得 IP
PRESTA:
    netpro|=atk_8266_netpro_sel(50,30,(u8*)ATK_ESP8266_CWMODE_TBL[0]);    //选择网络模式
    if(netpro&0X02)     //UDP
    {
                LCD_Clear(WHITE);
                POINT_COLOR=RED;
                Show_Str_Mid(0,30,"ATK-ESP WIFI-STA  测试",16,240);
                Show_Str(30,50,200,16,"正在配置 ATK-ESP 模块,请稍等...",12,0);
                if(atk_8266_ip_set("WIFI-STA  远端 UDP IP 设置",(u8*)ATK_ESP8266_WORKMODE_TBL[netpro],(u8*)portnum,ipbuf))goto PRESTA;   //IP 输入
                sprintf((char*)p,"AT+CIPSTART=\"UDP\",\"%s\",%s",ipbuf,(u8*)portnum);//配置目标 UDP 服务器
                delay_ms(200);
                atk_8266_send_cmd("AT+CIPMUX=0","OK",20);   //单链接模式
                delay_ms(200);
```

```
                LCD_Clear(WHITE);
                while(atk_8266_send_cmd(p,"OK",500));
    }
    else        //TCP
    {
        if(netpro&0X01)        //TCP Client       透传模式测试
        {
            LCD_Clear(WHITE);
            POINT_COLOR=RED;
            Show_Str_Mid(0,30,"ATK-ESP WIFI-STA  测试",16,240);
            Show_Str(30,50,200,16,"正在配置 ATK-ESP 模块,请稍等...",12,0);
            if(atk_8266_ip_set("WIFI-STA  远端 IP 设置
",(u8*)ATK_ESP8266_WORKMODE_TBL[netpro],(u8*)portnum,ipbuf))goto PRESTA;   //IP
输入
            atk_8266_send_cmd("AT+CIPMUX=0","OK",20);     //0：单连接，1：多连接
            sprintf((char*)p,"AT+CIPSTART=\"TCP\",\"%s\",%s",ipbuf,(u8*)portnum);       //
配置目标 TCP 服务器
            while(atk_8266_send_cmd(p,"OK",200))
            {
                    LCD_Clear(WHITE);
                    POINT_COLOR=RED;
                    Show_Str_Mid(0,40,"WK_UP:返回重选",16,240);
                    Show_Str(30,80,200,12,"ATK-ESP  连接 TCP 失败",12,0); //连接失败
                    key=KEY_Scan(0);
                    if(key==WKUP_PRES)goto PRESTA;
            }
            atk_8266_send_cmd("AT+CIPMODE=1","OK",200);         //传输模式为：透传

        }
        else                    //TCP Server
        {
                LCD_Clear(WHITE);
                POINT_COLOR=RED;
                Show_Str_Mid(0,30,"ATK-ESP WIFI-STA  测试",16,240);
                Show_Str(30,50,200,16,"正在配置 ATK-ESP 模块,请稍等...",12,0);
                atk_8266_send_cmd("AT+CIPMUX=1","OK",20);     //0：单连接，1：多连接
                sprintf((char*)p,"AT+CIPSERVER=1,%s",(u8*)portnum);       //开启 Server 模
式(0，关闭；1，打开)，端口号为 portnum
                atk_8266_send_cmd(p,"OK",50);
        }
    }
            LCD_Clear(WHITE);
            POINT_COLOR=RED;
            Show_Str_Mid(0,30,"ATK-ESP WIFI-STA  测试",16,240);
```

```
            Show_Str(30,50,200,16,"正在配置 ATK-ESP 模块,请稍等...",12,0);
            LCD_Fill(30,50,239,50+12,WHITE);            //清除之前的显示
            Show_Str(30,50,200,16,"WK_UP:退出测试   KEY0:发送数据",12,0);
            LCD_Fill(30,80,239,80+12,WHITE);
            atk_8266_get_wanip(ipbuf);//服务器模式,获取 WAN IP
            sprintf((char*)p,"IP 地址:%s  端口:%s",ipbuf,(u8*)portnum);
            Show_Str(30,65,200,12,p,12,0);              //显示 IP 地址和端口
            Show_Str(30,80,200,12,"状态:",12,0);       //连接状态
            Show_Str(120,80,200,12,"模式:",12,0);      //连接状态
            Show_Str(30,100,200,12,"发送数据:",12,0);  //发送数据
            Show_Str(30,115,200,12,"接收数据:",12,0);  //接收数据
            atk_8266_wificonf_show(30,180,"请设置路由器无线参数
为:",(u8*)wifista_ssid,(u8*)wifista_encryption,(u8*)wifista_password);
            POINT_COLOR=BLUE;

    Show_Str(120+30,80,200,12,(u8*)ATK_ESP8266_WORKMODE_TBL[netpro],12,0);
    //连接状态
            USART3_RX_STA=0;
            while(1)
            {
                            //获取距离
                length=GetHCSR04_Distance();
                            //获取温湿度
                DHT11_Read_Data(&temperature,&humidity);
                            //获取烟雾浓度
                adcx=Get_Adc_Average(ADC_Channel_5,20);//获取通道 5 的转换值，20 次
取平均
                ledstate=GPIO_ReadOutputDataBit(GPIOF,GPIO_Pin_3);
                ledstate1=GPIO_ReadOutputDataBit(GPIOF,GPIO_Pin_10);
                if(adcx >1500)
                {
                    GPIO_SetBits(GPIOF,GPIO_Pin_8);
                    delay_ms(300);
                    GPIO_ResetBits(GPIOF,GPIO_Pin_8);
                    delay_ms(300);
                }
                printf("\n");
                a=length;

                if(flag1==0&&a<15)
                {
                    while(1){
                    flag2=0;
                    delay_ms(10);
```

```
            LED0=0;
            LED1=0;
            delay_ms(num);
            DIR=1;
            flag++;
                number=7000/num;
            if(flag>=number){
                flag1=1;
                delay_ms(num);
            TIM_SetCompare1(TIM14,0);
                break;
            }
        else
            {
                TIM_SetCompare1(TIM14,led1pwmval);
                delay_ms(10);
            }
        }
}
if(flag1==1&&a>15)
    {
        while(1)
        {
            flag=0;
            delay_ms(1000);
            LED0=1;
            LED1=1;
            DIR=0;
            flag2++;
            if(flag2>=7){
                flag1=0;
                TIM_SetCompare1(TIM14,0);
                break;
            }
            else
            {
                TIM_SetCompare1(TIM14,led1pwmval);
                delay_ms(10);

            }

        }
}
    key=KEY_Scan(0);
```

```
                if(key==WKUP_PRES)              //WK_UP 退出测试
                {
                    res=0;
                    atk_8266_quit_trans();    //退出透传
                    atk_8266_send_cmd("AT+CIPMODE=0","OK",20);    //关闭透传模式
                    break;
                }
            else if(10==t)    //TCP Server
                {

    sprintf((char*)p,"%02d%02d%04d%1d%1d",temperature,humidity,adcx,ledstate,ledstate1);//
测试数据
                        Show_Str(30+54,100,200,12,p,12,0);
                        atk_8266_send_cmd("AT+CIPSEND=0,10","OK",200);  //发送指定
长度的数据
                        delay_ms(200);
                        atk_8266_send_data(p,"OK",100);  //发送指定长度的数据
                        timex=50;
                }else;

                if(timex)timex--;
                if(timex==1)LCD_Fill(30+54,100,239,112,WHITE);
                t++;
                delay_ms(10);
                if(USART3_RX_STA&0X8000)        //接收到一次数据了
                {
                    rlen=USART3_RX_STA&0X7FFF;    //得到本次接收到的数据长度
                    USART3_RX_BUF[rlen]=0;        //添加结束?


                    if('1'==USART3_RX_BUF[rlen-1])
                        {
                            formerstate='1';

                        }
                    else if('0'==USART3_RX_BUF[rlen-1])
                        {
                           formerstate='0';

                        }
                    else;

                if(formerstate=='0')
                    {
```

```
                                GPIO_SetBits(GPIOF,GPIO_Pin_10);
                        }
                    else
                        {

                                GPIO_ResetBits(GPIOF,GPIO_Pin_10);
                        }



                        printf("%s",USART3_RX_BUF);  //发送到串口
                        sprintf((char*)p,"收到%d 字节,内容如下",rlen);//接收到的字节数
                        LCD_Fill(30+54,115,239,130,WHITE);
                        POINT_COLOR=BRED;
                        Show_Str(30+54,115,156,12,p,12,0);              //显示接收到的数据长度
                        POINT_COLOR=BLUE;
                        LCD_Fill(30,130,239,319,WHITE);
                        Show_Str(30,130,180,190,USART3_RX_BUF,12,0);//显示接收到的数据
                        USART3_RX_STA=0;
                        if(constate!='+')t=1000;          //状态为还未连接,立即更新连接状态
                        else t=0;                                   //状态为已经连接了,10 秒后再检查
                    }
                    if(t==1000)//连续 10 秒钟没有收到任何数据,检查连接是不是还存在.
                    {
//                  //          LCD_Fill(30+54,125,239,130,WHITE);
//                              LCD_Fill(60,80,120,92,WHITE);
                        constate=atk_8266_consta_check();//得到连接状态
                        if(constate=='+')Show_Str(30+30,80,200,12,"连接成功",12,0);   //连接状
态
                        else Show_Str(30+30,80,200,12,"连接失败",12,0);
                        t=0;
                    }

                    atk_8266_at_response(1);

                }
    myfree(SRAMIN,p);       //释放内存
    return res;}
```