

西安邮电大学

毕业设计（论文）

题目： 基于物联网的建筑物能耗监控系统
——网关采集设计

学院： 自动化学院

专业： 自动化

班级： 自动 1403 班

学生姓名： 张鹏鹏

学号： 06141103

导师姓名： 王文庆 职称： 教授

起止时间： 2017 年 12 月 5 日 至 2018 年 6 月 10 日

毕业设计（论文）声明书

本人所提交的毕业论文《基于物联网的建筑物能耗监控系统—网关采集设计》是本人在指导教师指导下独立研究、写作的成果，论文中所引用他人的文献、数据、图件、资料均已明确标注；对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式注明并表示感谢。

本人完全理解《西安邮电大学本科毕业设计（论文）管理办法》的各项规定并自愿遵守。

本人深知本声明书的法律责任，违规后果由本人承担。

论文作者签名：

日期： 年 月 日

西安邮电大学本科毕业设计(论文)选题审批表

申报人	王文庆	职 称	教授	学 院	自动化学院			
题目名称	基于物联网的建筑物能耗监控系统—网关采集设计							
题目来源	科研				教学		其它	是
题目类型	硬件设计	是	软件设计		论文		艺术作品	
题目性质	应用研究		是		理论研究			
题目简述	<p>(为什么申报该课题)</p> <p>设计基于物联网的建筑物能耗监控系统，系统包括服务器、网站、网关采集节点，完成建筑物能耗的采集。网关采集节点完成以下功能： 1、本地能耗数据采集； 2、本地数据存储； 3、本地数据 TCP/IP 上传。</p>							
对学生知识与能力要求	<p>1. 熟练掌握终端 APP 开发；</p> <p>2. 熟练掌握 c#开发。</p>							
具体任务以及预期目标	<p>(应完成的具体工作，预期目标和成果形式)</p> <p>1. 完成相应的软硬件设计；</p> <p>2. 完成所设计的功能；</p> <p>3. 完成论文撰写。</p>							

<p>时间 进度</p>	<p>2017.12.5 至 2018.1.15 了解课题的背景知识，具体细节，明确所涉及的内容，确立毕业设计的题目，撰写开题报告；</p> <p>2018.1.16 至 2018.3.12 学习有关本课题设计的相关知识，包括 EDA 相关知识的学习；</p> <p>2018.3.13 至 2018.4.1 规划设计方案，硬件选型及购买，搭建硬件电路；；</p> <p>2018.4.1 至 2018.5.10 编写步进电机控制系统硬件代码、仿真，硬件调试；</p> <p>2018.5.10 至 2018.5.30 系统联调，完成实物设计，同时撰写写毕业设计论文，请指导老师评改；</p> <p>2018.6.1 至 2018.6.10 修改、打印毕业设计论文，准备 PPT 完成毕设答辩。</p>		
<p>系（教研室）主任 签字</p>	<p>年月日</p>	<p>主管院长 签字</p>	<p>年月日</p>

西安邮电大学本科毕业设计（论文）开题报告

学生姓名	张鹏鹏	学号	06141103	专业班级	自动 1403
指导教师	王文庆	题目	基于物联网的建筑物能耗监控系统——网关采集设计		
<p>选题目的（为什么选该课题）</p> <p>节能降耗，计量先行。在我国既有的约 430 亿平方米的建筑中，仅有 4% 的建筑采用了先进的能源效率改进措施。虽然当前市面上已经出现了用于建筑能耗监控系统，但这样的系统还不成熟，很多采用的设计方案较为陈旧，存在通用性、实用性不强的问题，不能满足当前实际的需要。面对建筑能耗监测方面巨大的缺口，同时考虑到政策支持等因素，建筑能耗监测有着潜在的需求，而作为其载体的智能仪表则存在着很大的市场空间。</p> <p>本课题设计的主旨在于基于物联网和智能仪表架构一个综合性的建筑物能耗采集平台，可满足国家相关技术规定和导则规定的技术条件。本设计涉及基于物联网的建筑物能耗监控系统——网关采集设计，通过能耗数据采集系统，采集解析多种不同通讯协议智能仪表（电表、水表、燃气表等）的数据，实现智能仪表的现场数据采集、本地存储、显示、TCP/IP 远程传输等功能。</p>					
<p>前期基础（已学课程、掌握的工具，资料积累、软硬件条件等）</p> <p>已学课程：《单片机原理及应用 A》、《高级语言程序设计(C 语言)》、《模拟电子技术基础 B》、《数字电路与逻辑设计 B》、《计算机控制技术》、《微型计算机原理》</p> <p>掌握工具：示波器、万用表、电烙铁、电脑；</p> <p>资料积累：《STM32F4 开发手册》、《SD 卡读写规范》、《LwIP 协议栈的设计与实现》、《FatFs 文件系统详解》、《FreeRTOS 开发手册》；</p> <p>软硬件条件：软件：Keil uVision5、protues、Altium Designer、Hyper Terminal； 硬件：STM32F429、RS485 总线电路、MBUS 通信接口。</p>					

要研究和解决的问题（做什么）

操作系统移植：采用软硬件协同设计方法，将 FreeRTOS 嵌入式操作系统移植到 STM32 的处理器上，操作系统全权调度各硬件，以 manage 任务为核心执行调度其余任务。系统开始运行后先行初始化，继而采集任务将采集到的数据发送到 manage 队列，由 manage 任务将采集数据再发送到存储、显示、网络任务完成一次循环，任务间通过消息队列进行通信使用信号量进行同步。

网络模块软件设计：从消息队列获取信息以后，需要对数据进行 JSON 格式打包，并且通过 HTTP 协议，上传至服务器。本系统采用轻量型的 TCP/IP 协议栈 LWIP，LwIP 在保持 TCP 协议主要功能的基础上减少对 RAM 的占用，它只需十几 KB 的 RAM 就可以运行，并且 LwIP 尽量避免内存复制，避免了此操作产生的性能损失。

存储模块软件设计：对于能耗数据采集系统，在实时的将采集到的能耗数据上传到服务器上的同时，还需要对采集到的数据进行备份，即在本地存储空间保存能耗数据，这样有利于本地对能耗数据进行分析。考虑到备份数据能够轻松取出，采用外部 SD 卡和内部 NAND Flash 作为存储介质。本系统通过采用 FatFs 文件系统，实现嵌入式系统中的 FAT 文件系统，FatFs 的编写遵循 ANSI C，因此不依赖于硬件平台。它可以嵌入到便宜的微控制器中

GUI 设计：能耗数据采集系统也需要预留配置端口，以方便维护，使用屏幕可以辅助设备的现场维护。在本设计中，采用 4.3 寸电阻式触摸屏来做界面显示。可在用户界面上完成服务器地址，仪表端口等能耗数据采集系统运行参数的设置，并可以通过 GUI 界面查看各仪表的参数与实时数据。

采集模块设计：在本系统所需仪表（电表、水表、气表）采用的通讯协议、接口具有一定差异。电表采用 MODBUS-RTU 通讯协议，支持 RS485 总线通信；水表采用 188 协议，具有 MBUS 通信接口；气表可支持 MODBUS-RTU、DTL645、188 协议，直接挂到 RS485 或 MBUS 总线上即可输出数据信号；通过 MBUS 转 RS485, 将水电气表都挂载到 RS485 总线上；从管理任务获取待采集仪表的端口、协议、所需要采集的数据地址等，配置板上串行接口资源，根据不同仪表通讯协议完成仪表数据的采集。并将采集完成消息通过消息队列发给管理任务。并且考虑到数据采集需要有多路串口同时进行采集，串口采集底层采用 DMA 发送与接收，以减少控制器的负担。

工作思路和方案（怎么做）

建筑能耗监控系统——网关采集设计系统结构如图 1 所示。

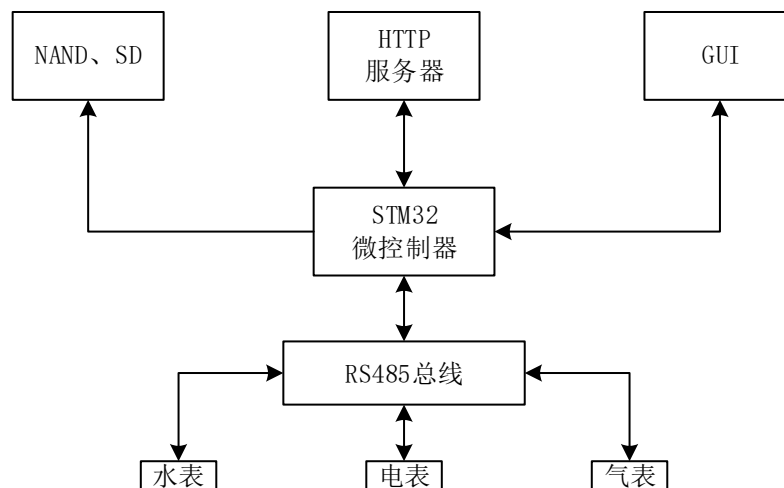


图 1 建筑能耗监控系统——网关采集设计系统结构图

基于物联网的建筑物能耗监控系统——网关采集设计以 STM32F407 微控制器作为核心控制单元，整个系统采用 FreeRTOS 实时系统，循环调度各个任务模块，存储模块采用 FatFs，对实时采集上来的数据进行本地 NAND 和 SD 存储，服务器上传采用 LWIP 协议栈，将用 JSON 格式打包好的数据进行 HTTP 上传。为了实现三表合一，将不同通讯接口的仪表，都挂载到 RS485 总线上，并且通过协议解析以达到控制器对仪表数据的精确采集。采用 Keil 5 集成编译环境编写代码，并使用串口调试工具作为辅助调试手段。

第 1 周一第 2 周：查找相关资料，选择各大模块所需芯片，进行电路基础方面以及各个仪表协议接口的学习；

第 3 周一第 4 周：初步搭建各模块的电路，实现其功能；

第 5 周一第 6 周：学习微控制器 STM32F429 的相关资料，使用 Keil 5 编写各模块的程序，并进行调试；

第 7 周一第 8 周：使用 Altium Designer 进行电路原理图和 PCB 的设计，并且将印刷好的 PCB 进行焊接调试；

第 9 周一第 10 周：搭建整个能耗监控系统，进行软件烧写和各功能模块实现；

第 11 周一第 12 周：进行能耗监控系统的调试；

第 13 周一第 14 周：撰写毕业论文。

指导教师意见

签字

2018 年 3 月 25 日

西安邮电大学毕业设计（论文）成绩评定表

学生姓名	张鹏鹏	性别	男	学号	06141103	专业 班级	自动 1403
课题名称	基于物联网的建筑物能耗监控系统——网关采集设计						
指导教师 意见	<div>(从开题论证、论文内容、撰写规范性、学习态度、创新等方面进行考核)</div> <div>评分（百分制）：<div>指导教师(签字)：</div><div>年 月 日</div></div>						
评阅 教师 意见	<div>(从选题、开题论证、论文内容、撰写规范性、创新和预期成果等方面进行考核)</div> <div>评分（百分制）：<div>评阅教师(签字)：</div><div>年 月 日</div></div>						
验收 小组 意见	<div>(从毕业设计质量、准备、操作情况等方面进行考核)</div> <div>评分（百分制）：<div>评阅教师(签字)：</div><div>年 月 日</div></div>						
答辩 小组 意见	<div>(从准备、陈述、回答、仪表等方面进行考核)</div> <div>评分（百分制）：<div>答辩小组组长(签字)：</div><div>年 月 日</div></div>						
评分比例	指导教师评分(20%) 评阅教师评分(30%) 验收小组评分(30%) 答辩小组评分(20%)						
学生总评 成绩	百分制成绩				等级制成绩		
答辩委员 会意见	<div>毕业论文(设计)最终成绩(等级)：</div> <div>学院答辩委员会主任(签字、学院盖章)：</div> <div>年 月 日</div>						

目 录

第一章 引言	1
1.1 课题背景	1
1.2 课题任务	1
1.3 论文结构	1
第二章 背景知识	3
2.1 STM32F429 微控制器	3
2.1.1 控制器片内外设丰富	3
2.1.2 控制器支持的嵌入式系统	3
2.1.3 控制器资料丰富程度	3
2.1.4 成本与功耗	3
2.2 STM32F429 主要特性	4
第三章 设计方案与功能介绍	5
3.1 系统方案	5
3.2 系统功能	5
3.3 通信方式设计	6
3.3.1 能耗数据监控系统上层数据链路设计	6
3.3.2 能耗数监控系统下层数据链路设计	6
3.4 主要模块设计方案	7
3.4.1 操作系统模块设计	7
3.4.2 采集模块设计	8
3.4.3 显示模块设计	8
3.4.4 存储模块设计	9
3.4.5 网络模块设计	9
第四章 系统硬件设计	10
4.1 整体硬件方案	10

4.2 控制器最小系统硬件设计	10
4.2.1 电源电路设计	10
4.2.2 复位电路设计	11
4.2.3 时钟电路设计	11
4.2.4 下载电路设计	12
4.3 采集模块电路设计	13
4.3.1 三表合一	13
4.3.2 电表采集电路设计	13
4.3.3 水表采集电路设计	14
4.3.4 气表数据采集电路设计	16
4.4 存储模块电路设计	16
4.4.1 存储模块介绍	16
4.4.2 NAND FLASH 设计	17
4.4.3 SD 卡电路设计	18
4.4.4 SDRAM 电路设计	19
4.5 网络模块电路设计	21
4.6 显示模块电路设计	24
4.6.1 LCD 介绍	24
4.6.2 TFT-LCD 原理图设计	24
第五章 系统软件设计	26
5.1 调试环境	26
5.2 实时操作系统	26
5.2.1 FreeRTOS 系统介绍	26
5.2.2 FreeRTOS 系统移植	26
5.3 采集模块设计	28
5.4 存储模块设计	30
5.4.1 配置文件读取	30
5.4.2 历史数据存储	31
5.5 显示模块软件设计	35
5.6 网络模块设计	37
第六章 调试	40

6.1 GUI 显示模块调试	40
6.2 网络上传模块调试.....	41
6.3 存储模块调试.....	42
结束语	43
致 谢	44
参考文献	45
附录	46
附录 1：电路原理图	46
附录 2：实物图	49
附录 3 数据传输的 JSON 数据格式	50

摘 要

节能降耗，计量先行。我国是世界上最大的能源生产国和消费国，统计显示，我国建筑能耗约占全国总能耗的 31%，每年新建的 20 亿平方米建筑中大多数是高能耗建筑，在如此严峻的形势下，我国需对建筑耗能进行全面的监控，建设部、财政部颁布的《关于加强国家机关办公建筑和大型公共建筑节能管理工作的实施意见》，其中提到建筑能耗监控系统是建筑节能监测的一个重要措施，但当前能耗监控系统的软硬件水平较低，对能耗监控系统的研究开发亟待加强。

项目组设计建筑物能耗采集系统，该系统基于物联网平台，利用嵌入式、现场总线、智能控制等技术，包括智能仪表层、现场数据采集层、网络传输层和上位机应用层。智能仪表层包括支持不同协议的水、电、气智能仪表；现场数据采集层的核心设备是数据采集器，其基于各类总线实现底层三表合一（三类智能仪表接口兼容、协议解析兼容）；能耗信息通过网络传输层进行远程传输；上位机对各类数据进行融合、分析、发布命令，系统最终实现了建筑物能耗的远程双向监测。

数据采集器在系统中起到关键作用，它内嵌 TCP/IP 网络协议，支持实现智能仪表数据采集、数据处理、协议转换、TCP/IP 网络协议的封装打包及数据上传、远程命令下发、本地存储调试等功能。数据采集器采用模块化设计方法，硬件主要包含：微控制器最小系统板、电源管理模块、时钟电路模块、数据存储模块、以太网网络模块、RS485 总线传输模块、接口转换模块、系统调试模块等。并利用独特的硬件设计，可实现 RS485 传输的自动收发控制。

采集器内嵌 FreeRTOS 操作系统，包括管理、采集、存储、网络、显示任务。为了方便调试，创新性的提供了指令式调试接口，可通过 UART、TCP、UDP 等多种通信方式调试。

本系统实现了三表合一的能耗数据采集，是一种全新的思路，同时采用工业化设计标准，运行良好稳定，为建筑能耗监控提供了稳定可靠的能耗数据来源。

关键词：物联网；能耗监控；三表合一；STM32；采集；

ABSTRACT

Saving energy and reducing consumption, and measurement. China is the world's biggest energy producer and consumer, statistics show that China's building energy consumption accounts for about 31% of the total energy consumption, a year to build 2 billion square meters of buildings is most energy-intensive, under such severe situation, need for building energy consumption in our country to conduct a comprehensive monitoring, the ministry of construction, ministry of finance issued "about strengthening the state organ office buildings and large public building energy efficiency management of the implementation opinions", mentions a monitoring system of building energy consumption is an important measure of building energy conservation monitoring, but the low level of the current energy consumption monitoring system hardware and software, research and development of energy consumption monitoring system needs to be strengthened.

Building energy consumption acquisition system project design, the system based on Internet of things platform, using the embedded, fieldbus and intelligent control technology, including smart meters, on-site data acquisition layer, application layer network transport layer and the upper machine.

Smart meters layer includes support for different protocols of water, electricity and gas intelligent instruments;

Is the core equipment of field data acquisition layer data collector, table and its implementation based on all kinds of bus bottom layer (three kinds of intelligent instrument interface compatibility, protocol parsing compatible);

Energy consumption information for remote transmission through the network transmission layer;

PC for all kinds of data integration, analysis, command, the system finally realize remote two-way monitoring of the building energy consumption.

Data collector played a key role in the system, the embedded TCP/IP network protocol, support intelligent meter data acquisition, data processing, protocol conversion, packaging of TCP/IP network protocol and the data to upload, a remote command issued, local storage and debugging, etc.

Data collector using modular design method, hardware mainly includes: the micro-controller minimum system board, power management module, the clock

circuit module, data storage module, module, RS485 bus transmission module, Ethernet network interface conversion module, system debugging module, etc.

And use the unique design of the hardware can send and receive RS485 transmission of automatic control.

Collector FreeRTOS embedded operating system, including management, collection, storage, network and display tasks.

In order to facilitate debugging, innovative provides instruction type debugging interface, through UART, TCP, UDP and other communication debugging.

This system has realized the energy consumption of the unity of three tables and data collection, is a kind of new train of thought, at the same time adopt industrial design standard, stable running, for building energy consumption monitoring provides a stable and reliable source of energy consumption data.

Key words: IOT(Internet of things); energy monitoring; three table one; STM32 and gathering;

第一章 引言

1.1 课题背景

随着我国经济的发展，建筑物高耗能的问题日渐突出。在我国现有的约四百多亿平方米的建筑中，仅有百分之四的建筑采用了先进的能源效率改良措施。虽然市面上已经出现了用于建筑物能耗监控的系统，但是这样的系统还不成熟，很多采用的设计方案比较陈旧，存在通用性、实用性不强的问题，这些远远不能满足当前实际的需要。那么如何实现当前能耗的远程采集就成了主要问题。

物联网技术的发展，为构建一个可靠、安全、方便的远程能耗数据监控系统，提供了良好的保障。能耗数据监控系统与传统的远程采集不同，传统的远程采集只是单纯的将数据采集，通过 GPRS 上传，而本能耗数据监控系统则是通过以太网将数据远传，并可以进行闭环控制。能耗数据监控系统可以实现对各种应用场合下的能耗计量装置进行更方便快捷且安全的远程采集，并将数据进行融合、本地存储、显示，同时可以远程通过网络进行浏览与分析，进而实施控制。

本项目为能耗数据监控系统的构建，实现对底层各类智能仪表协议兼容并进行数据采集、本地存储和显示，并且通过网络技术进行远程传输，服务器能对各类数据进行融合分析，最终实现远程监测。项目目标在于开发成本较低、通用性好、操作简便、性能稳定的能耗数据采集产品。

1.2 课题任务

本课题主要研究基于物联网的建筑物能耗监控系统一网关采集设计，主要实现了智能仪表数据的采集、本地显示、本地存储和以太网传输至服务器。该设计采用基于 ARM Cortex-M4 内核的 STM32F429 微控制器，其丰富的外围资源可以满足本课题的要求，并且其具有运算速度快，功耗低等优点。设计采用物联网技术以实现智能仪表数据的采集，并且将当前数据实时进行 GUI 液晶屏显示，以 Excel 表格形式存储至本地存储，同时将数据进行 JSON 格式打包，通过以太网上传至 HTTP 服务器。

1.3 论文结构

本文各章节安排如下：

第一章：引言：主要介绍课题背景和任务；

第二章：主要介绍主控单元与各模块的元器件；

第三章：设计方案与功能介绍：主要讲解系统功能，及其各模块设计方案；

第四章：硬件设计：主要讲解硬件整体方案，各个模块的硬件，及其设计原理与硬件描述；

第五章：软件设计：开发调试环境介绍，讲解主要程序，各功能模块的程序设计；

第六章：调试结果。

第二章 背景知识

2.1 STM32F429 微控制器

控制器是能耗监控系统的核心，选择适合的控制器，可以加快开发周期和降低成本，以及有助于避免开发过程中出现的许多问题。选择 STM32F429 微控制器^[6]，主要是从如下方面考虑。

2.1.1 控制器片内外设丰富

控制器的片内外设提供了可靠的、低成本的控制芯片上资源。在本能耗数据监控系统应用中，由于需要外接多个仪表 485 端口，同时可能采用基于 AT 指令的移动通信模块、电力线载波通信模块，所以对于串口资源要求较高。同时要实现高效的网络通信，需要控制器本身具有以太网控制器 MAC，控制器内部具有 SDRAM 控制器等也影响到能耗数据监控系统运行中的存储方式。

2.1.2 控制器支持的嵌入式系统

要加快开发周期，需要采用一款在控制器上成熟应用的嵌入式系统，这个主要体现在是否选择带有 MMU 的微控制器，仅有带有 MMU 的微控制器才能够运行 Linux，其他的控制器通常有 RTOS 支持。

2.1.3 控制器资料丰富程度

控制器拥有丰富的资料，对解决开发过程中遇到的问题十分有帮助，在选型中也需要考虑。

2.1.4 成本与功耗

能耗数据采集系统是一种嵌入式设备，在建筑能耗监控系统推广后，市场需求量很大，这就势必要降低产品成本来增加一定的竞争力，同时也要求能耗数据采集系统低功耗，选择控制器的时候要选择有多种低功耗模式的控制器。

结合题目要求、资料丰富程度等因素，选择 STM32F429 微控制器作为能耗数据监控系统的控制核心，其系统架构如图 2.1 所示。

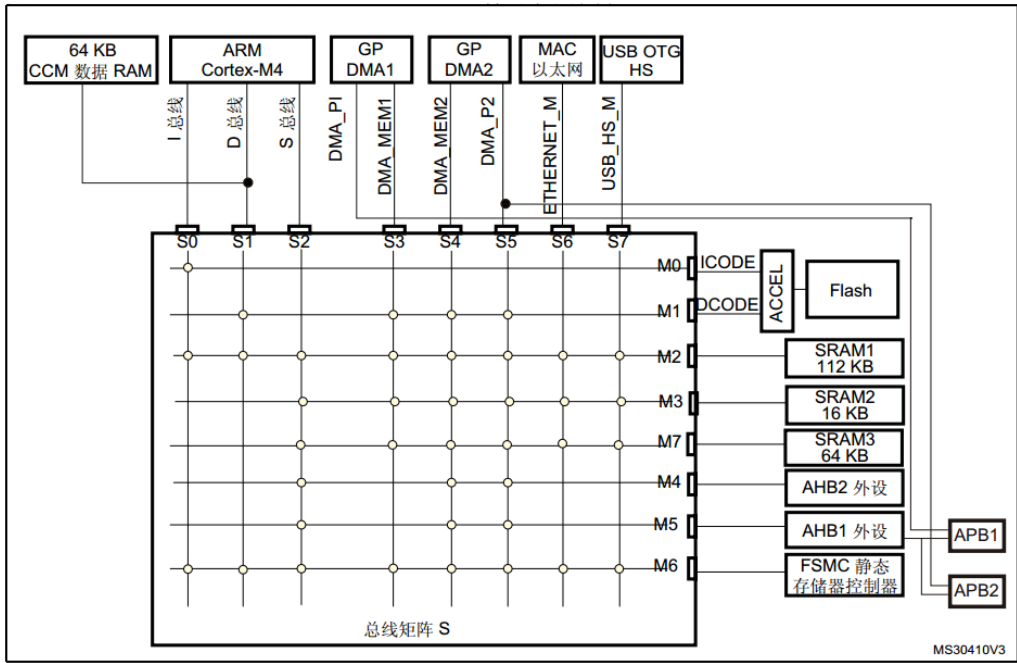


图 2.1 STM32 系统架构

2.2 STM32F429 主要特性

STM32F429 采用 ARM Cortex-M4 内核^[6]，具有 1MB 片上 FLASH 存储器，256KB 的 SRAM。该控制器时钟频率最大可达 180MHz，同时 STM32F4 系列还集成了单周期 DSP 指令和 FPU，极大提升了其浮点运算能力。除此以外，该控制器还具有许多有利于能耗数据监控系统的开发的特性，如图 2.2 所示

特性	说明
4xUSART,4xUART	提供足够多的串行接口，满足数据采集，调试，内外置 GPRS 的需要
1xEthernet MAC10/100	提供高速以太网接口
crypto/hash processor (加密/哈希硬件处理器)	支持硬件加速计算 AES-128/192/256，支持硬件 MD5 校验，为网络安全功能的实现提供了便利
1xSDIO	提供外置 SD 卡存储支持
LCD TFT Controller	提供片内 LCD 控制器（支持双图层支持、Chrom-ART 图形加速器），降低产品成本
2x12-bit DAC	为 4~20mA 模拟量输出提供支持
SDRAM Interface	提供低成本高密度的 RAM 解决方案

图 2.2 STM32F429 特性

第三章 设计方案与功能介绍

3.1 系统方案

本系统旨在设计一个安全、方便、可靠的能耗数据监控系统。该系统主要包括数据采集、网络协议智能转换、人机交互、服务器上传及本地存储这五部分。其系统结构图如图 3.1 所示：

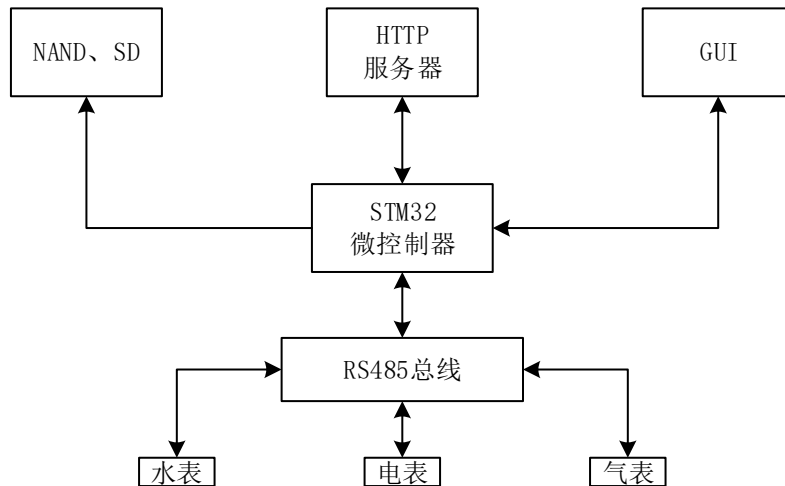


图 3.1 系统结构图

通过电表、水表及燃气表三种不同的智能仪表的信号采集，对用户的能耗使用情况进行采集，将采集到的数据通过总线传送到微处理器，微处理器经过协议解析及处理，将信息分别传送到本地存储、服务器及对应的用户界面，并且确保上位机可以实时更新数据。

3.2 系统功能

本系统的主旨在于基于物联网和智能仪表架构一个综合性的建筑物能耗监控平台，可满足国家相关技术规定^[9]和导则规定^[8]的技术条件。采集系统利用物联网技术与嵌入式技术，实现智能仪表（电表、水表、燃气表等）的现场数据采集、存储、本地 GUI 显示、以太网远程传输、上位机监测管理等功能。

3.3 通信方式设计

3.3.1 能耗数据监控系统上层数据链路设计

能耗数据监控系统向上层数据链路传输的是整理打包后的能耗数据，在住建部下发的相关能耗数据传输技术导则^[8]中，明确的规定了在能耗数据采集终端和远程数据中心之间，所要采用的协议需基于 TCP/IP 栈。本能耗监控系统在传输层使用 TCP 协议。考虑到服务器端的需求和实际应用，我们通过对 TCP 数据的封装，传输到 HTTP 服务器。本监控系统与服务器连接过程如图 3.2 所示

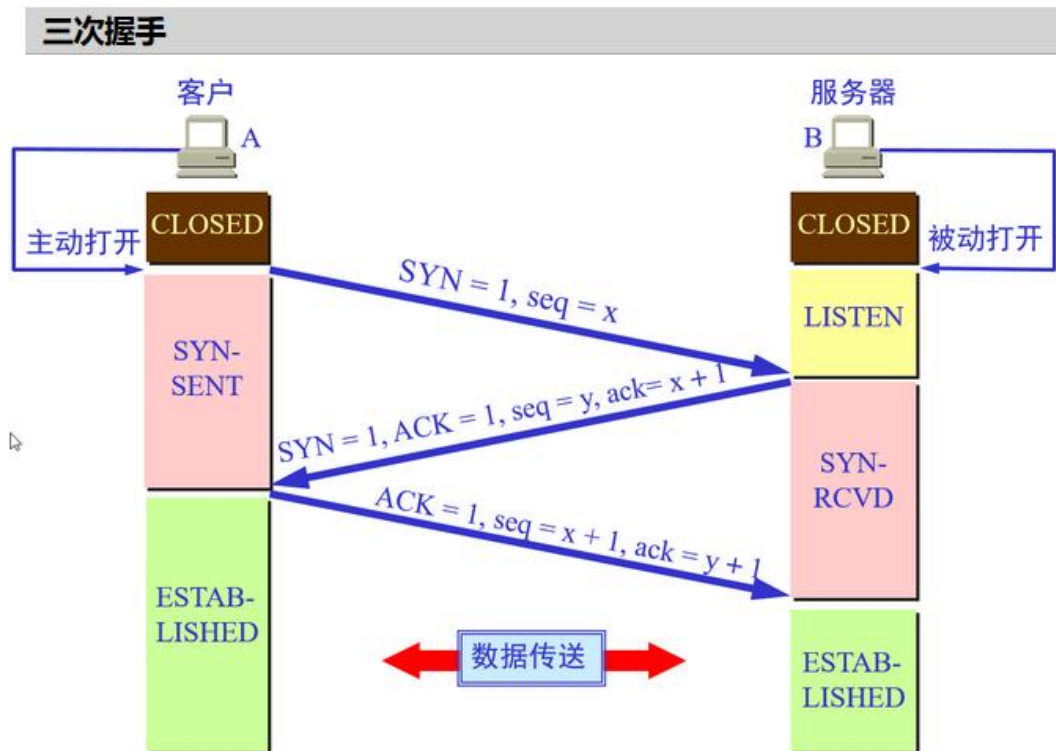


图 3.2 TCP 三次握手过程

3.3.2 能耗数监控系统下层数据链路设计

能耗数据监控系统应支持不同用能种类的智能仪表，并可以同时进行能耗数据采集，包括燃气表、水表、电能表等。智能仪表和能耗数据监控系统之间采用主-从结构的半双工通信方式，能耗数据监控系统是通信主机，智能仪表是通信从机，从机在主机的命令请求下应答。

大多数的智能仪表都可以采用 485 接口，其具有抗共模干扰，传输距离远的

特点。但是 485 总线的布线在老旧建筑中较难，可以在老旧建筑中采用其他通信方式，电力线载波通讯是一种方便较成熟的通信方式，在智能电表得到了广泛的应用，对于电能的分类分项计量，也可以采用电力线载波的方式传输。

能耗数据监控系统 and 智能仪表之间应采用符合各相关行业标准^[8]的通信协议。对各类仪表通信协议参照标准如图 3.3 所示。并且支持 Modbus 开放式协议，参照国家标准 GB/T 19582-2008，其规定了 Modbus 协议在串行链路上的实现指南。

仪表种类	行业标准
电表	DL/T 645-1997《多功能电表通信规约》
燃气表	CJ/T 188-2004《用户计量仪表数据传输技术条件》
水表	

图 3.3 智能仪表通信协议标准

3.4 主要模块设计方案

3.4.1 操作系统模块设计

本系统采用软硬件协同设计方法，将 FreeRTOS 实时操作系统移植到上主控制器上，FreeRTOS 嵌入式操作系统具备实时性，低成本，小型化，专用型和可靠性高，完全免费的特点，将它移植在处理器上，使用操作系统对数据进行操作，使任务之间的逻辑更加严谨，更加简单，方便。

操作系统以 manage 任务为核心通过消息队列将能耗数据传输到其余任务，从而达到调度其余任务。系统开始运行后先行初始化，创建各个任务和消息队列，继而采集任务开始运行，将采集到的数据发送到 manage 队列，由 manage 任务将采集数据再发送到存储、显示、网络任务完成一次循环，采集任务每两秒执行一次，任务间通过消息队列进行通信使用信号量进行同步。

在 FreeRTOS 上进行应用开发的大致框架如图 3.4 所示：

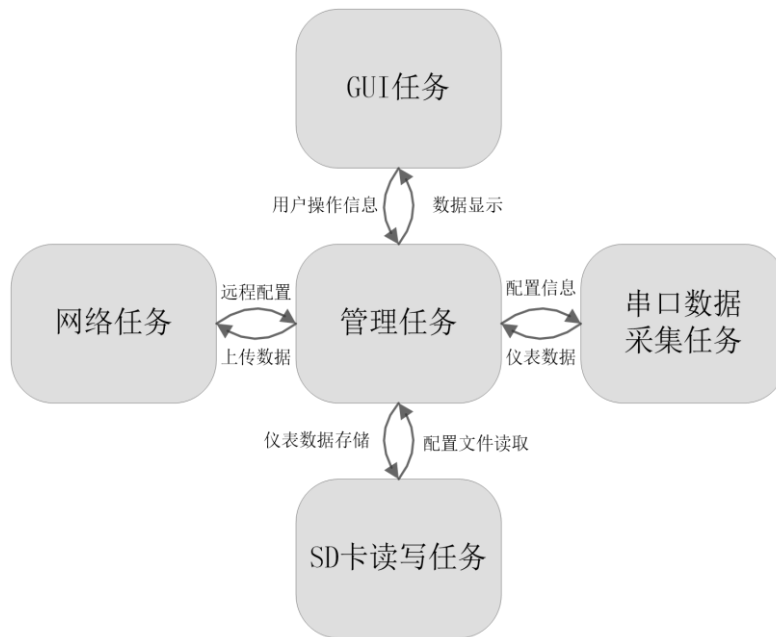


图 3.4 系统任务调度

3.4.2 采集模块设计

从管理任务获取待采集仪表的端口、协议、所需要采集的数据地址等，配置板上串口资源，将采集请求发送至各个仪表，从而仪表返回当前采集到的能耗数据，完成仪表数据的采集。然后将采集完成数据通过 manage 任务发送到其余任务。

考虑到数据采集需要有多路串口同时进行采集，串口采集底层采用 DMA 发送与接收，以减少控制器的负担。

3.4.3 显示模块设计

能耗数据监控系统所有采集的数据通过以太网上传到 HTTP 服务器，可通过服务器或上位机来查看能耗数据监控系统的工作状态。同时，能耗数据监控系统也需要预留配置端口，以方便维护，使用屏幕可以辅助设备的现场维护。在本设计中，采用 4.3 寸电阻式触摸屏来做本地 GUI 界面。可在用户界面上完成服务器地址，仪表端口等能耗数据采集系统运行参数的查看与设置，并可以通过 GUI 界面查看各仪表的参数与实时数据。

3.4.4 存储模块设计

对于能耗数据监控系统，在将实时采集到的能耗数据上传到服务器的同时，还需要对采集到的数据进行备份，即在本地保存能耗数据，这样有利于本地对能耗数据进行融合分析。考虑到备份数据能够轻松取出，主要采用外部 SD 卡存储，内部 NAND Flash 作为辅助存储介质。

系统启动后，能耗数据监控系统从 NAND 中读取与仪表相关的配置文件，若 NAND 读取失败，可以从 SD 卡中读取，并且将配置文件更新至 NAND 中。

能耗历史数据存储在 NAND 和 SD 卡中，采用制表位分隔的.xls 文件形式存储，采用.xls 格式存储方便了之后对数据的分析汇总。

3.4.5 网络模块设计

从消息队列获取信息以后，对数据进行 JSON 格式打包，然后 TCP 数据包的封装成 HTTP 数据包，通过 TCP 协议^[5]，上传至 HTTP 服务器。本系统采用轻量型的 TCP/IP 协议栈 LWIP，该协议栈保持了 TCP 协议的主要功能，占用内存小，只需要极小的 RAM 就能够运行，可适用于各嵌入式产品。并且 LwIP 尽量避免内存复制，由此也减少了此操作产生的性能损失。

第四章 系统硬件设计

4.1 整体硬件方案

数据采集器作为能耗监控系统的核心设备，本章重点介绍其硬件设计。数据采集器设计硬件框图如图 4.1 所示：

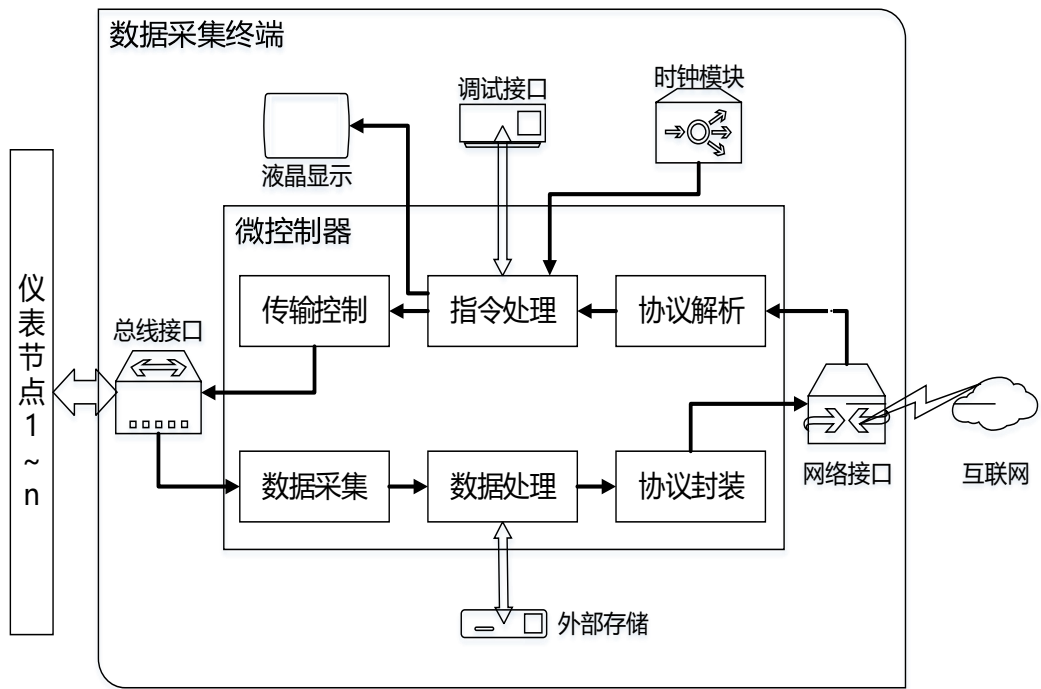


图 4.1 数据采集器设计硬件框图

设备通过 485 总线接口轮询终端节点，采集到的数据经过 STM32F429 的处理，在本地保存一份，同时在 GUI 界面上显示，并通过 JSON 打包由以太网传输到服务器上。

系统硬件设计采用局部式模块化的电路设计^[7]。以微控制器 STM32F429 为中心的局部化，以各类功能芯片为模块中心的电路设计。不仅加快了硬件设计，而且还能为后期硬件维护和升级提供良好的环境。

4.2 控制器最小系统硬件设计

4.2.1 电源电路设计

该系统用 DC 5V 来供电，除了电源接口供电外，还可以通过 Mini USB 接口进行供电。这里选用了 AS1117 芯片，用来电源的转换稳压，其电路设计如图 4.2

所示：

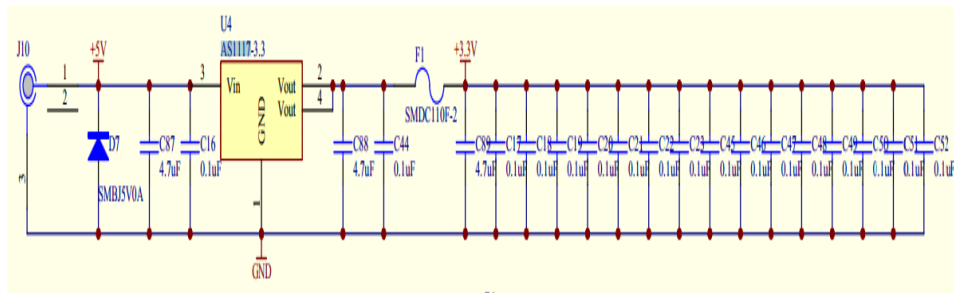


图 4.2 电源电路设计

4.2.2 复位电路设计

该系统采用按键复位，复位电路是各个系统必需的电路，当程序出现错误使系统没有响应时，可以在不断电的情况下，通过复位电路重新启动，本系统的复位电路如图 4.3 所示。

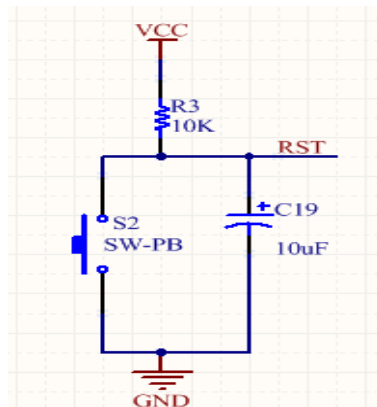


图 4.3 复位电路设计

4.2.3 时钟电路设计

该系统共有四个时钟源，其硬件设计如图 4.4—4.6 所示，分别为：

- Y1, 32.768K STM32F4x9IG 芯片的 RTC 晶振；
- Y2, 25MHZ STM32F4x9IG 芯片的主频晶振；
- Y3, 50MHZ 网络 PHY 的有源晶振，RMII 模式下使用；
- Y4, 25MHZ 网络 PHY 的无源晶振，MII 模式下使用。

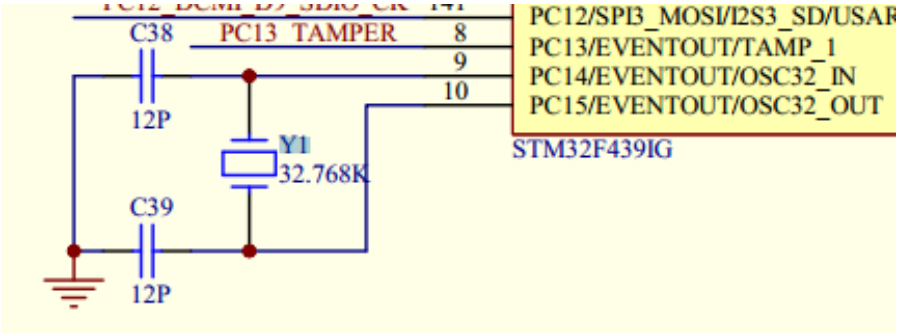


图 4.4 芯片的 RTC 晶振

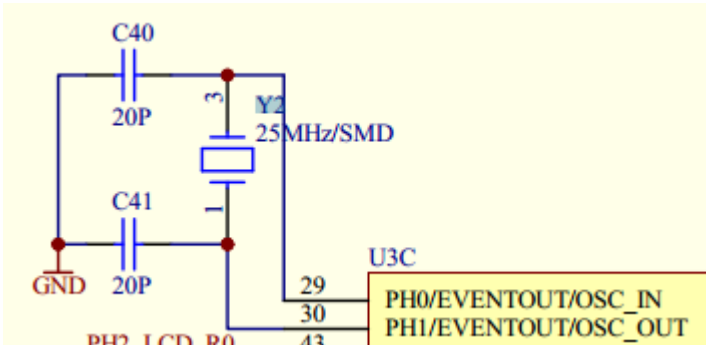


图 4.5 芯片的主频晶振

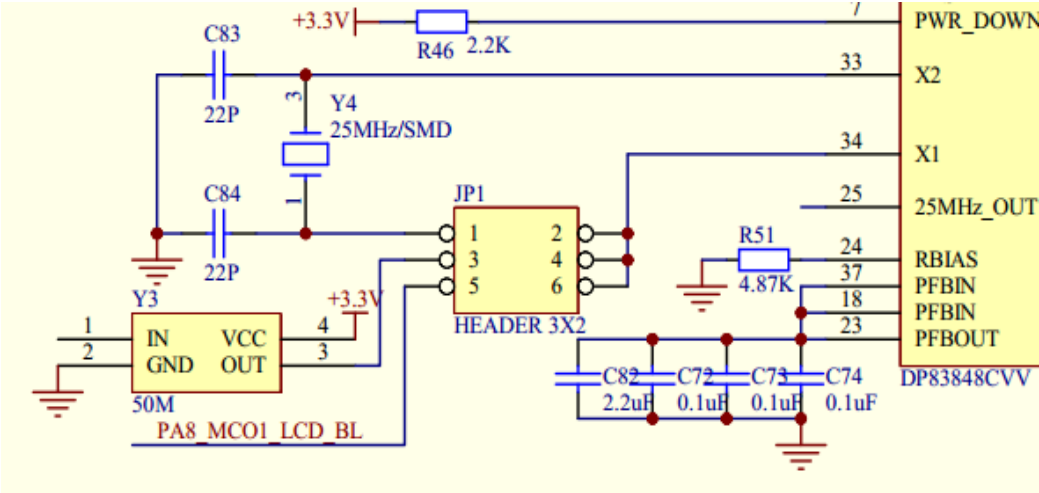


图 4.6 网络 PHY 晶振

4.2.4 下载电路设计

该系统配有 20 针 2.54mm 间距的仿真插座，可以接市场上通用的 J-link 及 U-link2 接口，通信模式方面可以选 JTAG 或 SW 两种模式，其硬件设计如图 4.7 所示。

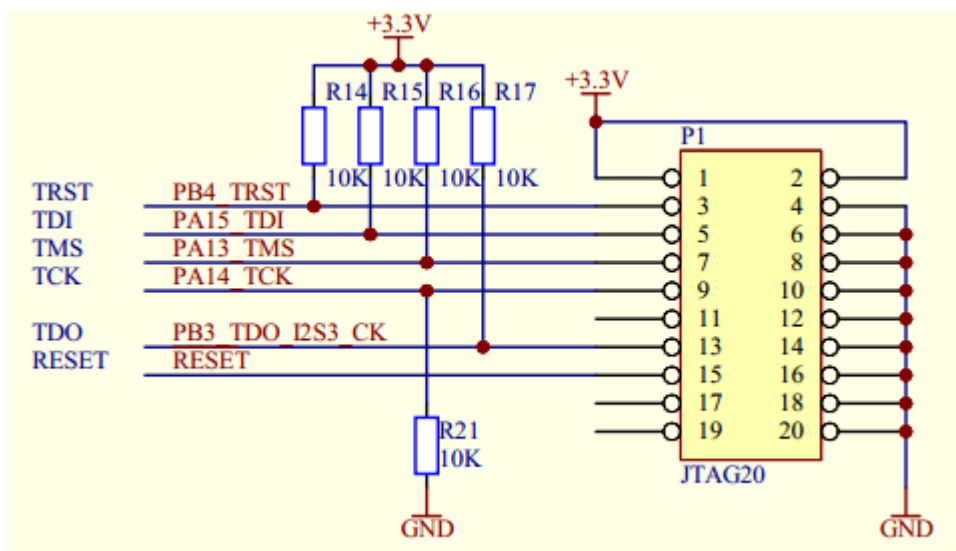


图 4.7 下载电路

4.3 采集模块电路设计

4.3.1 三表合一

考虑到能耗采集系统对市面上绝大多数智能仪表的兼容性，系统使用了 MODBUS、CJ/T 188、DL/T 645 三种不同的通信协议来分别对智能仪表进行数据采集与解析。

采集系统采集部分的关键是三表合一，水电气表都挂载到 RS485 总线上，通过不同的协议解析达到控制器对仪表数据的精确采集，大大方便了用户使用。其中电表使用 MODBUS 通信协议，水表使用 CJ/T 188 通信协议，考虑到市场上现有的智能燃气表可支持 MODBUS、DL/T 645、CJ/T 188 三种协议，为了采集平台的兼容性，我们为燃气表对应开发了三种不同的协议解析，用户燃气表接上监控系统后系统会以轮询的方式向气表发送数据，以此确定用户燃气表所用协议，而采集系统可通过发送向智能仪表发送设备识别码来确定表的种类，以此达到三表合一目的，如图 4.8 所示。

4.3.2 电表采集电路设计

本系统使用的电表为 XY194E 系列 LCD 型三相智能电表，该产品符合国家标准，广泛应用于各类建筑。采用标准数字通信接口 RS-485，和标准 BUS-RTU 通讯协议。

系统采用 Maxim 生产的 MAX3485，将串口转换为 RS485 接口，可支持 RS485 总线通信。其硬件电路设计如图 4.9 所示。

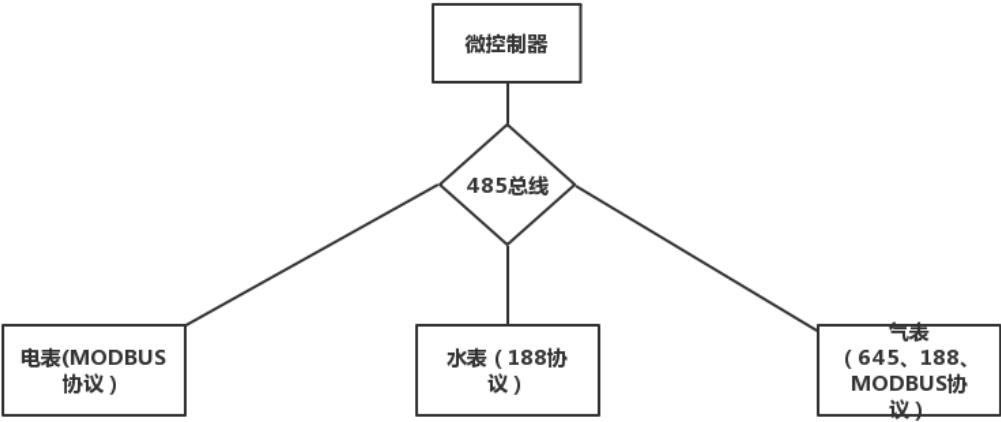


图 4.8 三表合一

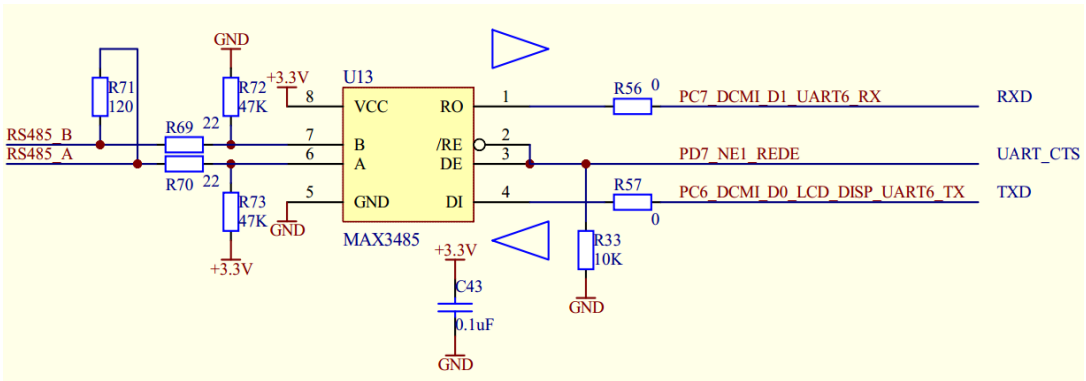


图 4.9 串口转 RS485 电路

4.3.3 水表采集电路设计

本监控系统使用的水表为 M-Bus 湿式光电直读式 (LXZD) 水表。产品符合国家标准 GB/T 778-2007 和建设部标准 CJ/T 224-2012。主要用于企事业单位及居民小区用水的计量与管理工作。

水表模块的硬件主要包括带有 USART 或 UART 功能接口的控制器、MBUS-TTL 电气转换板、具有 MBUS 通信接口的水表。

水表模块硬件设计结构如图 4.10 所示。

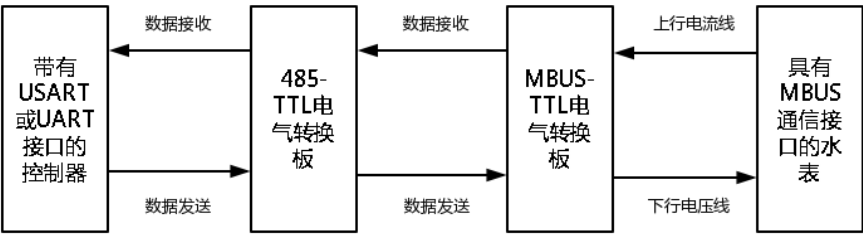


图 4.10 Mbus-485 电气转换

① M-Bus 主控器发送电路如图 4.11 所示。

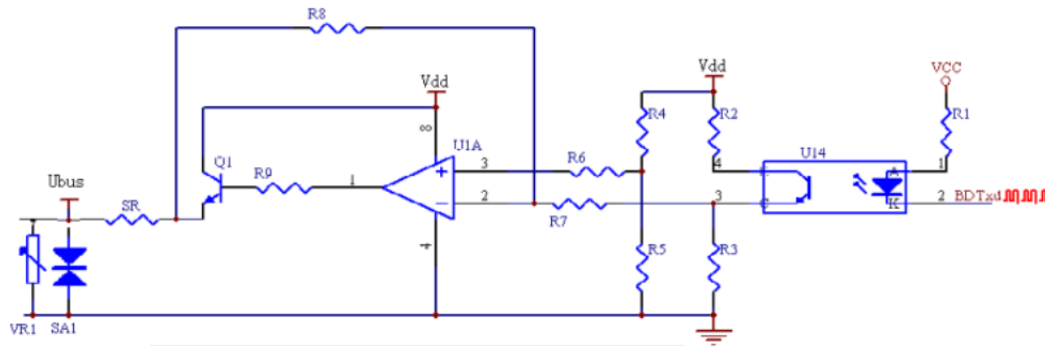


图 4.11 基于 M-bus 的主控制器发送电路

② 主控器接收电路如图 4.12 所示。

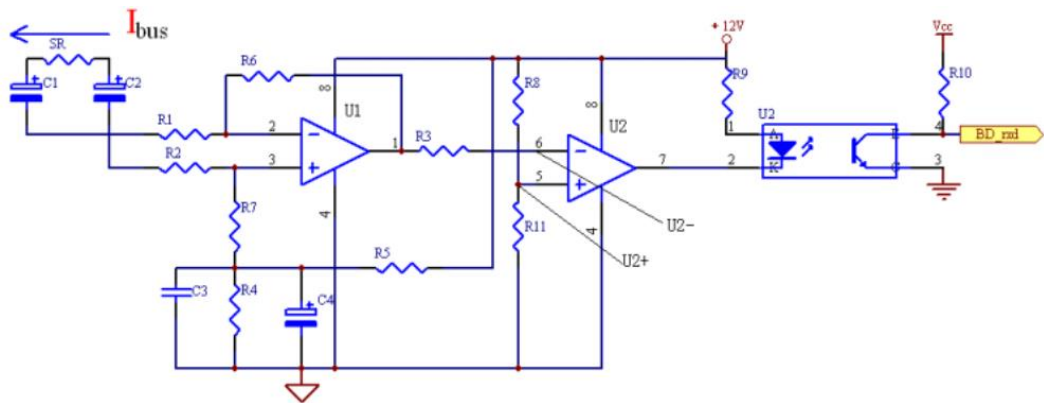


图 4.12 基于 M-bus 的总线主控制器接收电路

③ M-bus 节点的接收电路如图 4.13 所示。

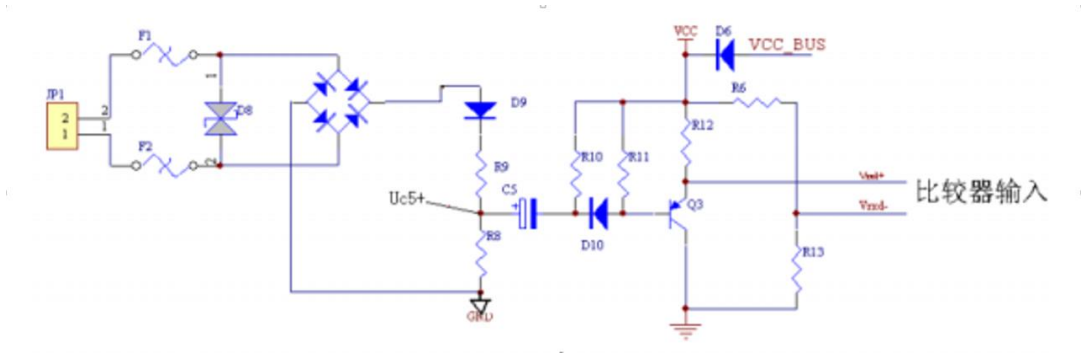


图 4.13 节点接收电路

④ 节点发送电路如图 4.14 所示。

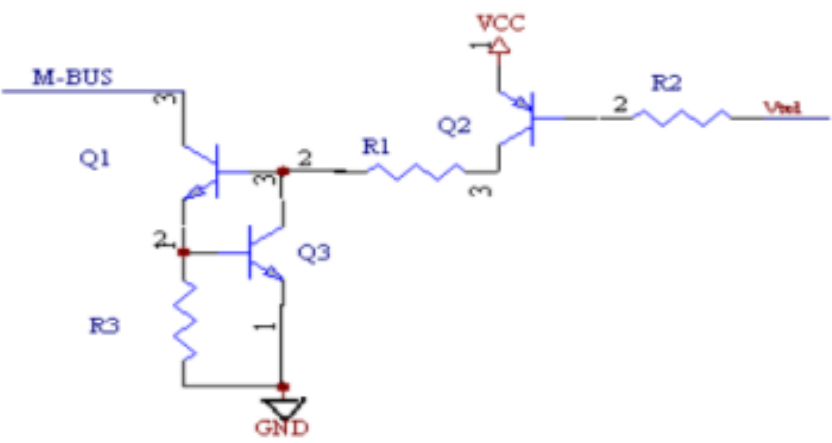


图 4.14 节点发送电路原理

4.3.4 气表数据采集电路设计

本监控系统使用的气表由基表 and 智能传感器两部分组成，基表可以按照用户需求选配任意国标气表，智能传感器的引线就是数据线，直接挂到 RS485 或 MBUS 总线上即可传输数据信号。

气表采集电路实现见上图 4.9。

4.4 存储模块电路设计

4.4.1 存储模块介绍

能耗数据采集系统需要完成数据的解析处理、网络上传任务，还需要屏幕显

示支持，控制器片内存储空间不能满足以上应用。

STM32F429 支持 SRAM、PSRAM、NOR、NAND 和 SDRAM 等多种存储器，可灵活选择外部存储方案。

片外 RAM 采用 SDRAM，主要用于存储显示模块的图像缓存信息。

片外非易失性存储采用 NAND Flash 和 SD 卡^[1]，主要用于存储能耗数据监控系统的配置文件，以及数据断点续传需要保存的临时数据等。

4.4.2 NAND FLASH 设计

本系统采用 NAND FLASH 为 mircon 公司的 MT29F4G08，大小为 4Gbit，其存储单元组织结构如图 4.15 所示：

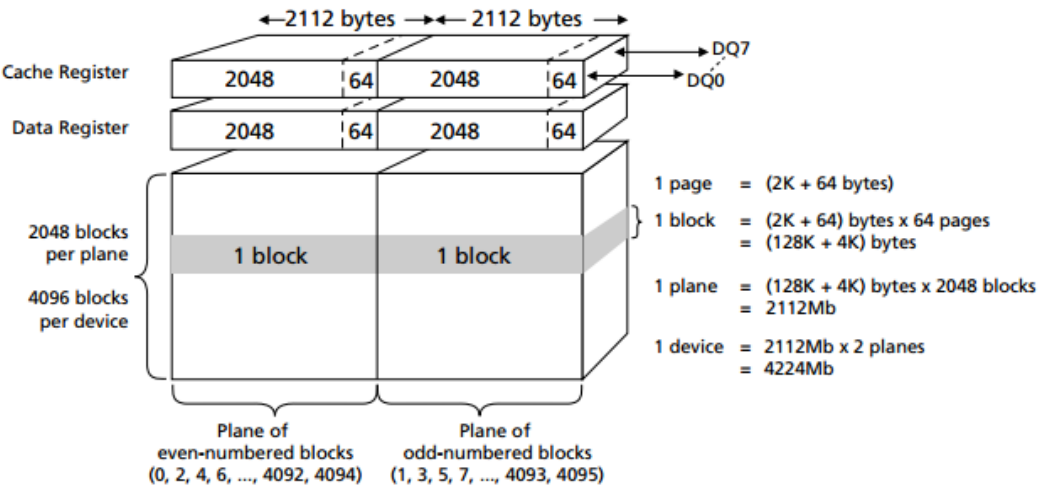


图 4.15 MT29F4G08 存储单元

NAND FLASH 的信号线如图 4.16 所示。

NAND FLASH 挂载到 STM32 特有的 FSMC 上。其接口电路如图 4.17。

Signal ¹	Type	Description ²
ALE	Input	Address latch enable: Loads an address from I/O[7:0] into the address register.
CE# CE#2	Input	Chip enable: Enables or disables one or more die (LUNs) in a target. For the 16Gb device, CE# controls the first 8Gb of memory; CE2# controls the second 8Gb of memory.
CLE	Input	Command latch enable: Loads a command from I/O[7:0] into the command register.
LOCK	Input	When LOCK is HIGH during power-up, the BLOCK LOCK function is enabled. To disable the BLOCK LOCK, connect LOCK to V _{SS} during power-up, or leave it disconnected (internal pull-down).
RE#	Input	Read enable: Transfers serial data from the NAND Flash to the host system.
WE#	Input	Write enable: Transfers commands, addresses, and serial data from the host system to the NAND Flash.
WP#	Input	Write protect: Enables or disables array PROGRAM and ERASE operations.
I/O[7:0] (x8) I/O[15:0] (x16)	I/O	Data inputs/outputs: The bidirectional I/Os transfer address, data, and command information.
R/B# R/B#2	Output	Ready/busy: An open-drain, active-low output that requires an external pull-up resistor. This signal indicates target activity. For the 16Gb device, R/B# indicates the status of the first 8Gb of memory; R/B# indicates the status of the second 8Gb of memory.
V _{CC}	Supply	V_{CC}: Core power supply
V _{SS}	Supply	V_{SS}: Core ground connection
NC	–	No connect: NCs are not internally connected. They can be driven or left unconnected.
DNU	–	Do not use: DNUs must be left unconnected.

图 4.16 NAND FLASH 信号线

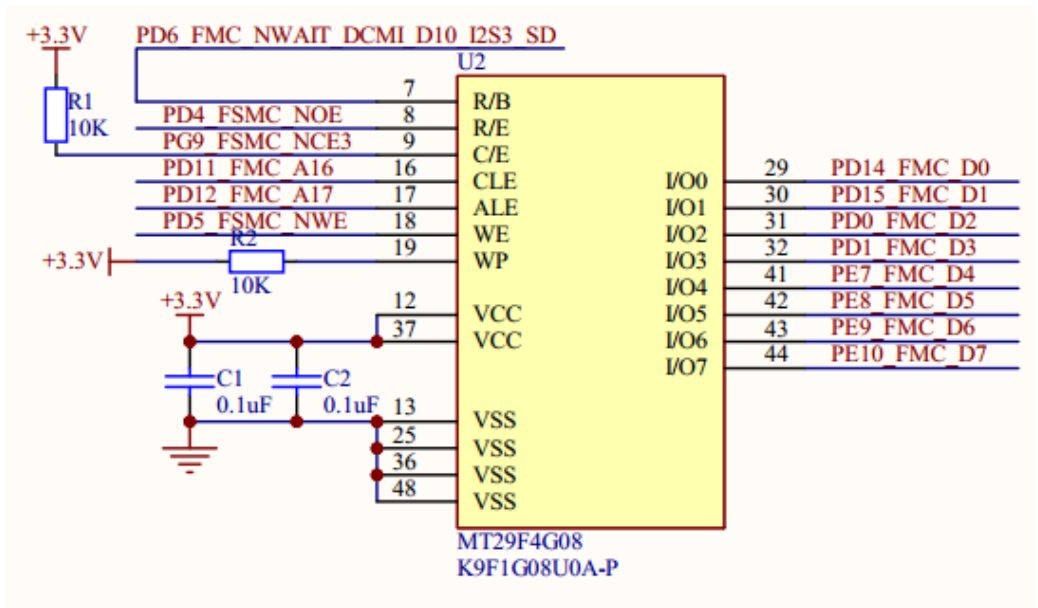


图 4.17 NAND Flash 接口电路

4.4.3 SD 卡电路设计

本系统采用 4bit 的 SDIO 接口连接 SD 卡^[1]并通过普通 I/O P18 来检测 SD 卡槽上是否有 SD 卡存在，SDIO 上使用的 I/O 部分与 DCMI 接口复用，同一时间智能实现其中一种，其管脚定义如图 4.18 所示。

SD 卡接口电路如图 4.19 所示。

管脚名称	SD 卡	摄像头
PC8	SDIO_D0	DCMI_D2
PC9	SDIO_D1	DCMI_D3
PC10	SDIO_D2	DCMI_D8
PC11	SDIO_D3	DCMI_D4
PC12	SDIO_CK	DCMI_D9
PD2	SDIO_CMD	DCMI_D11

图 4.18 SD 卡管脚定义

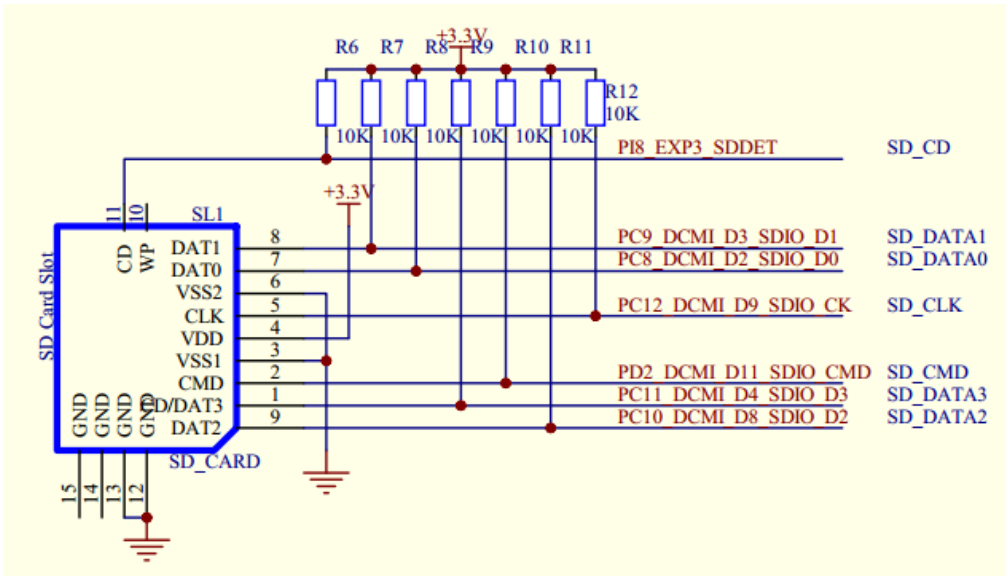


图 4.19 SD 卡接口电路

4.4.4 SDRAM 电路设计

本系统采用的 SDRAM 是 MT48LC8M16A2P，大小为 128Mb，数据宽度为 16 位。直接挂在到 STM32 特有的 FMC 上，其接口电路如图 4.20 所示。其原理框图如图 4.21 所示。其信号线如图 4.22 所示。

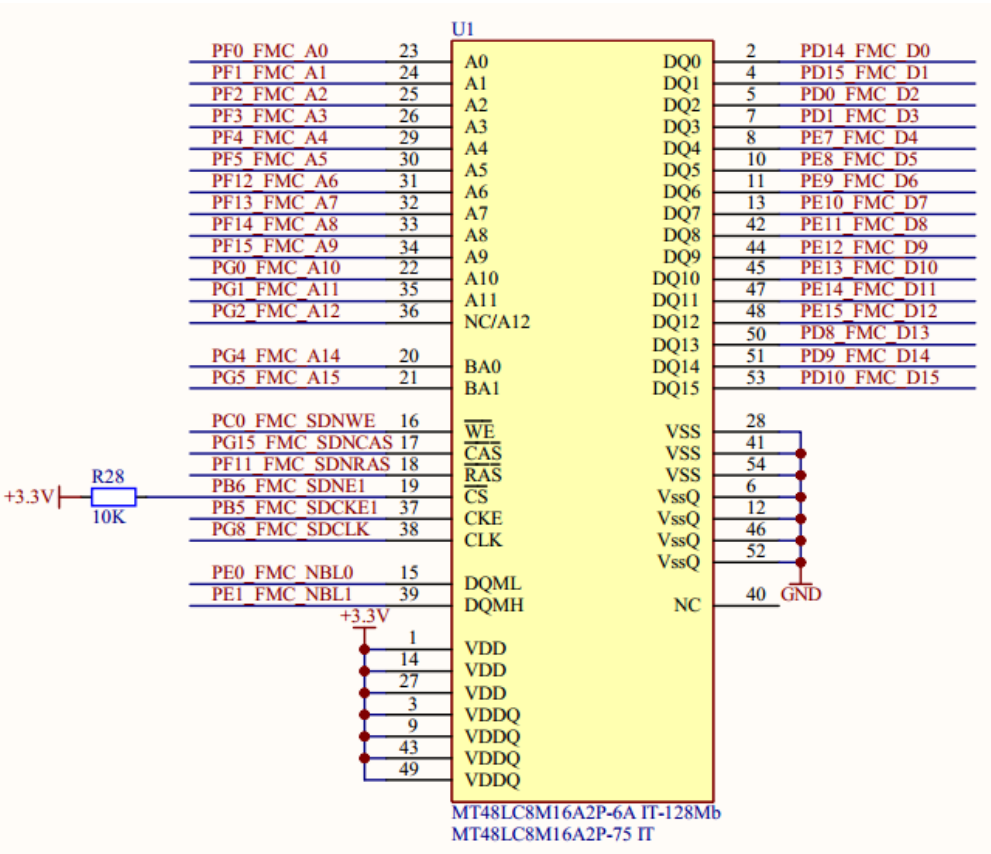


图 4.20 SDRAM 接口电路

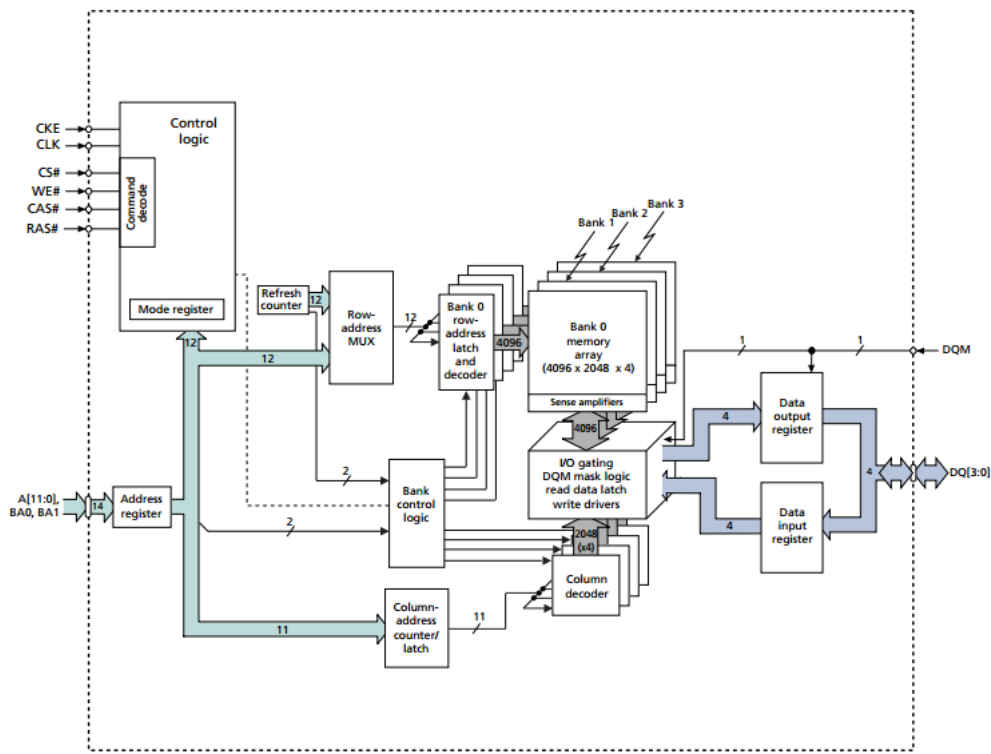


图 4.21 SDRAM 原理框图

Symbol	Type	Description
CLK	Input	Clock: CLK is driven by the system clock. All SDRAM input signals are sampled on the positive edge of CLK. CLK also increments the internal burst counter and controls the output registers.
CKE	Input	Clock enable: CKE activates (HIGH) and deactivates (LOW) the CLK signal. Deactivating the clock provides precharge power-down and SELF REFRESH operation (all banks idle), active power-down (row active in any bank), or CLOCK SUSPEND operation (burst/access in progress). CKE is synchronous except after the device enters power-down and self refresh modes, where CKE becomes asynchronous until after exiting the same mode. The input buffers, including CLK, are disabled during power-down and self refresh modes, providing low standby power. CKE may be tied HIGH.
CS#	Input	Chip select: CS# enables (registered LOW) and disables (registered HIGH) the command decoder. All commands are masked when CS# is registered HIGH, but READ/WRITE bursts already in progress will continue, and DQM operation will retain its DQ mask capability while CS# is HIGH. CS# provides for external bank selection on systems with multiple banks. CS# is considered part of the command code.
CAS#, RAS#, WE#	Input	Command inputs: CAS#, RAS#, and WE# (along with CS#) define the command being entered.
x4, x8: DQM x16: DQML, DQMH	Input	Input/output mask: DQM is sampled HIGH and is an input mask signal for write accesses and an output enable signal for read accesses. Input data is masked during a WRITE cycle. The output buffers are High-Z (two-clock latency) during a READ cycle. On the x4 and x8, DQML (pin 15) is NC; DQMH is DQM. On the x16, DQML corresponds to DQ[7:0] and DQMH corresponds to DQ[15:8]. DQML and DQMH are considered same-state when referenced as DQM.
BA[1:0]	Input	Bank address input(s): BA[1:0] define to which bank the ACTIVE, READ, WRITE, or PRECHARGE command is being applied.
A[11:0]	Input	Address inputs: A[11:0] are sampled during the ACTIVE command (row address A[11:0]) and READ or WRITE command (column address A[9:0] and A11 for x4; A[9:0] for x8; A[8:0] for x16; with A10 defining auto precharge) to select one location out of the memory array in the respective bank. A10 is sampled during a PRECHARGE command to determine whether all banks are to be precharged (A10 HIGH) or bank selected by BA[1:0] (A10 LOW). The address inputs also provide the op-code during a LOAD MODE REGISTER command.
x16: DQ[15:0]	I/O	Data input/output: Data bus for x16 (pins 4, 7, 10, 13, 42, 45, 48, and 51 are NC for x8; and pins 2, 4, 7, 8, 10, 13, 42, 45, 47, 48, 51, and 53 are NC for x4).
x8: DQ[7:0]	I/O	Data input/output: Data bus for x8 (pins 2, 8, 47, 53 are NC for x4 TSOP; balls A8, D8, D1, and A1 are NC for x4 FBGA).
x4: DQ[3:0]	I/O	Data input/output: Data bus for x4.
V _{DDQ}	Supply	DQ power: Isolated DQ power to the die for improved noise immunity.
V _{SSQ}	Supply	DQ ground: Isolated DQ ground to the die for improved noise immunity.
V _{DD}	Supply	Power supply: 3.3V \pm 0.3V.
V _{SS}	Supply	Ground.
NC	–	No connect: These should be left unconnected. For x4 and x8 parts, G1 is a no connect; it is A12 for 256Mb and 512Mb devices.

图 4.22 SDRAM 信号线

4.5 网络模块电路设计

本系统采用的网络芯片为 DP83848IVV，采用集成滤波器的 RJ45 网络接口 HR911105A，可以实现 10/100M 自适应的网络连接。通过配置 J1 上的短路帽，可以将网络设置成 MII 及 RMII 模式，网络 PHY 芯片运行在 MII 模式下使用 25M 的无源晶振占用较多的 IO，运行在 RMII 模式下则使用 50MHZ 的有源晶振占用较少 I/O，网络默认配置成 MII 模式。如图 4.23 为网络模式配置。

如图 4.24 和图 4.25 分别为 MII 模式下和 RMII 模式下用到的 I/O。





通信模式	J1	JP1
MII		
RMII		

图 4.23 网络 PHY 芯片模式配置

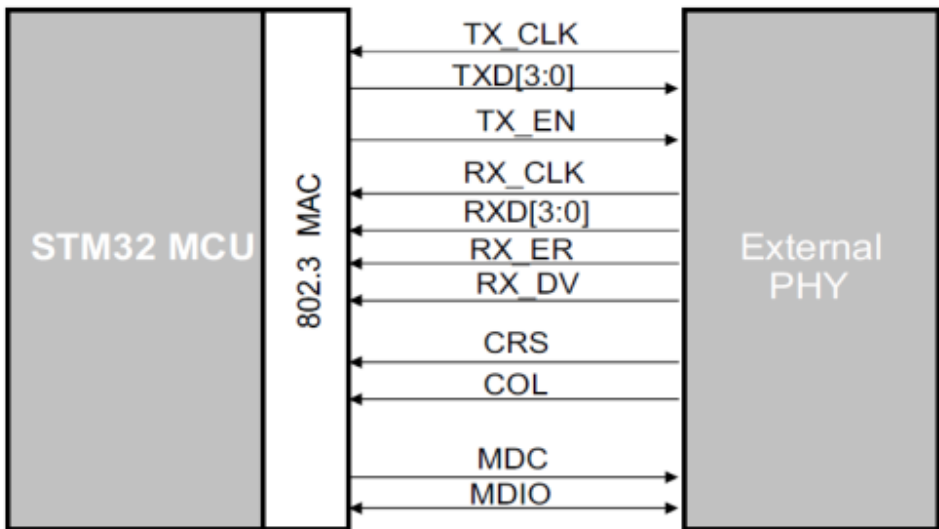


图 4.24 MII 模式下用到的 I/O

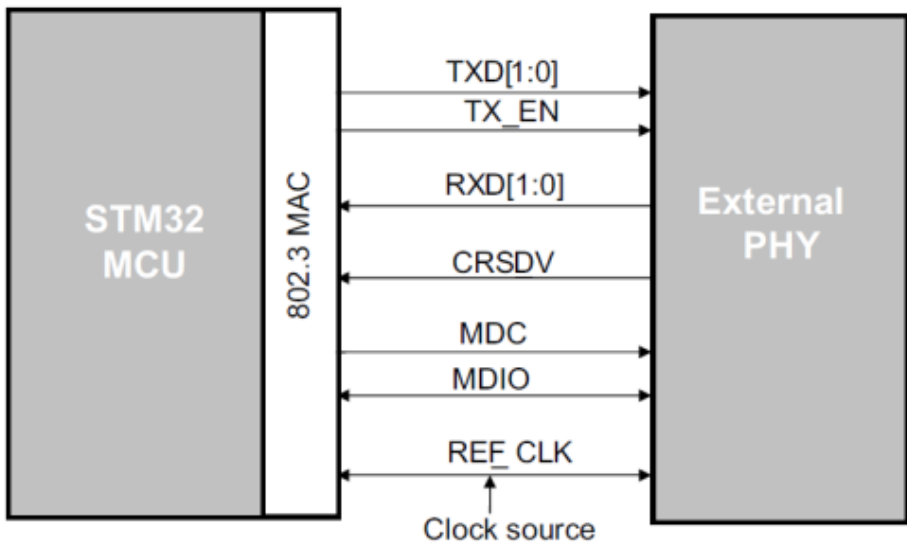
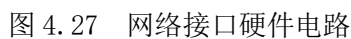
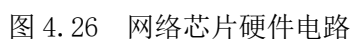


图 4.25 RMII 模式下用到的 I/O

如图 4.26 和图 4.27 分别为网络芯片 DP83848IVV 连接电路和网络接口



4.6 显示模块电路设计

4.6.1 LCD 介绍

显示方案我们使用了 TFT-LCD，TFT-LCD 采用每个像素设置一个薄膜晶体管的方式，极大的提高了其显示质量。实物图如图 4.28 所示：



图 4.28 LCD 实物图

4.6.2 TFT-LCD 原理图设计

如图 4.29 所示为 TFT-LCD 硬件接口电路，显示屏与外部控制器通过 24 位 RGB 并行接法相连接。

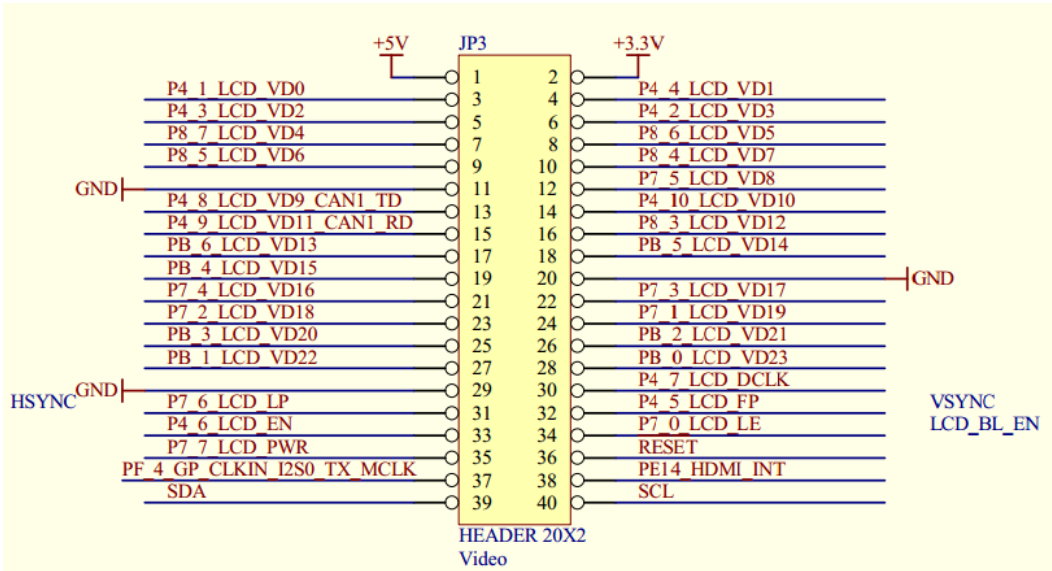


图 4.29 TFT-LCD 接口电路

主控单元通过 I2C 与触摸屏控制芯片 STMPE811 相连，如图 4.30 所示。

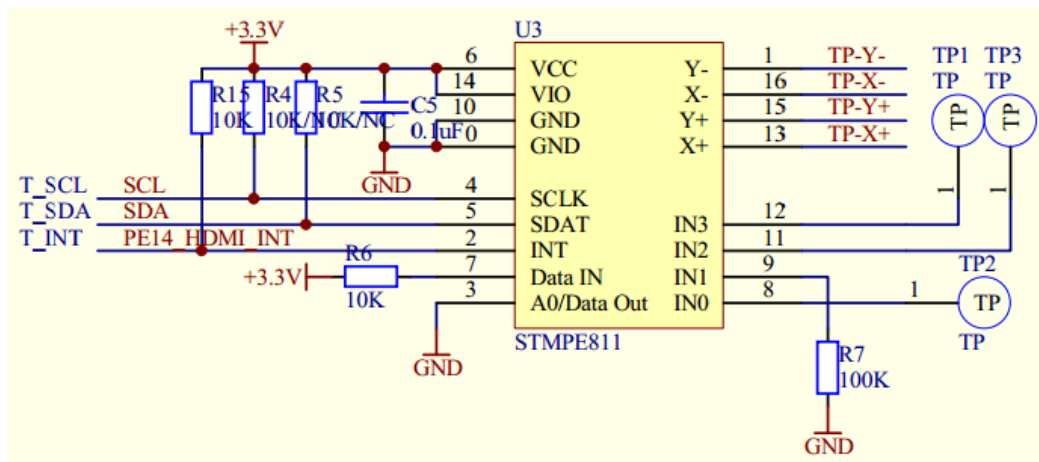


图 4.30 触摸屏控制芯片接口电路

第五章 系统软件设计

5.1 调试环境

本系统使用 Keil - MDK5 作为软件开发调试环境，是目前 ARM 内核微控制器开发的主流工具。

5.2 实时操作系统

5.2.1 FreeRTOS 系统介绍

本系统采用 FreeRTOS 实时操作系统^[10]。该系统是一个开源的小型嵌入式操作系统，但是其具有高效的多任务调度算法，大大降低了开发难度，以及其具有四种内存管理方式，高效的利用了控制器的片内外存储介质。如图 5.1 为 FreeRTOS 的任务状态转换关系。

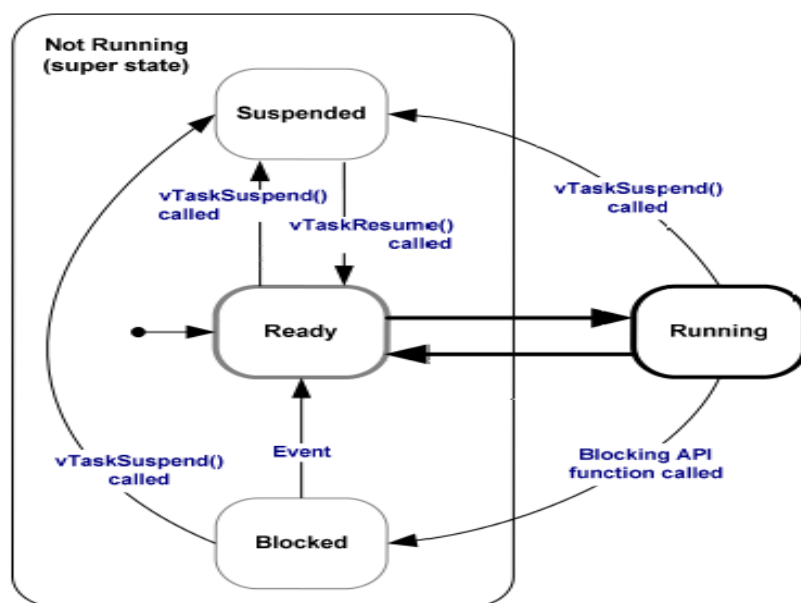


图 5.1 任务状态转换关系

为了实现能耗数据监控系统的功能，在 FreeRTOS 创建了多个任务，使得软件模块分明更加有条理。

5.2.2 FreeRTOS 系统移植

FreeRTOS 的移植是 FreeRTOS 使用的关键，关于 FreeRTOS 的移植主要由三个

文件实现，分别是.h文件，.c文件和.s文件。移植之后只需调用上层API函数，极大的降低了开发难度。

FreeRTOS 的移植过程：

从官网上下载 FreeRTOS 的库文件，将库文件存在合适的文件夹中；

与移植相关的代码主要有三个文件，分别为 port.c、portasm.s、portmacro.h，分别对应 C 接口部分，汇编处理部分和宏定义部分；

将 OS 相关文件移植过来后，然后根据系统资源，对 FreeRTOS 进行配置，主要配置文件为 FreeRTOSConfig.h；

移植完成后，就可以直接通过 API 进行任务创建，在任务中执行所需的功能，然后交给 OS 进行循环调度。

如图 5.2 所示为本系统几个主要任务的定义与创建。

```
//对 GUI 线程的定义
osThreadDef(gui, GUI_Thread, osPriorityBelowNormal, 0, configMINIMAL_STACK_SIZE * 15);
//对 GUI 线程的创建
GUIThreadHandle = osThreadCreate (osThread(gui), NULL);
//对存储线程的定义
osThreadDef(store, store_task, osPriorityNormal, 0, configMINIMAL_STACK_SIZE * 3);
//对存储线程的创建
StoreThreadHandle= osThreadCreate(osThread(store), NULL);
//对管理线程的定义
osThreadDef(manage_task, manageThread, osPriorityAboveNormal, 0, configMINIMAL_STACK_SIZE * 3);
//对管理线程的创建
ManageThreadHandle = osThreadCreate(osThread(manage_task), NULL);
//对采集线程的定义
osThreadDef(collect, Collect_Thread, osPriorityAboveNormal, 0, configMINIMAL_STACK_SIZE * 5);
//对采集线程的创建
CollectThreadHandle = osThreadCreate (osThread(collect), NULL);
//对网络线程的定义
osThreadDef(Start, StartThread, osPriorityNormal, 0, configMINIMAL_STACK_SIZE * 5);
//对网络线程的创建
osThreadCreate (osThread(Start), NULL);
//开启线程
osKernelStart (NULL, NULL);
}
```

图 5.2 任务创建

系统内的任务基本可分为 3 个层级，本其中第二层级任务较多采用分时间片的方式依序进行，如图 5.3 为系统任务调度的基本逻辑，可以清楚的看清楚网络

任务、GUI 任务和存储任务，是通过时间片轮转的方式一次进行。

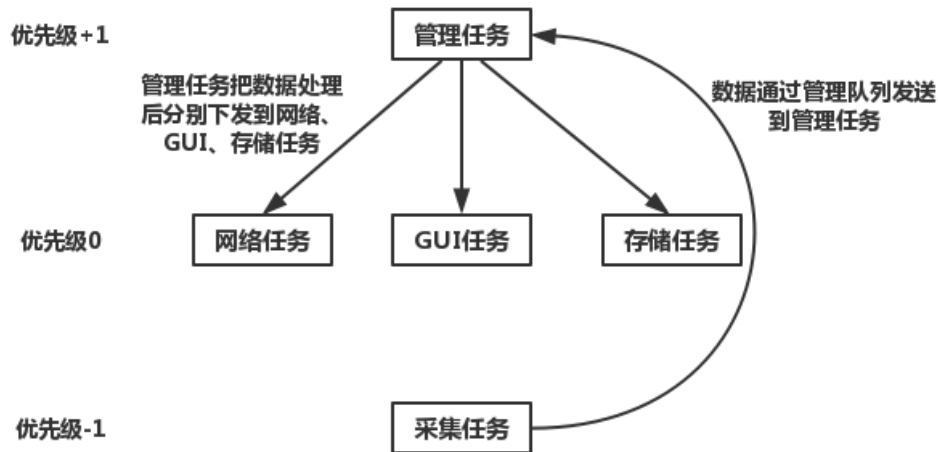


图 5.3 系统调度

5.3 采集模块设计

从 manage 任务获取待采集仪表的端口、协议、所需要采集的数据地址等、配置板上串行接口资源（具体参照 5.4.1 章节），根据不同仪表通讯协议完成仪表数据的采集。并将采集完成消息通过消息队列发给管理任务。

考虑到数据采集需要有多路串口同时进行采集，串口采集底层采用 DMA 发送与接收，以减少控制器的负担。

以水表为例，如图 5.4 为水表采集数据的流程图，采集任务会发送地址请求命令给智能仪表，以获取其地址并存储，仪表数据是通过发送读数据指令给智能仪表，从而获取当前能耗数据。

图 5.5 是采集任务部分代码，采集的过程中会对每个串口通信端口上所有的从机（智能仪表）的数据进行轮询采集。

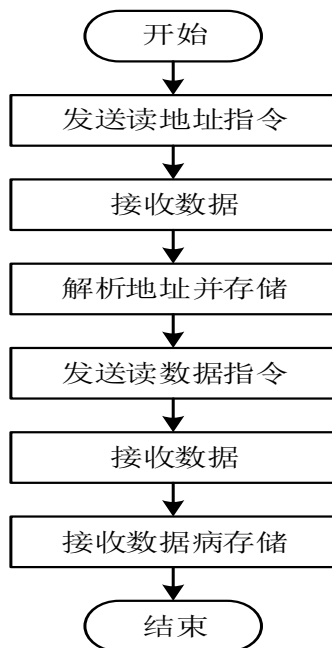


图 5.4 水表采集流程图

```

for(;;)
{
    uint8_t meter_index = 0;
    i = meterPort[port_index].number;
    if(i != 0)
    {
        while(meter_index != i)
        {
            modBus_t buf = {0};
            meterPort[port_index].meterNode[meter_index].meterAddr;
            p_item = meterPort[port_index].meterNode[meter_index].head;
            while(1)
            {
                if(p_item == NULL)
                {
                    break;
                }
                buf.regAddrHi = p_item->regAddrHi;
                buf.regAddrLo = p_item->regAddrLo;
                buf.lenLo = p_item->regSize;
                buf.funCode = p_item->funCode;
                if(BSP_ComSend(meterPort[port_index].port, (uint8_t*)&buf,
sizeof(buf)) != BSP_OK)
                {
                    Error_Handler();
                }
                flag[p_item->ParaId] = 1;
                Transport(p_item, bufRx.modBusRx.dat, NULL);
                osDelay(50);
                p_item = p_item->next;
            }
        }
    }
}

```

图 5.5 采集模块部分代码

5.4 存储模块设计

本系统采用了小型的文件系统 FatFS^[2]，用来管理系统的存储介质，其支持多个存储介质，有独立的缓冲区，可对多个文件进行读/写。FATFS 模块的层次结构如图 5.6 所示。

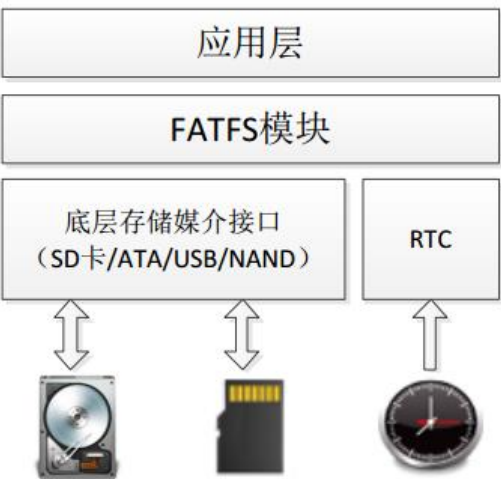


图 5.6 FATFS 层次结构

5.4.1 配置文件读取

在上电后能耗数据采集系统需要知道从哪个端口采集什么仪表，采集的数据在仪表中的地址是多少等一系列信息，此时就需要在存储中获取相关的信息。

下图 5.7 为 NAND FLASH 中配置文件，对采集一个多功能仪表所需的请求命令等。

```
<
1 1 1 1 0 1 1ABCD 0
0 0 26 1 0 04 2
1 0 27 1 0 04 2
2 0 28 1 0 04 2
4 0 20 1 0 04 1
5 0 21 1 0 04 1
6 0 22 1 0 04 1
8 0 33 1 0 04 4
9 0 45 1 0 04 5
10 0 59 2 0 04 8
>
```

图 5.7 配置文件

从配置文件的内容可以看出，“<”后的第一行为仪表信息，之后到“>”之前的所有的行，每行均表示该仪表的采集项目。

仪表信息及采集项目分别如表 5.1—5.2 所示。

表 5.1 仪表信息

仪表 ID	ModBus 表地址	电压变比 P	电流变比 T	串口端口号	通信协议	分类分项编码	异常标志
1	1	1	1	0	1	1ABCD	0

表 5.2 采集项目

参数 ID	寄存器高地址	寄存器低地址	寄存器大小	参数属性	功能码	参数计算方法
0	0	26	1	0	04	2

如图 5.8 所示为完成初始化时能耗数据监控系统运行参数的读取代码。

5.4.2 历史数据存储

存储任务软件流程如图 5.9 所示，此任务会将能耗历史数据按照不同用能种类仪表分别创建文件夹进行存储。在各文件夹内采用制表位分隔的.xls 文件形式存储，采用.xls 格式存储方便了之后对数据的分析汇总，能耗数据采集系统每隔一小时将会新建一个以时间命名的.xls 文件。

由于系统启动后，在管理任务中便需要在内存中读取配置文件 config.cfg，所以在管理任务中已经对存储方面进行了初始化（5.3 章节有说明），存储任务运行后，在进行相关变量的定义后，直接获取消息队列中的数据进行存储。

```

int32_t getMeterInfo(void)
{
    FIL ConfigFile;
    char fileName[] = "config.cfg";
    char index[COM_MAX] = {0};
    meterNode_t tmp;
    if(FATFS_LinkDriver(&SD_Driver, SDPath) == 0)
    {
        return -1;
    }
    else if(f_open(&ConfigFile, fileName, FA_READ) != FR_OK)
    {
        return -1;
    }
    else
    {
        while(readConfigFile(&ConfigFile, &tmp) == 0)
        {
            meterPort[0].port = METER_PORT1;
            meterPort[1].port = METER_PORT2;
            meterPort[2].port = DEBUG_PORT;
            switch(tmp.Port)
            {
                case METER_PORT1:
                    memcpy(&meterPort[0].meterNode[index[0]], &tmp, sizeof(meterNode_t));
                    index[0]++;
                    break;
                case METER_PORT2:
                    memcpy(&meterPort[1].meterNode[index[1]], &tmp, sizeof(meterNode_t));
                    index[1]++;
                    break;
                case DEBUG_PORT:
                    memcpy(&meterPort[2].meterNode[index[2]], &tmp, sizeof(meterNode_t));
                    index[2]++;
                    break;
            }
            meterPort[0].number = index[0];
            meterPort[1].number = index[1];
            meterPort[2].number = index[2];
        }
        f_close(&ConfigFile);
        return 0;
    }
}

```

图 5.8 读取初始化文件

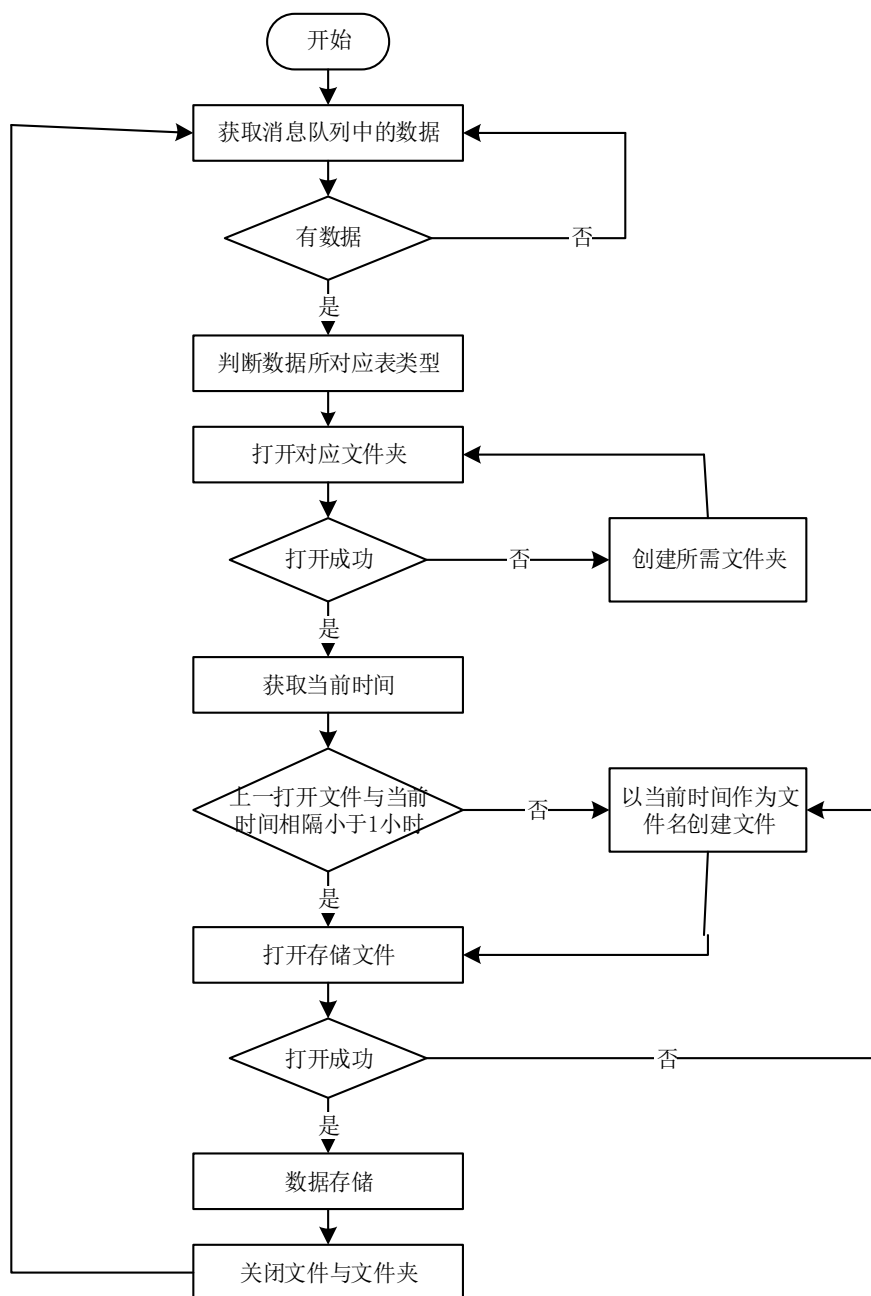


图 5.9 数据存储流程图

如图 5.10 可以看到在存储能耗数据是按照仪表的种类进行存储的。图 5.11 为 Excel 打开的历史数据，若电表不支持某种计量项目，则显示 INVALID 表示该数据不可用，可以看到，当前测量的为三项智能电表。


```

void store_task(void const *argument)
{
    meterNode_t* pMeterNode;          //电表的节点信息
    DEBUG_PRINTF("store_task begin.\n");

    while(1)
    {
        event = osMessageGet(SD_Q, osWaitForever); //消息队列
        DEBUG_PRINTF("store_task have data\n");
        if(event.status == osEventMessage)
        {
            pmsg = event.value.p;
            pMeterNode = pmsg->p_msg;
            sprintf(meterNodeId_s, "%d", pMeterNode->meter_Id);
            sprintf(meterNodePort_s, "%d", pMeterNode->Port);
            store_meterdata();
            memset(&event, 0, sizeof(osEvent));
        }
        osThreadYield();
    }
}

uint32_t store_meterdata(void)
{
    DIR ENandDir;
    DIR WNandDir;
    DIR GNandDir;
    DIR ESdDir;
}

```

图 5.12 存储部分代码

5.5 显示模块软件设计

本系统采用图形库 emWin，通过 GUIbuilder 图形设计软件设计功能界面，然后将那个文件转换为 C 代码，提供给 emWin API 进行调用；还可通过 BmpCvt 位图转换器将图片转换成 C 数组或者二进制文件，然后通过 emWin API 进行调用。在开发过程中，可以使用 vs2013 进行 PC 仿真调试，然后将代码移植进 MDK 工程。其软件系统结构流程图如图 5.13 所示。

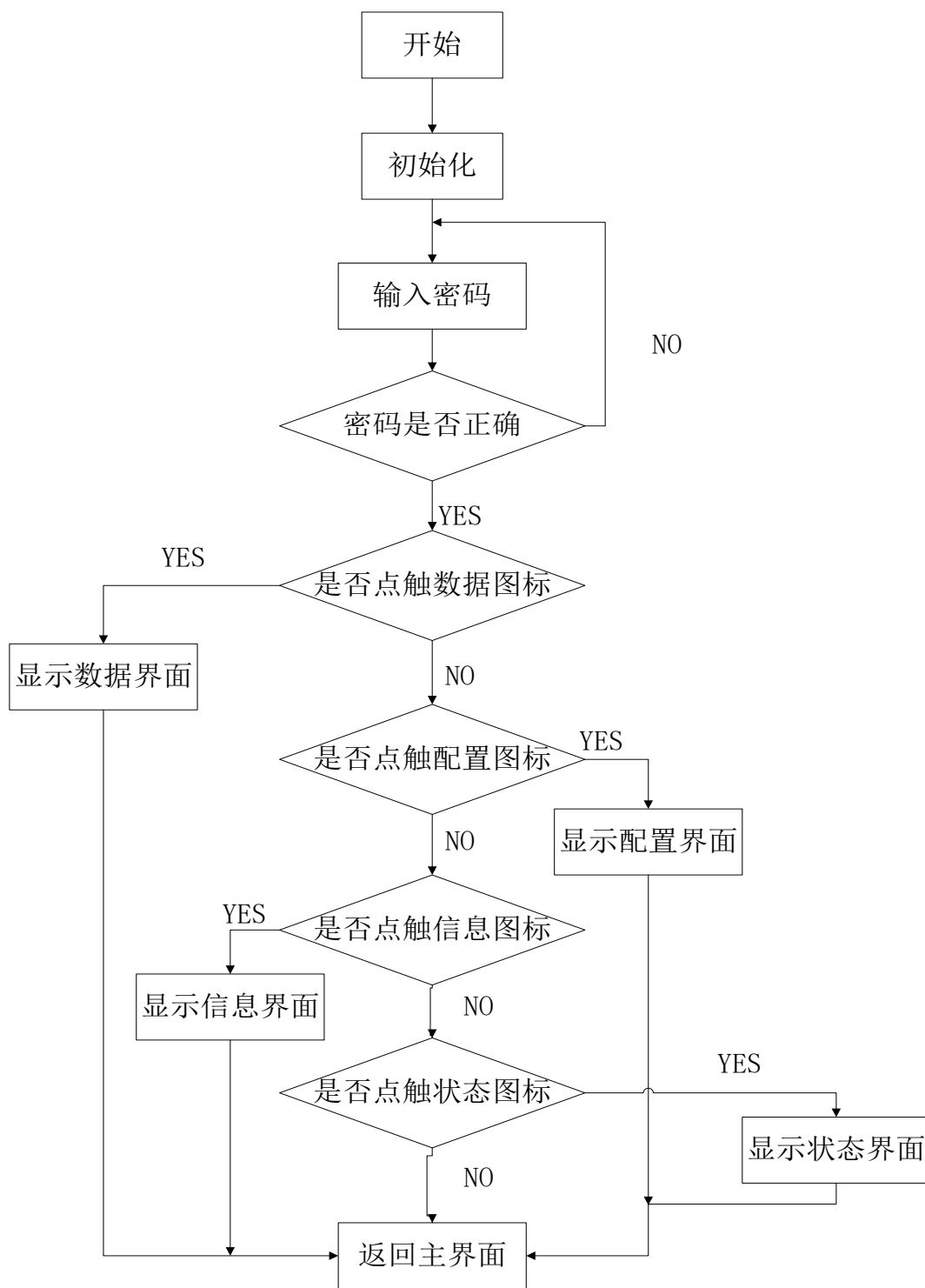


图 5.13 GUI 流程图

其部分代码如图 5.14 所示。

```

void MainTask(void) {
    WM_HWIN hDlg;
    GUI_Init();
    hDlg = CreateIPCONF();
    while (1)
        GUI_Delay(100);
}

if (CompareEqualString(sc, "123") == 1)
{
    GUI_EndDialog(pMsg->hWin, 0);
    for (i = 0; i < MASK_NUM; i++)
    {
        sc[i] = '\0';
        vc[i] = '\0';
    }
    k = 0;
    Task();

    osEvent event;
    msg_t* pmsg;
    event = osMessageGet(GUI_Q, 100);
    if(event.status == osEventMessage)
    {
        pmsg = osEventMessageGet(GUI_Q, 100);
        pMeterNode = pmsg->p_msg;
    }

    memcpy(OutputDisplay.I[0].str, &meterData.currentA, strlen((const char
}

```

图 5.14 GUI 部分代码

5.6 网络模块设计

由网络任务来完成能耗数据采集系统与服务器的通讯流程，在网络任务创建好之后，首先进行 TCP 三次握手^[3]，然后保持连接状态。在采集器采集仪表数据后，manage 任务通过消息队列发送数据至网络任务，网络任务对数据进行 JSON 数据格式打包，然后在 TCP 数据包之前加上 HTTP 头^[3]，封装为 HTTP 数据包，上传指 HTTP 服务器。并且网络任务中还有一个线程在接受服务器发送过来的数据，可以接受数据中心下发的配置信息、数据请求的数据包，并将接收到的信息交给管理任务处理。

其软件系统流程图如图 5.15 所示，可以看到，我们虽然将数据包最后封装成为了 HTTP 格式，但是最终还是用 TCP 进行的数据传输，主要原因是考虑到服务器方面的需求和应用。如果服务器有变化，可去掉 HTTP 头部，直接通过 TCP 发送 JSON 数据包^[4]即可。

如图 5.16 为网络部分代码

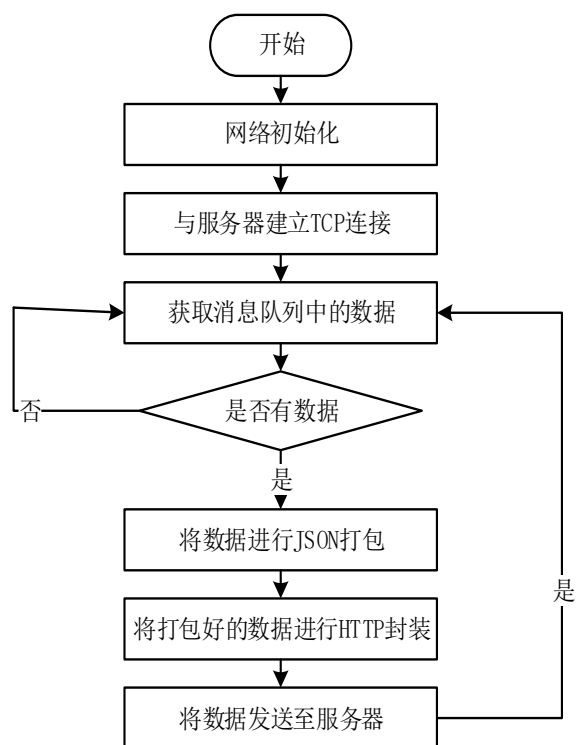


图 5.15 网络流程图

```

static void Netif_Config(void)
{
    struct ip_addr ipaddr;
    struct ip_addr netmask;

    struct ip_addr gw;

    IP4_ADDR(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
    IP4_ADDR(&netmask, NETMASK_ADDR0, NETMASK_ADDR1, NETMASK_ADDR2, NETMASK_ADDR3);
    IP4_ADDR(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);

    netif_add(&gnetif, &ipaddr, &netmask, &gw, NULL, &ethernetif_init, &tcpip_input);

    netif_set_default(&gnetif);

    if (netif_is_link_up(&gnetif))
    {
        netif_set_up(&gnetif);
    }
    else
    {
        /* When the netif link is down this function must be called */
        netif_set_down(&gnetif);
    }

    /* Set the link callback function, this function is called on change of link status */
    netif_set_link_callback(&gnetif, ethernetif_update_config);

    osSemaphoreDef(Netif_SEM);
    Netif_LinkSemaphore = osSemaphoreCreate(osSemaphore(Netif_SEM), 1);

    link_arg.netif = &gnetif;
    link_arg.semaphore = Netif_LinkSemaphore;
    osThreadDef(LinkThr, ethernetif_set_link, osPriorityNormal, 0,
configMINIMAL_STACK_SIZE * 5);
    osThreadCreate (osThread(LinkThr), &link_arg);
}

```

图 5.16 网络任务部分代码

第六章 调试

本章是关于基于物联网的建筑能耗监控系统—网关采集设计的调试，主要是对硬件系统进行搭建调试，分别检测各功能是否正常。

6.1 GUI 显示模块调试

GUI 显示主要是用来现场查看仪表数据，可以很好的辅助设备的现场维护，以及方便了开发过程中的调试改进，测试结果如下。



图 6.1 GUI 主界面

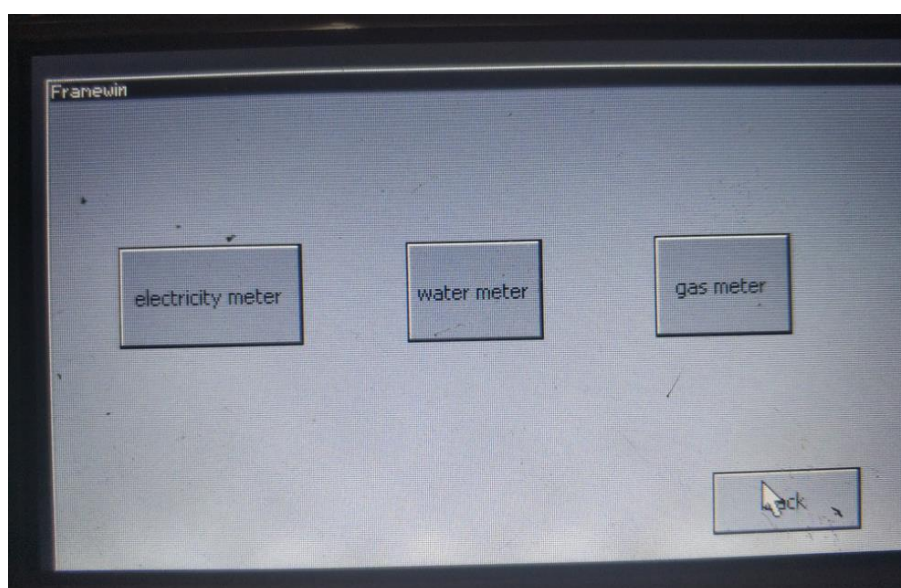


图 6.2 GUI 操作界面

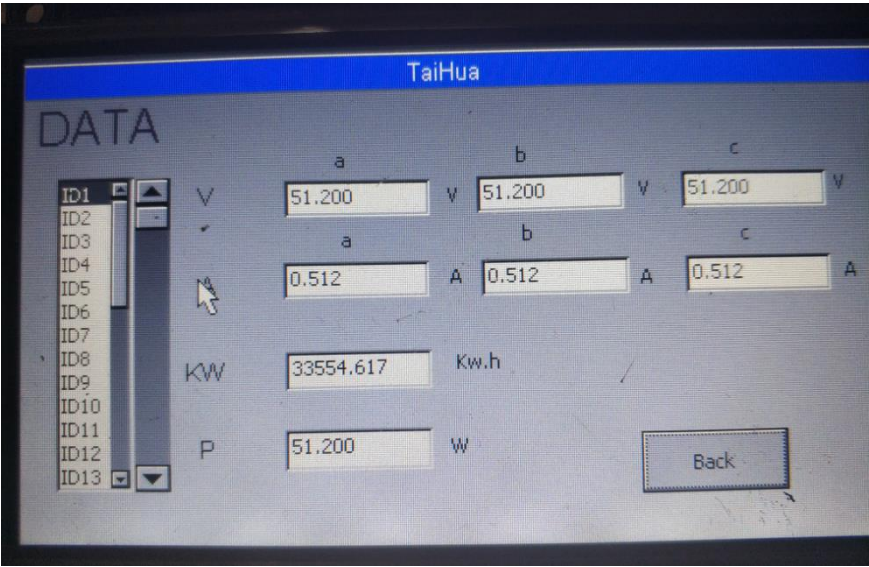


图 6.3 数据显示界面

6.2 网络上传模块调试

在从 manage 任务获取数据以后，网络任务会将数据进行 JSON 打包，上传指服务器，可以在服务器中对历史数据进行存储，以便服务器对数据进行融合分析，以及远程查看数据。本地使用网络助手测试结果如下。

```
网络数据接收
{
  "unit": "m^3"
}
POST /meter_parser HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json
Content-Length: 303
{
  "ewg": 1,
  "ID": 1,
  "Port": 0,
  "data": {
    "currentA": "0.000",
    "currentB": "0.000",
    "currentC": "0.000",
    "current": "",
    "voltageA": "0.000",
    "voltageB": "227.100",
    "voltageC": "0.000",
    "voltage": "",
    "activePower": "0.000",
    "powerFactor": "0.000",
    "activeEnergy": "47.408"
  }
}
POST /meter_parser HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json
Content-Length: 94
{
  "ewg": 3,
  "ID": 0,
  "add": "",
  "data": {
```

图 6.4 网络接收到的数据

6.3 存储模块调试

对于能耗监控系统，除了将数据进行 GUI 显示和远传服务器之外，还进行本地存储备份，这样有利于本地对能耗数据进行分析，测试结果如下。

名称	修改日期	类型	大小
elemeter		文件夹	
gasmeter		文件夹	
watmeter		文件夹	
config.cfg	2016/12/11 19:48	CFG 文件	1 KB

图 6.5 本地存储文件夹

名称	修改日期	类型	大小
2018052802.xls		Microsoft Excel ...	9 KB

图 6.6 本地存储文件

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
64	2.01E+11	0	1	0	0	0	INVALID	0	228.7	0	INVALID	0	0	47.408
65	2.01E+11	0	1	0	0	0	INVALID	0	228.7	0	INVALID	0	0	47.408
66	2.01E+11	0	1	0	0	0	INVALID	0	228.7	0	INVALID	0	0	47.408
67	2.01E+11	0	1	0	0.016	0.064	INVALID	6.4	229.3	0	INVALID	0	0.064	1095.984
68	2.01E+11	0	1	0.008	0.096	0.096	INVALID	1.6	229.3	0.8	INVALID	12.8	0.016	4241.712
69	2.01E+11	0	1	0.016	0.128	0.024	INVALID	0	229.2	0	INVALID	0	0.032	2144.56
70	2.01E+11	0	1	0	0	0.008	INVALID	6.4	229.5	4.8	INVALID	6.4	0	47.408
71	2.01E+11	0	1	0	0.016	0.064	INVALID	12.8	229.4	0	INVALID	0	0	8436.016
72	2.01E+11	0	1	0.024	0.048	0	INVALID	1.6	229.5	0	INVALID	0	0	4241.712
73	2.01E+11	0	1	0	0.016	0	INVALID	12.8	229.6	12.8	INVALID	0	0	3193.136
74	2.01E+11	0	1	0.032	0.128	0.032	INVALID	2.4	229.6	1.6	INVALID	3.2	0	571.696
75	2.01E+11	0	1	0	0.032	0.016	INVALID	9.6	229.8	0.8	INVALID	1.6	0.096	3193.136
76	2.01E+11	0	1	0	0	0.128	INVALID	0	229.8	12.8	INVALID	1.6	0.064	8436.016
77	2.01E+11	0	1	0.032	0.096	0	INVALID	1.6	229.7	6.4	INVALID	0	0.008	47.408
78	2.01E+11	0	1	0	0	0	INVALID	0	229.9	3.2	INVALID	1.6	0	2144.56
79	2.01E+11	0	1	0.128	0.008	0	INVALID	1.6	229.8	12.8	INVALID	3.2	0	47.408
80	2.01E+11	0	1	0.032	0.064	0	INVALID	4.8	229.5	3.2	INVALID	0	0.064	47.408
81	2.01E+11	0	1	0	0.128	0	INVALID	6.4	229.6	3.2	INVALID	6.4	0	8436.016
82	2.01E+11	0	1	0	0	0	INVALID	0	230.2	0	INVALID	0	0	47.408
83	2.01E+11	0	1	0	0	0	INVALID	0	230.2	0	INVALID	0	0	47.408
84	2.01E+11	0	1	0	0	0	INVALID	0	230.2	0	INVALID	0	0	47.408
85	2.01E+11	0	1	0	0	0	INVALID	0	230.3	0	INVALID	0	0	47.408
86	2.01E+11	0	1	0	0	0	INVALID	0	230.2	0	INVALID	0	0	47.408
87	2.01E+11	0	1	0	0	0	INVALID	0	230.2	0	INVALID	0	0	47.408

图 6.7 本地存储表格数据

结束语

建筑物能耗监控系统主要包括电表、水表、燃气表的监控，三表合一能最大程度的节约通信与基础设施硬件成本，是远程监控管理的必然选择与方向。

本课题是在满足国家相关技术规定^[9]和导则规定的技术条件下，基于物联网和智能仪表架构一个综合性的建筑物能耗采集平台。通过物联网技术与嵌入式技术进行智能仪表（电表、水表、燃气表等）的现场数据采集、存储、远程传输、上位机监测管理等功能。通过 Modbus 总线、电力线、无线网络及互联网络等多种方式，实现远距离、全自动、高可靠的智能仪表监测。

由于本次课题是在相对较短的时间和有限的经济支持下进行的，因此还有很多方面尚未达到市场需求，需要进一步完善系统功能和深入研究开发更多适用的新功能。经过对目前产品所达到的功能以及市场对产品功能的需求，现提出以下几个方面对未来产品功能完善的设想：

（1）对数据进行融合分析，智能判断，准确找出问题点，采取节能降耗，以及安保管理措施。

（2）不断修正及完善系统，使智能电表将能无缝接入到局域网或广域网中，并可以基于电力载波进行信息传递，以促进智能仪表的通用化。

（3）在数据转发的过程中，需要对数据进行有效性的检验、杜绝错误数据上传。

（4）当遇到通信中断等异常情况时，数据采集器使用本地存储采集到的数据，在通信恢复正常之后进行断点续传，并实现错误上报功能。

（5）进一步实现多通道数据采集及网络传输上需要在系统底层上进行优化，简化嵌入式中间件及接口函数，并需完成数据软件防干扰设计、数据非线性优化及通信的有限自愈能力。

致 谢

本设计的功能设计及实现，以及论文的撰写都是在导师王文庆老师的悉心指导下完成的。同时感谢实验室杨老师和实验室提供实验设备和实验环境的支持，221 实验室精益求精的工作态度和极具创新的研究精神，深深的感染和鼓励着我，让我能竭尽所能的完成这项设计。感谢王文庆导师以及实验室老师给予我的细心教导和不懈支持，以及无微不至的帮助与指导。

同时还要感谢李赵欣和王明凯等同学的不懈帮助，感谢实验室同学在我学习时提供的帮助和指导。并感谢自动化 217、221 实验室，感谢实验室同学们的理解与配合。在功能的调试过程中，需要用到大量的实验设备，占用了他们的实验资源。在此感谢各位同学的支持与鼓励。

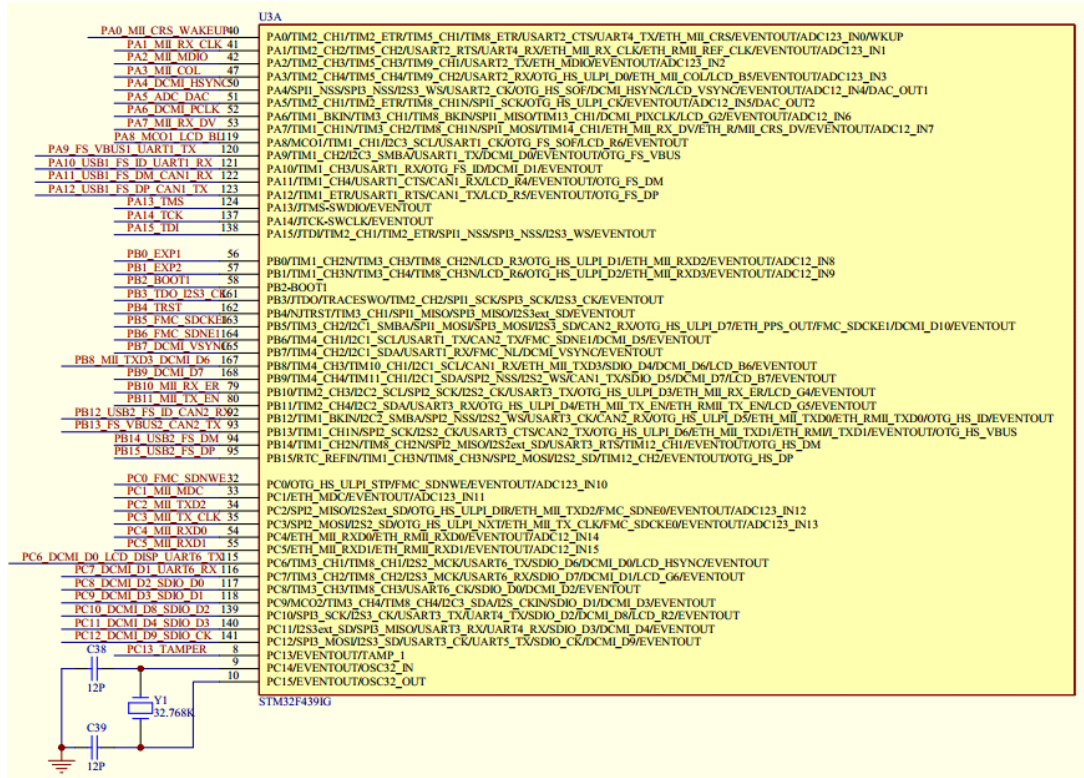
最后还要感谢我的母校——西安邮电大学。在此的大学四年，感谢能进入实验室进行学习与研究。同时还认识非常友好的朋友和老师，这都是我大学里最美好的回忆。

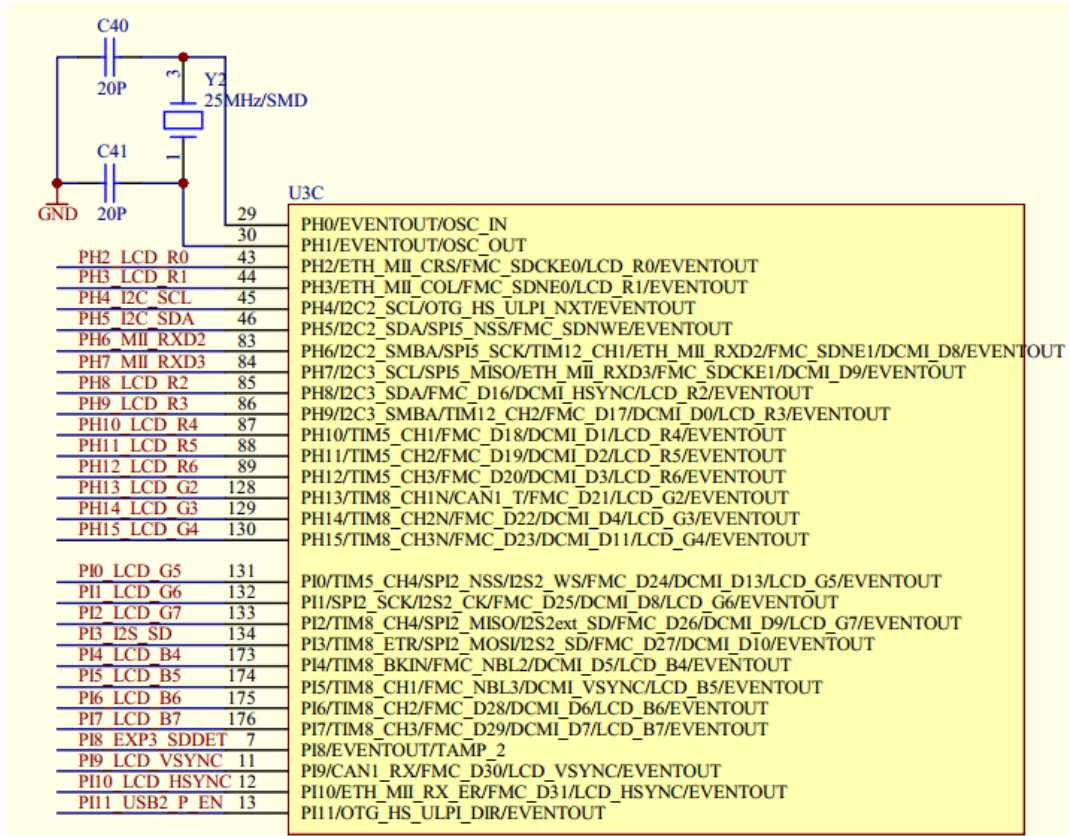
参考文献

- [1] SD Association. SD Specifications Part 1 Physical Layer Simplified Specification Version 4.10 [EB/OL]. www.sdcard.org, 2013
- [2] Chan. FatFs - Generic FAT File System Module ff10b[EB/OL].
http://elm-chan.org/fsw/ff/00index_e.html, 2014
- [3] 谢希仁. 计算机网络[M]. 电子工业出版社, 2008
- [4] Lindsay Bassett. JSON 必知必会[M]. 人民邮电出版社, 2016
- [5] W. Richard Stevens Bill Fenner Andrew M. Rudoff. UNIX 网络编程[M]. 人民邮电出版社, 2015
- [6] 姚文祥. ARM Cortex-M3 与 Cortex-M4 权威指南[M]. 清华大学出版社, 2015
- [7] 卡特索利斯(Catsoulis, J.). 嵌入式硬件设计[M]. 中国电力出版社, 2007
- [8] 中华人民共和国国家质量监督检验检疫总局. 基于 Modbus 协议的工业自动化网络规范 [M]. 中国标准出版社, 2008
- [9] 中华人民共和国建设部. 民用建筑能耗数据采集标准[M]. 中国建筑工业出版社, 2007
- [10] Richard Barry. Mastering the FreeRTOS Real Time Kernel A Hands-On Tutorial Guide[EB/OL]. www.freertos.org, 2016

附录

附录 1：电路原理图

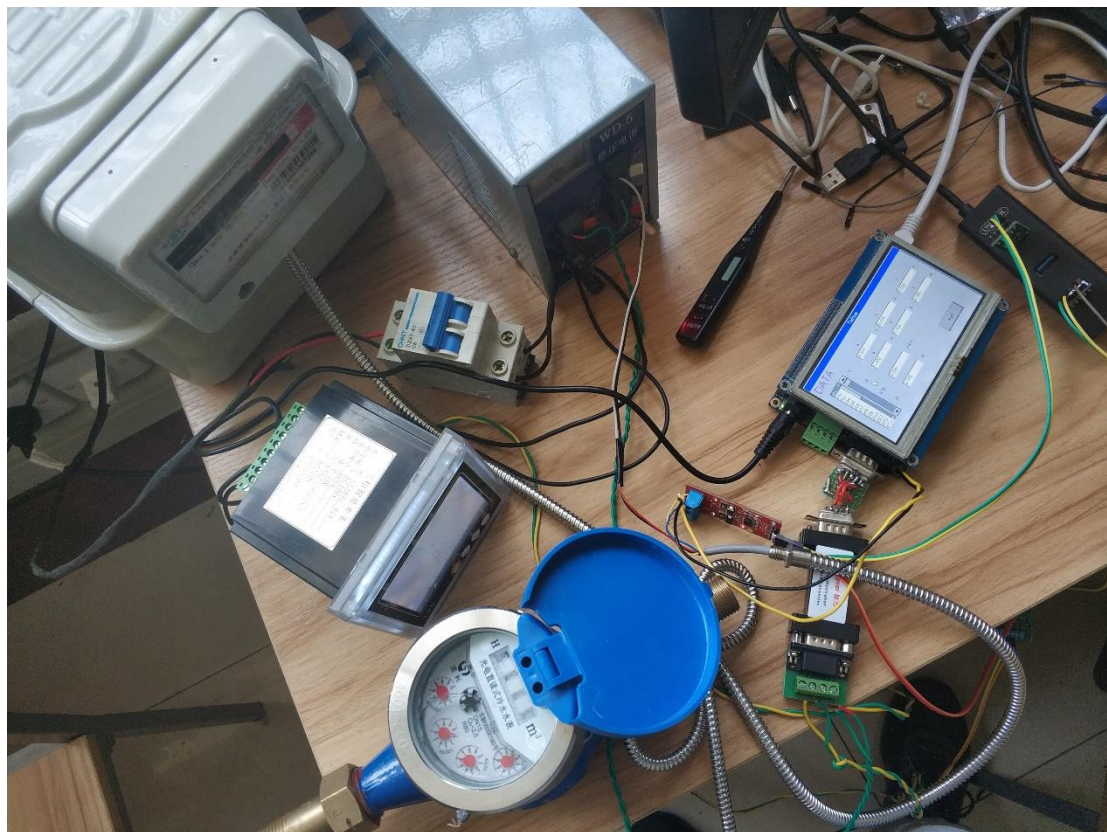




U3B		
PD0_FMC_D2	142	PD0/CAN1_RX/FMC_D2/EVENTOUT
PD1_FMC_D3	143	PD1/CAN1_TX/FMC_D3/EVENTOUT
PD2_DCMI_D11_SDIO_CMD	144	PD2/TIM3_ETR/UART5_RX/SDIO_CMD/DCMI_D11/EVENTOUT
PD3_DCMI_D5	145	PD3/SP2_SCK/I2S2_CK/USART2_CTS/FMC_CLK/DCMI_D5/LCD_G7/EVENTOUT
PD4_FSMC_NOE	146	PD4/USART2_RTS/FMC_NOE/EVENTOUT
PD5_FSMC_NWE	147	PD5/USART2_TX/FMC_NWE/EVENTOUT
PD6_FMC_NWAIT_DCMI_D10_I2S3_SD	150	PD6/SP3_MOS/I2S3_SSAI1_SD_A/USART2_RX/FMC_NWAIT/DCMI_D10/LCD_B2/EVENTOUT
PD7_NE1_REDE	151	PD7/USART2_CK/FMC_NE1/FMC_NCE2/EVENTOUT
PD8_FMC_D13	96	PD8/USART3_TX/FMC_D13/EVENTOUT
PD9_FMC_D14	97	PD9/USART3_RX/FMC_D14/EVENTOUT
PD10_FMC_D15	98	PD10/USART3_CK/FMC_D15/LCD_B3/EVENTOUT
PD11_FMC_A16	99	PD11/USART3_CTS/FMC_A16/EVENTOUT
PD12_FMC_A17	100	PD12/TIM4_CH1/USART3_RTS/FMC_A17/EVENTOUT
PD13_LED	101	PD13/TIM4_CH2/FMC_A18/EVENTOUT
PD14_FMC_D0	104	PD14/TIM4_CH3/FMC_D0/EVENTOUT
PD15_FMC_D1	105	PD15/TIM4_CH4/FMC_D1/EVENTOUT
PE0_FMC_NBL0	169	PE0/TIM4_ETR/UART8_Rx/FMC_NBL0/DCMI_D2/EVENTOUT
PE1_FMC_NBL1	170	PE1/UART8_Tx/FMC_NBL1/DCMI_D3/EVENTOUT
PE2_KEY_UP_INT	1	PE2/TRACECLK/SP4_SCK/SAI1_MCLK_A/ETH_MII_TXD3/FMC_A23/EVENTOUT
PE3_KEY_DOWN	2	PE3/TRACED0/SAI1_SD_B/FMC_A19/EVENTOUT
PE4_LCD_B0	3	PE4/TRACED1/SP4_NSS/SAI1_FS_A/FMC_A20/DCMI_D4/LCD_B0/EVENTOUT
PE5_LCD_G0	4	PE5/TRACED2/TIM9_CH1/SP4_MISO/SAI1_SCK_A/FMC_A21/DCMI_D6/LCD_G0/EVENTOUT
PE6_LCD_G1	5	PE6/TRACED3/TIM9_CH2/SP4_MOSI/SAI1_SD_A/FMC_A22/DCMI_D7/LCD_G1/EVENTOUT
PE7_FMC_D4	68	PE7/TIM1_ETR/UART7_Rx/FMC_D4/EVENTOUT
PE8_FMC_D5	69	PE8/TIM1_CH1N/UART7_Tx/FMC_D5/EVENTOUT
PE9_FMC_D6	70	PE9/TIM1_CH1/FMC_D6/EVENTOUT
PE10_FMC_D7	73	PE10/TIM1_CH2N/FMC_D7/EVENTOUT
PE11_FMC_D8	74	PE11/TIM1_CH2/SP4_NSS/FMC_D8/LCD_G3/EVENTOUT
PE12_FMC_D9	75	PE12/TIM1_CH3N/SP4_SCK/FMC_D9/LCD_B4/EVENTOUT
PE13_FMC_D10	76	PE13/TIM1_CH3/SP4_MISO/FMC_D10/LCD_DE/EVENTOUT
PE14_FMC_D11	77	PE14/TIM1_CH4/SP4_MOSI/FMC_D11/LCD_CLK/EVENTOUT
PE15_FMC_D12	78	PE15/TIM1_BKIN/FMC_D12/LCD_R7/EVENTOUT
PF0_FMC_A0	16	PF0/I2C2_SDA/FMC_A0/EVENTOUT
PF1_FMC_A1	17	PF1/I2C2_SCL/FMC_A1/EVENTOUT
PF2_FMC_A2	18	PF2/I2C2_SMB/AFMC_A2/EVENTOUT
PF3_FMC_A3	19	PF3/FMC_A3/EVENTOUT/ADC3_IN9
PF4_FMC_A4	20	PF4/FMC_A4/EVENTOUT/ADC3_IN14
PF5_FMC_A5	21	PF5/FMC_A5/EVENTOUT/ADC3_IN15
PF6_SAI1_SDB_SPI5_NSS	24	PF6/TIM10_CH1/SP15_NSS/SAI1_SD_B/UART7_Rx/FMC_NIORD/EVENTOUT/ADC3_IN4
PF7_SAI1_MCLKB_SPI5_SCK	25	PF7/TIM11_CH1/SP15_SCK/SAI1_MCLK_B/UART7_Tx/FMC_NREG/EVENTOUT/ADC3_IN5
PF8_SAI1_SCKB_SPI5_MISO	26	PF8/SP15_MISO/SAI1_SCK_B/TIM13_CH1/FMC_NIOWR/EVENTOUT/ADC3_IN6
PF9_SAI1_FSB_SPI5_MOSI	27	PF9/SP15_MOSI/SAI1_FS_B/TIM14_CH1/FMC_CD/EVENTOUT/ADC3_IN7
PF10_LCD_DE	28	PF10/FMC_INTR/DCMI_D11/LCD_DE/EVENTOUT/ADC3_IN8
PF11_FMC_SDNRA5	59	PF11/SP15_MOSI/FMC_SDNRA5/DCMI_D12/EVENTOUT
PF12_FMC_A6	60	PF12/FMC_A6/EVENTOUT
PF13_FMC_A7	63	PF13/FMC_A7/EVENTOUT
PF14_FMC_A8	64	PF14/FMC_A8/EVENTOUT
PF15_FMC_A9	65	PF15/FMC_A9/EVENTOUT
PG0_FMC_A10	66	PG0/FMC_A10/EVENTOUT
PG1_FMC_A11	67	PG1/FMC_A11/EVENTOUT
PG2_FMC_A12	106	PG2/FMC_A12/EVENTOUT
PG3_USB1_P_EN	107	PG3/FMC_A13/EVENTOUT
PG4_FMC_A14	108	PG4/FMC_A14/FMC_BA0/EVENTOUT
PG5_FMC_A15	109	PG5/FMC_A15/FMC_BA1/EVENTOUT
PG6_LCD_R7	110	PG6/FMC_INT2/DCMI_D12/LCD_R7/EVENTOUT
PG7_LCD_CLK	111	PG7/USART6_CK/FMC_INT3/DCMI_D13/LCD_CLK/EVENTOUT
PG8_FMC_SDCLKI12	112	PG8/SP16_NSS/USART6_RTS/ETH_PPS_OUT/FMC_SDCLK/EVENTOUT
PG9_FSMC_NCE3I52	152	PG9/USART6_RX/FMC_NE2/FMC_NCE3/EVENTOUT
PG10_LCD_B2	153	PG10/LCD_G3/FMC_NCE4_1/FMC_NE3/DCMI_D2/LCD_B2/EVENTOUT
PG11_LCD_B3	154	PG11/ETH_MII_TX_EN/ETH_RMII_TX_EN/FMC_NCE4_2/DCMI_D3/LCD_B3/EVENTOUT
PG12_LCD_B1	155	PG12/SP16_MISO/USART6_RTS/LCD_B4/FMC_NE4/LCD_B1/EVENTOUT
PG13_MII_TXD0	156	PG13/SP16_SCK/USART6_CTS/ETH_MII_TXD0/ETH_RMII_TXD0/FMC_A24/EVENTOUT
PG14_MII_TXD1	157	PG14/SP16_MOSI/USART6_TX/ETH_MII_TXD1/ETH_RMII_TXD1/FMC_A25/EVENTOUT
PG15_FMC_SDNCA560	160	PG15/USART6_CTS/FMC_SDNCA5/DCMI_D13/EVENTOUT

© 2013-2014

附录 2：实物图



附录 3 数据传输的 json 数据格式

```

char *add_eledata() ↓
{↓
char *p;↓
cJSON *pJsonRoot = NULL;↓
cJSON *data = NULL;↓
pJsonRoot = cJSON_CreateObject();↓
data = cJSON_CreateObject();↓
if(pJsonRoot == NULL || data == NULL)↓
DEBUG_PRINTF("*****error in makeJson*****");↓
↓
cJSON_AddNumberToObject(pJsonRoot, "ID", pEleMeterNode->meter_Id);↓
cJSON_AddNumberToObject(pJsonRoot, "Port", pEleMeterNode->Port);↓
cJSON_AddNumberToObject(pJsonRoot, "ewg", 1);↓
cJSON_AddItemToObject(pJsonRoot, "data", data);↓
cJSON_AddStringToObject(data, "currentA", (const char *)&meterData.currentA);↓
cJSON_AddStringToObject(data, "currentB", (const char *)&meterData.currentB);↓
cJSON_AddStringToObject(data, "currentC", (const char *)&meterData.currentC);↓
cJSON_AddStringToObject(data, "current", (const char *)&meterData.current);↓
cJSON_AddStringToObject(data, "voltageA", (const char *)&meterData.voltageA);↓
cJSON_AddStringToObject(data, "voltageB", (const char *)&meterData.voltageB);↓
cJSON_AddStringToObject(data, "voltageC", (const char *)&meterData.voltageC);↓
cJSON_AddStringToObject(data, "voltage", (const char *)&meterData.voltage);↓
cJSON_AddStringToObject(data, "activePower", (const char *)&meterData.activePower);↓
cJSON_AddStringToObject(data, "powerFactor", (const char *)&meterData.powerFactor);↓
cJSON_AddStringToObject(data, "activeEnergy", (const char *)&meterData.activeEnergy);↓
↓
p = cJSON_Print(pJsonRoot);↓
cJSON_Delete(pJsonRoot);↓
return p;↓
}↓
char *add_watdata() ↓
{↓
char *p;↓
cJSON *pJsonRoot = NULL;↓
cJSON *data = NULL;↓
pJsonRoot = cJSON_CreateObject();↓
data = cJSON_CreateObject();↓
if(pJsonRoot == NULL || data == NULL)↓
DEBUG_PRINTF("*****error in makeJson*****");↓
cJSON_AddNumberToObject(pJsonRoot, "ID", pmessage.serial);↓
cJSON_AddStringToObject(pJsonRoot, "add", (const char *)pmessage.add);↓
cJSON_AddNumberToObject(pJsonRoot, "ewg", 2);↓

```

```
cJSON_AddNumberToObject(pJsonRoot, "ewg", 2); ↓
cJSON_AddItemToObject(pJsonRoot, "data", data); ↓
cJSON_AddStringToObject(data, "counts", (const char *)pmessage.reading); ↓
cJSON_AddStringToObject(data, "unit", (const char *)&pmessage.unit); ↓
↓
p = cJSON_Print(pJsonRoot); ↓
↓
cJSON_Delete(pJsonRoot); ↓
return p; ↓
} ↓
```