华南农业大学

本科毕业论文(设计)

基于 i.MX6ULL 和 STM32 的智能家居系统设计

韦汉清

201621190517

指导教师　**严炳辉 讲师**

学 院 名 称	**电子工程学院** **人工智能学院**	专 业 名 称	**电子信息科学与技术**
论文提交日期	2020 年 4 月 28 日	论文答辩日期	2020 年 5 月 12 日

摘　　要

就像农业现代化的目的一样，住宅自动化、智能化的目的同样是减少重复的、无意义的、费时的人力活动，让生活更加方便、高效，同样对提高人们生活质量有着重要的现实意义，它的发展已成为一种必然趋势。近年来，智能家居行业有了一定的发展，甚至接近百花争鸣的状态，但现状不容乐观，落地面临困境。

本文研究分析智能家居行业发展的现状，并设计一个相对完善的智能家居系统，最后采用 NXP 公司的适用于家居自动化的 ARM Cortex A7 CPU 核心的 i.MX6ULL 系列 SoC 和 ST 公司的 STM32 系列 MCU/SoC，实现部分子系统。本文实现的智能家居子系统有三个：摄像头监控子系统、无线智能灯光控制子系统、无线温湿度监测子系统。摄像头监控子系统独立在 i.MX6ULL SoC 上，后两个子系统集成在同一个 STM32 MCU 上作为监测显示兼并智能控制功能的独立终端，两者不互相依赖，灵活搭配，搭配互联网、云服务器、网页端、手机客户端，在家居局域网内外安全、方便地使用。

关键词　　智能家居　　i.MX6ULL　　STM32　　摄像头监控

Design of Smart Home based on I.MX6ULL and STM32

Wei Hanqing

(College of Electronic Engineering, South China Agricultural University, Guangzhou 510642, China)

Abstract: Just like the purpose of agricultural modernization, the purpose of housing automation and intellectualization is to reduce repetitive, meaningless and time-consuming human activities and make life more convenient and efficient. It is also of great practical significance to improve people's life quality. Its development has become an inevitable trend. In recent years, the smart home industry has developed to a certain extent, even close to the state of hundred flowers contending, but the current situation is not optimistic, landing is facing difficulties.

This paper studies and analyzes the current situation of the development of smart home industry, and designs a relatively perfect smart home system. Finally, i.mx6ull series SOC of NXP company, which is suitable for home automation arm cortex A7 CPU core, and STM32 Series MCU / SOC of ST company are used to realize some subsystems. There are three subsystems in this paper: camera monitoring subsystem, wireless intelligent light control subsystem, wireless temperature and humidity monitoring subsystem. The camera monitoring subsystem is independent on i.mx6ll SOC, and the latter two subsystems are integrated on the same STM32 MCU as independent terminals for monitoring display and merging intelligent control functions. The two subsystems are independent and flexible. They are used in combination with Internet, cloud server, web page and mobile phone clients safely and conveniently inside and outside the home LAN.

Key words: smart home i.mx6ull stm32 camera monitoring

目　　录

1 前言

1.1 课题的研究背景

随着计算机技术、通讯技术等电子信息技术的快速发展下，我们的生活中不断地出现各种智能化的产品、设备。电子信息科学技术产品、设备在提高社会生产力和科学技术水平的同时它们的智能化程度也在不断地提高，使人们的生活、生产更加便利、高效，深深地影响着人们的生活习惯。

多年来，智能家居虽然有了一定的发展，但现状并不乐观，总体上看，许多的智能家居实现了自动化、网络化，但是离“安全、舒适、便利”还有很大的差距，离真正满足需求的智能还需要很多的改进。面对个性化的家居场景，就如家居无线网络化，用户体验十分依赖于网络连接的稳定性和可靠性，相对于普通自动化家居而言，首先它的安装成本相对高了不少，并且它对于售后和运维的人员素质要求较高、故障排查时效和难度也在倍增。其次，网络化的软件部署基本都沿用了传统互联网的软件，服务端如 redis、mysql 数据库、Apache Httpd（http 服务器）等，而相对于零散而小的家居传感网络数据而言，我们并不需要如此“庞大”的软件工具，我们需要针对物联网的定制化的耗费资源更小的软件工具，减少不必要的软件、硬件资源，综合起来降低部署、运营成本。智能家居的部署跟互联网相比，多了家居智能硬件这部分。传统互联网的通信数据包的收发基于统一标准的 TCP/IP 协议栈，因为标准统一，从而互联网在世界范围内可以迅速的被部署使用。而诸多不同种类的物联网家居产品是碎片化并且相互独立的，各个厂商的产品的各种标准不一致、互不兼容导致整个智能家居系统生态复杂，其落地实施环节也依然存在大量的安全隐患以及影响用户体验，针对这些由于厂商、产品的碎片化以及标准统一困难的问题，智能家居行业还在探索中，相关标准也还在商讨拟定中。

本文针对家居环境监测、灯光智能控制、视频监控进行智能家居系统设计，以及针

对智能家居设备研究和设计了更轻量的 http 服务器和数据库。

1.2 课题的研究意义

在过去，电子技术家居产品中不断发展，近年来不断地更新到了自动化式家居产品，而互联网、无线传感网络、自动化、人工智能等技术的飞速发展给现代人们的生活带来了全新的体验，家居设备的无线网络化、节能环保、个性化、智能化在不断地成为主流。就像现代化农业的发展一样，在互联网的影响之下，智能家居发展成为了一个新生产业，市场需求潜力巨大。

智能家居通过物联网技术将家中的各种设备连接到一起，提供无线控制、自动控制等多种功能。与普通家居相比，智能家居不仅具有传统的居住功能，兼备建筑、网络通信、信息家电、设备自动化，提供全方位的信息交互功能，甚至为各种能源费用节约资金（李天祥，2016）。

近年来，我国老龄化趋势日趋严重，老龄化问题已成为社会热点问题之一。为应对老龄化带来的社会问题，国家提出了“9073”的养老格局，即 90%的老年人在家中养老，7%的老年人在社区养老，3%的老年人在养老机构中养老（何迎朝,邢文华，2020）。随着大数据、互联网、智能技术的发展，智慧家居的养老方式改变了以往的养老发展方式。智慧居家养老，是指利用互联网、物联网、云计算等先进的信息技术，通过互联网服务、智能感知设备等手段，最大限度地实现老人的生理健康指标、位置信息、活动量等各类传感器通过互联网和数据中心随时随地实时连接，同时让其子女和家人、养老机构等服务机构和志愿者等相关人员能远程实现对居家老人的安全看护、健康监测、生活服务和精神关爱等服务，能让老人在日常生活中不受时间和地理环境的限制，在家中过上高质量老年生活（杨波等，2017）。

其次，从育儿方面来看，作为父母我们并不总是能够在孩子身边看护着他们，比如外出上班，而利用智能家居的一系列信息技术，家长可以在外出时远程地对孩子进行监护。通过摄像头，我们可以远程地监看着家中的孩子的情况；通过查看传感器数据，我们可以查看家里的物理环境是否安全并选择远程控制家居电器设备；通过智能音箱或电

话手表，我们可以跟孩子进行远程对话；通过智能手表或其它穿戴设备，我们可以随时查看孩子的体温、心率、血氧浓度、地理位置等有关安全的参数。在家长留小孩独自在家中时，智能家居技术简直是家长履行监护责任的一手“利器”。

智能家居能使人们的生活更加地安全、舒适、便利，这不仅对生活能源的统计管理和提高人民的生活水平具有重要意义，同时也是解决当今我国趋于严重的人口老龄化问题和家长育儿问题的重要的科学技术手段。

1.3 课题的方案分析

智能家居的场景主要有门厅、楼道、厨房、卧室、浴室，一个完善的智能家居系统应该包含家居环境监测、防盗自动报警、家具智能控制、近远程监控、语音控制、燃气泄漏远程报警等功能，同时还应考虑到安全性、稳定性、成本、环保和易用性。

综合地看，整个智能家居系统可分为数据采集与自动控制系统、视觉人机交互系统、语音交互系统、图像监控系统、报警系统五个子系统类别：

1).数据采集与自动控制系统对家居环境主要参数(如温湿度、自然环境光照状态、燃气状态，红外感应状态)进行环境数据采集和家居设备的自动控制；

2).视觉人机交互系统则将家居设备的数据和控制以屏幕、网络等方式对接到用户；

3).语音交互系统提供非常方便的、智能的家居设备语音控制等功能；

4).图像监控系统作为防盗系统前端以及作为查看家居室内近远程工具，如家长不在家时可以远程察看家里的子女及家居环境；

5).报警系统则作为安防系统的后端及时给予通知警告。

基于时间和工作量考虑，本文选择 1)、2)、4)进行智能家居设计和实现。

1.4 课题的主要内容

本课题的主要内容为针对家居环境监测、灯光智能控制、摄像头监控（视频监控前端系统），进行智能家居系统设计，并实现该智能家居系统设计。这个系统最终的目标

是实现：

(1).传感监控终端：自动收集室内环境的光照度、温度、湿度、照明灯状态的数据，能够控制灯光的开与关，并在 LCD 屏幕实时显示，同时用户也可以在手机打开网页或 APP 查看实时数据并控制室内灯具，需要建设网站、制作手机 APP；

(2).摄像头监控设备：利用摄像头对家居环境进行图像的抓取、处理，以及为上层应用程序提供操作接口或方式，配合报警软件系统可形成完整的视频监控系统，它在本设计的主要功能是在监控范围内检测移动物体并自动抓拍保存视频图像，用户可以通过浏览器或手机 APP 观看实时的视频图像，同样需要建设网站、制作手机 APP。

1.5 小结

本章节对智能家居的背景和意义进行了研究，并对完善的智能家居系统设计进行了分析，最后确定了本文的主要内容。

2 总体方案确定

2.1 嵌入式技术

相对地，与通用计算机技术相反，嵌入式（计算机）技术为定制化功能而生，是针对特定功能进行定制化的软件、硬件设计的一种技术归类，这种情景下的软件则一般被称为嵌入式软件。

在一个设备中，嵌入式软件与运行它的硬件（器件）构成的整体被称为嵌入式系统，换一句话说则是，运行嵌入式软件的硬件（器件）是一个嵌入式系统。如今，这种硬件的核心部分通常为一块单芯片。这种芯片通常集成了 CPU（Central Processing Unit）、中断控制器、CPU 总线、SRAM（Static Random Access，静态随机访问内存）、NOR FLASH（或非结构的半导体存储器/闪存）、供电电路、定时器、串行通信总线等器件。根据功能针对性和集成度的不同，它们被称为 MCU（Microcontroller Unit）或 SoC

（System(s) On a Chip）。这种针对定制化需求的、包含处理单元的芯片通常被归类为嵌入式处理器，以跟通用计算机处理器区分开来。

在嵌入式技术领域中的，英国 ARM（Advanced RISC Machines）公司设计的集成电路模块（尤其是 CPU）占据的绝大部分的市场。就像软件公司将自己的软件以库的形式授权给其他人的方式一样，ARM 公司将它的 CPU 等电路设计作为 IP 模块授权给世界各个芯片设计厂商。从 V7 版本架构开始，ARM 公司将它的 CPU IP 重新进行了分类：

(1).A 系列: Application profile，支持 ARM 和 Thumb 指令集，要求硬件的内存管理模型支持虚拟内存地址。

(2).R 系列: Real-time profile,支持 ARM 和 Thumb 指令集，要求硬件的内存管理模型只支持物理地址。

(3).M 系列: microcontroller profile,只支持 Thumb 指令集，指令的大小和操作的准确完成比绝对性能更为重要。（ARM）

芯片设计厂商们利用这些硬件 IP（可以看作为 Verilog 语言写的一个库）进行二次设计，根据功能加入一个或多个 CPU、添加 SRAM、添加中断控制器等功能模块展开芯片设计，然后芯片设计厂商（通过芯片制造商）将它们集成到一个或多个大规模集成电路，然后封装成一个或多个芯片。下游制造商则根据产品功能挑选芯片，设计电路板，开发软件。

在本设计中主要使用 ARM 公司 CPU 核心的芯片,我们需要对芯片进行对比和选择、对嵌入式软件技术进行对比和选择。

2.1.1　系统主芯片的比较和选型

(1).微处理器（microprocessor）

简单地说，在上世纪 80 年代，微处理器的概念初次出现，当时我们的技术仅仅能

将 CPU 集成、封装到一个芯片里，那时候人们将微处理器归类为一种单芯片（而不是由芯片组构成）的 CPU 芯片类型。如今，它通常作为微处理器单元 MPUs（Microprocessor Units）与其它器件集成到一起作为一个整体芯片的 Processor，当然，还是用来处理二进制系统中的数字和符号。

在上世纪（20 世纪）60 年代末，设计者们致力于将计算机中中央处理单元功能（central processing unit functions）集成到多个 MOS（metal-oxide-semiconductor，金属氧化物半导体）大规模集成电路芯片，这种芯片组合被称为 MPU（microprocessor unit）芯片组（Altman Laurence）。在 1968-1970 年间，多芯片的微处理器陆续出现。1971 年，第一个微处理器 Intel 4004 发布（于一块 MOS 大规模集成电路芯片，如图 1）（Augarten S., 1983），从此之后的数年，微处理器被用来形容单芯片的 CPU 芯片，微处理器单元则一夜成为了主流的 CPU 类型。如今除了个别爱好者或研究者可能会用 74LS138 这样的芯片群来组成单板 CPU 外，市面上所有的 CPU 类型都已经是 MPU。

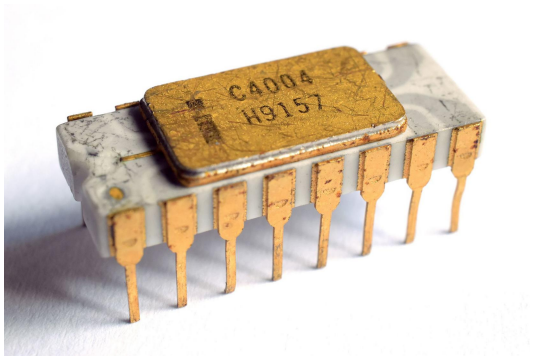


图 1　Intel C4004 微处理器

现在，像 Intel 4004 这样的单芯片微处理器已经几乎无迹可循了。随着集成电路技术的发展，我们已经不只是能把 CPU 集成、封装到单块芯片了。

通常地，现在我们更多的是用微处理器或微处理器单元来形容高性能的单芯片方案里的 CPU 或 CPU Platfrom（通常包含高性能 CPU 核、SRAM、中断控制器、内存管理单元、Program Trace Macrocell），它通用和中断控制器、定时器、用于缓存（cache）的 SRAM、MMU（Memeroy Management Unit）集成在一起。在通用计算技术领域中，微处理器单元通常和其它单元如 DRAM 控制器、GPU（graphic processor unit）封装在一起，也有极少数的厂商将微处理器单元和个人电脑中几乎所有需求的器件如 USB、SATA、DP/HDMI、PCIe 控制器等封装到一起做成一个单芯片解决方案，比如兆芯公司的无需主板芯片组的 x86 架构 KX-6000 系列 Processor，如下图 2 所示：

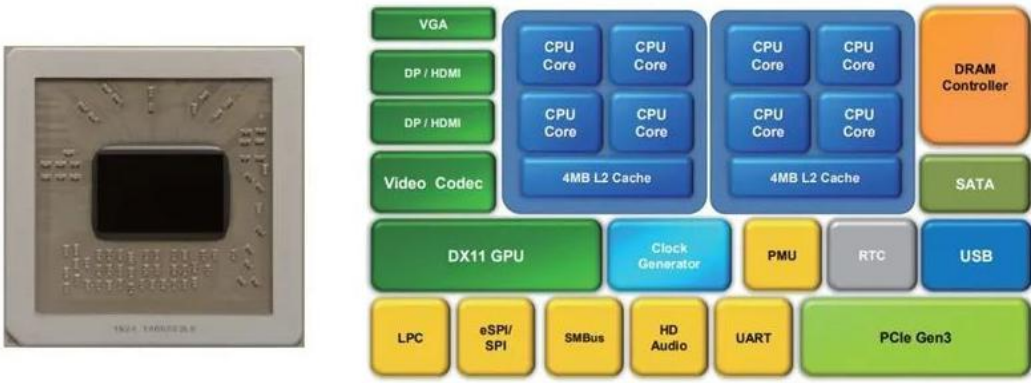


图 2　兆芯开先 KX-6000 系列处理器——芯片架构

而在嵌入式技术领域中，微处理器单元通常作为高性能 CPU 和其它计算机器件、其它可被集成的计算机外围器件被集成到一片或多片大规模集成电路，然后这样的一片或多片集成电路被封装成一块丰富功能的芯片，最后被芯片原厂命名为适用于高性能、低功耗的 XX MPU 系列的处理器，批量发布销售。可以说 MPU 在当代已经成为了“高性能”的代名词。

(2).单片机（单片微型计算机）

单片微型计算机，原名 Sigal-chip Microcomputer，与“单板机”相对应，概念诞生于上世纪，意如其名。单芯片的它内部一片或多片硅片上集成了 CPU、RAM、ROM、I/O 接口和中断系统电路，构成了完整的计算机，相比当年房子那么大的计算机，它非常地小巧，以至于它能够被嵌入各种设备里发挥它计算处理二进制数的功能，非常经典的代表作为上世纪 80 年代 Intel 公司发明的 8051 系列单片机。如今，随着技术的发展，越来越多的器件电路可以被集成到单晶硅片上，8051 也不再停滞当年，现在它们统统都换了个名字（根据应用领域）：Microcontroller（微控制器）。

“单片机”一词在国内仍被许多人沿用来形容 microcontroller（微控制器）或 microcontroller unit（MCU），实际上，国外（发源地）并没有沿用它来形容微控制器。单片机和微控制器（microcontroller）并不能等同，其中“微控制器”的针对性更强、集成度更高、集成内容更丰富。如一个 ARM Cortex R CPU 核心的 MCU，它可能是作为一个集成了计算机外围器件的 microcontroller 作为一个磁盘控制器，又或被集成在一个 SoC 里作为专门以软件方式进行电磁波信号调制调解器件的调制调解单元，又或是作为内置 NAND FLASH（与非结构半导体存储器/闪存）controller unit 的 microcontroller unit。

通常地，我们与一些老前辈、非专业人士用“单片机”意指 MCU、用“CPU”意指 SoC 进行沟通会更容易交流。由于此论文环境属于专业场合，为了专业性，针对微控制器、MCU 不再引用“单片机”一词。

(3).微控制器（microcontroller）

在现代，集成 CPU、RAM、地址总线、中断控制器、存储器、定时器、各种外设、外围器件以及其它器件的 IC 被称为 microcontroller 或 system on a chip（SoC），微控制器是从功能领域角度出发的命名。微控制器的内部组成的主要单元被称为 MCU

（microcontroller unit），当然，根据需求，它可能也集成了 DSP（Digital Sigal Processor，数字信号处理器）、FPU（Floating Point Unit，浮点运算单元）、传感器电路等等，它们被封装成的芯片实体即微控制器。正和其的命名的意思相同，它是专门针对控制领域的 IC 芯片。

相对于它的内部 CPU 而言，通常地，ARM 公司的 Cortex M 系列的 CPU 被应用于此，其指令的大小和操作的准确完成比绝对性能更为重要，非常适合在物联网传感层用作自动控制单元。常见的微控制器制造商有 ST（意法半导体）、NXP（恩智浦）、GD（兆易创新）、MTK（联发科技）、Allwinner（全志科技）、乐鑫、安信可等。

(4).片上系统（System(s) On a Chip）

它的定义参考上面“microcontroller”的定义，侧重于形容一个芯片里集成了一个或多个电路系统。它与微控制器的定义相似，实际上，它们之间最大的区别就是它们是对一个芯片从不同角度的命名。微控制器是从功能“控制”来针对性的一个定义，而 SoC 则是从“单片芯片”的角度来命名的，侧重点是“System”，它们的概念并不冲突，我们可以看到 Linux 基金会托管的 Zephyr OS 的主页对 ST 公司的 STM32L 系列微控制器以及 ESP 系列无线芯片定义为 SoC 型 MCU。下图 3 所示内容为 Linux 基金会托管的 Zephyr OS 的开发板支持列表的 ST Nucleo L496ZG board，从应用领域看，它主控芯片 STM32L496Z 是微控制器，从集成度来看，它集成了一个或多个 System (on a chip)。

图 3　STM32L496ZG SoC（MCU）评估板.（Zephyr OS 官网）

通常地，微控制器的系统复杂度一般相对较低，因为侧重于控制的它不需要集成太多电路系统即可满足需求，相反，倒不如说它需要集成更少的器件以满足它低成本的要求。一般我们口中的 SoC 则用来形容集成了复杂电路系统的单块芯片，通常用来运行复杂的应用程序，就如我们日常使用的手机芯片里集成了包含 ARM Cortex A 系列 CPU IP Core 的微处理器单元（MPU）在内的多种电路系统，相比微控制器，手机 SoC 内部集成了更多的 MCU（微控制器单元），因为手机里面需要被控制的片上/板上器件更多。

总的来说，它是对集成一个或多个电路系统的芯片（归类），通常地，这类芯片专门针对高性能需求被批量生产，十分适合运行复杂的应用程序，因此它非常适合在智能家居系统中用于运行复杂的视频监控程序。在嵌入式技术领域，它通常集成了至少一个 ARM Cortex A 系列的 CPU。

(5).模块系统（System-on-Module）

简称 SoM。为了将若干功能的电路系统集合到一起（模块化地）制造并向下游厂商销售，却在某些电路系统不能被集成到大规模集成电路上（并封装为单一芯片）的时候，

芯片原厂将集合一定功能的电路系统整合到一块单板上，这个整体作为一个模块进行销售，集合在里面的功能电路系统则被称为 System-on-Module。

(6).可编程逻辑门阵列（Field Programmable Logic Gate Array）

简称 FPGA，属于一种半定制电路，适合于定制高性能数据处理的需求，成本较高。成本高这一点就足以判定不适合本文中的智能家居系统（包括作为其中的音视频处理单元），因此本文不对其展开介绍。鉴于篇幅，其它类似电路也不展开介绍了。

2.1.2 嵌入式操作系统的对比和选择

在很少资源的可编程集成电路中我们可以方便地直接编码使用硬件资源，但是在更多硬件资源的时候，每次都人工一个个去编码排序我们的软件任务虽然可行，但是组织这样的庞然“大物”很费心神，前人已经编码出来了充分利用、组织各硬件资源的名为操作系统的软件，若自己编码写一个类似软件那得写到猴年马月，显然，“站在巨人的肩膀上”是个更好且非常棒的选择。

从时间角度看，操作系统主要分两种：实时操作系统（Real Time Operating System）、分时操作系统（Time Sharing Operating System）。其中实时操作系统对于执行任务的 CPU 时间片要求严格，比如在汽车控制中使用，一个操作的响应必须在某个时间内完成，否则整个系统可能会崩溃；分时操作系统的调度算法则是偏向平等、公平地给执行任务分配 CPU 时间片，更适合用来管理普通的应用程序。

目前，常见的用于嵌入式领域的操作系统有μC/OS、eCOS、FreeRTOS、Linux、Zephyr OS、RT-Thread、Mbed OS、Vxworks、QNX、NuttX、RTX 等，下面仅从适用于本设计的操作系统中介绍几个开源（Open Source）的操作系统。

(1).Linux

Linux 是由芬兰的 Linus Benedict Trovalds 于 1991 年开始开放发布的类 Unix 操作系统，是一个支持 POSIX（Portable Operating System Interface）、多用户、

多任务、多线程和多 CPU 的操作系统。简而言之，它是一个性能稳定、可自定义程度高的、拥有非常棒的生态的多用户网络操作系统。从 2.6.11 版本之后，Linux 内核的调度算法换成了追求平均、公平时间分配的 CFS（Completely Fair Scheduler），在此之后 Linux 操作系统完全地成为一个 TSOS（分时操作系统），非常适合运行应用系软件程序。Linux 内核的软中断 softirq 中对网络设备的中断优先级仅次于最高优先级别的 Timer，可谓是一个不折不扣的网络操作系统。

(2).μC/OS

μC/OS 是一个全功能的嵌入式操作系统，支持 TCP/IP、USB、CAN bus，并且包含鲁棒性的文件系统、人机图形界面组件。它现在的操作系统内核是μC/OS-III 和 μC/OS-II 实时操作系统内核，具有高移植性、可裁剪、多任务、实时可抢占的特点，面向微处理器、微控制器、DSPs（数字信号处理器）。其中μC/OS-III 内核的使用，相当一部分内容需要商业授权。

(3).FreeRTOS

性质跟μC/OS 一样，但是更简单，能在更低资源的设备上运行，并且在商业上是完全免费的，但是组件支持比μC/OS 少很多，稳定性比μC/OS 相对较差。

(4).RT-Thread

RT-Thread 是一款参考沿用了 Linux 操作系统设计风格的物联网操作系统、实时操作系统，同样地，它跟μC/OS 的性质也基本一样。简而言之，它是始于 2006 年的国内自研的实时操作系统，自带良好的中文支持，到目前已拥有较多的组件支持，并且支持非常多的 CPU 架构。

(5).Zephyr

Zephyr OS 跟 RT-Thread 非常类似，是近年出现的一款针对物联网低资源设备的 RTOS，托管于 Linux 基金会，借鉴 Linux 的设计思想、生态和资源，现在在迅速地发

展中。

2.2 无线通信技术

2.2.1 Wi-Fi

Wi-Fi是目前应用最广泛的无线通信技术，通信频段为2.4GHz、5GHz，传输距离在100-300M，速率可达300Mbps，功耗约10-50mA。

Wi-Fi是一种无线网络技术的名称，它使用无线电波提供无线高速互联网和网络连接。一个常见的误解是Wi-Fi是“wireless fidelity”的缩写，然而事实并非如此。Wi-Fi只是一个商标短语，意思是IEEE 802.11x（Beal V.）。许多家庭基本都装有带Wi-Fi功能的无线路由器，在智能家居设计中，这种家用路由器非常适合当作网关。

对于家庭摄像头监控设备来说，Wi-Fi是最好不过的技术，既满足图像传输的带宽要求，又直接兼容家庭网络，易于推广和部署。Wi-Fi技术的缺点是穿墙、过墙能力较差，不过该缺点可以通过一些方法来弥补，不成问题。另外一点就是，Wi-Fi相对其它无线通信技术来说功耗较高。本设计中的室内环境监测仪带有屏幕，与摄像头监控设备一样是接入外部电源，两个设备均使用Wi-Fi进行无线通信。

2.2.2 Zigbee

Zigbee 是 IEEE 802．15．4 协议的简称，是一个使用免授权通信频段 2.4GHz、868MHz、915MHz 的自组网协议，主要适用于无线控制领域。

它的优点是低功耗、低成本、掉线率低、组网能力强、工作频段灵活，缺点是传播距离近、低速率、数据有延迟，适用于物联网短距离、低功耗、低速率设备间通信、组网。

ZigBee 联盟的主要发展方向是建立一个基础构架，这个构架基于互操作平台以及配置文件，并拥有低成本和可伸缩嵌入式的优点。搭建物联网开发平台，有利于研究成

果的转化和产学研对，是实现物联网的简单途径（葛翠艳，珠海国芯云科技有限公司，2019）。

综合地看，Zigbee 不是本设计中的两个家居设备无线通信协议的最优解。

2.3 图像处理技术

限于篇幅，仅简单介绍本设计中涉及部分图像处理技术。进行视频监控，首先我们需要使用camera器件/传感器将环境的图像信息一帧帧的转换成二进制比特，然后进行编码、存储，继而对加工好的二进制图像信息进行一系列处理。为了实现移动物体检测的功能，最简单的方法是用当前图像帧减去上一个图像帧，但是就这样简单地对图像进行帧差处理的方法对设备部署的环境有要求，尤其是室内监控时，突然的开关灯、转动的风扇、闪烁的指示灯、夜晚低照度下sensor的随机噪点等情况是简单帧差无法较好处理的。

对于开关灯、日夜切换这种大面积的图像改变，我们可以将图像按照一定尺寸分成若干个画面框，如果整个画面所有框中都突然具有较大帧差则忽略检测触发，从而减少误报；

对于电子设备的呼吸灯、空中飞过的蚊子，同样将图像分成若干个画面框，每个框的前后帧相减再平方相加取平均数，即方差，与阈值方差相比，若小于对比值则说明图像的变化面积不足以触发阈值，减少误报。

硬件上，本次设计中选用了符合UVC（USB Video Class）标准的USB（Universal Serial Bus）摄像头，只需要在Linux内核中移植对应摄像头的驱动即可进行应用程序编程使用摄像头。软件上，本次设计使用开源的motion项目进行二次开发以加快进入测试阶段。

2.4 TCP/IP

TCP/IP 并不是指 TCP 和 IP 协议，而是指传输控制协议/网际协议(Transmission Control Protocol/Internet Protocol)协议族。TCP/IP 协议族分为四层（如图 4）：链路层、网络层、运输层和应用层，每一层各有不同的责任。在 TCP/IP 中，网络层和运输层之间的区别是最为关键的：网络层（IP）提供点到点的服务，而运输层（TCP 和 UDP）提供端到端的服务（W·Richard S., 2000）。

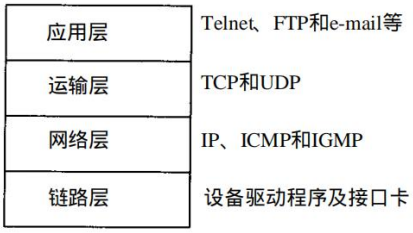


图 4　TCP/IP 协议族的四个层次（W·Richard S., 2000）

TCP/IP 是 Internet 的基础，本设计中的设备需要与路由器进行通信，TCP/IP 也是基础。

本次设计中使用运输层的 TCP 和 UDP 协议、应用层的 HTTP 协议。

2.5 服务器软件以及建站技术

2.5.1 服务器软件

在智能家居设计中，我们的家居设备只需要向浏览器客户端发送一些简单的网页标签、传感器数据、图像信息，因此，我们家居设备上的本地服务器只需要定制化少量的功能，而不是按照浏览器标准提供所有功能。

常见的嵌入式服务器有 Boa、lighttpd、shttpd 等，在对比各个嵌入式服务器之后，发现它们的功能还是不够精简，限于篇幅，不再对各个嵌入式服务器展开介绍、对比。

本设计中的嵌入式服务器采用完全自主设计开发的小型嵌入式服务器，大小可精简至十几 KB，分别部署到家居设备和云端服务器中，支持 HTTP、自定义协议。

2.5.2 建站技术

要使用浏览器作为人机交互工具，我们需要服务器和网站，在本设计中，我们的网站仅需要搭建几个网页即可满足（测试）需求。本设计中我们需要的技术：

(1).HTML（Hyper Text Markup Language）：称为超文本标记语言，一种标识性语言，是构成万维网世界的一砖一瓦。它定义了网页内容的含义和结构，除了 HTML 以外的其它技术则通常来描述一个网页的表现与展示效果，或功能与行为。

(2).层叠样式表（Cascading Style Sheets，缩写为 CSS）：它是一种样式表语言，用来描述 HTML 或者 XML（包括如 SVG、MathML、XHTML 之类的 XML 分支语言）文档的呈现。CSS 是开放网络的核心语言之一，它描述了在屏幕、纸质、音频等媒体上的元素应该如何被渲染的问题。

(3).Javascipt：简称 JS，是一门面向对象的、具有函数优先的、轻量的、解释性或即时编译型的计算机编程语言。Javascript 的标准是 ECMAScript，如今，所有的现代浏览器都已经完美地支持完整的 ECMAScript 2015，JS 已经是网页编程中的不而选择。

2.6 反向代理技术

2.6.1 NAT（Network Address Translation，网络地址转换）

互联网上的每一个接口必须有一个唯一的 Internet 地址（也称作 IP 地址）。目前作为 Internet 基础的 IP 协议仍为 IPv4（Internet Protocol version 4）。IPv4 地址使用 4 个字节，长 32bit，地址空间只有 4,294,967,296 个地址。Internet 地址并不采用平面形式的地址空间。IPv4 地址具有一定的结构，五类不同的互联网地址格式如下图 5 所示。

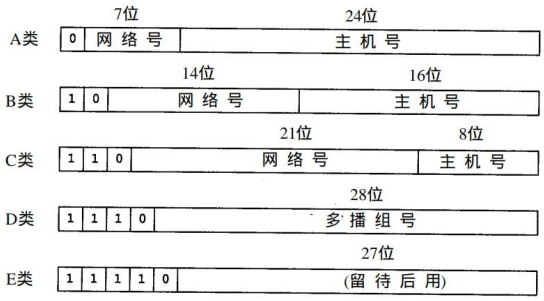


图 5　五类互联网地址（IPv4）

IPv4 的 40 多亿个地址空间一天天地面临枯竭，基于分类网络、无类别域间路由和网络地址转换（NAT）的地址结构重构显著地减少了地址枯竭的速度。

NAT 是 1994 年被提出的、使专用网内部主机能够访问因特网上的主机的方法和技术。在如今 IPv4 地址枯竭的情况下，运营商们给用户分配子网 IP，使用 NAT 方法来使用户共享 IPv4 一级地址（即公网 IP），因为这样，我们作为一般消费的普通消费者被分配的 IPv4 地址基本都是 NAT 主机下的子网 IP 地址。如果没有特意让运营商给自己建立端口映射规则，NAT 主机外是访问不了子网内的主机的，因为 NAT 主机虽然收到请求数据包，但它不知道这是发给哪个主机的。只有在内部主机向外访问时 NAT 主机才会为它建立临时的端口映射关系，这时候 NAT 主机才能将远端主机返回的数据包发给相应的内网主机端口。

2.6.2　反向代理

反响代理的意思是，公网服务器反过来扮演客户端的角色，作为其它客户端的代理者去访问一个处于内网之中的主机服务。

在运营商的 NAT 主机的子网下，在 IPv4 的条件下，当我们想让外部主机向本地子网主机发起请求的时候，我们只有两种选择：

(1).让运营商给我的主机和上层 NAT 主机记录一条或多条 IP+端口的映射关系；

(2).内部主机主动向外部主机发起请求，NAT 主机给我的 IP+端口建立临时映射关

系，此时 NAT 主机会将对应端口收到的数据包发回到下级主机端口。NAT 穿越就是基于此原理，但是至少需要有一个带公网 IP 的服务器。

显然，第二个比较可行。当然，如果你是运营商的大客户，直接让运营商给你分配公网 IP 也不是什么问题。

本设计中将根据第二个选择设计反向代理软件。公网（云）服务器虽然也不便宜，但对于一个智能家居供应商来说，这比方案一的部署成本简直低太多，除非客户都是运营商的大客户，自己就有或可以要一个公网 IP。

2.7 智能家居总体方案

2.7.1 硬件选型

(1).室内环境监测仪硬件选型（传感数据监控显示终端）

选用适用于自动控制的集成 ARM Cortex M CPU 核心的微控制器作为主控，本次测试使用 ST 公司的官方套件 STM32F429-DISC1 作为测试开发板，接入传感器进行测试版软件开发。

(2).视频监控设备硬件选型

选用性能均衡的集成 ARM Cortex A7 CPU 核心的 SoC 或 SoM（System on Module）的芯片作为主芯片，搭载 Linux 操作系统，运行视频监控相关的摄像头驱动、应用软件。本次测试使用飞思卡尔（已被 NXP 收购，在 NXP 作为一个部门存在）的 i.MX6ULL 系列 SoC 的单板计算机，其适用于家居自动化。

2.7.2 实施方案

将设计内容分开六个部分进行开发：传感数据监控显示终端、摄像头监控软件系统、小型嵌入式服务器、反向代理软件、web 端人机交互网页、安卓手机客户端。

(1).传感数据监控显示终端:将采集的传感器数据通过无线网络发送给局域网内小型

服务器，实现对家居室内环境光、温湿度的监测，并将数据在本地显示屏显示；

(2).摄像头监控软件系统:监测到移动物体时则开始抓拍一段时间的视频保存至本地服务器，既省去了人工查找有变化的视频记录也节省了大量不必要的存储资源；

(3).小型嵌入式服务器:根据功能自定义本地小型嵌入式服务器,去掉不必要的部件，减少硬件资源占用，有效降低成本，所有数据保存至本地，在局域网内以 web 服务方式提供家居设备及监控设备和数据的访问，更大限度的保证了安全和隐私以及实用性；

(4).反向代理软件：将本地服务器端口映射到公网服务器端口，使用户可在外网访问家居 web 服务；

(5).web 端人机交互网页：搭建以跨平台的 web 网页端作为人机交互平台，只要有浏览器就能连接，免去适配不同终端平台的大量工作量，同时用户也无需为此多安装一个 app；

(6).安卓手机客户端：本机一点击即打开，省去输入网址的步骤，舒心操作。

2.8 小结

本章节对本文智能家居系统设计所涉及的技术进行了对比和选择，并确定了硬件选型和实施方案。

3 理论分析及设计

3.1 传感数据监控显示终端

本系统设计采用模块化的方式进行设计，硬件电路、硬件驱动程序和应用程序分别模块化地进行开发，每个部分仅保持接口一致性，程序具有较高的可移植性。当硬件电路修改时，硬件驱动程序只需要进行少量的修改，而应用程序几乎不需要修改即可直接移植到新电路系统上。

3.1.1 系统硬件电路设计

(1).STM32F429I-DISC1

它是 ST 公司官方发布的 32F429IDISCOVERY Discovery（开发板）套件，该套件利用 STM32F429 高性能微控制器的功能，使用户可以通过高级图形用户界面开发丰富的应用程序。

本设计中使用该套件作为主要电路硬件进行传感监控显示终端的开发（以 C 语言为主）。该套件拥有如下资源（仅列出部分）：

• 以 LQFP144 尺寸封装的 STM32F429ZIT6 微控制器，包含 2Mbytes 的 Nor Flash 内存、256 Kbytes 的 RAM

• 2.4 寸的 QVGA TFT LCD

• Micro-A 和 Micro-B 接口的 USB OTG

• ST MEMS 运动传感器 3 轴数字输出陀螺仪 I3G4250D

• 64Mbit SDRAM

• 板载的 ST-LINK/V2-B

该官方套件如下图 6 所示：

Features

- STM32F429ZIT6 microcontroller featuring 2 Mbytes of Flash memory, 256 Kbytes of RAM in an LQFP144 package
- 2.4" QVGA TFT LCD
- USB OTG with Micro-AB connector
- I3G4250D, ST MEMS motion sensor 3-axis digital output gyroscope
- Six LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3 V power-on
  - Two user LEDs: LD3 (green), LD4 (red)
  - Two USB OTG LEDs: LD5 (green) V_{BUS} and LD6 (red) OC (over-current)
- Two push-buttons (user and reset)
- 64-Mbit SDRAM
- Extension header for LQFP144 I/Os for a quick connection to the prototyping board and an easy probing
- On-board ST-LINK/V2-B
- USB functions:
  - Debug port
  - Virtual COM port
  - Mass storage
- Mbed Enabled™ (see http://mbed.org)
- Board power supply: through the USB bus or from an external 3 V or 5 V supply voltage
- Comprehensive free software including a variety of examples, part of STM32CubeF4 MCU Package or STSW-STM32138, for using legacy standard libraries

Picture is not contractual.

图 6　stm32f4-discovery

(2).DHT11 数字温湿度传感器模块电路

它是将 DHT11 数字温湿度传感器模块化成的便于接入系统电路使用的模块电路。该电路的传感器是一款已校准的电容式温湿度复合传感器，它具有极高的可靠性和稳定性、超长的可达 20m 以上的信号传输距离、小体积、地功耗、连接方便等优点，第一版的该传感器的测量范围为 20~90%RH、0~50°C，第二版的测量范围为 5~95%RH、-20~+60，它采用简化的单总线通信，只需连接一个数据口。其原理图如下图 7 所示。

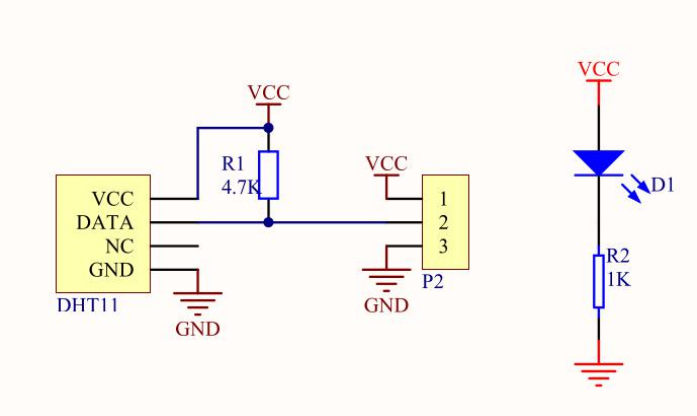


图 7　DHT11 模块原理图

(3).GY302 BH1750 光照度传感器模块电路

它是将 BH1750 光照度传感器模块化的电路模块。BH1750 是一款数字式的环境光照度传感器 IC，它内置了 16 bit 的 AD（模-数）转换器，直接数字输出，测量范围为 1-65535 lx（光照度单位）、测量精度可调且最小可达 0.11lx，通过 50Hz/60Hz 除光噪音功能实现稳定的测定，光源依赖性弱，受红外线影响很小，最小误差变动在-20%~+20%之间，且无需其他器件，是一个高精度、测量范围广、稳定的传感器电路。它采用 IIC（Inter-Integrated Circuit）总线进行通信，并有两种可选的 IIC 从机地址。该电路模块实物如下图 8 所示、原理图如下图 9 所示。

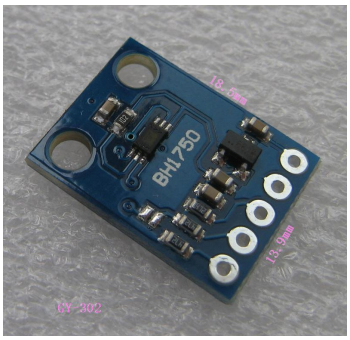


图 8　GY-302 BH1750 模块

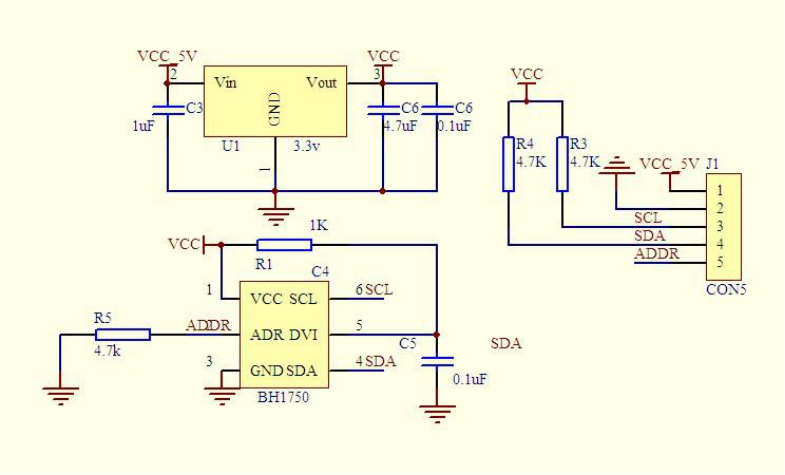


图 9　GY-302 BH1750 原理图

(4).电磁继电器若干

电磁继电器一般由铁芯、线圈、衔铁、触点簧片等组成，它是一种电子控制器，是一种控制不同电流的方法的具体实现和控制开关。通常，人们用它作为小电流控制大电流的控制开关。一个典型的电磁继电器原理图如下图 10 所示：

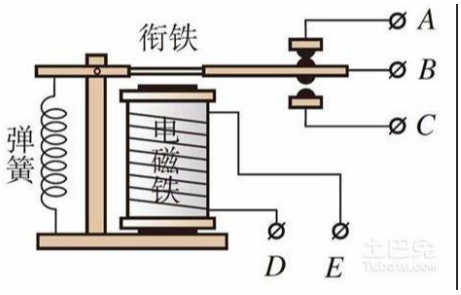


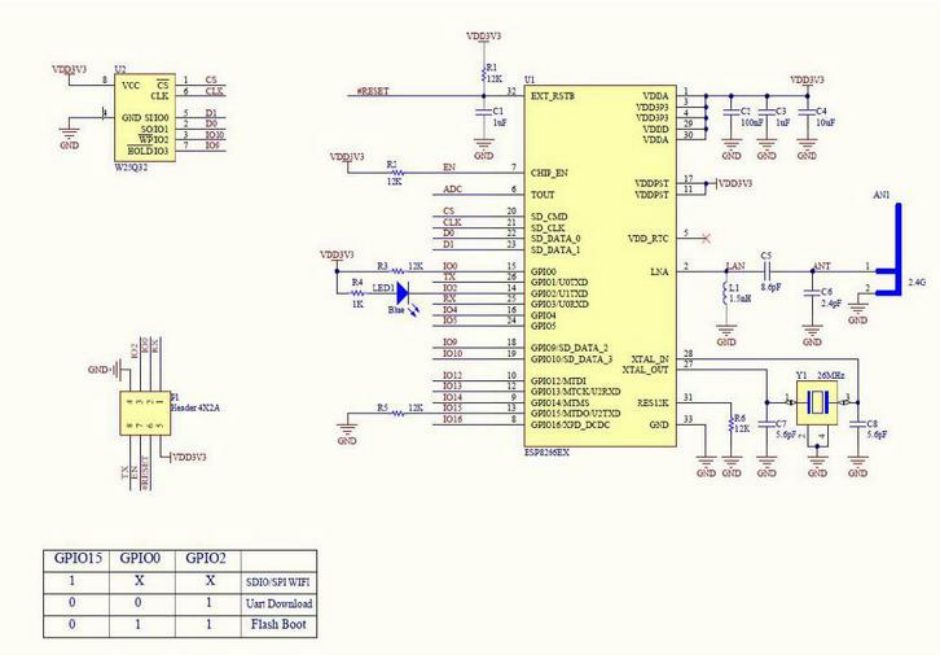
图 10　电磁继电器原理图

工作原理：在线圈两端加上一定的电压使线圈中流过一定的电流，从而产生电磁效应，衔铁在电磁力的吸引下克服弹簧拉力吸向铁芯，从而带动衔铁上的触电的吸合。当线圈的电流消失时，铁芯的电磁吸力也随之消失，衔铁就会在弹簧的拉力下返回原来的位置。

本设计中使用该类继电器让低电压的微控制器间接地控制 220V 交流电，从而控制本次测试的家居照明灯的开关。

(5).ESP-01S Wi-Fi 模块电路

ESP-01S 是 ESP8266 系列无线模块中的一种 WiFi SoC 模组，支持标准的 IEEE802.11 b/g/n 协议，原理图如下图 11 所示。



GPIO15	GPIO0	GPIO2	
1	X	X	SDIO/SPI WIFI
0	0	1	Uart Download
0	1	1	Flash Boot

图 11　ESP-01S 原理图

内置完整的 TCP/IP 协议栈，采用低功耗的 32bit 位宽 CPU，主频最高可达 160MHz，可兼作微控制器、应用处理器。它具有电路精简、高性价比等一系列优点，内置 10bit 高精度 ADC（analog-to-digital converter）、集成 Wi-Fi MAC/ BB/RF/PA/LNA、支

持 UART/GPIO/IIC/PWM/ADC/HSPI（通用异步首发/通用输入输出/集成电路总线/脉冲宽度调制输出/模-数转换/高速串行通讯）等接口。用户可使用它为现有设备添加联网功能，也可以构建独立的网络控制器。不过，ESP-01S 的可用 IO 只有两个，Flash 只有 8Mbit，不建议使用此型号直接二次开发更多功能。

3.1.2　硬件驱动程序设计

在本设备中不使用操作系统软件，因此，驱动程序和应用程序之间的接口按照一定的规范保持一致即可。

(1).DHT11 温湿度传感器驱动程序设计

DHT11 采用单总线方式通信，驱动程序只需要通过单总线向 DHT11 发送命令、接受测量数据、计算数据。DHT11 与主机（主控单元）的一次通讯时间在 4ms 左右，数据分为小数部分和整数部分，一次完整的数据传输为 40bit,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 校验和数据。传送正确时校验和数据等于“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”所得结果的末 8 位。

它的单总线通讯过程如下图 12 所示：

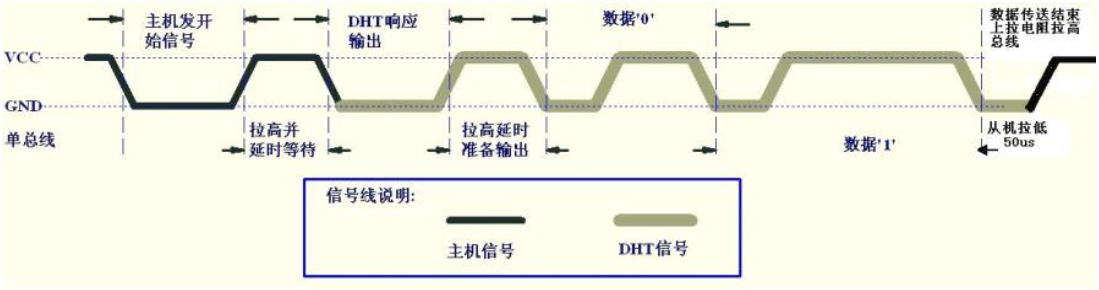


图 12　DHT11 与主机的通讯过程

总线空闲状态为高电平，主机把总线电平拉低至少 18ms 作为通信起始信号。

DHT11 检测到起始信号之后，等待主机起始信号结束（电平拉高），然后发送 80us 低电平响应信号。总线为低电平,说明 DHT11 发送了响应信号,DHT11 发送响应信号后，主机再把总线拉高 80us，准备发送数据，每一 bit 数据都以 50us 低电平时隙开始，高电平的长短定了数据位是 0 还是 1，如下图 13 和图 14 所示：

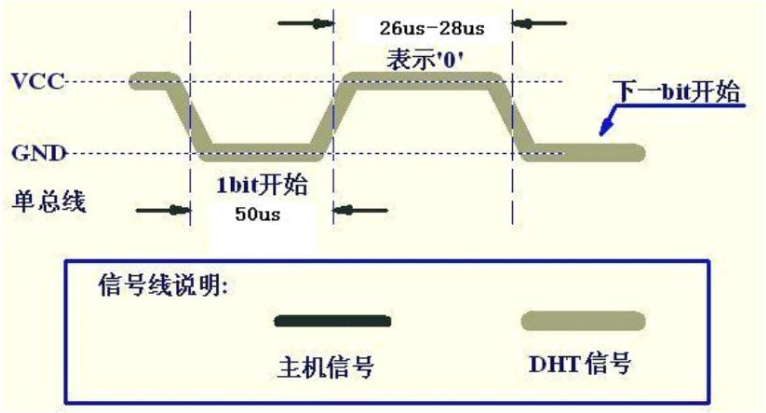


图 13　数字 0 信号表示方法

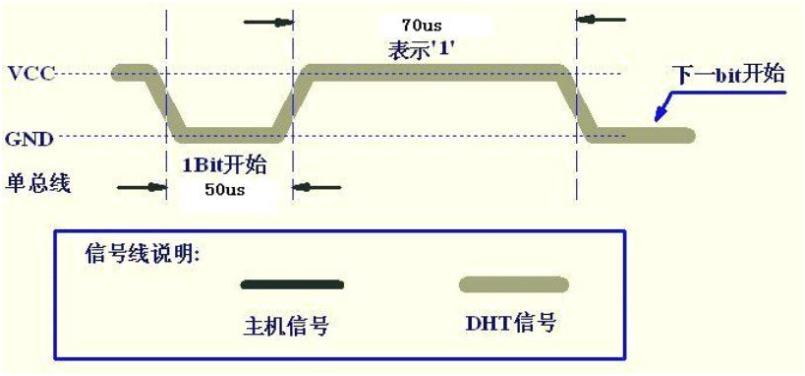


图 14　数字 1 信号表示方法

由此使用一个 GPIO（General Perpose Input/Output）引脚设计 DHT11 驱动程序，程序逻辑图如下图 15 所示：

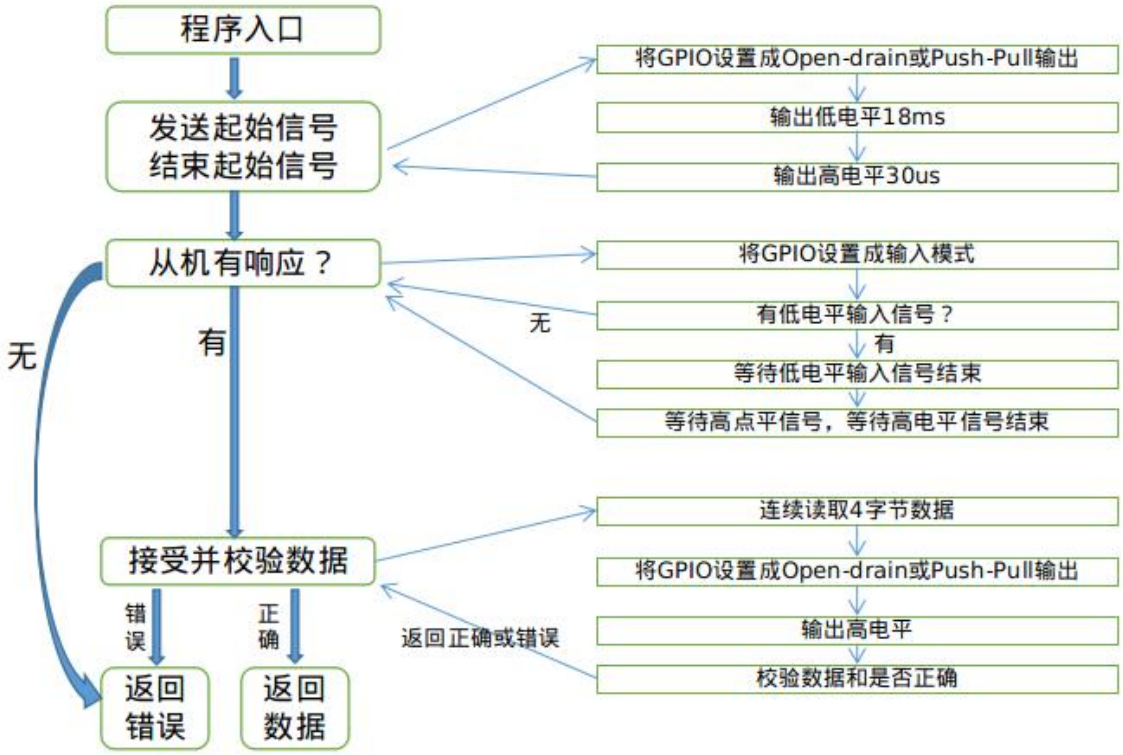


图 15　DHT11 驱动程序流程图

驱动程序只需向应用程序提供一个温湿度查询接口 get_temp_and_humi()即可，应用程序只需直接调用并传入用于保存结果的数据结构的指针。

(2).BH1750 光照度传感器驱动程序设计

该传感器采用标准 IIC 总线通信。同样需要查询它的产品数据手册获取具体细节信息。需要获取的关键信息是：从机地址、通讯时序、命令参数、对传输的数据的处理规则。从 datasheet 中查询得到的命令列表如下图 16 所示：

●Instruction Set Architecture

Instruction	Opecode	Comments
Power Down	0000_0000	No active state.
Power On	0000_0001	Waiting for measurement command.
Reset	0000_0111	Reset Data register value. Reset command is not acceptable in Power Down mode.
Continuously H-Resolution Mode	0001_0000	Start measurement at 1lx resolution. Measurement Time is typically 120ms.
Continuously H-Resolution Mode2	0001_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms.
Continuously L-Resolution Mode	0001_0011	Start measurement at 4lx resolution. Measurement Time is typically 16ms.
One Time H-Resolution Mode	0010_0000	Start measurement at 1lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.
One Time H-Resolution Mode2	0010_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.
One Time L-Resolution Mode	0010_0011	Start measurement at 4lx resolution. Measurement Time is typically 16ms. It is automatically set to Power Down mode after measurement.
Change Measurement time (High bit)	01000_MT[7,6,5]	Change measurement time. ※ Please refer "adjust measurement result for influence of optical window."
Change Masurement time (Low bit)	011_MT[4,3,2,1,0]	Change measurement time. ※ Please refer "adjust measurement result for influence of optical window."

※ Don't input the other opecode.

图 16　BH1750 命令参数

因为数据是实时显示，我们选择每秒测量一次即可，其次这里是驱动程序，无需也不应该多次测量去异常取均值。BH1750 的内部电路电源可以由 IIC 通信命令控制。选用一次测量-高精度模式一即可，开始测量就发送开机命令，测量完发送关机命令，节省不必要的电量消耗。程序流程图如下图 17 所示。

图 17　BH1750 驱动程序流程图

同样地，该驱动程序只需向应用程序提供一个查询接口 get_illuminance()，应用

程序调用时传入用于保存数据的数据结构指针即可。

(3).照明灯（继电器）驱动程序设计

控制继电器，只需要控制继电器控制端的电压使之产生电流，对于本设计的继电器，我们只需要使用一个 GPIO 引脚接上拉电阻来向继电器控制引脚输出高电平（5V）或低电平（接地）即可。

虽然应用程序在此处直接操作 GPIO 开、关灯也很简单，但将 GPIO 引脚的初始化和电平操作抽象成 light_init()和 light_on()、light_off()，应用程序只需要调用接口函数，不需要关心硬件细节，加上使用 C 语言“inline”关键字，在保持应用程序的阅读性、可移植性的同时，保持了原本的“编译后”程序尺寸。

统一开关灯接口，更方便后续接入语音控制系统。由于此部分主要用于接入后续开发的语音控制系统，红外人体感应、雷达感应电路只分别应用于楼道、室外，现本文仅关注室内房间和厅室的功能开发，红外人体感应与雷达感应电路暂未集成到系统中。其中已测试完成的红外人体感应电路模块如下图 18 所示：

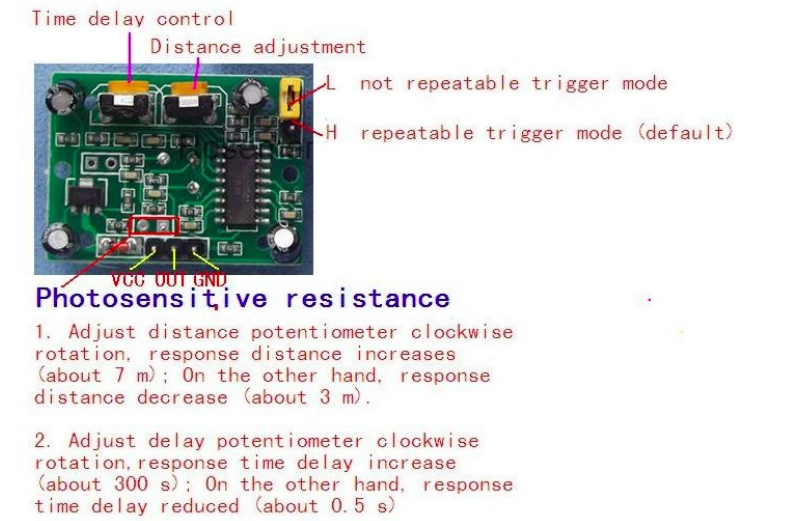


图 18　HC-SR501 红外人体感应电路模块

(4).ESP-01S Wi-Fi 模块驱动程序设计

ESP-01S 与主控电路间使用 UART 协议进行通信，两者之间短距离交叉相接接两条数据线，按照 TTL（transistor transistor logic）电平标准进行通信。因此，该驱动程序的第一步是初始化硬件或 GPIO 模拟的 UART（所使用的两个引脚）。两者初始化的 UART 的波特率必须保持一致，主机向该 Wi-Fi 模块发送 AT（attention）指令，后者按指示执行任务并返回状态或数据。

初始化了 UART 之后，发送 AT 指令让模块连接 WiFi，向服务器发起 TCP 双向连接请求，然后发送数据流/接受数据流，用完发送 AT 指令关闭 TCP 连接即可。其中将功能正常的模块的 CWMODE 设置为 3 可以同时开启热点和连接热点，使用此模式可以实现用户连接该热点重新设置 WiFi 连接的 SSID 和连接密码以连接新的 WiFi。连接一个名为 HOMEAP 密码为 123456 的 WiFi 并向服务器发送数据流的 AT 指令集如下：

AT+CWJAP="HOMEAP","123456"

AT+CIPSTART=4,"TCP","远端 IP",远端端口,本地端口,0（使用通道 4 进行连接，0 表示绑定远端 IP 和端口）

AT+CIPSEND=8

12345678（完成发送 8 个字符）

AT+CIPCLOSE =4（关闭通道 4 的连接）

此部分比较简单，将使用 UART 发送 AT 指令的步骤封装成 connect()、send()、close()接口供应用程序直接调用即可。

(5).TFT-LCD 驱动设计

TFT-LCD，薄膜晶体管液晶显示器，也叫真彩液晶显示器，它在液晶显示屏上的每一个像素点都有一个 TFT（薄膜晶体管），这有效地克服了非选通时的串扰，使液晶屏的静态特性与扫描线数无关，大幅提高图像质量。本设计中的 TFT_LCD 引脚原理图如

下图 19 所示。

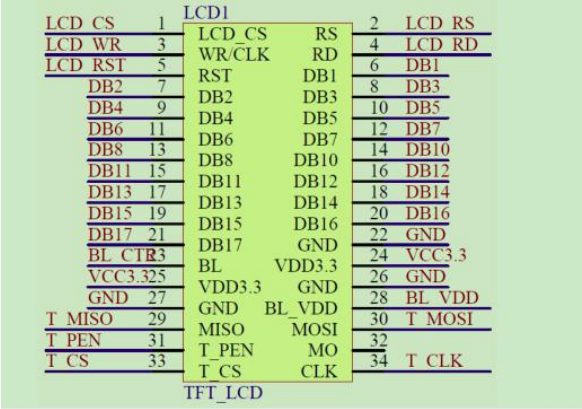


图 19　TFT-LCD 引脚图

要驱动该显示器显示图像，需要控制屏幕里的每一个像素点的三种颜色（三基色）的显示。按一定规则地一个个的点亮每一个像素，这个行为被称为扫描。根据图像 RGB 描述去扫描每一个像素点的工作量是很大的，我们独立了出来一部分处理单元专门负责控制像素点。一般地，LCD 方案的厂商会有将这种专门负责控制 LCD 像素点的显示的控制单元独立封装出来与 LCD 一起作为一个电路模块，方便客户使用，我们将这种封装出来的芯片归类为 LCD 控制器，往大的范围归类我们可以将它们归类为像素控制器；从另一种角度来看，若把像素点的解码功能也集成进去，那么它是一个负责将图像信息解码成 RGB 像素信息并控制像素点显示的图像信息处理器兼像素控制器，我们可以形象地称它为图形处理器。简而言之，要使 LCD 显示图像，通常是两种方法：

i.CPU 直接操作 LCD 引脚电平控制像素点；

ii.CPU 告诉（如果有）显示器控制器/图形处理器将被控制的显示器的参数、到哪里取图像信息，通常图像信息被约定好会在某段 RAM 地址中。

本设计中使用 ili9341 控制器作为 LCD 控制器，它将代替 CPU 去控制 LCD 像素点，具体它是怎么操作的，这点已经被专门负责该领域的人封装好功能了，不是本文关注的

重点。该控制器与主控制器之间使用 SPI 通信，我们需要配置好 FSMC（Flexible Static Memory Controller）进而能够使用外接的 SRAM，然后通过 SPI 告诉 ili9341 控制器我们要传送的 RGB 像素的格式、RAM 地址段，然后直接将 RGB 信息输出到约定好的 SRAM 地址段内即可驱动显示，流程如下图 20 所示。

图 20　TFT LCD 驱动流程图

使用前初始化 FSMC、外接 SRAM、SPI、ili9341，事先准备好字模、图像等 RGB 数据，然后应用程序通过驱动提供的接口 BSP_LCD_DisplayStringAtLine()并传入行和字符串即可显示字符。通过提前封装、运行时数学计算图形 RGB 数据等方法可以实现更丰富的图形显示操作。

3.1.3　应用程序设计

在驱动程序封装、屏蔽了底层硬件细节之后，应用程序仅需关注应用功能逻辑，代码也更加直观、可阅读性好、可移植性高。应用程序逻辑架构如图 21 所示：

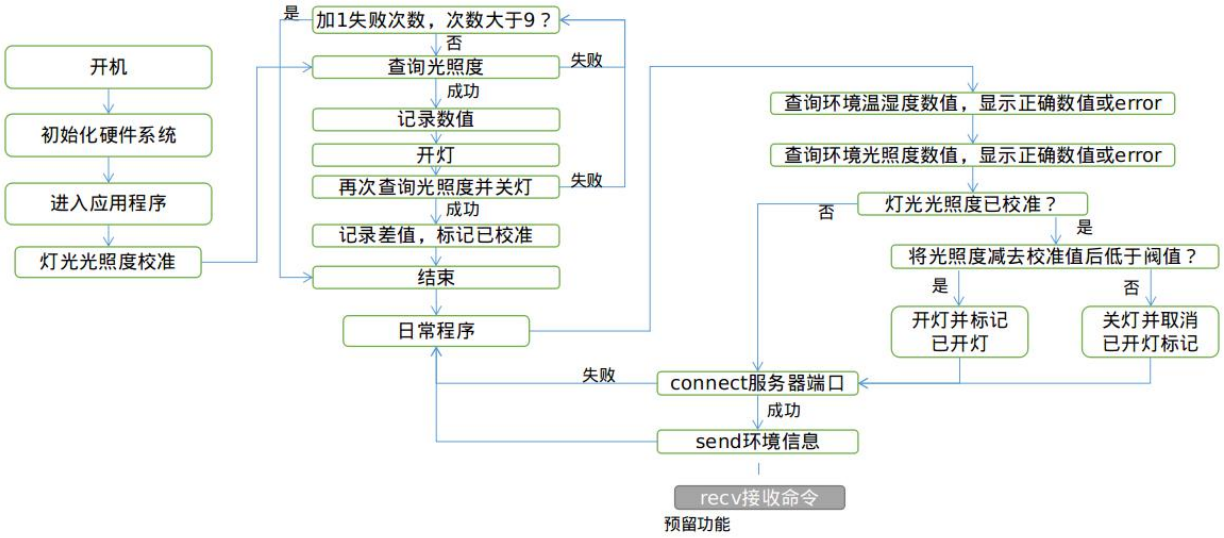


图 21　传感数据监控显示设备之应用程序设计

3.1.4　整体程序的编译和部署

(1).整体程序编译

软件逻辑使用 C 语言实现，开发环境为 x86_64（Manjaro） Linux PC，编译工具链为开源的 GNU GCC 编译工具链之 gcc-arm-none-eabi-toolchain。

打开 ARM 官网之 https://developer.arm.com,点击“Linux/Open Source”，找到 GNU Toolchain / GNU-R M Download，进去下载页选一个下载。例如下载了 7-2018-q2 版本，将下载的包解压到/opt,并添加到 PATH，打开命令行，操作如下：

```
$ cd /opt
$ sudo tar xjf ~/Downloads/gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2
$ export PATH=/opt/gcc-arm-none-eabi-7-2018-q2-update/bin:$PATH
$ arm-none-eabi-gcc -v
gcc version 7.3.1 20180622 (release) [ARM/embedded-7-branch revision 261907] (GNU Tools for Arm Embedded Processors 7-2018-q2-update)
```

编译环境即准备完成。

使用 VSCode 编辑器配合 Makefile 进行工程代码开发和编译，最终分别得到.hex/.elf/.bin 文件，其中.bin 文件中只包含最终的执行代码，使用该类型进行烧录时需指定具体的 ROM 或 RAM 地址。编译例图如图 22 所示。

```
arm-none-eabi-gcc -x assembler-with-cpp -c -mcpu=cortex-m4 -mthumb -mfpu=fpv4-sp-d16 -mfloat-ab
i=hard -DUSE_HAL_DRIVER -DSTM32F429xx -ICore/Inc -IDrivers/STM32F4xx_HAL_Driver/Inc -IDrivers/S
TM32F4xx_HAL_Driver/Inc/Legacy -IDrivers/CMSIS/Device/ST/STM32F4xx/Include -IDrivers/CMSIS/Incl
ude -IDrivers/BSP/STM32F429I-Discovery -Og -Wall -fdata-sections -ffunction-sections -g -gdwarf
-2 -MMD -MP -MF"build/startup_stm32f429xx.d" startup_stm32f429xx.s -o build/startup_stm32f429xx
.o
arm-none-eabi-gcc build/main.o build/stm32f4xx_it.o build/stm32f4xx_hal_msp.o build/stm32f4xx_h
al_spi.o build/stm32f4xx_hal_tim.o build/stm32f4xx_hal_tim_ex.o build/stm32f4xx_hal_rcc.o build
/stm32f4xx_hal_rcc_ex.o build/stm32f4xx_hal_flash.o build/stm32f4xx_hal_flash_ex.o build/stm32f
4xx_hal_flash_ramfunc.o build/stm32f4xx_hal_gpio.o build/stm32f4xx_hal_dma_ex.o build/stm32f4xx
_hal_dma.o build/stm32f4xx_hal_pwr.o build/stm32f4xx_hal_pwr_ex.o build/stm32f4xx_hal_cortex.o
build/stm32f4xx_hal.o build/stm32f4xx_hal_exti.o build/system_stm32f4xx.o build/gpio.o build/sp
i.o build/usart.o build/rtc-ds1302.o build/oled.o build/bh1750.o build/dht11.o build/stm32f429i
_discovery_lcd.o build/stm32f429i_discovery_sdram.o build/stm32f429i_discovery_ts.o build/stm32
f429i_discovery.o build/ili9341.o build/stm32f4xx_hal_uart.o build/ltdc.o build/stm32f4xx_hal_l
tdc.o build/stm32f4xx_hal_ltdc_ex.o build/stm32f4xx_hal_dsi.o build/dma2d.o build/stm32f4xx_hal
_dma2d.o build/fmc.o build/stm32f4xx_ll_fmc.o build/stm32f4xx_hal_sdram.o build/i2c.o build/stm
32f4xx_hal_i2c.o build/stm32f4xx_hal_i2c_ex.o build/tim.o build/startup_stm32f429xx.o -mcpu=cor
tex-m4 -mthumb -mfpu=fpv4-sp-d16 -mfloat-abi=hard -specs=nano.specs -TSTM32F429ZITx_FLASH.ld  -
lc -lm -lnosys  -Wl,-Map=build/F429DISC.map,--cref -Wl,--gc-sections -o build/F429DISC.elf
arm-none-eabi-size build/F429DISC.elf
   text    data     bss     dec     hex filename
  38908     200    2720   41828    a364 build/F429DISC.elf
arm-none-eabi-objcopy -O ihex build/F429DISC.elf build/F429DISC.hex
arm-none-eabi-objcopy -O binary -S build/F429DISC.elf build/F429DISC.bin
[whqee@whqee F429DISCI_HOMEAPP]$
```

图 22　编译结果

(2).程序部署

程序的烧写使用 OpenOCD。OpenOCD 是“Open On-Chip Debugger”的缩写，它是免费开放的片上调试、（芯片）系统内编程和边缘扫描测试的软件，通过调试器与芯片内部按照 JTAG（Joint Test Action Group）接口协议进行通信，支持非常多系列的芯片类别。

烧写（即对芯片内部存储器进行编程）只需一行命令“openocd -f board/stm32f429disc1.cfg -c "program $(find -name *.elf) reset exit"”，非常方便。其次，我们还可以自定义命令脚本实现各种丰富的自动化操作。不仅如此，我们还可以利用 OpenOCD 对代码进行单步调试，甚至在调试过程中可以查看任意寄存器值，包括 CPU 使用的 r0-r12、sp、lr、pc 寄存器以及各种外设的寄存器，功能十分强大。调试过程（VSCode+cortex-debug 插件+OpenOCD+arm-none-eabi-gdb）如下

图 23 所示：

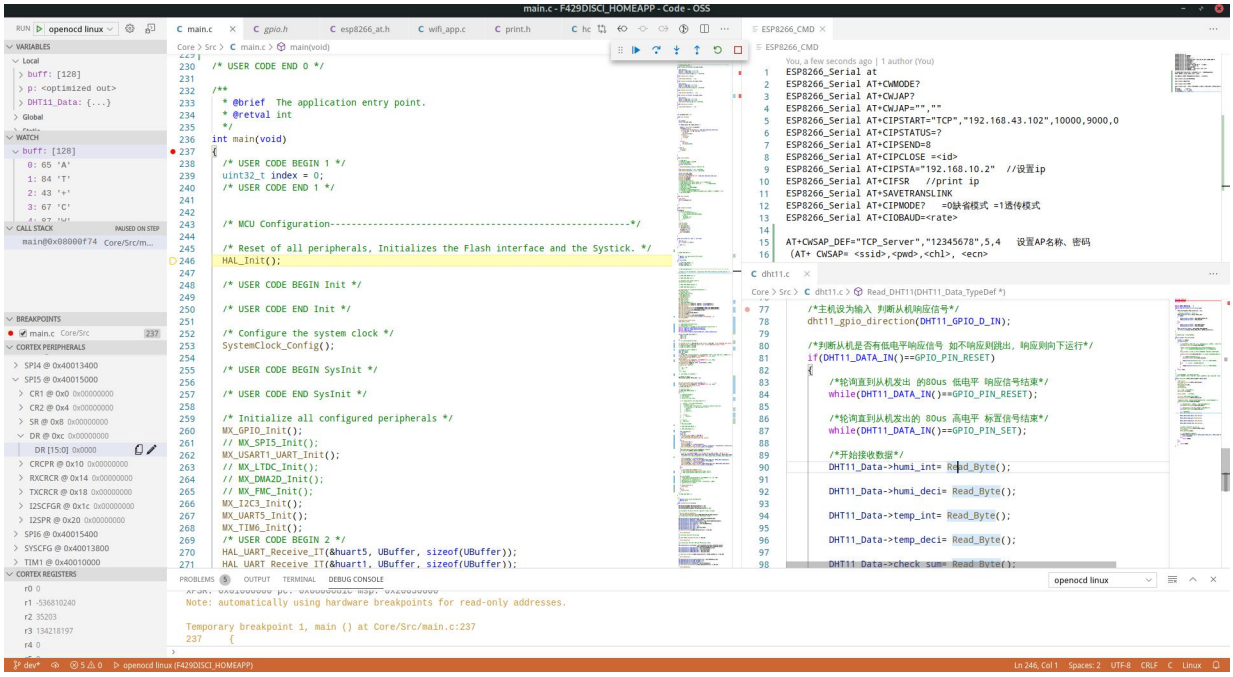


图 23　单步调试

整个程序烧写完，下一步即为测试。测试部分将统一排版在下一章节中。

3.2　摄像头监控子系统

摄像头监控子系统作为监控系统的前端，需要采集图像、处理图像，并向上层应用提供使用接口或方式，可以搭配报警系统软件形成完整的视频监控系统，也可以为其它系统功能所用。该子系统主要由摄像头模块电路、摄像头图像信息处理电路和系统软件组成。本子系统中的设备采用 Linux 操作系统搭载图像采集软件。

3.2.1　系统硬件电路设计

(1).MYS-6ULX-IOT 单板计算机

MYS-6ULX-IOT 是米尔科技推出的基于 NXP 公司 I.MX6ULL 系列处理器的面向物联网市场推出的单板计算机。其中 I.MX6ULL 系列处理器采用 ARM Cortex-A7 CPU 核

心，运行速度高达 528MHz，是一个高功效、高性价比、性能均衡的应用处理器系列。该系列处理器提供了多种存储器接口,其中包括 16 位 LPDDR2、DDR3、DDR3L、NAND Flash、NOR Flash、eMMC、Quad SPI 和各种其他接口,用于连接外围设备。如图 24 所示，该单板计算机在板卡上集成了以太网、USB OTG、USB Host、TF Card、WIFI、用户按键、LED、LCD 接口、IO 扩展接口等功能。

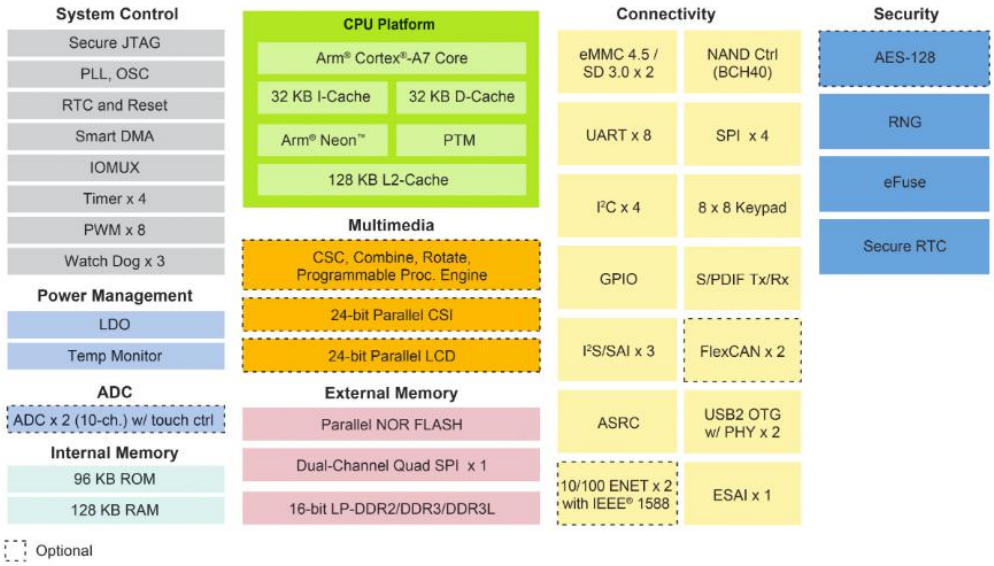


图 24　I.MX6ULL 功能结构图

本设计使用的单板机为标准版，板卡上搭载了 MCIMX6Y2DVM05A 处理器,配备了 256MB 高速 DDR3、256M NAND Flash，此外还集成了一路以太网、USB、WIFI、TF 卡等功能。MYS-6ULX-IOT 板载了最高通讯速率为 150Mbps USB WIFI 模块,板载 IPEX 天线接口，WIFI 内置芯片型号为 Realtek 公司的 RTL8188ETV，天线座子型号为泰科 1566230-1，如图 25 所示。

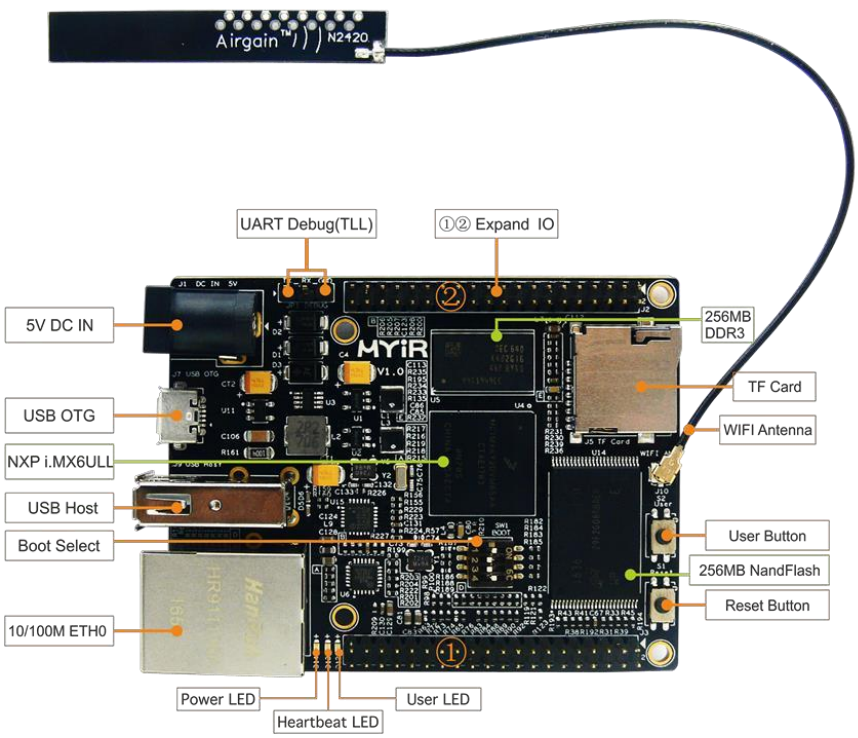


图 25　MYS-6ULX-IOT 资源标识图

(2).摄像头

摄像头采用 CMOS 传感器的 USB 摄像头模块，支持标准 UVC，支持 OTG，成本较低，作为图像采集模块，适用于监控设备、智能家居可视门铃、考勤机等设备，如图 26 所示。

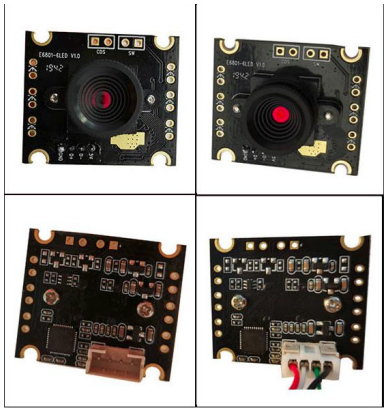


图 26　摄像头模组

3.2.2 Yocto 定制编译工具链

Linux 系统平台上有许多开源的系统构建框架,这些框架方便了开发者进行嵌入式系统的构建和定制化开发,目前比较常见的有 Buildroot, Yocto, OpenEmbedded 等等。其中 Yocto 项目使用更强大和系统化的方法,来构建出适合嵌入式产品的 Linux 系统。

Yocto 是一个开源的项目，它下面有很多个子项目，Yocto 只是把所有的项目整合在一起,同时提供一个参考构建项目 Poky，来指导开发人员如何应用这些项目，构建出嵌入式 Linux 系统。它包含 Bitbake，OpenEmbedded-Core，板级支持包,各种软件包的配置文件。通过 Poky，可以构建出不同类需求的系统，如最小的系统 core-image-minimal、带 GUI 的图形系统 fsl-image-gui、带 Qt5 图形库的 fsl-image-qt5。在构建完定制的系统镜像之后，我们可以使用 bitbake 工具利用构建好的软件包的动态库、静态库等定制一个 SDK。这个 SDK 不仅仅包含一个交叉编译工具链，还包含两个 sysroots（包含了目标机器的头文件和库、本地机器工具软件）的集合，还包含了设置编译时所需的必要的环境变量的脚本。

在本次设计中，图像采集软件要用到的软件库（包括动态库、静态库）比较多，若使用一般的编译工具链我们需要一个一个库去编译、搬运，这将花费非常非常多的时间。使用 Yocto 制作出来的 SDK 则直接包含了全部内容。

使用 Yocto 制作 SDK：

(1).使用 NXP 提供的 fsl-setup-release.sh 脚本，会新建一个工作文件夹,然后在此文件夹空间下构建镜像。

```
ISTRO=myir-imx-fb MACHINE=mys6ul14x14 source fsl-setup-release.sh \
-b build-myir
tree conf/
conf/
```

```
├── bblayers.conf
├── bblayers.conf.org
├── local.conf
├── local.conf.org
├── local.conf.sample
├── sanity_info
└── templateconf.cfg
```

(2).首先到 recipe-core 里 images 的 core-image-base.bbappend 中在添加所需的额外的软件包或库：hostapd、sqlite3、libwebp、libmicrohttpd、sqlite、jpeg、ffmpeg；

(3).source fsl-setup-release.sh -b build-myir/ ，然后 bitbake core-image-base 编译镜像 ；

(4).在编译完所需的软件包的时候即可开始制作 SDK：bitbake -c populate_sdk core-image-base，至此结束。

编译好的 SDK 在 tmp/deploy/sdk 目录中，运行脚本安装即可。安装完之后的目录树如下：

```
~/.../myir-imx-fb/core-image-base-toolchain >>> tree -L 3
.
└── 4.1.15-2.0.1
    ├── environment-setup-cortexa7hf-neon-poky-linux-gnueabi
    ├── site-config-cortexa7hf-neon-poky-linux-gnueabi
    ├── sysroots
    │   ├── cortexa7hf-neon-poky-linux-gnueabi
    │   └── x86_64-pokysdk-linux
    └── version-cortexa7hf-neon-poky-linux-gnueabi
```

4 directories, 3 files

~/.../myir-imx-fb/core-image-base-toolchain >>>

3.2.3 嵌入式 Linux 系统移植

一个 Linux 嵌入式系统的主要部件有：Bootloader，Kernel，System call interface，C-Runtime library，System shared libraries 和 Root filesystem（ALBERTO L., 2018），架构图如下图 27 所示。

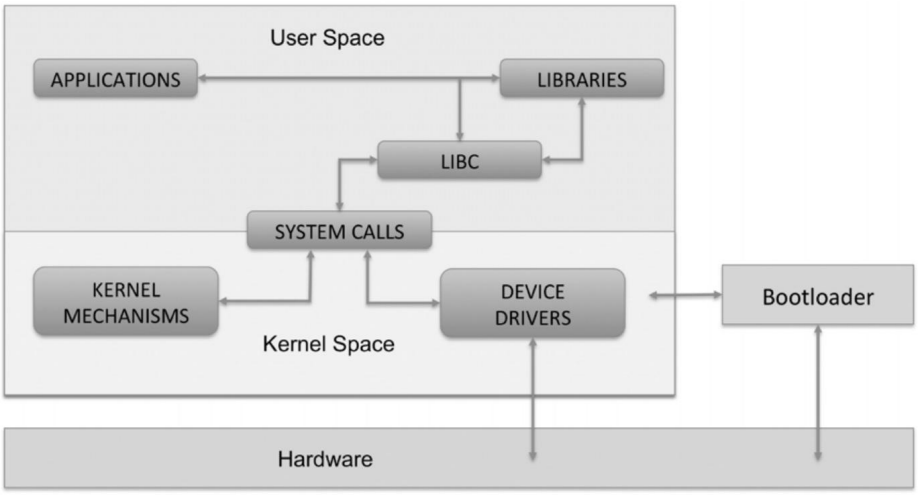


图 27　Linux 嵌入式系统主要部件（ALBERTO L., 2018）

本次设备中的存储设备为 NAND FLASH。在系统上电时，CPU 首先执行处理器内部的 boot ROM 中执行 FPL（First Program Loader），FPL 将判断用户选择从哪里启动下一个程序。启动设置为从 NAND FLASH 启动，则 FPL 将初始化 DDR memory controller、NAND flash controller，然后从 NAND flash 中将 SPL（Second Program Loader）程序搬运至 DDR 内存中，然后跳转执行 DDR 内存中的 SPL 程序。SPL 通常为 U-Boot。然后，U-Boot 将 kernel image、device tree binary 搬运至 DDR 内存中，然后跳转至 DDR 内存中执行 kernel 中 head.s 中的汇编级系统硬件初始化编码，开启 MMU 并创建内存叶表、设置缓存。最后，kernel 从 head.s 跳转到 init/main.c

中的 start_kernel()开始初始化系统核心功能、驱动程序，然后挂载 root filesystem，开始运行第一个用户进程 init，准备完整个系统环境，然后开始运行图像采集软件。除了 FPL 是原本处理器里有的之外，从第二层 bootloader 开始都是我们需要移植的内容。

在上面的 Yocto 编译中已经一次性的编译好了 U-Boot、Kernel、设备树以及包含了所需软件包和库的根文件系统，我们只需要将开发板的拨码开关调到从 USB 启动，接入 USB，使用 NXP 公司提供的 mfgtool 即可一键下载所有内容至 NAND flash 中。

3.2.4 Motion project 移植

简而言之，Motion 是一个开源的高度可配置的移动图像侦测软件，它支持 RTSP、RTMP、HTTP 协议传输视频流和 web 控制台，并且支持 tls 加密传输。

它的移动图像抓拍原理是在上一帧减去当前帧图像的基础上对图像进行了分割后分别处理，对每幅子图像进行帧差，对图像中不一样的面积进行方差计算，以在呼吸灯、日夜转换时的小块、大片图像变化时降低误报率。

进入 motion 源码目录，./configure --host=arm-linux，make -j8，make install 即可编译完成，将编译后的安装文件夹下载到开发板中即可运行。

至此，安防监控系统的前端系统即摄像头监控子系统搭建完成，接入报警系统即可组成完整的安防监控系统。

3.3 小型嵌入式服务器

3.3.1 服务器原理简介

简而言之，服务器和客户端之间是通过媒介手段按照双方约定好的规则进行信息交换，而我们使用的媒介手段则是以 TCP/IP 为基础的互联网通信。其中应用层软件主要以 TCP 和 UDP 作为基础，再按应用层自己的信息交换规则进行收发消息。

其中，TCP 是面向连接的、非常可靠的基于字节流的传输层协议，它是一种双向连

接，并确保收、发成功的每一个 bit 都是正确的。TCP 以字节流进行传输，没有包的概念，因此应用层需要自己进行解析。UDP 是一种无连接的传输协议，它只负责将数据包发出去，并不保证数据能发送到目的地，数据的校验需要应用层自己进行，适合发送实时的、数据准确率要求不高的数据，如传输直播视频数据。

本设计中使用 TCP 作为服务器和客户端通信基础，以 HTTP 作为通信规则。服务器的主要内容为解析浏览器端发送过来的资源请求、资源上传等信息头信息，并向浏览器作出回复，请求或回复均按照先发 HTTP 消息头再发 HTTP 消息体的顺序。HTTP 消息头是在，客户端请求（Request）或服务器响应（Response）时传递的，位请求或响应的第一行，HTTP 消息体（请求或响应的内容）是其后传输。其中一些关键的 HTTP 消息头如下表 1 和表 2 所示：

表 1　关键部分 HTTP Request Head

协议头	说明	示例
Accept	可接受的响应内容类型 (Content-Types)	Accept: text/plain
Connection	客户端（浏览器）想要优先使用的连接类型	Connection: keep-alive Connection: Upgrade
Content-Length	以 8 进制表示的请求体的长度	Content-Length: 348
Content-Type	请求体的 MIME 类型	Content-Type: application/json
Host	域名以及服务器监听的端口号，默认端口号可省略	Host: whqee.cn:80 Host: whqee.cn

Origin	发起跨域资源共享的请求	Origin: http://whqee.cn
User-Agent	浏览器的身份标识字符串	User-Agent: Mozilla/.....

表 2　关键 HTTP Response Head

响应头	说明	示例
Access-Control-Allow-Origin	指定哪些网站可以跨域源资源共享	Access-Control-Allow-Origin: *
Cache-Control	通知从服务器到客户端内的所有缓存机制，表明缓存最大有效时间（秒）	Cache-Control: max-age=3600
Connection	针对该连接所预期的选项	Connection: close
Content-Encoding	响应资源所使用的编码类型	Content-Encoding: gzip
Content-Language	响应内容语言格式	Content-Language: zh-cn
Content-Length	响应消息体长度，用 8 进制字节表示	Content-Length: 348
Content-Type	当前内容的 MIME 类型	Content-Type: text/html; charset=utf-8
Server	服务器的名称	Server: Tiny Server
Status	当前 HTTP 连接响应状态	Status: 200 OK

3.3.2　服务器软件

本文中的服务器基于 Linux 系统和 glibc 进行开发，因为是完全自主开发且时间有限，目前功能暂不全，仅支持 HTTP GET 和自定义动态请求以及自定义协议，项目地址位于 https://github.com/whqee/tiny_server，支持多线程，支持多域名-多站点绑定，这意味着你可以用一个 80 端口去为许多个不同的域名分别绑定不同的网站。编译后仅几十 KB，非常的小，十分适合用在低资源嵌入式设备中。

3.4　反向代理软件

3.4.1　NAT 原理

在现今路由器使用 NAT 技术扩展子网 IP 给用户使用的背景下，反向代理技术应需求而生。要实现反向代理，我们首先要理解 NAT 技术的原理。

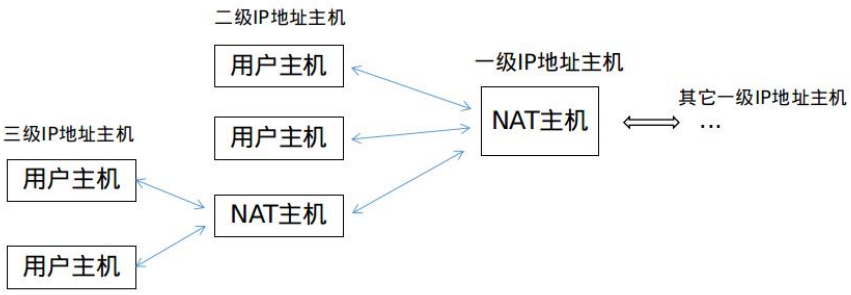


图 28　多级 IP 主机关系

如上图 28 所示，在 Internet 中，只有拥有一级 IP 地址，即公网 IP 地址的主机能够访问其它一级 IP 地址主机，二级 IP 地址主机想要访问一级 IP 地址主机就需要一台拥有一级 IP 地址主机来代其访问。同理，二级 IP 地址主机之于三级 IP 地址主机亦如此，原理都是上一级主机代替下一级主机去访问上一级的其它主机。

当下级主机发出“ip+端口”的请求的时候，上级主机记录下级主机的“ip+端口”

和目标“ip+端口”并把自己一个端口映射到下级主机的“ip+端口”、使用自己的“ip+端口”去访问目标端口，当目标“ip+端口”的数据包返回时，上级主机根据端口映射记录把数据包转发给对应的下级主机的“ip+端口”，在断开连接后，这个映射规则就会被清除。关系图如下图 29 所示：

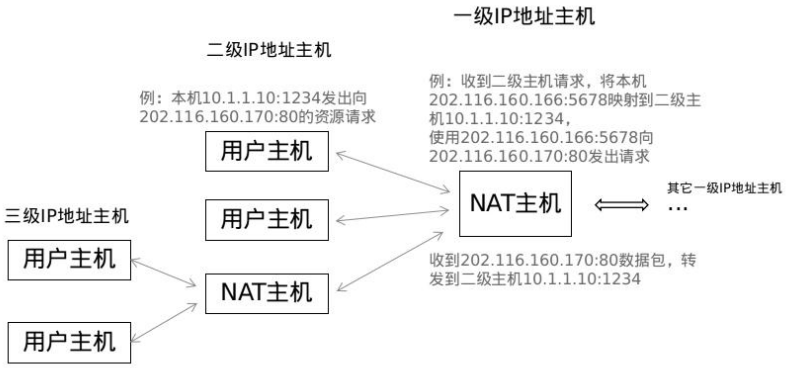


图 29　NAT 工作原理

当下级主机没有发出请求的时候，如果没有特意添加映射规则，NAT 主机是没有映射关系的，那么当外部主机发起请求或数据时，NAT 主机不知道这是发给谁的，所以 NAT 主机不对该外部请求作任何处理。因此，要让外部主机能在任意时候能够访问本地子网内主机，由子网主机发起的通道连接状态必须要一直保持，为此本地子网（内网）主机就需要不断地向外网主机发送心跳数据包以让 NAT 主机保持映射转发状态。

3.4.2　反向代理软件设计

简而言之，我们的服务器部署在子网内，外部网络的设备访问不到，我们通过一台能被内网主机主动发起双向连接的公网主机代替用户客户端进行对子网服务器的访问。内网主机向一台带公网 IP 的主机持续地发送心跳数据包保持连接状态，然后公网主机代替用户客户端将数据包发送至内网主机客户端，进而内网主机客户端将请求数据包转发

至内网服务器。软件整体架构如图 30 所示：

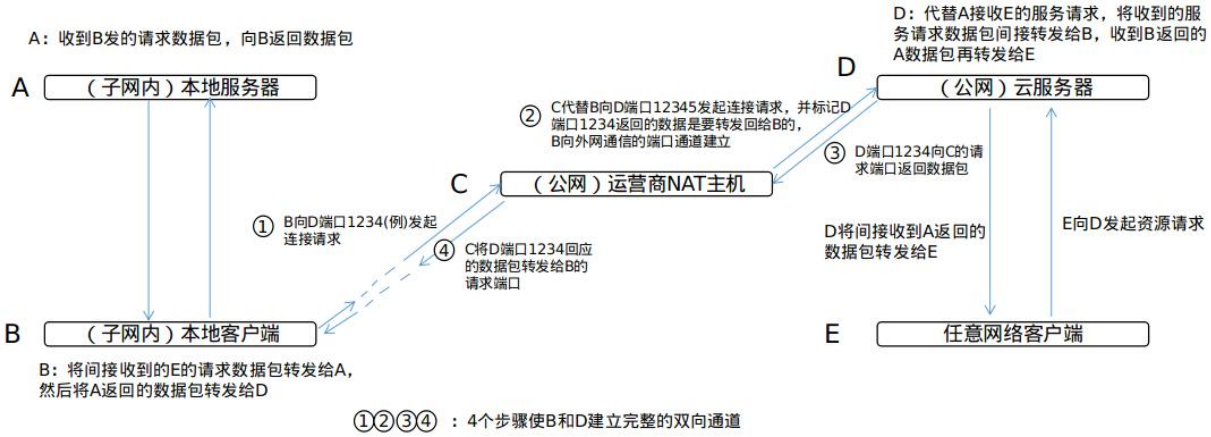


图 30　反向代理软件架构

反向代理软件只需使用 Linux C + glibc 库实现，分为公网服务器端、内网客户端，分别部署在内网摄像头监控设备、公网云主机，成对使用。

本次设计中自主开发的反向代理软件功能未能完整，由于时间关系，进而采用了开源的 FRP 反向代理软件。

3.5　web 端人机交互网页

利用 HTML、CSS 和 JS 对浏览器页面进行渲染，对页面进行布局，然后使用 ajax 异步地接受视频流或传感器数值，并对页面进行更新。本次页面较为简单。

3.6　安卓手机客户端

3.6.1　本文安卓客户端设计原理介绍

本设计中的手机 APP（Application）采用移动框架开发，即使用开发网站的方式开发。

原理简介：利用 WebView 框架，手机 APP 可以像浏览器一样打开任意网页，若在 APP 中开启一个小型的本地 HTTP 服务器，将网页部署在本地，然后直接访问本地 HTTP

服务器，这样就可以实现一个手机 APP。

除了像游戏领域这样的对速度非常敏感的 APP 之外，这种开发方式已经成为主流。我们只需要开发一套适应手机屏幕的 HTML5 页面布局配合后端逻辑就可以开发一套跨平台的 APP，可以直接一次在 Android、IOS、Windows、Linux 系统而不需要针对各个平台再独立开发。由于笔者没有苹果系列电脑，无法对进行 IOS 系统进行部署。现仅针对安卓（8 及以上）系统进行测试。

本文使用 Apache Cordova 进行 APP 开发。Apache Cordova 是一套使用 HTML、CSS 和 Javascript 构建移动应用的平台，也是一套代码（框架）。

3.6.2 搭建开发环境

(1).Android SDK

参考安卓官网 https://developer.android.google.cn/studio/即可，可直接安装 Android Studio 直接安装全套内容，也可自行从命令行按需安装。本文仅介绍从命令行安装 sdk：

i.下载命令行工具并配置初始环境：（先安装好 java 开发环境，比如 open-jdk）到安卓官网下载 command line tools，在任意位置新建一个文件夹，任意名称，推荐命名为 android_sdk，把下载的 commandlinetools 解压到这里。查看 usage:“tools/bin/sdkmanager --help”，“tools/bin/sdkmanager --sdk_root=. --list”可以看到一大页的列表，为了方便输入命令，起个别名：“alias sdkmanager="tools/bin/sdkmanager --sdk_root=. " ”

ii.通过命令行工具下载安卓平台和编译工具链:安装 android29 平台和工具链,当然你也可以安装其它的，“sdkmanager "platforms;android-29" "build-tools;29.0.3"”，文件夹内容如下所示：

[whqee@whqee android_sdk]$ tree -L 2

```
.
├── build-tools
│   └── 29.0.3
├── licenses
│   └── android-sdk-license
├── platforms
│   └── android-29
└── tools
    ├── bin
    ├── lib
    ├── NOTICE.txt
    ├── package.xml
    └── source.properties
8 directories, 4 files
```

至此，编译安卓客户端所需的 SDK 中的材料准备完成。

(2).Apache Cordova

Cordova 依赖于 Node.js，在 Arch 系 Linux 中直接“sudo pacman -S nodejs”即可。然后使用 npm 命令安装 cordova 即可：“sudo npm install -g cordova”。

其次，Cordova 使用 gradle 进行自动化编译 APP，同样地，直接“sudo pacman -S gradle”即可。其它类别 Linux、Mac OS 或 Windows 可参考官网。

3.6.3 编译安卓客户端

编译过程参考 Cordova 官网文档，下面列出关键步骤。

(1).使用 Cordova 创建 APP 项目工程：“cordova create SmartHomeApp”；

(2).进入 APP 文件夹：“cordova platform add android --save”；将 android sdk 中的 tools 和 platform-tools 文件夹路径添加到环境变量中，然后“cordova requirements”查看是否满足编译条件，在我的 zsh 命令行中，结果如下：

~/.../Cordova/SmartHomeApp >>> cordova requirements

Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: installed android-29,android-28
Gradle: installed /usr/share/java/gradle/bin/gradle

(3).编译并打包发布：

在修改完 APP 后，在根目录新建 build.json，内容如下：

```
{
    "android": {
        "release": {
            "keystore": "keystore",
            "alias": "android",
            "storePassword": "android",
            "password": "android"
        }
    }
}
```

然后“keytool -genkey -v -keystore keystore -alias android -keyalg RSA -validity 10000”生成密钥库，执行“cordova run android --release”即可编译完成，并且 gradle 根据 build.json 自动对 apk 打包签名，最终的 APP 路径为：platforms/android/app/build/outputs/apk/release/app-release.apk

至此，安卓客户端编译完成。

3.7 小结

本章节对本文所涉及的技术原理进行了介绍并对本文内容进行了设计。

4　测试与试验分析

4.1　传感数据监控显示终端测试

传感监控显示终端的主要内容为收集环境光照度、温度、湿度并显示以及上传服务器。其次它还拥有控制灯具开关的功能，并抽象出了控制接口方便后续接入语音控制功能。下列内容为测试内容。

在黑夜中测试（模拟厅室，不接红外人体感应电路模块），如图 31：

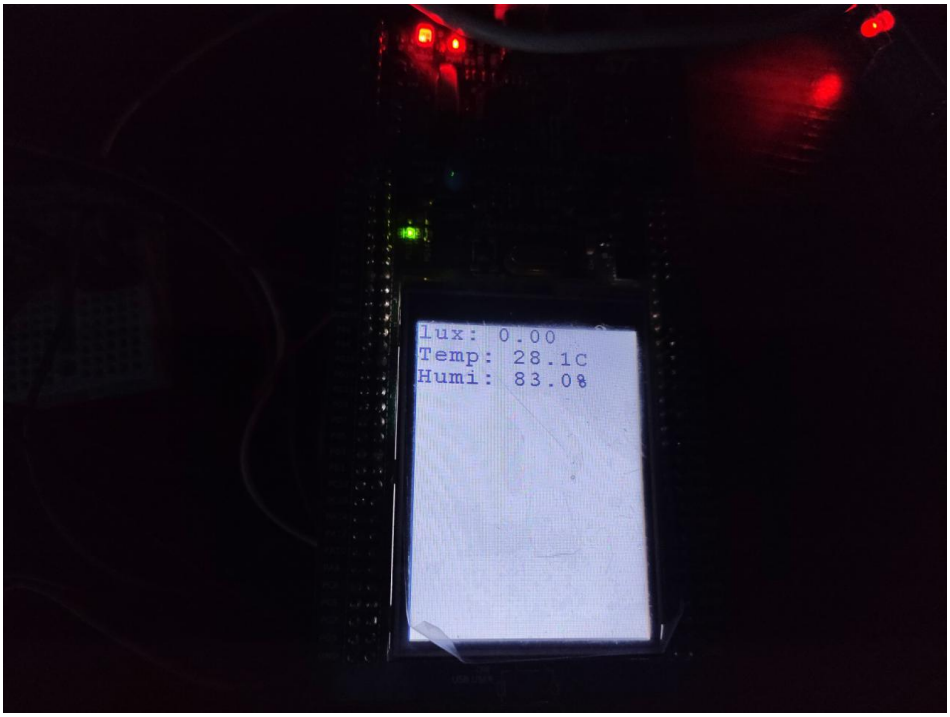


图 31　夜间测试 1

在黑暗中测试（模拟楼道，接入红外人体感应电路模块），如图 32：

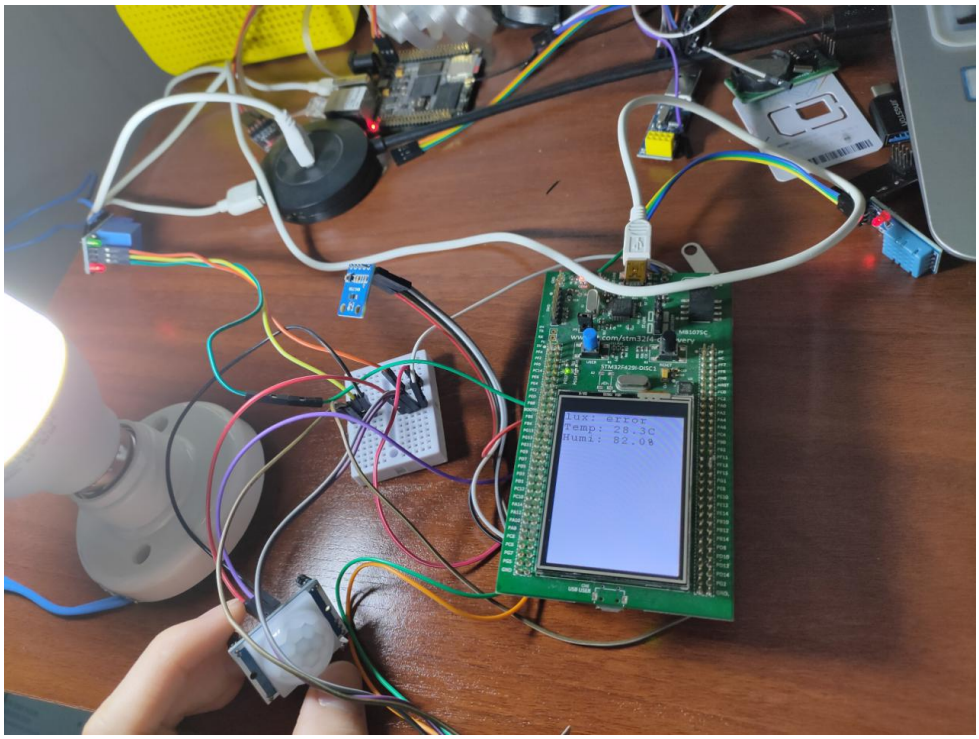


图 32　夜间测试 2

传感监控显示终端软件功能正常。但要作为真正的智能家居设备、使用用户拥有良好的体验还需要接入语音控制系统、美化显示界面。

4.2　摄像头监控子系统、反向代理、服务器和网站的综合测试

当摄像头监控子系统与反向代理软件运行后，我们可以从外网访问家居局域网里的监控摄像头，浏览器输入提前绑定的域名 iot.whqee.cn，如图 33 所示：

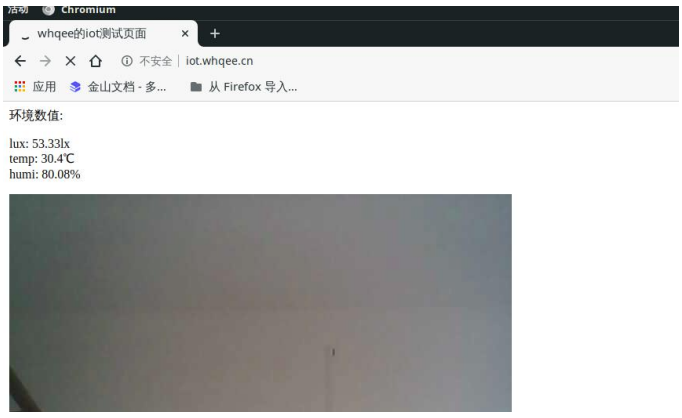


图 33　电脑端视频网页测试

手机输入网址 iot.whqee.cn 打开，如图 34 所示：

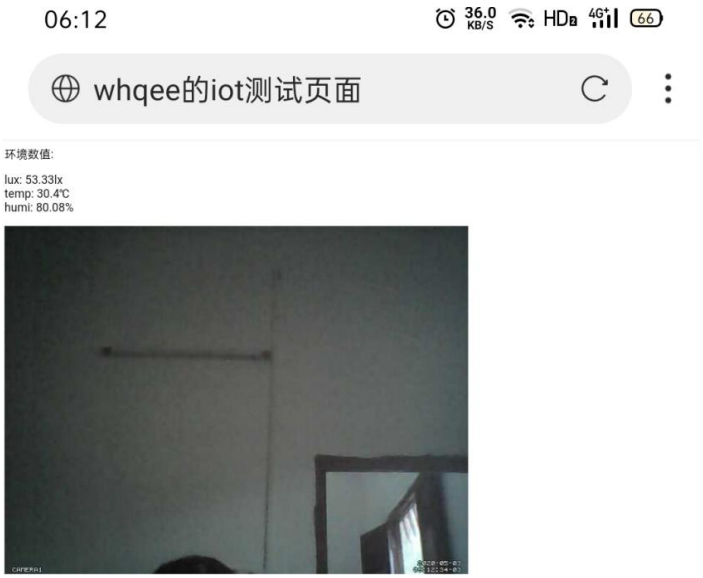


图 34　手机端视频网页测试

用手机输入 test.whqee.cn 打开单独的视频流页面，如图 35 所示：

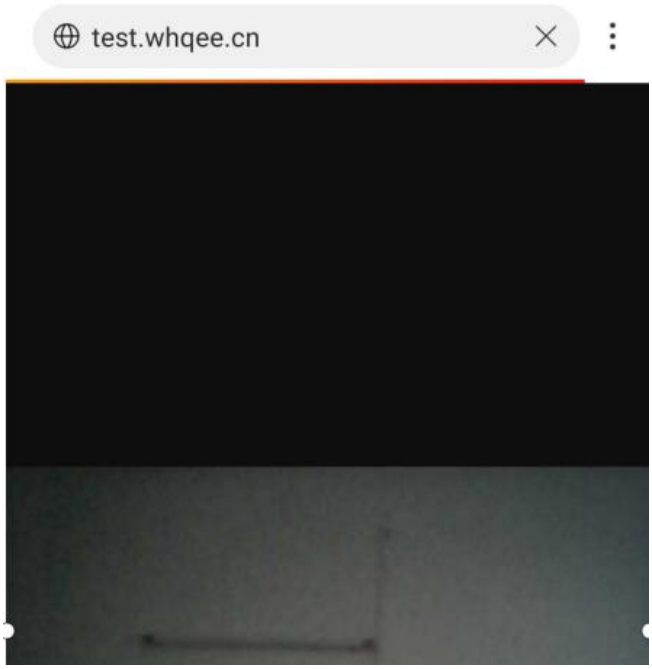


图 35　手机端视频流网页测试

用电脑浏览器打开 http://iot.whqee.cn/iot?sensor 测试网站动态请求链接，如图 36 所示：

图 36　电脑端网站测试

从浏览器查看数据的动态请求状态，如图 37 所示：

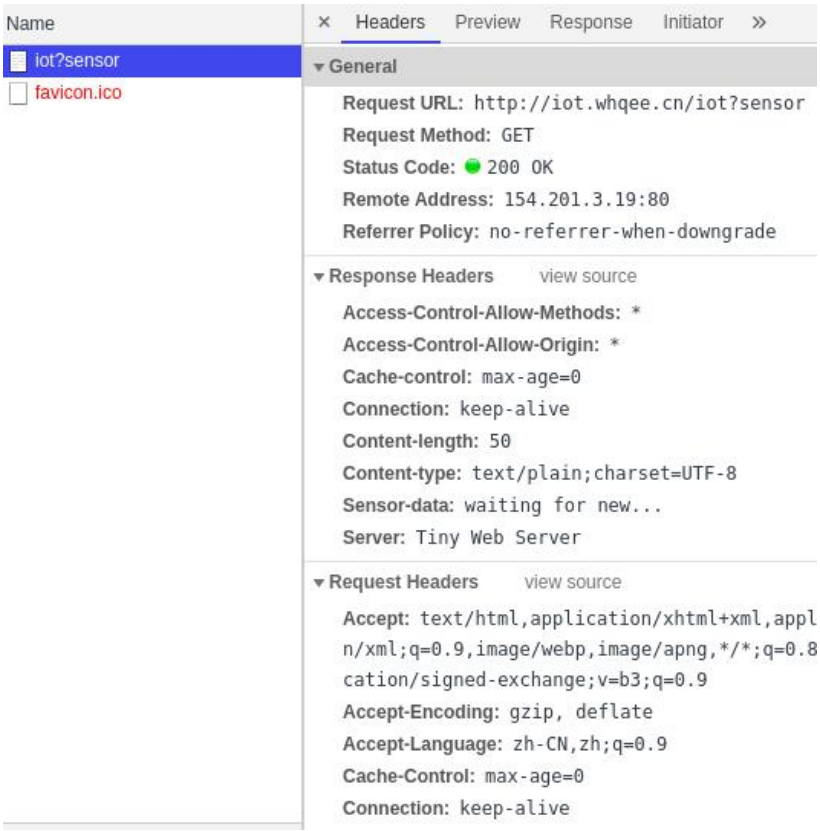


图 37　HTTP 动态请求测试

从上面可以看到“Server: Tiny Web Server”，以及 http://iot.whqee.cn/iot?sensor 的动态请求得到正常答复。

从综合测试结果可知，域名解析正常，摄像头监控软件、反向代理软件、自主开发服务器软件功能正常，网站需要优化页面布局。

4.3 安卓客户端测试

安卓客户端测试如图 38：

图 38 安卓客户端测试

客户端下载：http://iot.whqee.cn/app-release.apk 。安卓客户端正常，界面需要优化。

4.4 小结

通过对传感监控显示终端、摄像头监控子系统、服务器、反向代理以及安卓客户端的测试显示，上述内容构成的智能家居系统实现了家居环境监测、家居灯具控制、视频监控的功能。

5 结论

5.1 本文总结

本文主要的内容是智能家居系统设计，对智能家居系统的整体结构进行设计，并对部分子系统进行软件开发，其主要原理是利用微控制器技术、通信技术、计算机软件技术来进行设计和实现。

本文的主要工作：

(1).论文阐述了智能家居设计的研究背景和意义，并对设计方案作出了分析、确定了本文的探究方向和主要内容；

(2).比较了几种嵌入式技术、操作系统和无线通信技术的优缺点，并简短介绍了移动图像侦测技术和部分互联网技术，确立了本文设计的总体结构方案；

(3).对本文所涉及的硬件驱动程序设计、应用程序设计、操作系统移植、服务器开发、反向代理、建站技术、安卓移动 APP 构建技术的原理进行了阐述或简介，并进行了理论设计和构建；

(4).对论文设计内容进行了实施和测试，验证了本文理论和技术的可行性。

5.2 对本文设计的展望

本文对（完善的）智能家居系统中的传感数据收集显示、灯具的无线控制、安防系统前端的摄像头（视频）监控和针对低资源物联网设备的服务器等软件进行了理论探究，并初步实施了基本功能。

初步体验了自己设计的智能家居设计之后，初尝了甜头。其中感受最大的是无线网络技术让生活便利很多，其次是迫切地想对智能家居中的语音控制子系统进行设计实现并投入使用。根据本次经历总结，我认为，未来语音控制子系统将会是整个智能家居系统的灵魂所在。

回顾本文，本次设计中虽然完成了系统部分功能的测试，达到了预期目标，但还有很多需要完善的地方：

(1).大部分内容从零自主开发，许多细节未能做好，用户交互界面过于简单，移动物体抓拍的视频保存在了本地而用户访问方式却还没有开发；

(2).为购买的部分器件便宜但比较劣质，使系统未能如预期地稳定运行；

(3).系统安全性不足，家居数据连接没有加密，并且不用登录验证就可以访问系统资源，TLS 加密、自定义加密、自主开发的家居设备专用数据库未能预期上线，系统安全性有待完善；

(4).语音控制子系统、报警子系统待开发。

上述内容将是本系统未来改进、升级的主要研究方向，希望通过深入学习新技术知识能不断地完善该系统、不止于该系统。未来将继续学习、并应用所学知识在生活中创造更多的价值。

参 考 文 献

李天祥. Android 物联网开发细致入门与最佳实践[M]. 中国铁道出版社, 2016: 14-15.

何迎朝, 邢文华. 智慧居家养老技术采纳的影响因素及其使用效果研究: 文献综述的视角 [J/OL]. 信息资源管理学报, 2020.03.15: 1-13. http://kns.cnki.net/kcms/detail/42.1812.G2.20200308.1607.006.html.

杨波, 林毓铭, 丑建忠. 广州市智慧居家养老服务质量评价[J]. 社会保障研究, 2017(4): 21-35.

Altman L. *Single Chip Microprocessors open up a New World of Applications*[J]. Electronics, 1974.04.18: 81-87.

Augarten S. *State Of The Art: A Photographic History of the Integrated Circuit*[M]. New York: Houghton Mifflin, 1983: 30-48.

Beal V. "*What is Wi-Fi (IEEE 802.11x)? A Webopedia Definition*"[DB/OL]. Webopedia, 2012.

ARM. *ARMv7-M Architecture Reference Manual* [M]. ARM, 2014-10-02: 21.

葛翠艳, 珠海国芯云科技有限公司. 物联网中的无线传感 ZigBee 技术[J]. 《电子世界》, 2019(10).

[美]W·Richard S. TCP/IP 详解 卷 1:协议[M]. 范建华. 北京: 机械工业出版社, 2000: 1-14.

ALBERTO L. *Linux Driver Development for Embedded Processors*[M]. 第二版. Spanish: ALBERTO L., 2018: 19-55.

附录

/** 小型服务器主程序 **/

```
/* ----------------------
* update time: May.02 2020
* author: whqee
* e-mail: whqee@qq.com
* ----------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/epoll.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <assert.h>
#include <signal.h>
#include <pthread.h>
#include <sys/syscall.h>
#include <locale.h>

#define MULTI_CNAME_SERVICE

#define STRING_SIZE 1024
#define MAX_LISTEN 128 // n <= 128

static const char response_ok_head[] = "\
HTTP/1.0 200 OK\r\n\
Server: Tiny Web Server\r\n\
Connection: keep-alive\r\n\
Access-Control-Allow-Origin: *\r\n\
Access-Control-Allow-Methods: *\r\n\
";

static int send_file(int srcfd, int filesize, int sock)
{
if (filesize > 0) {
```

```
fprintf(stdout, "send_file(): sending file...\n"); // dbg msg
/* map a mem addr to the file, return the mem pointer of the start address */
void *fp = mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd, 0);
if (fp == MAP_FAILED) {
perror("map mem address failed in send_file()\n");
return -1;
}
/* send the file */
int nleft = filesize;
ssize_t nwritten;
char *bufp = fp;
while (nleft > 0) {
nwritten = write(sock, bufp, nleft);
if (nwritten <= 0) {
if (errno == EINTR || errno == EAGAIN || errno == EBUSY)
nwritten = 0;
else {
munmap(fp, filesize);
return -1;
}
}
nleft -= nwritten;
bufp += nwritten;
}
/* end of sending file */
munmap(fp, filesize);
return filesize;
}
return -1;
}

#ifdef MULTI_CNAME_SERVICE
static char g_host_binding_list[STRING_SIZE] = {0};

static void filter_comment(char *src, int src_size, char *buf, int buf_size)
{
char *p1 = src , *p2 = buf, *p2max = p2 + buf_size;
int i, flag_comment = 0;
while(src_size) {
if (*p1 == '#')
flag_comment = 1; // comment after
if (flag_comment) {
if (*p1 == '\n')
flag_comment = 0;
} else {
```

```
*p2++ = *p1;
if (p2 == p2max) {
fprintf(stdout, "buffer is full\n"); //dbg msg
fprintf(stdout, "buffer:\n%s\n", buf); //dbg msg
break;
}
}
p1++;
src_size--;
}
return;
}

static int parse_conf_from_file(int fd, char *buf, int size)
{
/* map a mem addr to the file, return the mem pointer of the start address */
int filesize = lseek(fd, 0, SEEK_END);
void *fp = mmap(0, filesize, PROT_READ, MAP_PRIVATE, fd, 0);
if (fp == MAP_FAILED) {
perror("map mem address failed in host_list_init() for file host.conf\n");
return -1;
}
filter_comment(fp, filesize, buf, size);
munmap(fp, filesize);
}

static void host_list_init()
{
int fd = open("/etc/tiny_server/host.conf", O_RDONLY, S_IREAD);
if (fd < 0) {
perror("/etc/tiny_server/host.conf");
fd = open("host.conf", O_RDONLY, S_IREAD);
}
if (fd < 0) {
perror("./host.conf");
return;
}
if (parse_conf_from_file(fd, g_host_binding_list, sizeof(g_host_binding_list)) <
0) {
perror("host_list_init(): failed. run default.\n");
}
close(fd);
}

static void parse_host(char *host, char *uri)
```

```
{
fprintf(stdout, "request host:%s\n", host);
char *s, path[STRING_SIZE];
s = strstr(g_host_binding_list, host);
if (s != NULL && s != g_host_binding_list)
do {
s = index(s, '\n');
if (!s)
break;
s = strstr(s+1, host);
} while (*(s-1) != '\n');
if (s == NULL) {
sprintf(path, "_site%s", uri);
sprintf(uri, "%s", path);
fprintf(stdout, "domain not matching, run default response\n"); // dbg msg
return;
}
sscanf(s, "%*[^ ] %s\n", path);
strcat(path, uri);
sprintf(uri, "%s", path);
fprintf(stdout, "path:%s\n", path); // dbg msg
}
#endif // MULTI_CNAME_SERVICE

static int security_check(char *p)
{
// char *p = uri;
/* check string ".." for security */
while(*p++ != '\0') {
if (*p == '.' && *p == *(p-1)) {
perror("String \"..\" is forbidden.\n"); // dbg msg
return -1;
}
}
/* end security check */
return 0;
}

static void http_uri_decode_to_utf8(char *uri)
{
if (uri == NULL) return;
printf("http_uri_decode_to_utf8()%d:%s\n", strlen(uri), uri); // dbg msg

char tmp[STRING_SIZE], *p1 = uri, *p2 = tmp;
memset(tmp, 0, sizeof(tmp));
```

```
do {
if (*p1 == '%') {
if (*(p1+1) < 0x41)
*p2 = (*(p1+1) - 0x30)<<4;
else
*p2 = (*(p1+1) - 55)<<4;
if (*(p1+2) < 0x41)
*p2 += (*(p1+2) - 0x30);
else
*p2 += (*(p1+2) - 55);
p1 += 3; p2++;
} else {
*p2++ = *p1++;
}
} while (*p1 != '\0');
sprintf(uri, "%s", tmp);
printf("after http_uri_decode_to_utf8()%d:%s\n", strlen(uri), uri); // dbg msg
}

/* Dynamic request - return 1
* Static request - return 0
* uri error - return -1
*/
static int parse_http_uri(char *uri, char *filetype)
{
const char *type[] = {".html",".css",".js",".json",".gif",".png",".jpg",".ico",".webp",".pdf"};
const char *http_type_table[] = {"text/html","text/css","application/js","application/json",
"image/gif","image/png","image/jpeg","image/ico","image/webp","application/pdf","text/plain"};
http_uri_decode_to_utf8(uri);
if (!strcmp(uri, "/")) {
sprintf(uri, "index.html");
sprintf(filetype, "text/html");
return 0;
}
fprintf(stdout, "parsing uri<%d>: %s\n", strlen(uri), uri); // dbg msg
if (strchr(uri, '?') != NULL) /* dynamic req */
return 1;
if (security_check(uri) < 0) {
perror("Unsafe uri request.\n");
return -1;
}
#ifdef MULTI_CNAME_SERVICE
```

```
if (uri[strlen(uri) - 1] == '/')
uri[strlen(uri) - 1] = '\0';
#else
sprintf(uri,"%s", uri+1);
#endif
struct stat buf;
if (lstat(uri, &buf) < 0)
return -1;
if (S_ISDIR(buf.st_mode)) { // if dir, append "/index.html" to uri
sprintf(uri, "%s/index.html", uri);
sprintf(filetype, "text/html");
return 0;
}
if (!S_ISREG(buf.st_mode)) {
perror("File does not exist or not allowed to read. Only normal files is readable.");
return -1;
}
char *p = rindex(uri, '.');
if (!p) {
fprintf(stdout, "file type doesn't exist, return file as text/plain.\n");
strcpy(filetype, "text/plain");
return 0;
}
for (size_t i = 0; i < 10; i++) {
if (strstr(p, type[i])) {
strcpy(filetype, http_type_table[i]);
break;
}
if (i == 10) strcpy(filetype, "text/plain");
}
printf("filetype: %s\n", filetype); // dbg msg
return 0;
}

static int order_reply_to_sensor = 0;
static char msg_reply_to_sensor[16];
static int sensor_pushed = 0;

static int dynamic_uri(int sock, char *uri)
{
int ret;
printf("recv dynamic req.\n");
char *p = strchr(uri, '?');
if (p == NULL)
```

```
return -1;
char req[64], path[64], response_head[STRING_SIZE];
memset(req, 0, sizeof(req));
memset(path, 0, sizeof(path));
if (sscanf(uri, "%*[^i]iot?%s", req) == -1)
return -1;
printf("%s\n", req); // dbg
if (!strcmp(req, "sensor")) {
memset(response_head, 0, sizeof(response_head));
sprintf(response_head, response_ok_head);
sprintf(response_head, "%sContent-type: %s;charset=UTF-8\r\n",
response_head, "text/plain");
sprintf(response_head, "%sCache-control: max-age=0\r\n", response_head);
sprintf(response_head, "%sSensor-data: waiting for new...\r\n",
response_head);
if (!sensor_pushed) {
sprintf(response_head, "%sContent-length: %d\r\n\r\n", response_head, 0);
return write(sock, response_head, strlen(response_head));
}
sscanf(uri, "%[^/]", path);
sprintf(path, "%s/iot.db", path);
printf("parsed path:%s\n", path);
int fd = open(path, O_RDONLY, S_IREAD);
if (fd < 0) {
fprintf(stderr, "open %s error.\n", path);
return -1;
}
int filesize;
filesize = lseek(fd, 0, SEEK_END);
sprintf(response_head, "%sContent-length: %d\r\n\r\n", response_head,
filesize);
if (ret = write(sock, response_head, strlen(response_head)) < 0) {
close(fd);
return ret;
}
printf("data path:%s\nsending data...",path);
ret += send_file(fd, filesize, sock);
close(fd);
return ret;
}
if (!strcmp(req, "lighton")) {
memset(msg_reply_to_sensor, 0, sizeof(msg_reply_to_sensor));
sprintf(msg_reply_to_sensor, "lon\r\n\r\n");
order_reply_to_sensor = 1;
goto reply_200ok;
```

```
}
if (!strcmp(req, "lightoff")) {
memset(msg_reply_to_sensor, 0, sizeof(msg_reply_to_sensor));
sprintf(msg_reply_to_sensor, "loff\r\n\r\n");
order_reply_to_sensor = 1;
goto reply_200ok;
}
printf("dynamic_uri: do nothing.");
return 0;
reply_200ok:
memset(response_head, 0, sizeof(response_head));
sprintf(response_head, response_ok_head);
sprintf(response_head, "%sContent-type: %s;charset=UTF-8\r\n",
response_head, "text/plain");
sprintf(response_head, "%sContent-length: %d\r\n", response_head, 0);
sprintf(response_head, "%sCache-control: max-age=0\r\n\r\n",
response_head);
return write(sock, response_head, strlen(response_head));
}

static int static_server(int sock, char *uri, char *filetype)
{
fprintf(stdout, "static request:%s\n", uri); // dbg msg
int ret, fd, filesize;
char response_head[STRING_SIZE];
fd = open(uri, O_RDONLY, S_IREAD);
if (fd < 0){
perror("File does not exist or permmition diny.\n"); // dbg msg
return fd;
}
else {
filesize = lseek(fd, 0, SEEK_END);
memset(response_head, 0, sizeof(response_head));
sprintf(response_head, response_ok_head);
sprintf(response_head, "%sContent-type: %s;charset=UTF-8\r\n",
response_head, filetype);
sprintf(response_head, "%sContent-length: %d\r\n", response_head,
filesize);
sprintf(response_head, "%sCache-control: max-age=30\r\n\r\n",
response_head);
if (ret = write(sock, response_head, strlen(response_head)) < 0) {
close(fd);
return ret;
}
ret += send_file(fd, filesize, sock);
```

```
close(fd);
fprintf(stdout, "my response(%d Byte):\n%s\n", ret, response_head); // dbg
msg
return ret;
}
}

static int handle_http_request(int sock, char * msg)
{
fprintf(stdout, "parsing http request:\n");
char method[STRING_SIZE], uri[STRING_SIZE], host[128],
connection[STRING_SIZE], response_head[STRING_SIZE],
filetype[STRING_SIZE];

sscanf(msg, "%s %s HTTP%*[^H]Host: %s\n", method, uri, host);
// printf("method:%s\nuri:%s\nhost:%s\n", method, uri, host); // dbg msg
if (!strcmp(method, "GET")) { /* only support GET */
#ifdef MULTI_CNAME_SERVICE
parse_host(host, uri);
#endif // MULTI_CNAME_SERVICE
if ( parse_http_uri(uri, filetype) ) { /* dynamic request */
return dynamic_uri(sock, uri);
}
else { /* static request */
return static_server(sock, uri, filetype);
}
}
sprintf(response_head, "HTTP/1.1 405 Method Not Allowed\r\n");
sprintf(response_head, "%sContent-length: 0\r\n", response_head);
sprintf(response_head,  "%sContent-type:  text/html;charset=UTF-8\r\n\r\n",
response_head);
sprintf(response_head, "%s405 Method Not Allowed\r\n\r\n",
response_head);
fprintf(stdout, "\nresponse:\n%s\n", response_head);
return send(sock, response_head, strlen(response_head), 0);
}

static int handle_message(int sock, char * msg)
{
fprintf(stdout, "handling message... ");
char method[STRING_SIZE], uri[STRING_SIZE], version[STRING_SIZE];
/** recv sensor data that starts with "IOT" and simply save at IOT/iot.db (for
testing) **/
if (*msg == 'I' && *(msg+1) == 'O' && *(msg+2) == 'T') {
if (order_reply_to_sensor) {
```

```
sensor_pushed = 1;
order_reply_to_sensor = 0;
if (send(sock, msg_reply_to_sensor, strlen(msg_reply_to_sensor), 0) <= 0) {
perror("Error in sending msg to sensor controller.");
/* reserved */
}
}
int fd = open("IOT/iot.db", O_WRONLY | O_TRUNC);
if (fd < 0)
return -1;
int n = write(fd, msg+3, strlen(msg)-3);
if (n == -1) {
perror("IOT write: error.\n");
close(fd);
return n;
}
if (n == 0) {
perror("IOT write: 0 byte.\n");
close(fd);
return n;
}
close(fd);
return n;
}
/** end sensor data **/
sscanf(msg, "%s %s %s", method, uri, version);
printf("%s %s %s...\n", method, uri, version);
if (!strcmp(version, "HTTP/1.1"))
return handle_http_request(sock, msg);
return -1;
}

static void * thread_handling_new_client(void * arg)
{
fprintf(stdout, "New thread started, PID %d, TID %d\n", getpid(),
(pid_t)syscall(SYS_gettid));
char buf[STRING_SIZE];
int client_sockfd = *(int*)arg;
while (1)
{
memset(&buf, 0, sizeof(buf));
int nbyte = recv(client_sockfd, buf, sizeof(buf)-1, 0);
if (nbyte == -1) {
perror("read error.\n");
close(client_sockfd);
```

```
break;
}
else if (nbyte == 0) {
fprintf(stderr, "client closed, fd=%d\n", client_sockfd);
close(client_sockfd);
break;
}
else {
fprintf(stdout, "fd=%d recieved %d bytes data:\n%s\n", client_sockfd, nbyte,
buf);

nbyte = handle_message(client_sockfd, buf);
if (nbyte == -1) {
fprintf(stderr, "fd=%d send error.\n", client_sockfd);
close(client_sockfd);
break;
}
}
}
fprintf(stdout, "thread exit, PID %d, TID %d\n", getpid(),
(pid_t)syscall(SYS_gettid));
pthread_exit(PTHREAD_CANCELED);
}

int main(int argc, char const *argv[])
{
printf("%s\n", setlocale(LC_CTYPE, "en_US.UTF-8"));
if (argc != 2) {
fprintf(stderr, "usage: %s <port>\n", argv[0]);
exit(1);
}
printf("argv[1]: %s\n", argv[1]);
// int port = atoi(argv[1]);
int server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket == -1) {
fprintf(stderr, "port %s busy.\n", argv[1]);
exit(1);
}
struct sockaddr_in server_addr;
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(atoi(argv[1]));

int reuse = 1, err;
```

```
err = setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse));
if (err == -1) {
fprintf(stderr, "bind port %s failed.\n", argv[1]);
exit(1);
}
err = bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
if (err == -1) {
fprintf(stderr, "bind port %s failed.\n", argv[1]);
exit(1);
}
err = listen(server_socket, MAX_LISTEN); // set max client (n <= 128) ;
if (err) {
fprintf(stderr, "listen port %s failed.\n", argv[1]);
exit(1);
}
fprintf(stdout, "Waiting for client...\n");
#ifdef MULTI_CNAME_SERVICE
host_list_init();
fprintf(stdout, "g_host_binding_list[]:\n%s\n", g_host_binding_list); // dbg msg
#endif // MULTI_CNAME_SERVICE
while (1) {
struct sockaddr_in client_addr;
socklen_t client_addr_len = sizeof(client_addr);

int client_fd = accept(server_socket, (struct sockaddr*)&client_addr, &client_addr_len);
if (client_fd < 0)
fprintf(stderr, "accept an error.\n");
else
fprintf(stdout, "accept a new client: %s:%d\n",
inet_ntoa(client_addr.sin_addr),client_addr.sin_port);
pthread_t t;
err = pthread_create(&t, NULL, thread_handling_new_client, &client_fd);
if (err) perror("pthread create failed.\n");
pthread_detach(t);
}
close(server_socket);
return 0;
}
```

/** bh1750 传感器驱动程序 **/

```
#include "bh1750.h"
#include "i2c.h"

#define i2c hi2c3

/** read 2 bytes from bh1750 to buf **/
int bh1750_read(unsigned char *buf)
{
int ret = HAL_I2C_Master_Transmit(&i2c, BH1750_Write_Address,
BH1750_CONTINUE_H_RES_MODE,1,100);
if (ret != HAL_OK) return ret;
HAL_Delay(120);
return HAL_I2C_Master_Receive(&i2c,BH1750_Read_Address,buf,2, 100);
}

int bh1750_read_lx(float *lx)
{
char buf[2];
int ret = bh1750_read(buf);
if (ret != HAL_OK) return ret;
*lx = (float)( ((uint16_t)buf[0]<<8) + buf[1] )/1.2;
return ret;
}

int bh1750_init()
{
int ret = HAL_I2C_Master_Transmit(&i2c, BH1750_Write_Address,
BH1750_POWER_ON,1,100);
if (ret != HAL_OK) return ret;
ret = HAL_I2C_Master_Transmit(&i2c, BH1750_Write_Address,
BH1750_RSET,1,100);
if (ret != HAL_OK) return ret;

ret = HAL_I2C_Master_Transmit(&i2c, BH1750_Write_Address, 0x42,1,100);
if (ret != HAL_OK) return ret;
ret = HAL_I2C_Master_Transmit(&i2c, BH1750_Write_Address, 0x65,1,100);
if (ret != HAL_OK) return ret;
// 0x42 0x65 -- transmittance 100% for 1750
// 0x44 0x6a -- transmittance 50% for 1750
return HAL_I2C_Master_Transmit(&i2c, BH1750_CONTINUE_H_RES_MODE,
0x65,1,100);
}


/** dht11 传感器驱动程序 **/
```

```
#include "dht11.h"
#include "gpio.h"
#include "tim.h"


#define DHT11_GPIO_D_IN 0
#define DHT11_GPIO_D_OUT 1
static void dht11_gpio_direction(int mode)
{
GPIO_InitTypeDef GPIO_InitStruct = {0};

/*Configure GPIO pin : PtPin */
GPIO_InitStruct.Pin = Sensor_DHT11_Pin;
if (mode)
{
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
}
else
{
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
}
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(Sensor_DHT11_GPIO_Port, &GPIO_InitStruct);
}

/*

* 从 DHT11 读取一个字节，MSB 先行

*/
static uint8_t Read_Byte(void)
{
uint8_t i, temp=0;
for(i=0;i<8;i++)
{

/*每 bit 以 50us 低电平标置开始，轮询直到从机发出 的 50us 低电平 结束*/

while(DHT11_DATA_IN()==GPIO_PIN_RESET);


/*DHT11 以 26~28us 的高电平表示“0”，以 70us 高电平表示“1”，

*通过检测 x us 后的电平即可区别这两个状 ，x 即下面的延时

*/
```

```
Delay_us(40); //延时 x us 这个延时需要大于数据 0 持续的时间即可

if(DHT11_DATA_IN()==GPIO_PIN_SET)/* x us 后仍为高电平表示数据“1” */
{
/* 等待数据 1 的高电平结束 */
while(DHT11_DATA_IN()==GPIO_PIN_SET);

temp|=(uint8_t)(0x01<<(7-i)); //把第 7-i 位置 1，MSB 先行
}
else // x us 后为低电平表示数据“0”
{
temp&=(uint8_t)~(0x01<<(7-i)); //把第 7-i 位置 0，MSB 先行
}
}
return temp;
}

/*
* 一次完整的数据传输为 40bit，高位先出
* 8bit 湿度整数 + 8bit 湿度小数 + 8bit 温度整数 + 8bit 温度小数 + 8bit 校验
和
*/
uint8_t Read_DHT11(DHT11_Data_TypeDef *DHT11_Data)
{
/*输出模式*/
dht11_gpio_direction(DHT11_GPIO_D_OUT);
/*主机拉低*/
DHT11_DATA_OUT(LOW);
/*延时 18ms*/
HAL_Delay(18);

/*总线拉高 主机延时 30us*/
```

```
DHT11_DATA_OUT(HIGH);

Delay_us(30); //延时 30us

/*主机设为输入 判断从机响应信号*/
dht11_gpio_direction(DHT11_GPIO_D_IN);

/*判断从机是否有低电平响应信号 如不响应则跳出，响应则向下运行*/
if(DHT11_DATA_IN()==GPIO_PIN_RESET)
{
/*轮询直到从机发出 的 80us 低电平 响应信号结束*/
while(DHT11_DATA_IN()==GPIO_PIN_RESET);

/*轮询直到从机发出的 80us 高电平 标置信号结束*/
while(DHT11_DATA_IN()==GPIO_PIN_SET);

/*开始接收数据*/
DHT11_Data->humi_int= Read_Byte();

DHT11_Data->humi_deci= Read_Byte();

DHT11_Data->temp_int= Read_Byte();

DHT11_Data->temp_deci= Read_Byte();

DHT11_Data->check_sum= Read_Byte();


/*读取结束，引脚改为输出模式*/
dht11_gpio_direction(DHT11_GPIO_D_OUT);
/*主机拉高*/
DHT11_DATA_OUT(HIGH);

/*检查读取的数据是否正确*/
```

```
if(DHT11_Data->check_sum == DHT11_Data->humi_int + DHT11_Data->humi_deci + DHT11_Data->temp_int+ DHT11_Data->temp_deci)
return SUCCESS;
else
return ERROR;
}
else
{
return ERROR;
}
}
```

/** 定时器实现精准延时代码 **/

```
/**
******************************************************************************
**
* File Name : TIM.c
* Description : This file provides code for the configuration
* of the TIM instances.
******************************************************************************
**
* @attention
*
* <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
* All rights reserved.</center></h2>
*
* This software component is licensed by ST under BSD 3-Clause license,
* the "License"; You may not use this file except in compliance with the
* License. You may obtain a copy of the License at:
* opensource.org/licenses/BSD-3-Clause
*
******************************************************************************
**
*/

/* Includes ------------------------------------------------------------------*/
#include "tim.h"

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

TIM_HandleTypeDef htim6;
```

```
/* TIM6 init function */
void MX_TIM6_Init(void)
{
TIM_MasterConfigTypeDef sMasterConfig = {0};

htim6.Instance = TIM6;
htim6.Init.Prescaler = 90-1;
htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
htim6.Init.Period = 1;
htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
{
Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if   (HAL_TIMEx_MasterConfigSynchronization(&htim6,   &sMasterConfig)   !=
HAL_OK)
{
Error_Handler();
}

}

void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
{

if(tim_baseHandle->Instance==TIM6)
{
/* USER CODE BEGIN TIM6_MspInit 0 */

/* USER CODE END TIM6_MspInit 0 */
/* TIM6 clock enable */
__HAL_RCC_TIM6_CLK_ENABLE();
/* USER CODE BEGIN TIM6_MspInit 1 */

/* USER CODE END TIM6_MspInit 1 */
}
}

void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* tim_baseHandle)
{

if(tim_baseHandle->Instance==TIM6)
```

```
{
/* USER CODE BEGIN TIM6_MspDeInit 0 */

/* USER CODE END TIM6_MspDeInit 0 */
/* Peripheral clock disable */
__HAL_RCC_TIM6_CLK_DISABLE();
/* USER CODE BEGIN TIM6_MspDeInit 1 */

/* USER CODE END TIM6_MspDeInit 1 */
}
}

/* USER CODE BEGIN 1 */
void Delay_us(uint16_t us)
{
uint16_t differ=0xffff-us-5;

HAL_TIM_Base_Start(&htim6);
__HAL_TIM_SetCounter(&htim6,differ);
while(differ < 0xffff-5)
{
differ = __HAL_TIM_GetCounter(&htim6);
}
HAL_TIM_Base_Stop(&htim6);
}

/* USER CODE END 1 */


/** uart 驱动以及中断转发代码  **/

/**
******************************************************************************
**
* File Name : USART.c
* Description : This file provides code for the configuration
* of the USART instances.
******************************************************************************
**
* @attention
*
* <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
* All rights reserved.</center></h2>
*
* This software component is licensed by ST under BSD 3-Clause license,
* the "License"; You may not use this file except in compliance with the
```

```
* License. You may obtain a copy of the License at:
* opensource.org/licenses/BSD-3-Clause
*
******************************************************************************
**
*/

/* Includes ------------------------------------------------------------------*/
#include "usart.h"

/* USER CODE BEGIN 0 */
unsigned char UBuffer[1] = {0};
// esp01s baud 74880
/* USER CODE END 0 */

UART_HandleTypeDef huart5;
UART_HandleTypeDef huart1;

/* UART5 init function */
void MX_UART5_Init(void)
{

huart5.Instance = UART5;
huart5.Init.BaudRate = 74880;
huart5.Init.WordLength = UART_WORDLENGTH_8B;
huart5.Init.StopBits = UART_STOPBITS_1;
huart5.Init.Parity = UART_PARITY_NONE;
huart5.Init.Mode = UART_MODE_TX_RX;
huart5.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart5.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart5) != HAL_OK)
{
Error_Handler();
}
// HAL_UART_Receive_IT(&huart5, UBuffer, sizeof(UBuffer));
}
/* USART1 init function */

void MX_USART1_UART_Init(void)
{

huart1.Instance = USART1;
huart1.Init.BaudRate = 115200;
huart1.Init.WordLength = UART_WORDLENGTH_8B;
huart1.Init.StopBits = UART_STOPBITS_1;
```

```
huart1.Init.Parity = UART_PARITY_NONE;
huart1.Init.Mode = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart1) != HAL_OK)
{
Error_Handler();
}
// HAL_UART_Receive_IT(&huart1, UBuffer, sizeof(UBuffer));
}

void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{

GPIO_InitTypeDef GPIO_InitStruct = {0};
if(uartHandle->Instance==UART5)
{
/* USER CODE BEGIN UART5_MspInit 0 */

/* USER CODE END UART5_MspInit 0 */
/* UART5 clock enable */
__HAL_RCC_UART5_CLK_ENABLE();
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOD_CLK_ENABLE();
/**UART5 GPIO Configuration
PC12 ------> UART5_TX
PD2 ------> UART5_RX
*/
GPIO_InitStruct.Pin = GPIO_PIN_12;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF8_UART5;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

GPIO_InitStruct.Pin = GPIO_PIN_2;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF8_UART5;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/* UART5 interrupt Init */
HAL_NVIC_SetPriority(UART5_IRQn, 0, 14);
HAL_NVIC_EnableIRQ(UART5_IRQn);
```

```
/* USER CODE BEGIN UART5_MspInit 1 */
HAL_UART_Receive_IT(&huart5, UBuffer, sizeof(UBuffer));
/* USER CODE END UART5_MspInit 1 */
}
else if(uartHandle->Instance==USART1)
{
/* USER CODE BEGIN USART1_MspInit 0 */

/* USER CODE END USART1_MspInit 0 */
/* USART1 clock enable */
__HAL_RCC_USART1_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
/**USART1 GPIO Configuration
PA9 ------> USART1_TX
PA10 ------> USART1_RX
*/
GPIO_InitStruct.Pin = STLINK_RX_Pin|STLINK_TX_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USART1 interrupt Init */
HAL_NVIC_SetPriority(USART1_IRQn, 0, 13);
HAL_NVIC_EnableIRQ(USART1_IRQn);
/* USER CODE BEGIN USART1_MspInit 1 */
HAL_UART_Receive_IT(&huart1, UBuffer, sizeof(UBuffer));
/* USER CODE END USART1_MspInit 1 */
}
}

void HAL_UART_MspDeInit(UART_HandleTypeDef* uartHandle)
{

if(uartHandle->Instance==UART5)
{
/* USER CODE BEGIN UART5_MspDeInit 0 */

/* USER CODE END UART5_MspDeInit 0 */
/* Peripheral clock disable */
__HAL_RCC_UART5_CLK_DISABLE();
/**UART5 GPIO Configuration
PC12 ------> UART5_TX
PD2 ------> UART5_RX
```

```
*/
HAL_GPIO_DeInit(GPIOC, GPIO_PIN_12);

HAL_GPIO_DeInit(GPIOD, GPIO_PIN_2);

/* UART5 interrupt Deinit */
HAL_NVIC_DisableIRQ(UART5_IRQn);
/* USER CODE BEGIN UART5_MspDeInit 1 */

/* USER CODE END UART5_MspDeInit 1 */
}
else if(uartHandle->Instance==USART1)
{
/* USER CODE BEGIN USART1_MspDeInit 0 */

/* USER CODE END USART1_MspDeInit 0 */
/* Peripheral clock disable */
__HAL_RCC_USART1_CLK_DISABLE();
/**USART1 GPIO Configuration
PA9 ------> USART1_TX
PA10 ------> USART1_RX
*/
HAL_GPIO_DeInit(GPIOA, STLINK_RX_Pin|STLINK_TX_Pin);

/* USART1 interrupt Deinit */
HAL_NVIC_DisableIRQ(USART1_IRQn);
/* USER CODE BEGIN USART1_MspDeInit 1 */

/* USER CODE END USART1_MspDeInit 1 */
}
}

/* USER CODE BEGIN 1 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
if(huart->Instance == USART1) {
HAL_UART_Transmit(&huart5, UBuffer, sizeof(UBuffer), 100);
HAL_UART_Receive_IT(&huart1, UBuffer, sizeof(UBuffer));
}
if(huart->Instance == UART5) {
HAL_UART_Transmit(&huart1, UBuffer, sizeof(UBuffer), 100);
HAL_UART_Receive_IT(&huart5, UBuffer, sizeof(UBuffer));
}
}
/* USER CODE END 1 */
```

/********************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/

/** 安卓客户端主要代码 **/

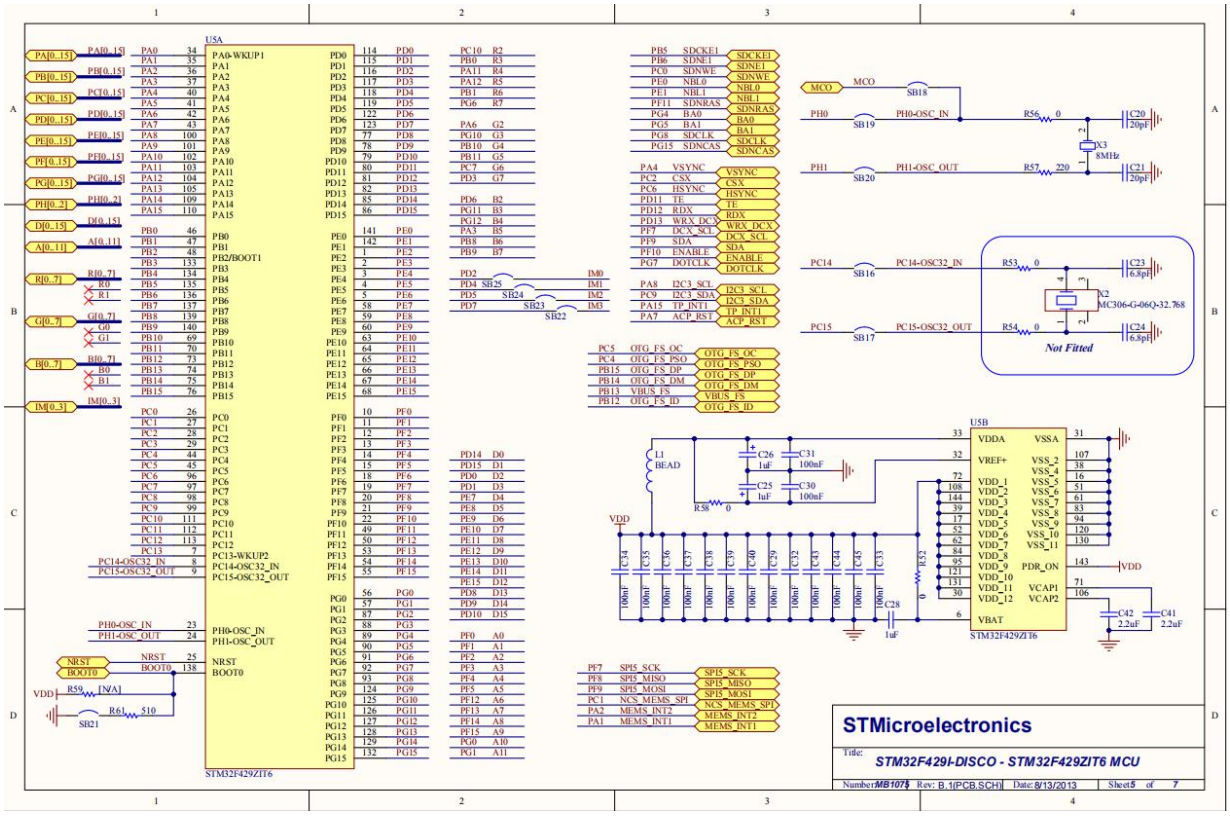
```
// index.js
var app = {
// Application Constructor
initialize: function() {
document.addEventListener('deviceready',    this.onDeviceReady.bind(this),
false);
},

// deviceready Event Handler
//
// Bind any cordova events here. Common events are:
// 'pause', 'resume', etc.
onDeviceReady: function() {
this.receivedEvent('deviceready');
// Here, we redirect to the web site.
var targetUrl = "http://iot.whqee.cn";
var bkpLink = document.getElementById("bkpLink");
bkpLink.setAttribute("href", targetUrl);
bkpLink.text = targetUrl;
window.location.replace(targetUrl);
},

// Update DOM on a Received Event
receivedEvent: function(id) {
var parentElement = document.getElementById(id);
var listeningElement = parentElement.querySelector('.listening');
var receivedElement = parentElement.querySelector('.received');

listeningElement.setAttribute('style', 'display:none;');
receivedElement.setAttribute('style', 'display:block;');

console.log('Received Event: ' + id);
}
};

app.initialize();

// index.css
* {
```

```
-webkit-tap-highlight-color: rgba(0,0,0,0); /* make transparent link selection, adjust last value opacity 0 to 1.0 */
}

body {
-webkit-touch-callout: none; /* prevent callout to copy image, etc when tap to hold */
-webkit-text-size-adjust: none; /* prevent webkit from resizing text to fit */
-webkit-user-select: none; /* prevent copy paste, to allow, change 'none' to 'text' */
background-color:#E4E4E4;
background-image:linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
font-family: system-ui, -apple-system, -apple-system-font, 'Segoe UI', 'Roboto', sans-serif;
font-size:12px;
height:100vh;
margin:0px;
padding:0px;
/* Padding to avoid the "unsafe" areas behind notches in the screen */
padding: env(safe-area-inset-top, 0px) env(safe-area-inset-right, 0px) env(safe-area-inset-bottom, 0px) env(safe-area-inset-right, 0px);
text-transform:uppercase;
width:100%;
}

/* Portrait layout (default) */
.app {
background:url(../img/logo.png) no-repeat center top; /* 170px x 200px */
position:absolute; /* position in the center of the screen */
left:50%;
top:50%;
height:50px; /* text area height */
width:225px; /* text area width */
text-align:center;
padding:180px 0px 0px 0px; /* image height is 200px (bottom 20px are overlapped with text) */
margin:-115px 0px 0px -112px; /* offset vertical: half of image height and text area height */
/* offset horizontal: half of text area width */
}

/* Landscape layout (with min-width) */
@media screen and (min-aspect-ratio: 1/1) and (min-width:400px) {
.app {
background-position:left center;
```

```
padding:75px 0px 75px 170px; /* padding-top + padding-bottom + text area
= image height */
margin:-90px 0px 0px -198px; /* offset vertical: half of image height */
/* offset horizontal: half of image width and text area width */
}
}

h1 {
font-size:24px;
font-weight:normal;
margin:0px;
overflow:visible;
padding:0px;
text-align:center;
}

.event {
border-radius:4px;
-webkit-border-radius:4px;
color:#FFFFFF;
font-size:12px;
margin:0px 30px;
padding:2px 0px;
}

.event.listening {
background-color:#333333;
display:block;
}

.event.received {
background-color:#4B946A;
display:none;
}

@keyframes fade {
from { opacity: 1.0; }
50% { opacity: 0.4; }
to { opacity: 1.0; }
}
@-webkit-keyframes fade {
from { opacity: 1.0; }
50% { opacity: 0.4; }
to { opacity: 1.0; }
}
```

```
.blink {
animation:fade 3000ms infinite;
-webkit-animation:fade 3000ms infinite;
}


// MainActivity.java
package io.cordova.hellocordova;

import android.os.Bundle;
import org.apache.cordova.*;

public class MainActivity extends CordovaActivity
{
@Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);

// enable Cordova apps to be started in the background
Bundle extras = getIntent().getExtras();
if (extras != null && extras.getBoolean("cdvStartInBackground", false)) {
moveTaskToBack(true);
}

// Set by <content src="index.html" /> in config.xml
loadUrl(launchUrl);
}
}

// config.xml
<?xml version='1.0' encoding='utf-8'?>
<widget id="io.cordova.hellocordova" version="1.0.0"
xmlns="http://iot.whqee.cn"
xmlns:cdv="http://cordova.apache.org/ns/1.0">
<allow-navigation href="http://iot.whqee.cn" />
<name>SmartHomeApp</name>
<description>
A sample Apache Cordova application.
</description>
<author email="whqee@qq.com" href="http://iot.whqee.cn">
Whqee
</author>
<content src="index.html" />
<plugin name="cordova-plugin-whitelist" spec="1" />
```

<access origin="*" />
<allow-intent href="http://*/*" />
<allow-intent href="https://*/*" />
<allow-intent href="tel:*" />
<allow-intent href="sms:*" />
<allow-intent href="mailto:*" />
<allow-intent href="geo:*" />
<platform name="android">
<allow-intent href="market:*" />
</platform>
<platform name="ios">
<allow-intent href="itms:*" />
<allow-intent href="itms-apps:*" />
</platform>
</widget>



附图 1 STM32 开发板主要原理图

致　　谢

首先感谢这一年来愿意陪我的同学们，万万没想到当初年少猖狂的自己会像近年这个样子，在这些自闭不出门、几乎没去过教室的日子里，隔壁宿舍、楼上的你们就是我的大街小巷，尤其感谢愿意被我拉着一起去夜跑、以及和我一起去自习的同学们，你们的陪伴给了我莫大的“能量”补给和坚持下去的动力；

感谢图书馆、感谢去图书馆找了不算少的书看和晚上睡不着拿多本 Linux 驱动英文原著当小说看的自己，起码还不算白过大学几年；

感谢红满堂工作室在我大一大二时对我的培养，同时也感谢工作室中的 Linux 驱动详解一书吸引了我的注意，感谢各位运维的师兄师姐们让我接触到了 Linux 操作系统，同时也感谢所有的 Linux 的贡献者；

感谢严炳辉老师给了我自己出题的机会，让我能按自己的想法发挥，以及耐心地指导我编写、纠正论文，同时也感谢刘勇老师在资源上对我的支持和帮助；

感谢贾维清老师布置的关于 ARM 架构的大作业，它让我费劲心思翻遍 ARM 官网、架构体系参考手册，也让我着迷，让我开始理清了底层芯片架构的知识，感觉水平骤升一个台阶；同时也感谢其他各个领域的老师的悉心帮助；

最后，感谢父母的养育之恩和身边人们的耐心相待，希望所有人包括自己以后各方面都继续成长。