



物联网核心技术丛书

物联网 实验教程

徐勇军 主 编

任 勇 徐朝农 安竹林 编 著



- ❑ 融中科院、清华大学等**20余家科研院所**多年的技术开发经验
- ❑ 从“**物联网三层架构**”体系出发，全面设计物联网技术的实验
- ❑ 包括**21项**单项技术实验、**5项**演示实验以及**7个**实际应用系统
- ❑ 涵盖RFID、ZigBee、WiFi、RF、自组网、异构网接入等技术
- ❑ 涵盖Android、Qt、安全、搜索、同步、定位、无线视频等技术
- ❑ 本书编写得到了“**感知中国**”**物联网联盟**的大力推动和支持



参考文档、实验源程序、实验环境及配置文档、实验手册等其他实验相关资料



机械工业出版社
CHINA MACHINE PRESS

物联网核心技术丛书

物联网实验教程

徐勇军 主编

任勇 徐朝农 安竹林 编著



机械工业出版社

本书从实验的角度介绍物联网相关技术, 针对物联网的感知与标识、通信与网络、接入与处理 3 个不同层面的技术, 精心设计了 21 项实验, 并提供了内容丰富的配套光盘(包括参考文档、实验源程序、实验环境及配置文档、实验手册等其他实验相关资料), 还结合智能医疗、智能物流、环境监测等领域的应用场景, 介绍了 5 个演示实验和 7 个应用系统, 为读者进一步开发物联网应用解决方案提供思路。最后, 针对物联网教学及研究实验室建设的实际需求, 特别是不同专业或行业背景, 提供了一些具体的参考建议, 希望能够对广大读者从事相关工作起到借鉴作用。

本书既可作为高等学校物联网工程、传感网工程等相关专业, 以及无线通信、网络技术、电子、微电子、单片机、嵌入式系统等课程的实验教材, 也可作为物联网应用系统工程开发人员的入门指导书。

图书在版编目(CIP)数据

物联网实验教程 / 徐勇军主编. —北京: 机械工业出版社, 2011.4 (2012.1重印)
(物联网核心技术丛书)

ISBN 978-7-111-34147-5

I. ①物… II. ①徐… III. ①互连网络—应用—教材②智能技术—应用—教材 IV. ①TP393.4②TP18

中国版本图书馆 CIP 数据核字(2011)第 070194 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)

责任编辑: 郝建伟

责任印制: 杨曦

保定市中国画美凯印刷有限公司印刷

2012 年 1 月第 1 版·第 2 次印刷

184mm×260mm·17.25 印张·426 千字

4001—6000 册

标准书号: ISBN 978-7-111-34147-5

ISBN 978-7-89451-940-5 (光盘)

定价: 43.00 元(含 1DVD)

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

电话服务

网络服务

社服务中心: (010) 88361066

门户网: <http://www.cmpbook.com>

销售一部: (010) 68326294

教材网: <http://www.cmpedu.com>

销售二部: (010) 88379649

读者购书热线: (010) 88379203

封面防伪标均为盗版

物联网核心技术丛书

编审委员会

(以姓氏笔画为序)

名誉主任 刘海涛

主任 马建

副主任 牛志升 杨志强 张宏科 周立军

委员 马晓东 马华东 牛建伟 王文东 王双全

孙利民 刘越 刘军 任勇 任丰原

朱珍民 吴健康 陈益强 陈灿峰 张宝贤

张林 郑军 罗海勇 赵立君 隆克平

钱跃良 徐勇军 梁久祯 彭力 阚志刚

秘书长 胡毓坚

出版说明

伴随工业化向信息化的发展，智能化管理和服务已进入快速发展的阶段，物联网正是在这样的背景下发展起来的新兴产业。物联网被预言其拥有万亿级市场应用前景，已被世界各国作为应对金融危机、振兴经济的重点领域。物联网的发展，将促进传统生产生活方式向现代方式转变，将大大提高生产力，并提升人们的生活质量，极大拓展信息网络和信息技术应用空间，并将成为我国加快培育发展战略性新兴产业的重要突破口和未来经济发展的重要引擎。

为了适应物联网技术发展的新趋势和新特点，以及新时期对物联网专业技术人才培养工作的需要，机械工业出版社邀请来自全国从事物联网相关研究和教学工作的单位，如无锡物联网产业研究院、中国科学院、北京邮电大学、清华大学、东南大学、电子科技大学、北京航空航天大学、江南大学等，以及产业界，如中国移动、国家电网、诺基亚等的首席科学家、校长、院长、资深研究员、教授、高工、研究及教学团队，成立了丛书编委会，共同策划了这套涵盖物联网核心技术的丛书，并且将陆续出版。在编写过程中，得到了无锡市新区管委会、国家传感信息中心管委会等政府机构的支持和指导，特别是得到了“感知中国”物联网联盟的大力推动和支持。

这套丛书将力求做到：内容全面、客观、公正、系统，对物联网产业的发展具有一定的指导意义，有助于促进物联网知识的普及和物联网产业的健康有序发展。通过讲述物联网的概念、作用、本质和发展现状，将给人民大众的工作、生活和学习带来革命性的影响。丛书的出版将对物联网相关产业发展规划的制定、物联网应用的推广、物联网知识的普及起到有益的促进作用，将使更多的人通过了解物联网进而投身物联网的研究与应用，从而助力于推动物联网技术及产业在中国的发展。

这套丛书的选题是开放式的。随着物联网技术日新月异的发展，我们将不断更新和补充选题，使这套丛书及时反馈物联网领域的新发展和新技术。我们也欢迎在科研、生产和教学第一线有丰富经验的科技人员和教师积极参与这项工作。

物联网技术发展迅速而且涉及领域非常宽，这套丛书的选题和编审如有缺点和不足之处，诚恳希望各位读者提出宝贵意见，以利于今后不断改进。

机械工业出版社

前 言

2009年8月7日，温家宝总理在江苏无锡提出“加快推进传感网发展”、“尽快建立中国的传感信息中心”，后被称为“物联网”的这一科技新宠迅速引起了社会各产业界、学术界的高度关注和广泛共鸣。人们开始从各自的专业领域和实际需求，演绎出了形形色色的“物联网”。而其无处不在的普适性，也给物联网的高速发展，以及向各行各业的渗透铺平了道路。虽然物联网诞生不久，但其概念已经深入人心，人们开始探讨物联网相关的理论、技术、标准、应用等一系列问题。实际上物联网还是太年轻，其概念、内涵在科技界和工业界都没有达成共识，让我们在这么短的时间内，用科学实验这么具体的方式来展示物联网，这是一个巨大的挑战，我们确实感到战战兢兢、如坐针毡。

经反复讨论，编写团队整理了大量和物联网（特别是无线传感器网络）相关的实验。考虑到可能作为广大读者迈入物联网实验神圣殿堂的第一步，并成为对物联网相关技术进行深度研究的基石，本书首先着重介绍相关开发环境建立、软件开发、程序调试、实验原理等相关基础知识，然后立足当前国内外厂商的主流芯片及实现方案，以物联网不同技术层面的实验原理为主线，精心设计20余项实验，并将具体的实验过程、实验代码等通过配备内容丰富的配套光盘呈现出来，以在有限的篇幅内，展示更多的实验技术。另外，本书还结合智慧医疗、智能物流、智能家居等领域，编制了5个演示实验和7个应用系统的介绍，可以进一步加深读者对物联网应用的认知，减少用户从理论到实践的周期。通过对本书的部分或全部实验进行适当选修，可以让学生或工程技术人员在最短的时间内，深入理解物联网各层面的知识，同时也为物联网从实验室走向工程应用提供参考。

为了能够尽量全面展示物联网不同层面的技术核心，本书为物联网的感知与标识、通信与网络、接入与处理3个不同的技术层面设计了传感器技术实验、RFID技术实验、短距离无线通信实验、无线自组网实验、IEEE 802.15.4实验、ZigBee网络实验、公网接入实验、网络安全通信实验、室内实时定位实验、自组网时间同步实验及数据索引及搜索实验等20多个实验。为了便于读者系统掌握，本书把所有实验聚焦到单片机、嵌入式及智能终端等3大平台，并详细给出了各自平台环境的搭建方法和基础实验，以供读者入门以及进一步提高。另外，本书还结合智能医疗、智能物流、环境监测等领域的具体应用需求，给出了基于物联网的一些综合演示实验以及应用演示系统，为读者开发物联网应用解决方案提供思路。最后针对高校及企业建立物联网教学及研究实验室，提供了一些具体的建议，希望能够起到抛砖引玉的作用。

本书由中国科学院计算技术研究所徐勇军组织编写，清华大学任勇为本书相关实验的设计进行了大量的指导，并完成了第6章和第8章相关内容的编写；中国石油大学徐朝农与中国科学院计算技术研究所团队密切合作，完成了书中物联网硬件环境的开发；中国科学院计算技术研究所安竹林为实验内容的精选与整理进行了大量的工作。由于本书涉及的软硬件平台较多（包括单片机平台、嵌入式平台和智能终端平台）、实验开发工作量大、知识结构跨度

较宽(包括感知层、网络层和应用层),争取到了团队内外诸多专家、老师及同学的大力支持。这里包括中国科学院计算技术研究所罗海勇、林以明、陈彦明、姜鹏、王鹏、李培培、赵登昌、王飞、吴琳等,在计算所联合培养的北京师范大学李超、吕芸帙,中国石油大学于德龙,武汉理工大学汪三青、付小丽、昆明理工大学尹喜等。虽然最终选进教材的实验只有 20 余项,但我们实际开发的实验远远不止这些。限于篇幅的原因,很多很好的实验没有被选进本书,我们将通过其他途径同希望进一步了解物联网实验的读者共享。

本书还得到了其他相关企业、高校的大力支持。诺基亚(中国)研究院陈灿峰博士在智能终端平台方面,中国联通研究院张云勇博士在智能家居系统方面,中国电子器材总公司赵惟副总经理和郭达博士在城市防洪系统方面,国土资源部地质环境监测研究院周平根教授在地质监测系统方面,中国人民解放军某部张杰博士、中国科学院软件研究所朱红松博士在相关应用演示系统方面,灵玖中科软件(北京)有限公司技术总监于满泉和刘治华博士在智能搜索方面,艾威梯科技(北京)有限公司卫俊平博士智慧医疗方面,深联致远(北京)科技有限公司的李镜华、刘峰、黄建英工程师在 RFID 实验方面,北京奥尔斯电子科技有限公司张鑫工程师在智能家居方面,北京师范大学李朱峰博士在实验室建设方面,北京理工大学张华平博士和祝烈煌博士在海量智能搜索和网络安全方面都付出了相当多的工作,中国科学院计算所传感器网络实验室主任崔莉研究员也为本书的撰写提供了很多支持和建议,还有很多没有一一提到的领导、朋友、同事及同学,他们在这艰苦的几个月中,为本书的顺利出版付出了辛勤的汗水,在此一并表示由衷的感谢。

本书作为《物联网核心技术丛书》中的一本,与其他各本都有密切联系。通过本书相关实验的动手操作,可以加深读者对《物联网技术概论》中与物联网概念及内涵、体系结构、物联网关键技术的理解,也可以进一步加深读者对物联网的网络协议及其实现的原理和方法等内容的理解。另外,本书第 6 章和第 7 章结合若干行业应用,从演示方案的角度与《物联网技术应用》进行呼应。本书内容还涉及传感器技术、RFID 技术、无线传感器网络、短距离无线通信、物联网安全、物联网定位等内容,这些都将与相关丛书形成呼应。可以说,《物联网实验教程》是丛书的一个映射,也是整个物联网领域的一面镜子。

物联网具有很强的实践性,要想熟练地掌握这些实验技术,实验操作是非常重要的环节。本书所涉及的所有硬件平台、软件平台、软件开发环境、软件源代码等都将在光盘中一并提供。我们专门开设了新浪博客(<http://blog.sina.com.cn/wulianlab>),并设立了专业的网站(<http://www.wulianlab.cn/>),对书中的具体问题和读者在实际操作中的问题进行解答,同时还随时更新实验内容以及可能发生的错误或不完善的地方。另外,读者还可以通过电子邮件(wulianlab@gmail.com)与我们沟通。物联网技术是新兴的技术,将在近几年获得飞速发展,本书也将随时更新相关内容,以与最新技术保持同步,使读者能够循序渐进地掌握物联网各层次的开发技术,为以后更深层次的理论研究和应用实践打下坚实的基础。

需要指出的是,物联网领域的研究尚处于起步阶段,许多技术还不成熟,其硬件平台和技术方案也千差万别,因此要编写一个全面完善的实验教程非常困难。

由于水平有限,书中疏漏甚至错误之处在所难免,希望广大读者批评指正。

编 者

目 录

出版说明

前言

第 1 章 物联网实验概述	1
1.1 物联网的诞生	1
1.2 物联网的内涵	2
1.3 实验设计思路	3
1.3.1 关键技术简介	3
1.3.2 本书实验设计	4
1.4 本书结构安排	6
第 2 章 物联网基础实验平台	8
2.1 单片机实验平台	8
2.1.1 硬件环境简介	8
2.1.2 软件环境简介	10
2.1.3 操作系统简介	12
2.1.4 实验一 单片机平台基础实验	13
2.2 嵌入式实验平台	19
2.2.1 硬件环境简介	19
2.2.2 软件环境简介	22
2.2.3 操作系统简介	22
2.2.4 实验二 嵌入式平台基础实验	24
2.3 智能移动终端平台	34
2.3.1 主流移动智能终端平台简介	34
2.3.2 实验三 智能移动终端平台基础实验	37
第 3 章 物联网感知与标识实验	44
3.1 感知与标识实验概述	44
3.2 典型传感器感知实验	44
3.2.1 实验四 A 单片机平台高精度温湿度传感器实验	45
3.2.2 实验四 B 嵌入式平台高精度温湿度传感器实验	51
3.2.3 实验五 A 单片机平台运动感知实验	59
3.2.4 实验五 B 嵌入式平台运动感知实验	64
3.2.5 实验六 图像采集实验	69
3.2.6 实验七 步进电动机实验	76

3.2.7	实验八 键盘与显示屏实验	79
3.2.8	实验九 嵌入式平台基于 MiniGUI 的简易编辑器实验	85
3.3	条形码与 RFID 标识实验	94
3.3.1	条形码及 RFID 技术概述	94
3.3.2	实验十 高频 RFID 读/写实验	98
3.3.3	实验十一 超高频 RFID 读/写实验	104
第 4 章	物联网通信与网络实验	112
4.1	通信与网络实验概述	112
4.2	短距离无线通信实验	112
4.2.1	短距离无线通信规范简介	112
4.2.2	实验十二 433MHz 无线通信实验	115
4.2.3	实验十三 2.4GHz 无线通信实验	122
4.3	无线自组网实验	128
4.3.1	无线自组网技术简介	128
4.3.2	实验十四 433MHz 低功耗自组网实验	129
4.4	IEEE 802.15.4 实验	140
4.4.1	IEEE 802.15.4 协议以及 TinyOS 简介	140
4.4.2	实验十五 IEEE 802.15.4 协议通信实验	143
4.5	ZigBee 网络实验	153
4.5.1	ZigBee 协议简介	153
4.5.2	实验十六 ZigBee 协议组网通信实验	156
第 5 章	物联网接入与处理实验	166
5.1	接入与处理实验概述	166
5.2	公网接入实验	167
5.2.1	通信网接入简介	167
5.2.2	实验十七 Android 系统下互联网接入实验	167
5.3	网络安全通信实验	173
5.3.1	椭圆曲线及群组密钥协商方法简介	173
5.3.2	实验十八 基于椭圆曲线的群组密钥协商协议通信实验	175
5.4	室内实时定位实验	181
5.4.1	WiFi 网络协议简介	181
5.4.2	实验十九 基于射频指纹匹配的室内实时定位实验	182
5.5	自组网时间同步实验	189
5.5.1	无线传感器网络时间同步简介	190
5.5.2	实验二十 TPSN 时间同步实验	191
5.6	数据索引及搜索实验	196
5.6.1	基于倒排索引的全文搜索技术简介	196
5.6.2	实验二十一 数据库全文精准搜索实验	198
第 6 章	物联网演示实验	204

6.1	演示一 环境监测演示实验	204
6.2	演示二 出入管理演示实验	211
6.3	演示三 智能家居演示实验	214
6.4	演示四 智能医疗演示实验	219
6.5	演示五 智能抄表演示实验	223
第 7 章	物联网应用示范系统	227
7.1	系统一 智能远程医疗急救系统	227
7.2	系统二 城市防洪应急指挥系统	232
7.3	系统三 智能家居示范系统	238
7.4	系统四 智能闸口示范系统	242
7.5	系统五 海量地址匹配与搜索系统	248
7.6	系统六 室内位置服务示范系统	250
7.7	系统七 地质灾害监测预警系统	257
第 8 章	物联网实验室建设	261
8.1	物联网实验室环境建设	261
8.2	物联网实验室实验内容	262
8.3	相关课程和预备知识	265

第1章 物联网实验概述

1.1 物联网的诞生

人类自诞生之日起，就在不断地认识世界和改造世界，而工具的使用是其发展水平的显著标志。随着社会生产力的发展，人类不断通过提升科学技术的水平来拓展自己的视野，这集中表现为创造出越来越多的工具，并有效地利用这些工具为人类社会服务。随着近几十年来信息技术，特别是网络技术的飞速发展，人类获取、保存、处理及传播信息的能力获得了迅猛的提升，从此目光变远了，触觉更灵敏了，地球也变“小”了。人类的生理功能获得了大幅度延伸，开始更从容地去面对自然。

以计算机为代表的计算世界（Cyber World）在物理世界（Physical World）和人类社会（Human Society）当中扮演着重要角色，它们已经被合称为三元世界（Ternary Universe）。第一台电子计算机 ENIAC（Electronic Numerical Integrator And Computer）诞生时，计算机是一个庞然大物，是极少数人能享用的“奢侈品”。随着 CMOS 半导体电路技术按摩尔定律的速度发展，计算机系统正在朝着功能不断增强，可靠性、集成度、能源效率、信息密度不断提高，用户界面更加友好的方向演进。一方面出现了具有超强计算能力和存储能力的超级计算机，另一方面也开始出现体积小、成本低、便于人机交互的各种便捷计算设备，信息技术开始进入社会生产和日常生活，开始为传统产业提供更方便快捷的强大支撑。

借助计算机这一有力工具，人类对客观物理世界的认知和利用大概经历了以下几个阶段。

1) 数字化阶段。随着传感器的规模化应用，物理世界的信息通过采样、量化、编码，可以变成在计算机中表示、存储和处理的数据，传感器拉近了人们和物理世界的距离。

2) 信息化阶段。随着处理和存储能力的增强，人们所积累的原始数据及加工数据逐步增多、种类更加丰富、数据更加实时，信息开始逐步脱离客观的物理世界，并成为人们生产和生活中的重要资料，计算世界开始独立于物理世界而形成。

3) 网络化阶段。随着大量信息日复一日地被创造出来，信息也开始成为人们交互的不可或缺的物质资料，甚至开始成为重要的商品。网络化不但为人们相互交换信息、实现信息共享提供了便利，更通过大量分布式的信息采集和处理，进一步扩展了信息获取的途径。网络化让更多人加入到了信息的交互中，与信息相关的产品和服务也越来越多。

4) 泛在化阶段。随着信息和网络不断发展成人类社会生活中必不可少的物质基础，人们随时随地都需要和大量数字化信息打交道，网络化达到无时无处不在的程度，人们再也无法回避信息化的生活方式，人类社会也将迈入知识信息的泛在化阶段，而海量的信息也为计算世界的智能化提供了可能，智能化让人类社会和物理世界融为一体。无线传感器网络和物联网在网络化和智能化的背景下应运而生！



无线传感器网络（Wireless Sensor Network, WSN）简称传感网，它在人类与自然界之间架起了一座桥梁，是人类借助信息化工具认知物理世界的新式武器。传感网综合了信息感知、信息处理、信息传输等功能的分布式自组织的计算系统，传感网节点自组织地以多跳的方式完成互联与协同传输处理，并通过网关节点与互联网、移动通信网等公网连接，从而实现远程信息的获取和计算。传感网的出现，使信息获取的能力得到大幅度提高，多传感器协同感知和传感数据的信息融合则帮助人们克服了传统传感器感知区域小等缺陷，可视为人类感官的延伸，因而被称为“神经末梢”。

2009年8月7日，温总理在江苏无锡谈到太湖水质监测时，首次提出了“感知中国”的概念，从此“物联网”开始为普通百姓所熟知。“物联网”通过物与物之间的智能互联，将人们认识自然的水平提升到一个崭新的阶段。传统的互联网和移动通信网均是以信息为中心，仍然是信息网络；而物联网是直接以人的需求为中心的网络，通过各种“神经末梢”与物理世界相连，从而达到了无时无处不在的感知，是一种更为体贴、更为友好的网络。物联网并不仅仅是一项技术，也是人类自身认识世界的需要，它不但拥有射频识别（Radio Frequency Identification, RFID）标签、传感器以及传感网等载体的感知触角，而且拥有互联网、移动通信网等传输中枢，还拥有海量信息存储与处理的云计算及智能信息处理等系统支撑，通过综合使用各种信息技术手段，完成对物理世界真正意义上的智能感知，从而最大限度地丰富和提高了人类通过技术手段对于自然界的认知能力、分析综合判断能力和决策支持能力。

物联网是人类认识世界理想化的新境界。这与IBM公司在2009年初提出的“智慧的地球（Smart Planet）”以及Nokia公司所提出的“情境感知计算（Context-aware Computing）”等概念有着异曲同工之妙。虽然是从不同的角度出发提出的，但殊途同归，都是将简单的信息处理提升到人类社会与物理世界的融合上，而不再是纯粹的以具体技术升级为背景。“物联网”的提出给了我们一个重要的信号，技术水平的提升最终是为人类自身服务，其最高境界是让人不再察觉到信息技术手段的介入过程，信息技术“潜移默化”地融入人们的生活中，在丝毫不被察觉的情况下完成使命，从而在真正意义上实现了人类对自然界丰富信息的随心所欲的感知，从最大程度上消除了人与自然的距离，达到人与自然的和谐共生。

1.2 物联网的内涵

关于物联网的概念，在学术界和产业界都没有统一的定义。早期的物联网是指依托于射频识别（RFID）技术的物流网络。然而，随着技术和应用的发展，物联网的内涵远远超出这一范畴。现阶段，物联网是指在物理世界的实体中部署具有一定感知能力、计算能力和执行能力的各种信息传感设备，通过网络设施实现信息传输、协同和处理，从而实现广域或更大范围的人与物、物与物之间的信息交换和互联。基于以上物联网的内涵，学术界基本一致认可物联网包括“感知、网络、应用”3个基本要素，即将物联网划分为感知（互动）、网络（传输）、应用（服务）3个层次（如图1-1所示）。

1）感知（互动）层主要实现智能感知功能，包括信息采集、捕获、识别，并对信息进行初步的融合等功能。其关键技术包括传感器、执行器、RFID、自组织网络、短距离无线通信、低功耗路由、数据融合等，它是物联网的智能前端，也是神经末梢。

2）网络（传输）层主要实现信息的传输和交换，又包括接入层和核心层两个子层。网



网络层可依托公众电信网和互联网，也可以依托行业专用网络，还可以同时依托于二者，是多种网络融合后的大网。网络层构建在强大的基础设施之上，提供四通八达的信息高速公路，将海量的感知信息进行全面的共享。

3) 应用(服务)层则主要包含各类应用服务，例如监控服务、智能电网、工业自动化、绿色农业、智能家居、环境保护、公共安全等应用和服务。应用层还包括了中间件层，主要实现了网络层与物联网应用服务间的接口和功能调用平台，包括对业务的分析整合、信息共享、智能处理、网络管理等，具体体现为一系列业务支撑平台、管理平台、信息处理平台、智能计算平台、信息服务平台等。

通过感知层、网络层和应用层的相互作用，物联网可以实现物理世界与人类应用需求的联通，可以实现真正意义上的物物互联。

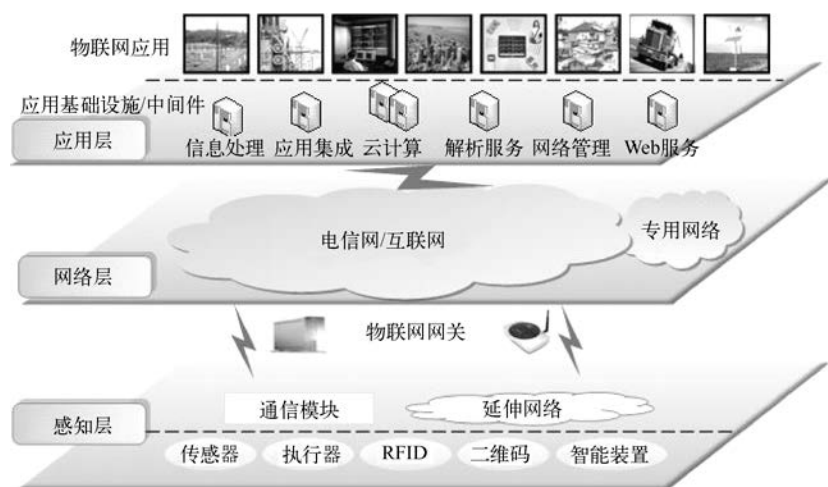


图 1-1 物联网网络架构示意图

1.3 实验设计思路

物联网内涵丰富，涉及计算、自动化通信、控制、微电子、微机电、网络等诸多领域，几乎囊括了信息技术及相关领域的各个方面，而且通过与具体应用背景的结合，物联网更是深入到社会生产和生活的每个角落，并强势渗透到各行各业。这里首先简单介绍物联网的关键技术，然后给出本书系列实验及系统的设计思路，让读者对物联网实验及应用技术有一个宏观的了解，也为读者进一步选修其中的实验内容进行准备。

1.3.1 关键技术简介

物联网的感知层、网络层、应用层均有很多核心关键技术，这也是本书努力通过实验进行展示的主要目标。由于纷繁复杂的物联网技术在丛书中的其他书中有详细描述，下面仅选择与本书实验相关的内容进行简单介绍。

1) 感知层：主要包括条码、二维码、RFID、传感网等内容，是与物理世界进行交互的



触角。在这一层，材料、工艺（生产和加工）是困扰各项技术发展的重要瓶颈，特别是高精度、高灵敏度、高可靠性或专业领域的传感器，以及传感网及 RFID 芯片工艺，我国和国外在其设计、生产、测量等方面，也有相当大的差距，需要突破的技术内容还很多。感知层也包括了控制器和执行器，特别是面向工业应用，这要求有较高的实时性、可靠性、安全性，通过上层网络反馈控制信息，完成与自然界的协同交互。另外，与传感网相关的技术很多，包括各层无线通信协议、网络安全、时间同步、定位、服务质量保证等。我国在这方面的研究较多，也取得了较多的成果，很多高校和研究所在这些领域都有一些特色技术，但很多研究都是以仿真为主，缺少实际实验的验证。

2) 网络层：包括互联网、通信网等骨干网络，也包括各领域定制的专用网络，实现海量信息的传送和通信（也有人将传感器网络放在这一层）。在互联网方面目前的研究已经从 IPv4 网络逐步过渡到 IPv6 网络，并开始进入后 IP 时代，对于下一代新型网络的研究也成为各国研究界争相占领的高地。通信网络是物联网信息传递和服务支撑的重要基础设施，涵盖传统的接入网、核心网和业务网等多个层面的内容，通信技术、频管技术、异种异构网络融合与接入技术等都是该层当前的研究热点。

3) 应用层：包括各类通用或定制的服务，有基于行业应用的，也有基于计算、存储、通信、网络等资源的服务。海量感知信息的计算与处理是物联网的核心支撑，而“云计算”正是这一层的重要解决方案。应用层重点研究面向服务且支持节能和安全的智能化存储体系、支持云存储等存储服务的架构、动态数据对象管理和资源共享、存储服务等，为物联网提供核心的计算环境。在这里，支持海量感知数据的中间件和智能搜索技术是众多服务平台的支撑技术，也是各大厂商争相占领的技术制高点。

物联网实验的目标就是基于当前通用的软硬件技术平台，通过实验全面地展现这些技术。不但将各个技术点逐一分解到不同的软硬件实验系统中，让初学者可以一步步实现出来，而且从技术发展的角度，给出可扩展的系统方案，方便用户进一步开发和完善，在此基础上开发出性能更好、实用性更强的特色技术和产品。这是理想目标，也是本书努力的方向。但是，由于实验室环境的条件受限，并且考虑到时间、成本、复杂度等限制，物联网的实验设计只能在这些技术中进行适当筛选，将那些能够满足这些因素的实验纳入本书中。

1.3.2 本书实验设计

为了能够让读者在较短的时间内就能完成物联网实验的入门，本书不太可能将各层的所有技术都一一通过实验来呈现，而是在物联网的不同层次选取一些有代表性的核心技术进行集中展示，所以后续章节的名称并没有与物联网的层次进行对应，而是为了清晰起见，本书章节以技术内容来命名，其他部分可以通过读者反馈进行添加。物联网实验分布如图 1-2 所示。

1) 在感知层，主要涉及智能感知和标识相关的技术，包括各类传感器实验（高精度温湿度传感器实验、运动感知实验、图像采集实验、步进电动机控制实验、键盘与显示屏交互实验）、RFID 实验（高频及超高频无源 RFID 实验）。

2) 在网络层，主要涉及无线通信和网络协议的设计。由于骨干网络环境复杂，在实验室呈现的难度及成本较高，本书主要以短距离网络为例来设计实验，包括短距离无线通信实验（433MHz、2.4GHz）、无线自组网实验、IEEE 802.15.4 及 ZigBee 协议实验等，可以从原理及概念上让读者理解网络协议的设计及实现。



图 1-2 物联网实验分布

3) 在应用层, 主要涉及网络接入和智能处理的技术。虽然应用层的核心在于中间件和服务平台, 但我们在这方面的开发研究工作正在进行, 由于本书编写时间仓促, 就没有以实验的形式展现, 我们将在后续的版本中进一步展现相关工作。本书在应用层的实验包括公网接入实验、安全网络通信实验、室内实时定位实验、自组网时间同步实验、海量数据搜索实验等。由于这些应用服务在实验中涉及较多的数据及网络资源, 本书在选取相关内容时也进行了一定的简化。应用层所涉及的内容较多, 特别是在行业领域的应用, 本书特别选取了若干典型行业领域, 对其物联网的示范方案进行阐述, 有实际需求或者学有余力的读者可以根据实验内容自行搭建, 无论是对应用开发还是学习理解都有非常大的帮助。

由于上述实验可能涉及的硬件平台、软件平台、实验环境种类繁多, 为了不至于让读者由于理解大量实验平台细节而导致对技术原理的理解和消化吸收, 本书将这些实验聚焦到 3 个实验平台上, 包括单片机平台、嵌入式平台及智能终端平台, 本书所有实验都分布在这 3 类平台上, 如图 1-3 所示。由于硬件环境的要求, 部分实验可能会跨越多个平台, 但只要掌握了这 3 个平台, 就可以设计出丰富多彩的实验和系统, 而不局限于本书内容。

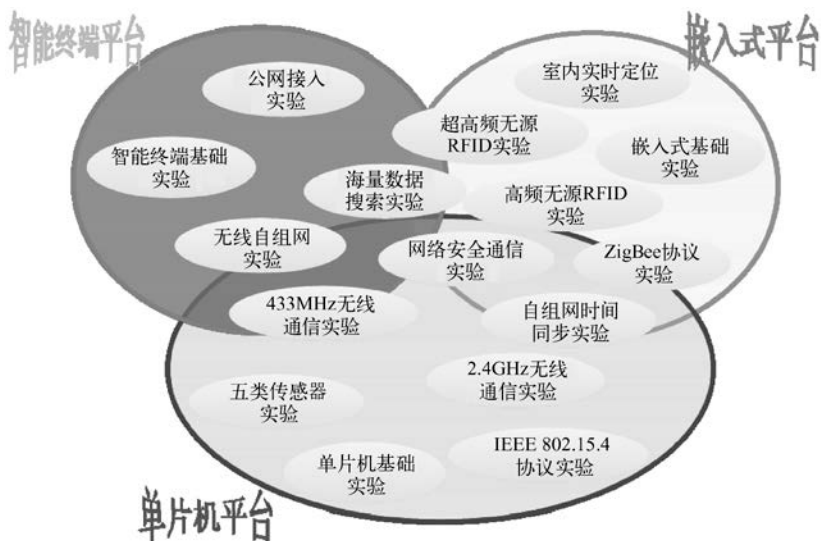


图 1-3 物联网 3 类实验平台及实验分布



本书的全部实验均在一系列的硬件设备上完成，主要硬件设备如表 1-1 所示。

1) 单片机实验平台是由单片机实验母板外加一系列传感器及外围模块组成的。

2) 嵌入式实验平台是由 ARM 实验母板和传感器及外围模块组成的。

3) 智能终端平台，包括 QT 及 Android 两个软件平台，可以在个人计算机 (PC) 上直接通过软件仿真实现，当然也可以下载到支持这些平台的智能终端上 (本书中的相关实验已经在相关智能手机上进行验证)。

表 1-1 中的硬件 4~7 是分别在单片机实验平台和嵌入式实验平台的基础上开发的，只是在体积和封装上做了一些改变，以方便进行实验部署，降低功耗及成本。

表 1-1 本书主要硬件设备介绍

	英文名	中文名	设备描述	相关实验
1	IOT-SCMMB	单片机实验母板	ATmega128 单片机实验母板，具备 IOT-MODULES 所有模块的硬件接口 (包括十多种传感器、5 种射频模板、键盘与液晶等)，另外还配有丰富的编程及调试接口，方便进行二次开发	实验一、四 A、五 A、六、七、八、十二、十三
2	IOT-ARMMB	ARM 实验母板	ARM7 实验母板，很容易扩充到 ARM9 及以上性能芯片，具备 IOT-MODULES 所有模块的硬件接口 (包括十多种传感器、5 种射频模板、键盘与液晶等)，IOT-SCMMB 实验均可在此平台上完成，另外还配有丰富的编程及调试接口，方便进行二次开发	实验二、四 B、五 B、九
3	IOT-MODULES	传感器及外围模块	各种传感器及外围设备，本书中使用的包括高精度温湿度传感器、三轴加速度传感器、图像传感器、步进电动机、433MHz 和 2.4GHz 通信模块、键盘与液晶等模块	实验一、二、四 A、四 B、五 A、五 B、六、七、八、九
4	IOT-RFID	RFID 实验硬件	RFID 实验设备，主要包括高频 (HF) 和超高频 (UHF) 无源 RFID 实验硬件、2.4GHz 有源 RFID 实验硬件、RFID 读/写手持设备及相关编程及调试硬件	实验十、十一
5	IOT-NODE433	433MHz 节点硬件	ATmega128 处理器、CC1000 无线收发，可用于进行单跳通信实验、自组织网络实验及各层次短距离无线通信实验开发，全面支持 TinyOS 1.x 及以上版本	实验十二、十四
6	IOT-NODE24	2.4GHz 节点硬件	ATmega128 处理器、CC2420 无线收发，可全面支持 ZigBee 协议、IEEE 802.15.4 协议开发，全面支持 TinyOS 1.x 及以上版本	实验十三、十五、二十
7	IOT-ZBJ	ZigBee 协议实验硬件	JN5139 ZigBee 片上系统级实现方案，主要用于快速学习开发 ZigBee 协议栈，并组建 ZigBee 网络	实验十六
8	IOT-NODE24T	2.4GHz 增强节点硬件	MSP430 处理芯片、CC2420 无线收发，集成度高，处理能力较强，可支持 ZigBee 协议开发，支持 TinyOS2.x	实验十八
9	IOT-WIFI	低功耗 WiFi 实验硬件	GAINSPAN 公司的 GS10xx 低功耗 WiFi 平台，可以方便用于低功耗 WiFi 协议的开发，本书采用基于低功耗 WiFi 的室内实时定位实验	实验十九

1.4 本书结构安排

本书的第 1 章主要介绍本书总体情况，包括物联网基本概念、物联网关键技术的介绍，还包括了本书的实验在物联网不同层次、不同实验平台上的分布情况。对于本书在关键技术和实验设计基础知识方面，读者可以根据需要在学习实验前深入选读。

第 2 章介绍本书所涉及的单片机、嵌入式及智能终端平台，每一部分都讲述了各平台的基础知识，包括硬件、软件、开发环境相关知识，最后针对本书中所使用的平台，设计了专门的涉及平台方面的基础实验。本章实验是全书的基础，这些实验基本都是在这 3 个平台上，通过对硬件的组合和增强、软件及协议的修改和开发来实现的。



第3章是物联网感知与标识实验。首先介绍物联网感知层的一些基础知识,然后从传感器和RFID两个方面设计了两组实验。传感器实验包括高精度温湿度传感器实验、运动感知实验、图像采集实验、步进电动机控制实验、键盘与液晶显示交互实验;RFID实验包括高频及超高频无源RFID实验。由于有源RFID的实验类似于后面的点到点无线通信,这里不再赘述。本章实验内容较多,读者可根据专业及项目需要进行选修。

第4章是物联网通信与网络实验。本章集中介绍与通信和网络相关的实验,首先是在ISM频段选择了433MHz和2.4GHz两个典型短距离通信实验,较详细地介绍了实现点到点无线通信的基本方法,然后基于这两个频段设计了无线自组网实验、IEEE 802.15.4及ZigBee网络实验。由于每部分还要涉及相当多的基础知识,建议读者适当结合丛书的其他内容和专业学术论文进行深入学习,并适当结合思考题进行深化学习。

第5章是物联网接入与处理实验。本章基于第4章的通信和网络实验,设计了更贴近应用的实验方案,主要包括公网接入实验、网络安全通信实验、室内实时定位实验、自组网时间同步实验、数据索引及搜索实验等。虽然很难从实验的角度展示物联网的应用和服务,但读者可以将本章的实验结合后面的演示实验及应用系统,对物联网的规模化行业应用有一定的认识。本章实验为物联网行业应用的技术支撑。

第6章给出5个典型物联网应用的演示实验,内容涉及环境监测、出入管理、智能家居、智能医疗和智能抄表等。这些系统级演示实验,更贴近于实际,同时也能够反映出前面各章节所包含的基础实验的具体应用,有利于建立全局的系统的整体的概念。另外,智能家居和智能医疗将在第7章从系统的角度进行详细介绍,读者也可以结合起来学习。

第7章是本书物联网演示实验的升级,也是本书切入实际行业应用的基础,给出更接近应用水平的系统设计方案。由于时间的限制,主要给出了智能家居、智能物流、智能医疗等系统或子系统的设计,实际的系统设计应和具体应用需求密切相关。本章的内容都是相关作者在各自的应用背景下做出的,仅供读者自行设计时参考。

第8章结合物联网专业设置和实验室组建的需求,给出实验室建设的建议。当然,由于行业与领域的千差万别,读者可以在掌握本书实验原理及内容的基础上,根据自身的行业及专业需求,设计出具有显著特色的物联网实验室建设方案。

第 2 章 物联网基础实验平台

本章主要介绍目前物联网领域中常用的单片机、嵌入式及智能终端平台 3 种开发环境，这是全书所有实验的基础。首先简述了各平台的硬件、软件及开发环境相关基础知识，然后针对每一种平台，设计了一个入门实验。通过这些实验的深入学习，读者可以迅速掌握各平台软硬件开发的方法及过程，为深入学习物联网各层次的应用实验打下基础。

2.1 单片机实验平台

单片机在早期的物联网，特别是传感网相关的硬件方案中扮演着重要的角色。作为物联网神经末梢的无线传感器网络，其典型的特点在于低功耗、低成本、小体积、自组织等。在目前的工艺条件下，单片机是搭建原型系统和产品的主要工具。当然，随着材料和工艺的发展，集成芯片、微机电芯片等将逐步走上舞台，并有可能最终成为主导。

2.1.1 硬件环境简介

单片机按照 CPU 处理数据的位宽可分为 4 位、8 位和 16 位机。1971 年 Intel 公司研制出世界上第一个 4 位的微处理器 Intel 4004，主要应用在工业控制领域。随着工艺的发展，出现了更高性能的单片机，4 位单片机的市场逐步缩小。8 位单片机则由于内部构造简单、体积小、成本低等优势，应用更为广泛。即便到了 21 世纪，8 位单片机在单片机的应用中仍占有相当的份额。16 位单片机虽然计算能力和功能比 8 位要强得多，但由于成本和应用场合的限制，尤其是近年来嵌入式 ARM 技术的发展进一步挤占了 16 位单片机的应用空间，因此目前 16 位单片机的应用并不如 8 位单片机那样广泛。

目前，世界各大电子电器公司基本上都有自己的单片机系列产品，如三星公司的 KS86 和 KS88 系列 8 位单片机，Philips 公司的 P89C51 系列 8 位单片机，Atmel 公司的 AT89 系列 8 位单片机等。目前，在物联网领域应用较为广泛的有 TI 公司的 MSP430 系列，Atmel 公司的 AVR 系列、51 系列，Microchip 公司的 PIC 系列等。除了单片机含有的外设种类和数量存在一定差异外，处理器核的差异是体现这些单片机性能差异的主要原因。

1. 51 系列

应用最广泛的 8 位单片机当首推 Intel 公司的 51 系列。由于该产品硬件结构合理，指令系统规范，加之生产历史悠久，存在着先入为主的优势，因此在我国得到了广泛应用。许多芯片公司都购买了 51 芯片的核心专利技术，并在其基础上进行性能和功能上的扩充（如加入片内 Flash、各种丰富的外设等），使得芯片功能和种类得到进一步的完善，形成了一个庞大的体系，并随着应用需求而不断升级。从学习者的角度来说，51 系列最大的好处莫过于其设计的标准化。它有完备的地址总线 and 数据总线，便于外部扩展，指令处理方式基本与 Intel



公司高端处理器的处理方式一致，适合于学习及应用。

51 系列单片机的另一个优点是它有一套完整的位处理器，或称为布尔处理器。其处理对象不是字或字节，而是位。布尔处理器不仅能对片内某些特殊功能寄存器的某些位进行单独处理，如传送、置位、清零、测试位等，还能进行位的逻辑运算，其功能十分完备，对位运算的支持非常有力。相对而言，虽然其他种类的单片机也具有位处理功能，但能进行单个位逻辑运算的实属少见，这为用户进行细粒度的开发提供了极大的方便。

51 系列的第三个优点是具有硬件乘法和除法指令，这给编程也带来了便利。虽然其乘除法指令受到运算精度的限制，但是相对而言，很多的 8 位单片机都不具备乘除法功能，只能用一段子程序来模拟实现乘除法，效率很低且十分不便。

当然，51 单片机也有它的缺点。例如，对于传统的 51 单片机来说，最快的指令需要 12 个时钟周期，指令执行速度太慢；由于它采用了复杂指令集（CISC）结构，不利于流水线操作，因此执行效率较低。这些缺点与其他精简指令集（RISC）的单片机相比显得十分逊色。然而近年来，许多厂商争相推出 6 个时钟周期的 51 单片机，甚至单时钟周期的 51 单片机，还有厂商提供在线调试及编程（ISP）功能，给开发调试带来很大的便利。51 单片机的生命周期又出现了新的生机，预计未来，51 单片机仍会占据 8 位单片机的主要市场。

2. PIC 系列

PIC 单片机系列是 Microchip 公司的产品，是当前市场份额增长最快的单片机之一。最近几年，PIC 单片机尤其在消费类、安防控制类电子市场已经占有非常大的份额。

PIC 单片机采用 RISC 结构，分别有 33、35、58 条指令，视其级别而定（而 51 系列有 111 条指令，AVR 单片机有 118 条指令，指令系统都更为复杂）。PIC 单片机采用哈佛双总线结构，使程序存储器的访问和数据存储器的访问并行处理，这种指令流水线结构，在一个周期内同时完成指令执行和预取两部分工作，从而极大地加速了指令的执行效率。此外，它还具有低工作电压、低功耗等特点。

PIC 系列单片机的 I/O 口是双向的，其输出电路为 CMOS 互补推挽输出电路。灌电流达 25mA，拉电流可达 20mA，有相当大的驱动能力，从而可以不用外部驱动电路就可以直接驱动数码管显示这样的设备，这无疑会给用户降低很多成本。

3. AVR 系列

AVR 单片机是 Atmel 公司推出的一款较为新颖的单片机，其显著特点为高性能、高速度和低功耗。它取消了机器周期，而以时钟周期为指令周期，实行流水作业，从而缩短指令的平均执行时间。AVR 单片机指令以字为单位，且大部分指令都为单周期指令，通常时钟频率范围为 4~8MHz，ATmega 128 作为 AVR 系列单片机的代表被很多无线传感器网络节点所采用。

AVR 单片机的通用寄存器共有 32 个（R0~R31），但前 16 个寄存器（R0~R15）都不能存储立即数，因而通用性有所下降。由于寄存器数量相对较少，因此当程序复杂时，通用寄存器就略显不足（与此相比，51 系列的通用寄存器最多可达 128 个）。

AVR 的 I/O 脚类似 PIC，它也有用来控制输入或输出的方向寄存器，在输出状态下，拉电流在 10mA 左右，灌电流为 20mA。虽然，它在驱动能力上不如 PIC，但比 51 系列强。

4. MSP430 系列

MSP430 系列单片机是 16 位单片机，采用了 RISC 结构，具有简洁的 27 条内核指令以及大量的模拟指令。除片内大量的寄存器外，片内数据存储器也可参加多种运算。在 8MHz



晶体驱动下指令周期为 125ns。片内多功能的硬件乘法器和优化的查表指令，使得在其上可以轻松实现数字信号处理的相关算法，如 FFT 算法等。MSP430 系列单片机的中断源较多，并且可以任意嵌套，因此使用时灵活方便。

MSP430 单片机最大的特点当属其独特的低功耗性能。首先，电源电压为 1.8~3.6V，在系统中有一种活动模式（AM）和五种低功耗模式（LPM0~LPM4）。在 1MHz 的时钟条件下活动模式运行时，芯片电流只有 200~400 μ A；在等待方式下，耗电为 0.7 μ A；在节电方式下，最低可达 0.1 μ A。其低功耗特点的一个主要技术支撑在于其独特的时钟系统设计。在 MSP430 系列中有两个不同的时钟系统：基本时钟系统和锁频环（FLL 和 FLL+）时钟系统或 DCO 数字振荡器时钟系统。由时钟系统产生 CPU 和各功能所需的时钟，并且这些时钟可以在指令的控制下打开和关闭，从而实现对总体功耗的控制和优化。因而采用不同的工作模式，芯片的功耗有着显著的不同。此外，当系统处于省电的备用状态时，用中断请求将它唤醒只用 6 μ s。这也是 MSP430 单片机超低功耗的重要技术支撑点。

单片机是物联网，特别是无线传感器网络节点广泛使用的核心处理器，在物联网应用中不可避免要涉及。对于单片机型号的选择依赖于具体的物联网应用场景，应用对系统提出的主要参考指标包括速度、功耗、体积、外设种类、易用性等。作为用户，可以在满足功能及性能需求的前提下，结合自身的喜好而进行科学的选配。

2.1.2 软件环境简介

对于单片机的开发环境，软件方面涉及对编程语言、编辑编译和调试环境的选择问题。根据应用对象的特点选择合适的开发编程语言和开发工具，是解决问题的首要任务。本节结合常用的单片机类型，简单介绍有代表性的软件开发环境。

1. 编程语言的选择

早期的单片机开发均使用汇编语言。把汇编语言源程序变为单片机可执行的机器码（一般为 .bin 和 .hex 等文件格式）有两种方法：一种是手工汇编；另一种是机器汇编。手工汇编完全参照处理器的指令表，逐条地将汇编语句翻译成机器指令。这种方法显然效率极低，目前几乎不再使用。机器汇编是通过汇编器（Assembler）将源程序变为机器码，例如用于 MCS-51 单片机的 A51 汇编软件。机器汇编极大地减少了人工参与的过程，效率显然更高。

随着单片机开发技术的不断发展，高级语言，特别是嵌入式 C 语言，成为了单片机开发的主要编程语言。与汇编语言相比，C 语言在功能、结构、可读性和可维护性上有明显的优势，尤其对于那些使用过汇编语言后，再使用 C 语言来开发的程序员来说，会觉得 C 语言非常易学易用。当然，采用 C 语言编程存在着生成代码效率问题，这取决于编译器和程序员的编程风格。由于 C 语言编译器已经对单片机部分硬件资源进行了封装，如堆栈、子程序的跳转、中断处理时的现场保护等。C 代码中的函数、控制语句在编译时都会被编译器编译成相应的汇编指令，这些语句生成的代码量对各种不同的 C 语言编译器不尽相同。一般来说，同样功能的总代码生成量比使用汇编语言略高 20%。可以这样说，任何一款编译器都不会比一个经验丰富的汇编语言程序员编写的汇编程序效率高。

无论是采用 C 语言，还是采用汇编语言，都各有其利弊。虽然对汇编语言的娴熟使用需要一定的时间，并且调试起来困难很大，但其程序执行效率高是不争的事实。C 语言虽易学易用，但对于一些底层和重复性操作，采用 C 语句实现起来效率偏低。所以在开发过程中，



推荐采用 C 语言和汇编语言相结合的编程方式,以充分发挥这两者的优势。例如,通常用汇编语言来编写底层的对硬件的操作,把与硬件无关或相关性较小的部分用 C 代码实现。当然,要充分发挥两者的性能优势,需要程序员对 C 编译器有一定了解,并注重平时的积累。

2. 开发平台的选择

单片机开发平台的基本功能是实现源代码的编译、连接并生成目标代码,提供目标代码下载的功能或接口,并支持仿真以及目标调试功能。目前,已经有一些较为通用的单片机集成开发环境,如 ICC (Imagecraft C Compiler)、IAR Embedded Workbench、CodeVision AVR (或称 CVAVR)、Keil μ Vision、GCC 等。这些开发平台虽然针对某些特定品种而定制,它们包括集成环境 IDE、处理器专家库、可视化参数显示工具、项目工程管理器、C 编译器、宏汇编、连接/定位器,以及目标文件生成、库管理及功能强大的仿真调试器等,是一种集成化的文件管理编译环境,具有良好的用户接口,并且都普遍支持汇编语言和 C 语言的混合编程。

ICC 的集成开发环境包括一个叫 Application Builder 的代码生成器,可通过设置 MCU 所具有的中断、内存、定时器、IO 端口、UART、SPI 等外围设备,从而自动生成初始化外围器件的代码,这为简化初始配置提供了便利。ICC 集成开发环境还有一个终端程序,可以发送和接收 ASCII 码,这为设备的调试提供了方便。此外,它还提供了常用的器件库和运算库代码,以方便用户的编程调用,可以提高开发效率。

CVAVR 是一个针对 AVR 单片机的集成开发环境。它有一个 CodeWizard 的代码生成器,可生成外围器件的相应初始化代码,支持位变量操作,并且与 Keil C51 的代码风格最为相似,集成较多常用外围器件的操作函数以及一个集成代码生成向导,并且集成了串口/并口 AVRISP 等下载烧写功能,开发起来也非常方便。

IAR Embedded Workbench 可支持 AVR、MSP430、MCS-51、ST8 等多种单片机。经过反复实验证明,IAR Systems 的 C 编译器可以生成高效可靠的可执行代码,并且应用程序规模越大,效果越明显,生成的代码尺寸远远小于其他同类编译器生成的代码尺寸。此外,IAR Embedded Workbench 还提供了 Visual State 和 IAR MakeApp 两套图形开发工具帮助开发者完成应用程序的开发,它可以根据设计需求自动生成应用程序代码,使开发者摆脱这些耗时的任务并且也保证了代码的质量。

Keil 公司主要开发、制造和销售嵌入式 8051、251、ARM、XC16x/C16x/ST10 等微控制器软件开发工具、调试和模拟器、综合评估板等。2005 年,Keil 公司被 ARM 公司收购。其早期的 Franklin C51 交叉编译器是专门为 8051 系列单片机设计的简单、高效 C 语言编译器, Franklin 软件包主要由 C51 交叉编译器、A51 交叉汇编器、L51 连接器、OHS51 代码转换器及 DSCOPE-51 软件仿真器等模块组成,其输入/输出库函数 scanf 和 printf 默认是通过 8051 单片机的串行口实现的,其他输出方式可以通过编写特定的输入/输出函数来实现,以方便用户的板上调试。Keil μ Vision 则是把上述编辑、编译、连接和仿真等模式打包成集成开发环境,还包含项目管理和调试器等功能,其中包含的微处理器模拟器可以模拟被支持的微处理器设备,包括指令集、片上外设,甚至还可模拟外部激发信号。

GCC 是一款优秀的 C 编译软件,针对一些特定的单片机型号,通过在其上的移植而得到针对该处理器的 C 编译器。GCC 支持多种操作系统,在 MS-DOS、Windows、Linux 下都可以使用。在 AVRGCC 编译器的 Win32 版本中,有一个软件包 WinAVR。它虽然不是集成开发环境,但不仅包括 C 编译器,还包括了编辑、编译流程中所需的所有软件。GCC 最大



的特点是开源，发展较快，并且免费使用。

可以说，单片机的开发环境虽然品种繁多，但对于普通用户的使用来说，并不一定能体会到其底层的细微差别。主要感觉到的区别体现在与用户的交互接口上，因此用户可以根据使用习惯和应用需求，综合进行开发平台的选择。

2.1.3 操作系统简介

与传统个人计算机不同，单片机系统在系统实时性、高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有较为突出的特点，并且随着硬件功能与软件规模的不断发展，在裸机上直接使用 C 语言或汇编语言开发越来越困难，因此单片机的操作系统应运而生。操作系统可以将应用与硬件隔离开来，让用户可以更多地关注应用的开发。

由于单片机需要管理的资源比较有限，因此其操作系统代码量也比较小，例如最小的 μCOS 内核只有 2KB，并且由于相对 Linux、WinCE 等嵌入式操作系统而言，这些在单片机上运行的操作系统更强调实时性而非用户界面，因而被广泛称为实时操作系统（Real Time Operating System, RTOS）。

最早的 RTOS 是 VRTX。第一个商业版本的 VRTX 1.0 发表于 1981 年。在整个 20 世纪 80 年代，VRTX 在全世界占领了大多数市场，有超过 100 万用户的产品，包括 AT&T、Motorola、Siemens 的通信和手机产品，以及波音、麦道和空客的飞机控制装置。VRTX 是一个真正意义上的 RTOS。20 世纪 80 年代，它还只能支持一些 16 位的微处理器，如 68000、8086 等，到 90 年代时，VRTX 已经可以支持 68000、x86、960、Sparc 等 16、32 位的单片机和嵌入式微处理器，当时的产品除 VRTX 外，还有 IPI 公司的 MTOS 和 80 年代末 ISI 公司的 PSOS，主要用于军事和电信设备。

进入 20 世纪 90 年代，现代操作系统的设计思想，如微内核设计技术和模块化设计思想，开始渗入 RTOS 领域。老牌的 RTOS 厂家（如 Ready System）也推出了新一代的 VRTXsa 实时内核，新一代的 RTOS 厂家 Windriver 推出了 Vxwork。另外，在这个时期，各公司都力求摆脱完全依赖第三方工具的制约，而通过自己收购、授权或使用免费工具链等商业方式，组成一套完整的开发环境。例如，ISI 公司的 Prismt、著名的 Tornado（Windriver）和 Spectra（VRTX 开发系统）等。这些操作系统均属于商品化产品，价格昂贵且由于源代码不公开导致了诸如对设备的支持、应用软件的移植等一系列的问题。从而，开放源码的 RTOS 凭借其在成本和技术上特有的优势异军突起，在 RTOS 领域占有越来越重要的地位。可以说，开源系统已经成为了目前单片机操作系统的主流。这里介绍 RTX51、 $\mu\text{C/OS-II}$ 、eCos 源代码公开的实时操作系统，希望给读者在选择时以参考。

1. $\mu\text{C/OS-II}$

$\mu\text{C/OS-II}$ 的前身是 $\mu\text{C/OS}$ ，最早出自于美国嵌入式系统专家 Jean Labrosse 在《嵌入式系统编程》杂志 1992 年 5 月和 6 月刊上刊登的文章连载，并把 $\mu\text{C/OS}$ 的源代码发布在该杂志的 BBS 上。之后世界上数以千计的工程技术人员将 $\mu\text{C/OS}$ 应用到了各个领域，如照相机业、发动机控制、网络接入设备、高速公路电话系统、ATM 机和工业机器人等。许多大学将 $\mu\text{C/OS}$ 作为教材，用于实时系统的教学工作。

$\mu\text{C/OS}$ 和 $\mu\text{C/OS-II}$ 是专门为计算机的嵌入式应用设计的，绝大部分代码是用 C 语言编写的。CPU 硬件相关部分是用汇编语言编写的，总量约 200 行的汇编语言部分被压缩到最低



限度，为的是便于移植到任何一种其他的 CPU 上。用户只要有标准的 ANSI C 交叉编译器、汇编器、连接器等软件工具，就可以将 $\mu\text{C}/\text{OS}$ 嵌入到开发的产品中。

从技术角度来说， $\mu\text{C}/\text{OS}$ 具有执行效率高、占用空间小、实时性能优良和可扩展性能好等特点，最小内核可编译至 2KB。 $\mu\text{C}/\text{OS-II}$ 是一个抢占式的内核，即已经准备就绪的高优先级任务可以剥夺正在运行的低优先级任务的 CPU 使用权。因此它的实时性非常好。目前， $\mu\text{C}/\text{OS-II}$ 已经被移植到了几乎所有知名的 CPU 上，从使用角度来说，已经非常方便。

2. eCos

eCos (embedded Configurable operating system)，即嵌入式可配置操作系统，最初起源于美国的 Cygnus Solutions 公司。Cygnus 公司于 1998 年 11 月发布了第一个 eCos 版本 eCos 1.1，当时只支持有限的几种处理器结构。1999 年 11 月，RedHat 公司收购了 Cygnus 公司。在此后的几年里，eCos 成为单片机及嵌入式领域的关键产品，得到了迅速的发展，2003 年 5 月正式发布了 eCos 2.0，获得了很大的成功。

虽然 eCos 是 RedHat 公司的产品，但是 eCos 并不是 Linux 的派生，eCos 可以提供实时嵌入式应用所需的基本运行基件，而只占用几十 KB 或几百 KB 的内存空间。从 eCos 的名称可以看出，其最大的特点在于它是一个配置灵活的系统。eCos 的核心部分是由不同的组件组成的，包括内核、C 语言库和底层运行包等，每个组件都能提供大量的可配置选项。利用 eCos 提供的配置工具可以很方便地进行配置，通过不同的配置使得 eCos 能够满足不同的嵌入式应用需求。

3. RTX51

RTX51 是美国 Keil 公司开发的一种小型的应用于 MCS51 系列单片机的实时多任务操作系统，它可以工作在所有 8051 单片机以及派生家族中。RTX51 有两种模式：RTX51 完全模式和最小模式。RTX51 最小模式版是 RTX51 完全版的一个子集，可以很容易地运行在 8051 系统上而不需要外部 RAM。RTX51 完全模式有 4 个任务优先级，可以和中断函数并行处理，各个任务之间可进行信号和消息的传递。

RTX51 有两类任务：快速任务和标准任务。相对于标准任务，快速任务有很快的响应速度。每个快速任务使用一个单独的寄存器组，并且有自己的堆栈区域。RTX51 支持最大同时有三个快速任务，已经基本满足单片机的应用需求。

2.1.4 实验一 单片机平台基础实验

1. 实验目的

本实验在单片机实验母板 (IOT-SCMMB) 上使用定时器、LED 和串口实现了一个简单的演示程序。该母板以无线传感器网络节点中常用的 Atmel 公司的 ATmega128L 芯片为处理器，扩充了多种传感器及外围模块。本实验通过定时器中断交替点亮开发板上黄蓝 LED 灯，并向串口输出已点亮 LED 灯的编号 (黄色 LED 编号为 0，蓝色 LED 编号为 1)。通过该演示例子，使读者对单片机开发有初步了解，掌握在 WinAVR 集成开发环境下进行单片机程序的编写和编译的步骤，并能使用 AVRStudio 软件将程序下载到开发板上。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。



- 传感器及外围模块 (IOT-MODULES) 1 套。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- 串口调试助手。
- WinAVR20050214。
- AVRStudio4.18。

3. 实验原理

本实验的演示程序使用了单片机实验母板 (IOT-SCMMB) 上的基本部件, 其主要围绕 Atmel 公司的 ATmega128 芯片构建的开发板, 主要功能组成包括了多种传感器、射频及外部接口, 开发板的实物照片如图 2-1 所示。

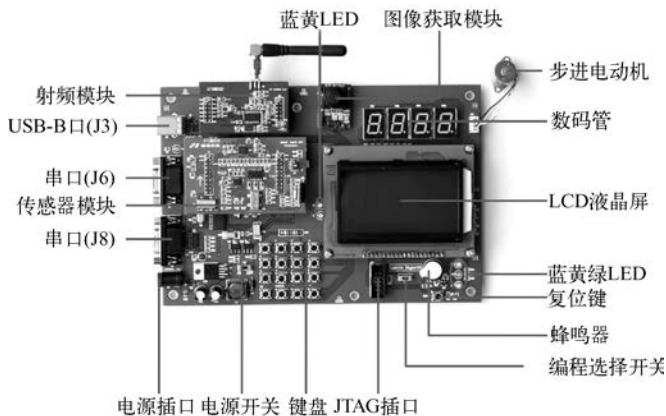


图 2-1 单片机开发板实物照片

本实验的程序使用了该开发板上 ATmega128 单片机 U4 的 8 位定时器 0、蓝黄 LED 以及 USB-B 口 J3。在程序的主函数文件 mian.c 中, 首先设置单片机与 LED 的连接端口, 然后设置串口的参数并对其进行初始化, 接下来配置计数器 0 (采用 64 分频, 将 7.3728MHz 的晶振频率降为 115200Hz), 最后打开全局中断, 并使程序进入一个循环。相关代码如下:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define uchar unsigned char
#define Leds_yellowOn PORTD = 0x40; //黄色 LED 灯亮
#define Leds_blueOn PORTD = 0x20; //蓝色 LED 灯亮

static uchar counter = 0; //LED 灯闪烁频率控制变量
static uchar i = 0; //i 为 1 时表示黄灯亮, i 为 0 时表示蓝灯亮

/*****
*名称 main()
*****/
```



```

*功能 初始化 I/O、串口及定时器 0，定时时间为 0.5s
*入口参数 无
*出口参数 无
*****/
int main(void)
{
    //初始化 LED
    DDRD = _BV(PD5)|_BV(PD6);    //设置与两个 LED 相连的 PD5 和 PD6 口为输出
    PORTD = _BV(PD5)|_BV(PD6);    //置高 PD5 和 PD6 口

    //初始化串口
    UBRR0L=7;                    //设置波特率为 57.6kbit/s
    UCSRB=_BV(RXEN0)|_BV(TXEN0); //使能串口的发送和接收

    //配置计数器
    TCNT0=0;                      //计数器清零
    TCCR0=_BV(CS02);              //采用 64 分频，并启动计数器
    TIMSK=_BV(TOIE0);            //打开计数器中断

    sei();                        //打开全局中断
    while(1)                      //进入循环
        ;
}

```

8 位计数器溢出时，会触发溢出中断。在中断处理函数中，对 LED 灯的状态进行设置，并输出到串口。分频后的 115200Hz 依然不便于观察，因此引入了一个 8 位的变量 counter，每次计数器溢出时该变量加 1。当该变量值为 225 时改变 LED 灯的状态。这样通过控制该变量的大小就可以调节 LED 灯闪烁的周期，即

$$2^8/115200\text{Hz}\times 225=0.5\text{s}$$

相关的代码如下：

```

/*****
*名称 SIGNAL()
*功能 定时器/计数器 0 溢出中断
*入口参数 SIG_OVERFLOW0
*出口参数 无
*****/
SIGNAL(SIG_OVERFLOW0)
{
    //产生中断周期 T = 256*(7.3728M / 64)
    //T*225 等于 0.5s
    if(++counter>224)                //中断 225 次 0.5s
    {
        if(i)
        {
            Leds_blueOn;            //蓝色 LED 灯亮

```



```
        serialputc(i);                //向串口发送 1
        i = 0;
    }
    else
    {
        Leds_yellowOn;                //黄色 LED 灯亮
        serialputc(i);                //向串口发送 0
        i = 1;
    }
    counter = 0;
}
}
```

serialputc()为向串口发送数据的函数，参数 i 为待发送的数据，其实现代码如下：

```
*****
*名称 serialputc()
*功能 向串口发送数据
*入口参数 待发送的字符
*出口参数 无
*****/
void serialputc(uchar c)
{
    while(!(UCSR0A&(1<<UDRE0)))    //数据发送已准备好
        ;
    UDR0 = c;                        //待发送的数据放入缓冲器，发送数据
}
```

4. 实验内容

安装 WinAVR20050214 和 AVRStudio4.18 软件，学习如何使用这两个软件进行单片机程序的开发、编译和下载。

初步掌握基于开发板的单片机程序设计方法，能够设计简单的程序操作开发板上的各个模块。

5. 实验步骤

(1) 安装 WinAVR20050214 和 AVRStudio4.18

分别单击 WinAVR-20050214-install.exe 和 AvrStudio4Setup.exe 图标，按照默认的设置进行安装。

(2) 编译演示程序

启动 WinAVR20050214 编辑环境 Programmers Notepad。选择 File→New→Project，新建一个工程，并保存在工程文件夹中，工程名称设置为 demoexperiment。新建一个 C 语言源代码文件，即选择 File→New→C/C++，然后选择菜单中的 Save as 将其保存为一个新的文件 demoexperiment.c。用鼠标右键单击左侧窗口中新建的工程 demoexperiment，选择 Add Files，在 Add Files 对话框中选择 demoexperiment.c，并单击打开，即可将该文件添加到工程文件中。此时，就可以在 demoexperiment.c 中编辑程序了。将本实验提供的源代码复制到编辑窗口，然



后单击 Tools→[WinAVR] MakeAll 编译文件。此时会出现如图 2-2 所示的对话框。

“No rule to make target `all'. Stop.”表示未找到 Makefile 文件。下面将介绍如何使用 WinAVR 建立项目工程的 Makefile 文件。

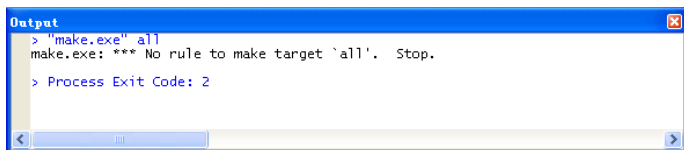


图 2-2 未找到 Makefile 文件对话框

通常由编译器（泛指高级语言编译器、汇编器、连接器等）、项目管理和文本编辑器构成一个完整的编程环境。项目管理器负责为每一个源程序文件调用编译器生成目标文件，然后用连接器将它们组合在一起，生成可执行文件。WinAVR 使用一个叫做 Makefile 的文件来管理程序的编译连接，Makefile 是个通用脚本文件，可执行文件 make.exe 负责解析它并根据脚本内容来调用编译器、连接器或其他的工具，最终生成可执行代码文件。每次调用 make 时它会比较目标文件与源文件的更新时间，如果源文件比目标文件新，它会执行 Makefile 内的相关指令更新目标文件，如果目标文件与源文件同样新，它就跳开这个源文件的编译，避免重复工作，这对于较大的工程来说，是节省编译时间的有效方法。

WinAVR 包中有一种简单的 Makefile 文件生成工具叫做 MFile，利用它就可方便地生成合适的 Makefile 文件。选择程序→WinAVR→MFile，即可打开 MFile。下面对 MFile 的 Makefile 菜单下的各个选项进行简单的介绍。

Main file name...菜单项指定主程序文件，它将决定主源程序文件名及输出文件名。选择该选项，在对话框中输入主程序的主文件名，这里使用 demoexperiment。注意，文件名中不要包含扩展名，如.c 等。

MCU type 根据所选择的处理器进行设置，这里选择 ATmega128。

Output format 菜单项用于选择最终生成的可执行代码格式，根据编程器支持格式选择即可，这里选择 ihex。

Optimization level 指定 C 代码的优化级，s 代表按最小代码量编译，选择 s。

C/C++ source file(s) 和 Assembler source files(s) 用于在项目添加其他 C、C++和汇编程序源文件。本实验只有一个文件，因此可以不设置。

通常，设置以上几项就可以了。在编写程序的过程中往往需要在项目添加源文件，这时只需在生成的 Makefile 中 SRC 变量后添加源文件名即可。

当上述选项设置好之后，选择 File→Save As，将其保存到工程文件夹中。再单击 Tools→[WinAVR] Make All 编译文件，就可以在 Output 对话框中看到如图 2-3 所示的结果。

图中的“Process Exit Code: 0”表示编译成功。此时在工程文件夹会生成 demoexperiment.hex 文件。

(3) 将编译后得到的.hex文件下载到开发板上

硬件连接：首先，将编程器的 JTAG 插头插到开发板上的 JTAG 插口中，并将 JTAG 插口右侧的白色开关拨到左侧，表示将文件下载到单片机 U4 上；然后，将编程器通过串口连接到计算机；最后，分别连接编程器和开发板的电源，并打开电源开关。



软件操作：首先选择程序→Atmel AVR Tools→AVR Studio 4，打开 AVR Studio。如果出现开始界面（显示最近使用过的文件或工程以供用户选择），选择 **Cancel**，关闭此界面。之后会进入该软件的主界面，选择 **Tools→Programme AVR→Connect...** 或者直接单击界面上的 **AVR** 按钮，会弹出一个工作界面，这个界面有 8 个选项页，分别是 **Main**，**Program**，**Fuses**，**LockBit**，**Advanced**，**HW Settings**，**HW Info** 和 **Auto**，需要用到的是 **Main** 和 **Program**，后面的几项保持默认。

在 **Main** 选项下首先需要在 **Device and Signature Bytes** 的第一个框的下拉列表中选择单片机的型号，节点使用的是 **ATmega128**，需要选中对应的设备。在 **Program** 选项下 **Device** 框中有两项是可选的，它们是 **Erase Device Before** 和 **Verify Device After Program**。如果选中了第一个，在编写程序时会先将 **ATmega128** 的 **128KB** 的 **Flash** 全部擦除，然后再进行程序下载；第二个选项表示在编写程序完成以后进行一些校验和验证的工作，确保编程的数据是无误的。建议将这两个选项都选上。最后在 **Program** 选项下的 **Flash** 框中选择实验程序的 **.hex** 文件，单击 **Flash** 框中的 **Program** 按钮就可以将该 **.hex** 文件下载到 **ATmega128** 中。程序下载成功之后可以看到实验开发板上黄蓝灯轮流闪烁，在上位机上打开串口调试助手（波特率设为 **57 600bit/s**，以十六进制显示），会输出当前正点亮的 **LED** 的编号，如果黄色 **LED** 亮则输出 **0**，蓝色 **LED** 亮则输出 **1**。

```
Output
> "make.exe" all

----- begin -----
avr-gcc (GCC) 3.4.3
Copyright (C) 2004 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiling: demosexperiment.c
avr-gcc -c -mmcu=atmega128 -I. -gdwarf-2 -DF_CPU=8000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums
Linking: demosexperiment.elf
avr-gcc -mmcu=atmega128 -I. -gdwarf-2 -DF_CPU=8000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wl
Creating load file for Flash: demosexperiment.hex
avr-objcopy -O ihex -R .eeprom demosexperiment.elf demosexperiment.hex
Creating load file for EEPROM: demosexperiment.eep
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" \
--change-section-lma .eeprom=0 -O ihex demosexperiment.elf demosexperiment.eep
Creating Extended Listing: demosexperiment.lss
avr-objdump -h -S demosexperiment.elf > demosexperiment.lss
Creating Symbol Table: demosexperiment.sym
avr-nm -n demosexperiment.elf > demosexperiment.sym

Size after:
demosexperiment.elf :
section      size      addr
.data        0      8388864
.text       372      0
.bss         2      8388864
.noinit      0      8388866
.eeprom      0      8454144
.debug_aranges 20      0
.debug_pubnames 58      0
.debug_info 204      0
.debug_abbrev 154      0
.debug_line 291      0
.debug_str  129      0
Total      1230

AVR Memory Usage:
-----
Device: atmega128

Program:   372 bytes (0.3% Full)
(.text + .data + .bootloader)

Data:      2 bytes (0.0% Full)
(.data + .bss + .noinit)

----- end -----

> Process Exit Code: 0
```

图 2-3 使用 WinAVR 编译实验代码通过结果

6. 思考题

- 1) 如何自己写一个 Makefile 文件?
- 2) 如何将程序下载到 EEPROM 中?



7. 补充阅读

- [1] 徐勇军, 安竹林, 等. 无线传感器网络实验教程[M]. 北京: 北京理工大学出版社, 2007.
- [2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.

2.2 嵌入式实验平台

虽然单片机在感知层扮演了最主要的角色, 但毕竟其处理能力及存储容量均十分有限, 在很多应用场景下显得捉襟见肘, 这就需要功能更强的嵌入式系统来完成。嵌入式系统包括了功能较强的 CPU、DSP、FPGA 及其他专用芯片。它们在性能上的优势可以弥补在功耗、成本等方面的不足, 特别适用于能量不十分敏感, 对性能及可靠性要求较高的场景, 但又不适合使用体积更大的通用计算机或服务器, 是单片机系统的有力补充。

2.2.1 硬件环境简介

与个人计算机系统不同, 嵌入式系统具有面向用户、面向产品和面向应用, 具有定制性和非通用性的特点。嵌入式处理器是嵌入式系统硬件的核心。与 PC 中的通用处理器不同, 嵌入式处理器的选择必须要根据应用的需要, 综合考虑其功耗、体积、成本、可靠性、速度、处理能力、电磁兼容性等多方面的综合因素。对于开发者来说, 根据应用的需求, 合理地选择适合的嵌入式处理器, 是项目能够成功的先决条件。

按照嵌入式处理器的用途, 嵌入式处理器可以分成下面几类: 嵌入式微控制器 (Microcontroller Unit, MCU)、嵌入式微处理器 (Embedded Micro-processor Unit, EMPU)、嵌入式数字信号处理器 (Embedded Digital Signal Processor, EDSP)、嵌入式片上系统 (System on Chip, SoC) 及其他专用处理器。

嵌入式微控制器也是单片机, 实际应用中的界限并不十分明确。微控制器强调控制功能而弱化计算功能, 因此其控制类指令要比计算类指令的功能要强得多。由于计算功能有限, 它只能适合于较为简单的控制场合。由于控制算法、控制对象和用户接口的日益复杂, 嵌入式微控制器已经不能满足应用的需求, 而通用 CPU 又不能满足嵌入式系统对处理器的功耗、封装以及资源的要求, 因此嵌入式微处理器应运而生。

嵌入式微处理器一般指的是具有 32 位以上的处理器。与计算机处理器不同的是, 在实际嵌入式应用中, 只保留和嵌入式应用紧密相关的功能硬件, 去除其他的冗余功能部分, 从而以最低的功耗和最少的资源实现嵌入式应用的特殊要求, 这样可以进一步降低成本和功耗, 并提高系统性能。嵌入式微处理器一般具备如下 4 个特点:

- 1) 对实时和多任务有很强的支持能力, 能完成多任务并且有较短的中断响应时间。
- 2) 具有功能很强的存储区保护功能。
- 3) 具有可扩展的处理器结构, 可以迅速地扩展出满足应用需求的高性能嵌入式微处理器。
- 4) 在保证性能的前提下, 具有极低的功耗。

目前, 主要的嵌入式处理器类型有 Am186/88、386EX、SC-400、Power PC、68000、MIPS、ARM/ StrongARM 系列等, 下面进行简单介绍。



1. ARM系列

ARM (Advanced RISC Machines), 既可以认为是一个公司的名字, 也可以认为是对一类微处理器的通称, 还可以认为是一种技术的名字。1991 年 ARM 公司成立于英国剑桥, 主要出售芯片设计技术的授权。ARM 处理器的主要特点是: 耗电少功能强、16 位/32 位双指令集和众多合作伙伴。目前, 采用 ARM 技术 IP 核的微处理器, 即通常所说的 ARM 微处理器, 已遍及工业控制、消费类电子产品、通信系统、网络系统、无线系统等各类产品市场, 基于 ARM 技术的微处理器应用约占据了 32 位 RISC 微处理器 75% 以上的市场份额, 在工业控制、无线通信、网络、消费类电子、安全产品中都得到了广泛的应用。

作为采用 RISC 架构的处理器, ARM 微处理器具有一般 RISC 处理器的特点, 如 Load/Store 结构、寻址方式灵活简单、执行效率高、大量使用寄存器、指令执行速度快等。此外, 它还具有体积小、低功耗、低成本、支持 Thumb (16 位)/ARM (32 位) 双指令集, 兼容 8 位/16 位器件, 指令长度固定等优点。

ARM 微处理器目前包括下面几个系列: ARM7、ARM9、ARM9E、ARM10E、ARM11、SecurCore、Cortex。其中, ARM7、ARM9、ARM9E 和 ARM10 为 4 个通用处理器系列, 每个系列提供一套特定的性能, 以满足设计者对功耗、性能和体积的需求; SecurCore 系列专门为安全应用而设计, 主要在安全算法上做了较多考虑。此外, Intel 公司通过购买 ARM 的 IP 核先后生产出 Xscale 和 StrongARM 系列, 除了维持 ARM 的指令集不变外, 在功能和性能上对这两款处理器进行了改进和提高。

2. PowerPC

PowerPC 也是一种 RISC 架构的 CPU, 其基本的设计源自 IBM 公司的 Power 架构。Power 是由 Apple 公司、IBM 公司、Motorola 公司组成的 AIM 联盟在 1991 年所发展出的微处理器架构, 以对抗在市场上具有压倒性优势的 x86 处理器。在 2006 年前, 基于 Power PC 架构的 CPU 一直只能由 IBM 公司和 Motorola 公司生产。2006 年后, 它们开放了 Power PC 的授权。虽然授权开放得晚, 但由于其出色的性能和高度整合性的优势, Power PC 正在侵占原先 ARM 和 MIPS 占据主导地位的市场, 尤其在高端嵌入式设备场合, Power PC 更是具有很大优势。

目前, 主流的 Power PC 处理器制造商有 IBM 公司、Freescale 公司、AMCC 公司、LSI 公司等。其中以 IBM 公司和 Freescale 公司的 Power PC 处理器最为流行。IBM 公司目前共有 3 个主要的 PowerPC 处理器系列: Power、Power PC 和 CELL, 另外还有一个 Star 系列。Freescale 公司提供了数量众多的含集成化外设的 Power PC 处理器, 在网络设备市场取得了成功。目前 Freescale 公司已经调整了其 Power PC 产品策略, 将其划分成 3 个主要市场: 网络设备、汽车电子 (MPC5xx 系列) 和工业控制, Power PC 由于其性能优良、安全可靠, 其应用也十分广泛。

3. MIPS

MIPS 也是一种 RISC 处理器。MIPS 的意思是“无内部互锁流水级的微处理器” (Microprocessor without Interlocked Piped Stages), 其机制是尽量利用软件办法避免流水线中的数据相关问题。它最早是在 20 世纪 80 年代初期由斯坦福大学 Hennessy 教授领导的研究小组研制的。MIPS 公司的 R 系列就是在此基础上开发的 RISC 工业产品的微处理器。MIPS 公司在 1986 年推出 R2000 处理器, 1988 年推出 R3000 处理器, 1991 年推出第一款 64 位商用微处理器 R4000。之后又陆续推出 R8000 (于 1994 年)、R10000 (于 1996 年) 和 R12000 (于 1997 年) 等型号。随后, MIPS 公司的战略发生变化, 把重点放在嵌入式系统。1999



年, MIPS 公司发布 MIPS32 和 MIPS64 架构标准, 新的架构集成了所有原来 MIPS 指令集, 并且增加了许多更强大的功能。

在通用计算方面, MIPS R 系列微处理器用于构建 SGI 的高性能工作站、服务器和超级计算机系统。在嵌入式方面, MIPS K 系列微处理器是目前仅次于 ARM 的用得最多的处理器之一, 其应用领域覆盖游戏机、路由器、激光打印机、掌上电脑等各个方面。MIPS 公司的商业赢利模式和 ARM 公司类似, 也是研发处理器内核, 并将知识产权授权给其他芯片厂商。国产处理器龙芯就是获得了 MIPS32 和 MIPS64 架构的授权, 借此开发通用 CPU 的。

4. 嵌入式DSP

数字信号处理器 (Digital Singnal Processor, DSP) 是一种独特的微处理器, 有自己的完整指令系统, 运算功能异常强大。一个数字信号处理器在一块不大的芯片内包括有控制单元、运算单元、各种寄存器以及一定数量的存储单元等, 在其外围还可以连接若干存储器, 并可以与一定数量的外部设备互相通信, 功能全面。

DSP 采用的是哈佛结构, 即数据总线和地址总线分开, 使程序和数据分别存储在两个分开的空间, 允许取指令和执行指令完全重叠, 这大大提高了微处理器的速度。它最值得称道的两大特色是其强大的数据处理能力和高运行速度。由于它运算能力强, 速度快, 体积小, 以及采用软件编程而带来的高度灵活性, 使得 DSP 成为解决各种复杂应用的一条有效途径。

在数字滤波、FFT、谱分析等方面, DSP 算法目前正在大量进入嵌入式领域, 也是传统嵌入式处理器难以满足的功能。从而, DSP 应用正从在通用单片机中以普通指令实现 DSP 功能的方式过渡到采用嵌入式 DSP 处理器进行开发的方式。嵌入式 DSP 处理器有两个发展来源, 一个是 DSP 处理器经过单片化、EMC 改造、增加片上外设成为嵌入式 DSP 处理器, 如 TI 的 TMS320C2000/C5000 等; 另一个是在通用单片机或 SOC 中增加 DSP 协处理器, 例如 Intel 的 MCS-296 和 Infineon(Siemens)的 TriCore。

比较有代表性的嵌入式 DSP 产品是 Texas Instruments 公司的 TMS320 系列和 Motorola 公司的 DSP56000 系列。TMS320 系列处理器包括用于控制的 C2000 系列, 移动通信的 C5000 系列, 以及性能更高的 C6000 系列和 C8000 系列。DSP56000 目前已经发展成为 DSP56000、DSP56100、DSP56200 和 DSP56300 等几个不同系列的处理器。

5. 嵌入式片上系统

作为嵌入式片上系统 (System on Chip, SOC) 的主要承载体, FPGA (Field Programmable Gate Array) 的性能直接决定了 SOC 系统的可用性。FPGA 是在 PAL、GAL、PLD 等可编程器件中集成度最高的一种。基于其静态可重复编程和动态在系统重构的特性, 使得硬件的功能可以像软件一样通过编程来修改。作为专用集成电路领域中的一种半定制电路, FPGA 既解决了定制电路的不足, 又克服了原有可编程器件门电路数有限的缺点。在硬件设计阶段, 工程师可以通过传统的原理图输入法, 或是硬件描述语言自由设计一个数字系统; 通过软件仿真, 可以事先充分验证设计的正确性; 在印制电路板 (PCB) 完成以后, 还可以利用 FPGA 的在线修改能力, 随时修改设计而不必改动硬件电路。可以说, FPGA 芯片是小批量系统提高系统集成度和增强可靠性的最佳选择之一。目前 FPGA 的品种很多, 有 Xilinx 公司的 XC 系列、TI 公司的 TPC 系列、ALTERA 公司的 FIEX 系列等。

随着 EDA 技术的推广和 VLSI 设计的普及, 以及半导体工艺的迅速发展, 出现了在一个硅片上实现一个复杂系统的趋势, 这就是 SOC。在 SOC 中, 作为嵌入式系统核心的嵌入



式微处理器内核不再作为一个单独的芯片存在，而是作为普通的标准库，和许多其他嵌入式系统外设一样，成为超大规模集成电路（Very Large Scale Integrated Circuits, VLSI）设计中一种标准的器件。用户在开发时，只需定义出整个应用系统，仿真通过后就可以将设计图交给半导体工厂制作样品。这样，除个别无法集成的器件（尤其是模拟电路）外，整个嵌入式系统被集成到一块或几块芯片中去，应用系统的电路板将变得很简洁，这对于减小体积和功耗、提高可靠性非常有利。

SOC 可以分为通用和专用两类。通用系列包括 Infineon 公司的 TriCore，Motorola 公司的 MCore，还有某些 ARM 系列器件，如 Echelon 公司和 Motorola 公司联合研制的 Neuron 芯片等。专用 SOC 一般专用于某个或某类系统中，不为一般用户所知。例如 Philips 公司的 Smart XA，它将 XA 单片机内核和支持超过 2048 位复杂 RSA 算法的计算单元制作在一块硅片上，形成一个可加载 Java 或 C 语言的专用的 SOC，可用于公众互联网（如 Internet）的安全方面。

2.2.2 软件环境简介

对于基于嵌入式 DSP 和 SOC 的系统开发来说，软件开发环境往往与所选用的处理器/FPGA 型号直接相关，一般由厂商提供，并不具有通用性。相对来说，基于嵌入式微处理器的系统开发流程则具有一定程度的通用性。本节则以基于嵌入式微处理器的嵌入式系统开发为主线介绍其软件开发环境。下面以 Linux 应用为例进行说明。从开发人员的角度，如在专用的嵌入式板卡上运行基于 Linux 的应用，软件可以分为以下 4 个层次。

1) 引导加载程序。包括固化在固件中的 boot 代码和 Boot Loader 代码两部分。其中 boot 代码与开发人员基本无关。

2) Linux 内核。特定于嵌入式板子的定制内核以及内核的启动参数。

3) 文件系统。包括根文件系统和建立在 Flash 内存设备之上的文件系统。

4) 用户应用程序。特定于用户的应用程序。有时在用户应用程序和内核层之间可能还会包括一个嵌入式图形用户界面。常用的嵌入式 GUI 有 MicroWindows 和 MiniGUI 等。

引导加载程序（Boot Loader）是系统加电后运行的第一段软件代码。简单地说，Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，可以初始化硬件设备和建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。

Boot Loader 严重地依赖于硬件，包括处理器和板级结构、各种计算及外设资源。因此，在嵌入式世界里建立一个通用的 Boot Loader 几乎是不可能的。然而，这并不表示 Boot Loader 每次都需要重新编写。在一些成熟的 Boot Loader 的源代码的基础上进行适当修改或添加，就可以达到事半功倍的效果。

2.2.3 操作系统简介

嵌入式系统是以应用为中心，软硬件可裁减的，适用于对功能、可靠性、成本、体积、功耗等综合性要求严格的专用计算机系统。具有软件代码小、高度自动化、响应速度快等特点，特别适合于要求实时和多任务的体系。

嵌入式操作系统作为嵌入式系统中极为重要的组成部分，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。嵌入式操作系



统具有通用操作系统的基本特点，能够有效管理越来越复杂的系统资源；能够把硬件虚拟化，使得开发人员从繁忙的驱动程序移植和维护中解脱出来；能够提供库函数、驱动程序、工具集以及应用程序。与通用操作系统相比，嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有独特之处。

嵌入式操作系统伴随着嵌入式系统的发展经历了以下4个阶段。

第一阶段：无操作系统的嵌入算法阶段，以单芯片为核心的可编程序控制器形式的系统，具有与监测、伺服、指示设备相配合的功能。应用于一些专业性极强的工业控制系统中，通过汇编语言编程对系统进行直接控制。系统结构和功能都相对单一，处理效率较低，存储容量较小，几乎没有用户接口。

第二阶段：以嵌入式微控制器为基础、简单操作系统为核心的嵌入式系统。其通用性比较差，系统开销小，效率高；一般配备系统仿真器，操作系统具有一定的兼容性和扩展性；应用软件较专业，用户界面不够友好；系统主要用来控制系统负载以及监控应用程序运行。由于需要管理的资源有限，其操作系统代码量小，这些运行于嵌入式微控制器的嵌入式操作系统更强调于实时性而非界面，因而被广泛称为实时操作系统。其典型代表为 RTX51。

第三阶段：通用的嵌入式实时操作系统阶段，以嵌入式操作系统为核心的嵌入式系统，能运行于各种类型的嵌入式微处理器上，兼容性好；内核精小，效率高，具有高度的模块化和扩展性；具备文件和目录管理、设备支持、多任务、网络支持、图形窗口以及用户界面等功能；具有大量的应用程序接口（API）；嵌入式应用软件丰富。

第四阶段：以基于 Internet 为标志的嵌入式系统。这是一个正在迅速发展的阶段。目前大多数嵌入式系统还孤立于互联网之外，但随着 Internet 的发展以及 Internet 技术与信息家电、工业控制技术等结合日益密切，特别是物联网技术的出现，嵌入式设备与 Internet、通信网的融合将代表着嵌入式技术的真正未来。

下面介绍几种具有代表性的嵌入式操作系统。

1. VxWorks

VxWorks 操作系统是美国 WindRiver 公司于 1983 年设计开发的一种嵌入式实时操作系统（RTOS），是 Tornado 嵌入式开发环境的关键组成部分。良好的持续发展能力、高性能的内核以及友好的用户开发环境，使其在嵌入式实时操作系统领域占据一席之地。VxWorks 具有可裁剪微内核结构、高效的任務管理、靈活的任務間通信、微秒級的中斷處理、支持 POSIX 1003.1b 實時擴展標準、支持多種物理介質及標準的完整的 TCP/IP 網絡協議等，VxWorks 由於具有極高的實時性和可靠性，在工業控制領域廣泛應用。

然而，VxWorks 價格昂貴。由於操作系統本身以及開發環境都是專有的，價格一般都比較高。不提供源代碼，只提供二進制代碼。由於它們都是專用操作系統，需要專門的技術人員掌握開發技術和維護，所以軟件的開發和維護成本都非常高，並且支持的硬件數量也很有限。

2. Windows CE

Windows CE 與 Windows 系列有較好的兼容性，從而成為 Windows CE 的一大優勢。其中 WinCE 3.0 是一種針對小容量、移動式、智能化、32 位、模塊化的實時嵌入式操作系統。它能在多種處理器體系結構上運行，通常適用於那些對內存占用空間具有一定限制的设备。操作系統的基本內核需要至少 200KB 的 ROM。然而，從技術角度上講，Windows CE 作為嵌入式操作系統有很多的缺陷：沒有開放源代碼，使應用開發人員很難實現產品的定制；在



效率、功耗方面的表现并不出色，而且和 Windows 一样占用过多的系统内存，运行程序庞大；以及版权许可费等都是需要考虑的因素。

3. 嵌入式Linux

Linux 最大的特点是源代码公开并且遵循 GPL 协议，在近年来成为应用热点。由于其源代码公开，人们可以任意修改，以满足自己的应用，并且编码调试也很容易。遵从 GPL，无须为每例应用交纳许可证费。有大量的应用软件可用，由于开源和免费，使得软件的开发和维护成本很低。它还具有优秀的网络功能。从技术上来说，稳定性是 Linux 本身具备的一个很大优点，内核精悍，运行所需资源少，十分适合嵌入式应用。此外，它支持的硬件数量庞大，在通用市场上很有竞争力。

嵌入式 Linux 系统中常用的 BootLoader 有 ArmBoot、RedBoot、Blob、U-Boot 等。其中 ArmBoot 适合于采用 ARM 架构 CPU 的开发板，PPC-boot 用于采用 Power PC 架构 CPU 的开发板。目前这两种 BootLoader 都已停止开发。U-Boot 是 sourceforge 网站上的一个开放源代码的项目。它可支持 PowerPC MPC5xx、MPC8xx、MPC82xx、MPC7xx、MPC74xx、ARM (ARM7、ARM9、StrongARM) 处理器和 VxWorks、NetBSD、QNX、RTEMS、ARTOS、LynxOS 等嵌入式操作系统。事实上，U-Boot 对 Power PC 系列处理器的支持最完善，对 Linux 操作系统的支持最好。

嵌入式 Linux 也并非完美无缺。在嵌入式系统上运行 Linux 的一个缺点是 Linux 体系提供实时性能时需要添加实时软件模块，而这些模块运行的内核空间正是操作系统实现调度策略、硬件中断异常和执行程序的部分。由于这些实时软件模块是在内核空间运行的，因此代码错误可能会破坏操作系统，从而影响整个系统的可靠性。这对于实时应用将是一个非常严重的弱点。

4. μ C/OS-II

μ C/OS-II 是著名的源代码公开的实时内核，是专为嵌入式应用设计的，可用于 8 位、16 位和 32 位单片机或 DSP。它是在原版本 μ C/OS 的基础上做了重大改进与升级，并有了近十年的使用实践，有许多成功应用的实例。它具有如下主要特点：

- 1) 公开源代码。很容易将操作系统移植到各个不同的硬件平台上。
- 2) 可移植性。绝大部分源代码是用 C 语言编写的，便于移植到其他微处理器上。
- 3) 可固化。
- 4) 可裁剪性。有选择地使用需要的系统服务，可以减少所需的存储空间。
- 5) 抢占式。完全抢占式的实时内核，即总是运行在就绪条件下优先级最高的任务。
- 6) 多任务。可管理 64 个任务，任务的优先级必须是不同的，不支持时间片轮转调度法。
- 7) 可确定性。函数调用与服务的执行时间具有其可确定性，不依赖于任务的多少。
- 8) 实用性和可靠性。成功应用该实时内核的实例，是其实用性和可靠性的最好证据。

由于 μ C/OS-II 仅是一个实时内核，这就意味着它不像其他实时操作系统那样提供给用户一些 API 函数接口，而是需要用户自己去完成很多工作。

2.2.4 实验二 嵌入式平台基础实验

1. 实验目的

本实验主要搭建了一个基于 ARM7 下 μ Clinux 开发平台，并在 ARM 实验母板 (IOT-ARMMB) 上使用蜂鸣器、串口实现了一个简单的演示程序。该程序通过 μ Clinux 控制蜂鸣



器可以实现每隔 2s 鸣叫一次，并同时通过串口在控制台输出提示信息。通过该演示实验，可以使读者对开发板和 μClinux 有初步的了解，并掌握在 ARM7 上 μClinux 开发平台的构建和使用 GNU 工具链进行 μClinux 开发的步骤。

2. 实验设备

(1) 硬件

- ARM 实验母板 (IOT-ARMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- EasyJTAG-H 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- Linux 和 Windows 操作系统。
- H-JTAG。
- μClinux 2.4x 发行包和 arm-elf-gcc 交叉编译环境。

3. 实验原理

本实验的演示程序使用了 ARM 实验母板 (IOT-ARMMB) 的基本部件，其主要外围接口丰富，可以满足物联网相关实验的全部功能，开发板的实物照片如图 2-4 所示。

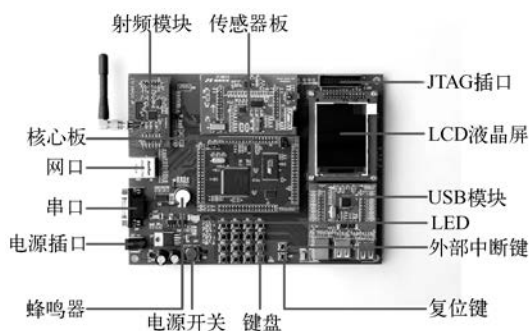


图 2-4 嵌入式 ARM 开发板实物照片

μClinux 对每个任务都会建立 3 个内部定时器。

ITIMER_REAL: 实时定时器，不管进程在何种模式下运行（甚至在进程被挂起时），它总在计数。定时到达，向进程发送 **SIGALRM** 信号。

ITIMER_VIRTUAL: 这个不是实时定时器，当进程在用户模式（即程序执行时）计算进程执行的时间。定时到达后向该进程发送 **SIGVTALRM** 信号。

ITIMER_PROF: 进程在用户模式（即程序执行时）和核心模式（即进程调度用时）均计数。定时到达产生 **SIGPROF** 信号。**ITIMER_PROF** 记录的时间比 **ITIMER_VIRTUAL** 多了进程调度所花的时间。

定时器在初始化时，被赋予一个初始值，随时间递减，递减至 0 后发出信号，同时恢复初始值。在任务中，可以使用一种或者全部 3 种定时器，但同一时刻同一类型的定时器只能使用一个。

在本程序中，使用了 **TIMER_PROF** 定时器，每秒产生一次定时信号，将蜂鸣器控制口



取反，也就是每隔 2s 蜂鸣器将会鸣叫一次，并输出提示信息。在程序的主函数 `main()` 中，首先通过调用 `init_sigaction()` 对 `struct sigaction` 类型的变量 `act` 进行设置，设定信号的处理函数，完成信号的初始化；然后通过调用 `init_time()` 对 `struct itimerval` 类型的变量 `val` 进行设置，设定定时值，完成定时器的初始化；紧接着启动定时器。相关代码如下：

```
//信号初始化
void init_sigaction(void)
{
    struct sigaction act;
    act.sa_handler = led_info;           //指定信号处理函数
    act.sa_flags = 0;
    sigemptyset(&act.sa_mask);         //将信号集初始化并清空
    sigaction(SIGPROF, &act, NULL);    //设定对 SIGPROF 的信号处理函数
}
//定时器初始化
void init_time(void)
{
    struct itimerval val;
    val.it_value.tv_sec = 1;           //定时时间为秒
    val.it_value.tv_usec = 0;         //定时时间为微秒
    val.it_interval = val.it_value;
    setitimer(ITIMER_PROF, &val, NULL); //启动定时器
}
//定时器初始化
int main(void)
{
    init_sigaction();
    init_time();
    while(1);                          //进入循环，等待蜂鸣器
    exit(0);
}
```

每当定时到达就产生 `SIGPROF` 信号，触发信号处理函数 `led_info(int signo)`，执行 GPIO 操作，打开 P0 引脚，并将蜂鸣器对应的 P0.7 引脚取反，相关的代码如下：

```
#define BEEP 7                          //P0.7 控制蜂鸣器
uint8 ledcon = 0;
//信号处理函数
void led_info(int signo)
{
    int fd;
    printf("timeout!\n");               //输出提示信息
    ledcon = ~ledcon;
    fd = open("/dev/P0", O_RDONLY);    //打开 P0 引脚
    if(fd == -1)
    {
```



```

printf("\nCan't open P0!\n");
exit(-1);
}
//取反蜂鸣器控制口
ioctl(fd, GPIO_SET_PIN_OUT, BEEP);
if (ledcon)
    ioctl(fd, GPIO_SET_PIN, BEEP);
else
    ioctl(fd, GPIO_CLR_PIN, BEEP);
close(fd);
}

```

4. 实验内容

构建 ARM 下的 μ Clinux 开发平台，初步掌握移植 μ Clinux 到基于 ARM 的嵌入式开发平台的一般步骤。

初步掌握 ARM 下 μ Clinux 程序设计方法和使用 GNU 工具链在交叉开发环境下进行 μ Clinux 开发的一般步骤，能够设计简单的程序操作开发板上的各个模块。

5. 实验步骤

(1) 建立交叉开发环境

由于嵌入式 Linux 的开发母板资源有限，不可能在开发母板上运行开发和调试工具。通常需要以交叉编译调试的方式进行，即“宿主机+目标机（开发母板）”的形式。目标机和宿主机一般采用串口连接，亦可同时通过网口连接，如图 2-5 所示。

宿主机是运行 Linux 的 PC，可以是安装了 Linux 操作系统的本地机，也可以是 Linux 服务器（模型假定其 IP 为 192.168.0.10）。宿主机和目标机的处理器通常情况下都不会相同。宿主机需要建立适合于目标机的交叉编译环境。程序在宿主机上编译—连接—定位，得到的可执行文件则在目标机运行。在开发调试过程中，通常的操作界面如图 2-6 所示。

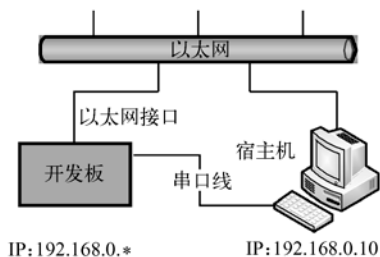


图 2-5 嵌入式 Linux 应用程序调试模型

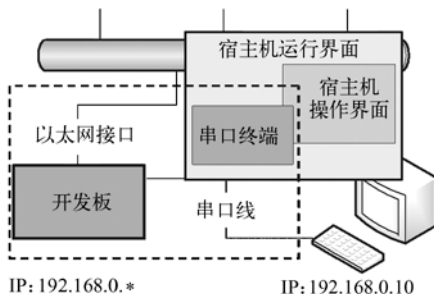


图 2-6 嵌入式 Linux 开发调试界面

在宿主机上一般需要运行两个窗口：宿主机本机操作窗口和串口终端窗口。宿主机操作窗口可以是本机的操作终端（安装 Linux 的本地机），也可以是 Windows 下通过远程登录软件（如 Telnet）登录到 Linux 服务器的操作界面。宿主机只能编译连接程序，不能运行产生的 Flat 文件（GCC 模式的交叉编译调试方式）。串口终端可以是 Linux 的 Minicom，也可以是 Windows 的超级终端。目标机可以被看成一台计算机，串口终端就相当于这台计算机的显示



器，作为人机交互界面。将在宿主机编译连接后得到的可执行文件下载到目标机上运行。

由于目标板平台处理器是 ARM，因而需要在宿主机上安装 GCC 针对 ARM 的编译器。适用于 μ Clinux 的编译器为 arm-elf-gcc。arm-elf-gcc 安装步骤：

将光盘附带的文件 arm-elf-tools-20100702.sh 复制到目录（如/usr/src）下，为 arm-elf-tools-20100702.sh 增加可执行权限（如果已经具有可执行权限，这步可省略）。注意：增加权限必须具有 root 权限，若没有 root 权限，请先使用 su 命令为用户增加 root 权限。增加权限的命令如下：

```
#chmod 755 arm-elf-tools-20100702.sh //命令提示符“#”表示 PC 上的 root 用户命令
```

安装 arm-elf-gcc。执行 arm-elf-tools-20100702.sh。

```
#!/arm-elf-tools-20100702.sh
```

安装完毕，查看/usr/local/bin 目录下是否存在 arm-elf-gcc 等文件，如果存在，基本可以确定交叉编译器安装成功。

添加交叉编译器的路径。一般情况下安装过程会自动执行，并在下次启动时仍有效。

```
#export PATH=$PATH:/usr/local/arm-elf/bin
```

（2）安装 μ Clinux

将光盘附带的 μ Clinux 源代码包 μ Clinux-dist-20040408.tar.gz 复制到目录（如/usr/src）下，并解压。

```
#tar xzvf uClinux-dist-20040408.tar.gz
```

解压后，在当前目录下会多出文件夹 μ Clinux-dist，即 μ Clinux 源代码目录。

为 IOT-ARMMB 开发母板和 LPC2200 芯片打补丁。将光盘附带的补丁文件 μ Clinux-dist-20040408-lpc-chy-cmj.patch 复制到当前目录下，并执行：

```
#cat uClinux-dist-20040408-lpc-chy-cmj.patch | patch -p1 -d uClinux-dist
```

为 LPC2200 打补丁后，在 μ Clinux 平台内核配置时将会有 LPC2200 的选项。

（3）配置 μ Clinux 和文件系统

新建终端（或者在控制台 \langle Ctrl $\rangle + \langle$ Alt $\rangle + \langle$ F1 $\rangle \sim \langle$ F6 \rangle 其中之一进入 6 个控制台中的 1 个），进入 μ Clinux 源代码所在的目录（如/usr/src/ μ Clinux-dist）。

```
# cd /usr/src/uClinux-dist
```

如果不是进行第一次编译，需要先清除以前编译产生的辅助文件和目标文件，以保证编译的正确性。如果是第一次编译则可省略。

```
# make distclean
```

通过文本菜单方式配置 μ Clinux。



```
# make menuconfig
```

执行“make menuconfig”进行厂商/产品、内核版本和函数库的设置。各选项的含义以及设置如表 2-1 和表 2-2 所示。

表 2-1 μ Clinux Vendor/Product 选择设置

选 项	描 述	设 置
Vendor	厂商选择	选择 PHILIPS
Product	产品选择	选择 lpc2200

表 2-2 μ Clinux Kernel/Library/Defaults 等的选择设置

选 项	描 述	设 置
Kernel Version	内核版本	选择 Linux-2.4.x
Libc Verson	函数库版本	选择 μ Clibc
Default all Settings	恢复默认设置	选择（保留补丁信息）
Customize Kernel Settings	定制内核	根据需要选择，暂时不用
Customize Vendor/User Settings	定制厂商和用户设置	根据需要选择，暂时不用
Update Default Vendor Settings	更改设置为默认厂商配置	根据需要选择，一般不用

按照上述表格内容进行设置完毕之后，选择 Default all Settings 将此设置作为默认设置，之后选择 Exit，然后在提示是否保存时选择保存。以后如果用户在配置过程中进行了不恰当的配置，导致编译出错，选择 Default all Settings，即可恢复保存的设置。选择该项，为 LPC2200 所打的补丁信息依然保留。

建立源代码文件依赖关系（第一次必须执行）：

```
# make dep
```

编译内核：

```
# make
```

这两步的时间都比较长。编译结束，会在 μ Clinux 安装目录下建立 romfs 和 images 两个目录。其中，romfs 为 romfs 文件系统的临时存放目录，里面有很多实用程序；images 为生成的目标代码目录，其中 romfs.img 为 romfs 文件系统的映像文件，image.bin 为文件系统和内核代码的目标文件。在 linux-2.4.x 目录下会有一个 Linux 文件，是编译好的内核文件，为 elf 格式。这些文件可用于进行 μ Clinux 内核实验。至此，宿主机的交叉编译环境已经建立完成。

制作 .bin 格式的内核文件 linux.bin：

```
# cd /usr/src/uclinux-dist/linux-2.4.x
#arm-elf-objcopy -O binary linux linux.bin
```

（4）烧写 BootLoader 和下载 μ Clinux 内核

烧写 BootLoader 需要在 Windows 环境下通过 H-JTAG 软件进行。



首先，将 EasyJTAG-H 编程器和开发母板相连。设置跳线 BANK0-Flash，BANK1-RAM。选择 LPC2200_flash.hfc 配置文件，然后打开 H-Flasher 的 Programming 选项，如图 2-7 所示。

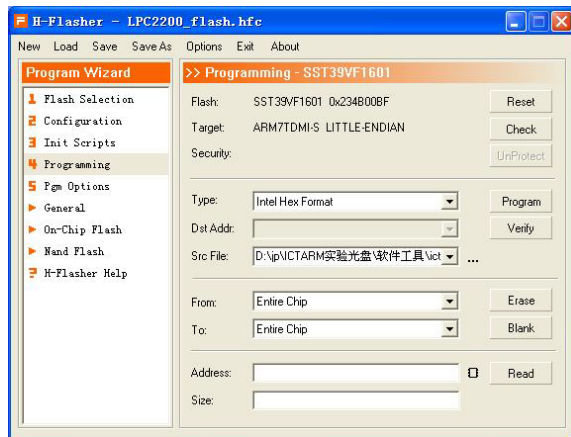


图 2-7 H-Flasher 烧录程序到目标地址

在 Type 下拉列表中选择 Intel Hex Format，如果是二进制的程序，则选择对应的二进制文件选项。

接着选择程序文件，单击 Src File 右边的按钮“...”，找到烧录的程序文件。

接下来，单击“Erase”按钮，擦除 Flash 内容（Nor Flash 必须在擦除后才能写入）。擦除完毕，单击“Program”按钮，等待几秒钟就可以将程序烧录到对应的地址中。

下载μClinux 内核既可以在 Windows 环境下进行，也可以在 Linux 环境下进行，这里介绍在 Linux 环境下的下载方法。

在 Linux 下，打开终端（也可在控制台进行），启动 minicom:

```
#minicom -s //加“-s”选项设置minicom
```

选择菜单中的“Serial Port Setup”，按〈Enter〉键，进入 minicom 操作菜单。进入“Serial Prot Setup”菜单，首先按具体硬件连接设置“Serial Device”（如果使用串口 1，则输入/dev/ttyS0，如果使用串口 2，则输入/dev/ttyS1，如果使用的是 USB 转串口，则输入/dev/ttyUSB0）；然后设置“bit/s/par/Bits”（波特率），将波特率设置为 115200bit/s；最后设置“Hardware Flow Control”为“NO”。

设置完毕，按〈Enter〉键返回到串口设置主菜单，选择“Save Setup as Dfl”，按〈Enter〉键保存刚才的设置。再选择“Exit”退出设置模式。刚才的设置被保存到“/etc/minirc.dfl”。

进行正确的串口设置后，将开发母板串口 0 和 PC 串口相连，按开发母板复位键 RST，将进入 ICT_BOOT 界面，如图 2-8 所示。

选择 1，则使用用户 IP 进行 FTP 连接；选择 2，使用默认 IP（192.168.15.115）进行 FTP 连接；输入 F，则对 NAND Flash 进行格式化。现在选择 2，使用默认 IP 进行 FTP 连接。



图 2-8 ICT_BOOT 启动界面

对宿主机进行网络设置，将 IP 设置为 192.168.41.42，将子网掩码设置为 255.255.0.0，将默认网关设置为 192.168.0.1（具体设置可不同，只要保证与 IOT-ARMMB 开发板的 IP 在同一网段内即可）。由于接下来要使用 NFS 服务，所以需要关闭宿主机的防火墙。

在 Linux 下，可以通过命令行和图形界面进行 FTP 下载。使用图形界面相对直观和简单，因而在这里介绍使用 gFTP 软件进行下载的方法。在 Linux 下新建终端，输入 gftp，启动 gFTP 软件。

```
#gftp
```

通过网线将开发母板接入局域网，或者通过网络交叉线直接将 IOT-ARMMB 开发母板和 PC 相连。在 gFTP 中进行 FTP 设置：在 Host 文本框中填入正确的 IP 地址，如在 ICT_BOOT 启动界面选择 2（使用默认 IP），则填入 192.168.15.115，并输入端口号 21。

进入光盘中存放内核和文件系统的目录，选择要下载的文件，然后单击传输按钮进行 FTP 上传，大约 1min 后，传输完毕。传送成功后的界面如图 2-9 所示。

下面在 Linux 环境下使用 minicom 连接开发板启动 μ CLinux。首先将开发母板的串口 0 和 PC 相连。仍然采用上文提到的方法对 minicom 进行设置。设置完毕，按复位键，启动开发母板，将得到如图 2-9 所示的界面。

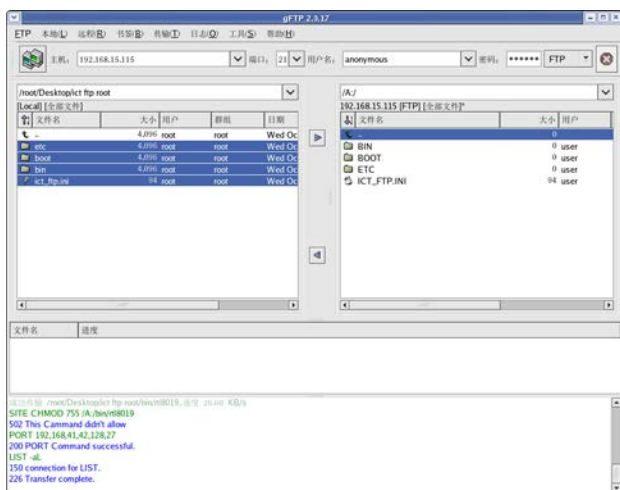


图 2-9 使用 gFTP 传输文件完成



选择 3，启动 μ Clinux，启动之后，会出现如图 2-10 所示的界面。

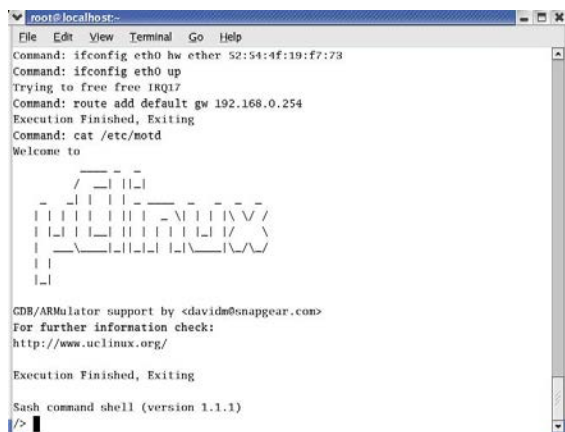


图 2-10 μ Clinux 启动界面

在命令提示符“/>”下输入 μ Clinux 支持的命令，即可对其进行操作。

(5) 配置 NFS 服务

在宿主机建立共享目录“/home/armwork”，修改/etc/exports 文件对 NFS 服务进行设置：

```
/home/armwork 192.168.*.*(rw, sync,no_root_squash)
```

设置共享目录为“/home/armwork”，设置允许访问 IP 为“192.168.*.*”，表示处于网段 192.168.x.x 的主机均可对共享目录“/home/armwork”进行访问，后面括号内为共享目录的权限设置，如“读/写”等，可以根据需要进行设置。最后输入如下命令重启 NFS 服务：

```
# services nfs restart
```

接着用开发板去 ping 服务器：

```
/> ping 192.168.15.3 //命令提示符“/>”表示开发板上的用户命令
```

确定 ping 通服务器后，在开发板上输入如下命令，即可将服务器上的/home/armwork 目录下的文件映射到开发板的/mnt 目录下：

```
/> mount -t nfs 192.168.15.3:/home/armwork /mnt -o nolock
```

下面来检测在开发板上是否映射成功，先在服务器上输入：

```
#cd /home/armwork  
#mkdir test  
#cd test  
#echo "this is a test file." > testfile
```

然后在开发板上输入如下命令即可验证映射是否成功。

```
/> cd /mnt
```




6. 思考题

- 1) μ Clinix 文件结构下的文件都可起到什么作用?
- 2) 在本开发套件中 BootLoader 都完成了什么功能, 应当如何实现?
- 3) 如何自己完成一个 GPIO 的驱动程序?
- 4) 如何自己写一个 Makefile 文件?

7. 补充阅读

- [1] 田泽. ARM7 μ Clinix 开发实验与实践[M]. 北京: 北京航空航天大学出版社, 2006.
- [2] Getting started μ Clinix with PC22xx. <http://www.cn.nxp.com/>.

2.3 智能移动终端平台

以智能手机、PDA 等为代表的便携式智能移动终端在近几年应用逐步加快, 已经成为消费类电子的重要组成部分, 其性能已经基本达到桌面系统水平同时其良好的用户界面、优越的应用软件及移动便携的产品形态, 也使其迅速成为大众消费品。智能移动终端可以方便地接入移动通信网和互联网, 将成为物联网的重要人机接口。本节以诺基亚、苹果等智能手机为例介绍智能终端平台, 并在 Nokia 的 Qt 平台上安排了一个实验。另外, 还结合 Android 平台, 在第 5 章安排了一个网络接入实验。

2.3.1 主流移动智能终端平台简介

近年来, 随着通信技术和计算机技术的飞速发展, 以及需求、竞争和科技三大动力的推动, 以语音、数据和视频为基础的新型电信业务层出不穷且发展迅猛, 最终对通信终端提出了迫切的需求。特别是随着 3G 网络环境的日趋成熟、移动业务的飞速发展以及第三方应用程序的快速增长, 移动终端的形式多样化, 功能增强化和使用普及化已是大势所趋, 越来越多的移动终端被人们广泛使用。智能终端是物联网和云计算环境中最重要的用户界面之一。

移动智能终端(以智能手机为例)的广泛使用促进了移动智能终端平台的快速发展。当前主要的移动智能终端平台包括 Symbian、Maemo、MeeGo、Windows Mobile、Android、Linux、Palm、iOS 等。下面集中对常见的系统进行介绍。

1. Symbian OS

进入智能手机的时代, Symbian 应该算是各平台的鼻祖。Symbian OS 是 Symbian 公司为手机设计的专用操作系统, 它包含联合的数据库、使用者界面架构和公共工具的参考实现, 其被 Nokia 收购之后, 将其转移到 Symbian 基金会, 以开放源代码的形式释出。它的前身是 Psion 的 EPOC, 并且独占式地执行于 ARM 处理器, 包含由 Symbian 公司所提供的相关函数库、用户界面架构和通用工具的参考实现。

Symbian OS 在移动智能终端上拥有强大的应用能力以及通信能力, 因为它具有非常健全的核心、强大的对象导向系统、企业用标准通信传输协议以及完美的 Sun Java 语言支持等。在 Symbian 看来, 无线通信装置除了要提供语音沟通能力外, 同时也应具有其他沟通方式, 如触笔、键盘、触摸等。在外观和硬件设计上, 它可以提供许多不同风格的外形, 像使用真实或虚拟的键盘; 在软件功能上可以容纳许多功能, 包括和他人互相分享信息、浏览网页、传输与接收电子邮件、传真以及个人生活行程管理等。此外, Symbian OS 提供了很好的可扩展



性, 这为各商家的自行扩展提供了条件; EPOC 操作系统还可以细分成 Pearl、Quartz、Crystal 3 种类型, 分别对应普通手机、智能手机、Hand Held PC 场合的应用。

Symbian OS 是一个实时的、多任务的 32 位操作系统, 具有功耗低、内存占用少等特点, 非常适合手机等移动设备使用, 随着其功能的不断完善, 目前已经支持 GPRS、蓝牙、WiFi 以及 3G 技术等。最重要的是它是一个标准化的开放式平台, 任何人都可以为支持 Symbian 的设备开发软件。与微软公司产品不同的是, Symbian 公司将移动设备的通用技术, 也就是操作系统的内核, 与图形用户界面技术分开, 能很好地适应不同方式输入的平台, 也使厂商可以为自己的产品制作更加友好的操作界面, 符合个性化的潮流, 这也是用户能见到不同样子的 Symbian 系统的主要原因。

2. Windows Phone 7

Windows Mobile 是 Windows CE 在移动终端发展的后期版本, 也是在手机上使用较为广泛的一个版本。Windows CE 操作系统是一个 32 位、多任务、多线程, 具有很好的可扩展性和开放性的嵌入式操作系统, 它对多种设备提供支持。Windows CE 可以应用在各种嵌入式系统中, 或是硬件规格比较低的计算机系统, 例如有限的内存配置、较低档的 CPU 等。Windows CE 还集成了电源管理功能, 使移动设备具有较长的待机时间。Windows CE 嵌入了标准通信协议, 支持 Internet 的接入、电子邮件的收发, 以及浏览万维网等。此外, Windows CE 还提供了友好、方便的图形化用户界面。

广为人知的采用 Windows CE 架构的操作系统是 Windows Mobile。Windows Mobile 是由微软公司为智能手机和移动设备开发的操作系统。Windows Mobile 分为 3 类, 分别是 Windows Mobile Classic、Windows Mobile Standard 和 Windows Mobile Professional。

为了让手机产品能够真正契合用户生活的脚步、满足社交需求并提供各种无缝体验, 微软公司最新推出了 Windows Phone 7 操作系统, 它是对 Windows Mobile 系统的重大突破, 让使用者有完全不一样的体验。全新的操作系统集成了 Xbox 的 Live 游戏部分、Zune 的音乐和视频部分的主要优势, 将这些高端技术第一次运用到移动平台上。另外, Windows Phone 7 加入 Bing 搜索, 提供专业的搜索服务, 在统一操作界面的同时, 微软还统一软件界面不再支持合作厂商间的深度定制。

Windows Phone 7 的用户界面与此前系统的界面完全不同, 采用了全新的流动体系设计, 用户不但可以根据自己的喜好来进行个性设计, 还可以随时更新界面设置, 能够更好地体现真正的一体化设计, 令网络内容和应用服务可以更好地呈现出来。例如, 在可自定义化的操作界面上实现简洁、即时的动态卡片效果, 用户在添加好友的同时便可获得这位好友的社交网站以及博客等全部信息。

Windows Phone 7 的功能集中在 6 大中心。People Hub (人物中心), 重点在于社交网络功能, 每一位联系人的通讯方式、照片、博客以及在社交网络上的最新更新都会在这部分随时更新; Pictures Hub (图片中心), 可以搜索和编辑、上传本地与网络上的照片及视频; Games Hub (游戏中心), 吸收了 Xbox Live 游戏平台的优势功能, 支持本地以及网络游戏; Music + Video Hub (影音中心), 融入了强大的 Zune 掌上媒体播放器的功能与界面, 可以直接播放本地、网络影音节目; Marketplace Hub (软件商店中心), 可搜索和下载 Windows Phone 第三方软件; Office Hub (办公中心), 例如 Office Mobile、Office SharePoint 和 Office OneNote 等软件的应用, 可以帮助用户更好地在手机上进行文档处理。



Windows Phone 7 对硬件配置要求较高, 支持多点触摸、WiFi、蓝牙以及 3G 等技术。它完全兼容 Silverlight 应用以及 XNA 框架的游戏开发, 其本身与微软公司的其他产品及服务都是紧密相连的, 汇集了微软公司的很多业务。其特点在于简单美观的界面、整合良好的 Office Mobile、顺畅的使用体验, 加上很酷的界面动画、整合社交网络以及云端服务 (像是 Facebook、Xbox Live、SkyDrive、Windows Live 等), 也能无线同步多媒体内容。

2011 年 2 月, 诺基亚公司与微软公司正式宣布, 双方将达成广泛的战略合作关系, 诺基亚高端手机将采用 Windows Phone 7 系统, 或许会在一定程度上提高 Windows Phone 7 的用户数量。

3. Android

Android 一词的本意是指“机器人”, 在 2007 年 11 月 5 日被广为人知。这一天 Google 公司推出了目前最为流行的开源手机操作系统 Android。Android 基于 Linux 的开放平台, 由操作系统、中间件、用户界面和应用软件组成, 号称是首个为移动智能终端打造的真正开放和完整的移动软件。Android 是由 Google 公司与开放手机联盟合作共同开发的, 这大大降低了新型手机设备的研发成本。这个联盟包括 Google、中国移动、摩托罗拉、HTC、T-Mobile、高通、三星和 LG 等公司。开放手机联盟表示, Android 平台可以促使移动设备的创新, 让用户体验到最优质的移动服务, 同时开发商也得到了一个新的开放级别, 更方便地进行协同工作, 进而保障新型移动设备的研发速度。因此, Android 系统可以被看做是第一个完整的、开放的、免费的移动智能终端平台。

Android 系统能够受到用户的青睐, 因为具有以下特点:

1) 开放性。Google 公司与开放手机联盟合作开发了 Android。Google 公司通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作关系, 希望通过建立标准化、开放式的移动电话软件平台, 在移动产业内形成一个开放式的生态系统。

2) 应用程序无界限。Android 上的应用程序可以通过标准 API 访问核心移动设备功能。通过互联网, 应用程序可以声明它们的功能可供其他应用程序使用。

3) 应用程序是在平等条件下创建的。移动设备上的应用程序可以被替换或扩展, 即使是拨号程序或主屏幕这样的核心组件。

4) 应用程序可以轻松嵌入网络。应用程序可以轻松嵌入 HTML、JavaScript 和样式表, 还可以通过 WebView 显示网络内容。

5) 应用程序可以并行运行。Android 是一种完整的多任务操作系统, 应用程序可以在其中并行运行。在后台运行时, 应用程序可以生成通知以引起注意。

4. iOS

iOS 是苹果公司开发的 iPhone 操作系统, 后来部署到 iPod Touch、iPad 和 Apple TV 上。但是 iOS 不被允许运行在第三方的硬件上。iOS 用户界面的开发设计基于直接操作的理念, 使用多点触摸技术。界面控制方法包括滑动、轻触开关和按键。它能够很快地响应用户的输入并提供一个绚丽的界面。与 iOS 的交互包括滑动、轻按、挤压以及旋转, 而且通过其内置的加速器, 可以令其旋转装置改变屏幕的方向, 这样的设计效果更加易于使用。

iPhone 和 iPod Touch 使用的是基于 ARM 的处理器, 与苹果公司的个人计算机所用的处理器不同, 因此 Mac OS 上的应用程序不能直接应用到 iOS 上, 需要对其进行重写, 然后才能在 iOS 上运行。从 iOS 2.0 开始, 第三方应用程序在获得审核后放到 App Store 进行发布和下载。同时, iOS 自带有丰富的应用程序, 例如 SMS、日历、照片、相机、Youtube、



股市、地图、天气、时间、计算器、备忘录、系统设置、iTunes、App Store、Game Center 以及网络通信等应用程序。

2.3.2 实验三 智能移动终端平台基础实验

1. 实验目的

本实验介绍了 Qt 应用程序框架。通过实验，使读者初步了解 Qt 的特点和基本开发流程，并能够编写一个简单的应用程序在仿真器上查看运行效果。

2. 实验设备

(1) 硬件

- PC 1 台。

(2) 软件

- Nokia Qt SDK。

3. 实验原理

Qt 是一个跨平台的 C++ 应用程序开发框架。它包括跨平台类库、集成开发工具和跨平台 IDE。使用 Qt 只需一次性开发应用程序，无须重新编写源代码，便可跨不同桌面和嵌入式操作系统部署这些应用程序。

Qt 最初由 Haavard Nord 和 Eirik Chambe-Eng 开发而成。1991 年，Haavard 和 Eirik 开始一起合作设计、编写 Qt 的核心类，并且在随后的一年中，Eirik 提出了“信号和槽”的设想这一简单有效的强大图形用户界面编程规范。信号和槽机制允许回调，并支持对象之间在彼此不知道对方信息的情况下进行合作，这使 Qt 非常适合于真正的组件编程。

之所以选择字母“Q”作为类前缀，是因为该字母在 Haavard 的 Emacs 字体中看起来非常漂亮，随后添加的字母“t”代表工具包 (toolkit)。

1995 年 5 月， TrollTech 公司公开发布 Qt 的第一个版本 Qt 0.90。从第一天起，Qt 就提供了两个版本的软件许可协议：商业许可协议版和自由软件许可协议版。1997 年，Matthias Ettrich 利用 Qt 开发 KDE 的决定，使 Qt 成为 Linux 环境下开发 C++ 图形用户界面的事实标准。

2000 年，公司发布了 Qt/Embedded Linux，用于 Linux 嵌入式设备。它提供了自己的窗口系统，并且可以作为 x11 的轻量级替代产品。同年底，发布了 Qtopia 的第一版，这是用于移动电话和掌上电脑的环境平台。

2005 年夏，Qt 4.0 发布，它大约有 500 个类和 9000 多个函数，比以往的任何一个版本都要全面和丰富。如今，Qt 已经超越了作为图形用户界面工具包的界限，逐渐成长为一个成熟的应用程序开发框架。

2008 年 6 月，TrollTech 公司被 Nokia 收购。目前，Qt 是 Nokia 的 Qt Development Frameworks 部门的产品。

Qt 支持的平台包括 Embedded Linux、Mac OS X、Windows、Linux/X11、Windows CE/Mobile、Symbian、Meego。使用 Qt 开发的程序包括 KDE、Google Earth、Opera、Skype 等。

下面介绍如何使用 Qt 编写一个应用程序，显示 LCD 数字，并且可以用滑动滑块的方式来改变数字。首先，介绍程序中涉及的，同时也是 Qt 中非常重要的 Q_OBJECT 宏，以及 Qt 的两个常用模块 QtCore 和 QtGui；然后，结合实验程序讲解 Qt 中的常用类以及信号与槽的概念；最后，给出使用方面的内容——建立一个完整 Qt 应用程序的方法。



(1) Q_OBJECT宏

对于所有定义了信号和槽的类，在类定义开始处都必须声明 Q_OBJECT 宏。一般建议不论是否定义了信号和槽，都在类定义开始处声明 Q_OBJECT 宏。

Q_OBJECT 宏提供了很多的 Qt 特色。当一个类声明了自己的信号与槽或者使用了元对象系统 (Meta-Object System) 所提供的其他服务 (如文本翻译) 时，Q_OBJECT 宏必须出现在类定义的私有区。

Q_OBJECT 宏声明了在每一个 QObject 子类中必须实现的一些内省函数：metaObject()、tr()等，Qt 的 moc 工具生成了用于由 Q_OBJECT 宏声明的所有函数和所有信号的实现。

在程序中的声明如下：

```
class MyWidget : public QWidget
{
    Q_OBJECT //宏
    ..... //其他语句
};
```

Qt 的主要成就之一就是使用了元对象机制对 C++进行了扩展，并且使用这种机制创建了独立的软件组件。这些组件可以绑定在一起，但任何一个组件对于它所要连接的组件的情况实现都一无所知。标准 C++没有对 Qt 的元对象系统所需要的动态元信息提供支持。Qt 通过提供一个独立的 moc 工具解决了这个问题，moc 解析了 Q_OBJECT 宏的定义并且通过 C++函数来提供可供使用的信息。

注意：Q_OBJECT 宏只能在 QObject 的子类中使用。如果不是 QObject 的子类，需要元对象系统支持时，可以使用 Q_GADGET 来代替 Q_OBJECT。

(2) Qt常用模块

Qt 4 提供了一些模块，它们以文件形式被分成了几个类库。最为重要的模块有 QtCore、QtGui、QtNetwork、QtOpenGL、QtScript、QtSql、QtSvg 和 QtXml 等。比较常用的有 QtCore 和 QtGui 模块。

- QtCore。QtCore 模块包含了核心的非 GUI (图形用户接口) 功能。所有其他 Qt 模块都依靠这个模块。
- QtGui。QtGui 模块扩展了 QtCore 在 GUI 方面的功能。在 <QtGui>头文件中为构成 QtCore 和 QtGui 组成部分的所有类进行了定义。其中包含了 QMainWindow、QDockWidget、QPushButton、QCanvas、QDialog 等窗口部件类，QLayout、QBoxLayout、QFormLayout 等布局管理类，更多其他的类请参考帮助文档。

需要使用模块中的类时，可以在程序开始包含相应的模块。

例如，需要使用 QtGui 模块时，只要在代码头包含 QtGui 模块：

```
#include <QtGui>
```

另外，也可以只包含模块中使用到的类，而不包含整个模块。在本实验中，主要使用的



模块如下：

```
#include <QFont>
#include <QLCDNumber>
#include <QPushButton>
#include <QSlider>
#include <QVBoxLayout>
#include <QWidget>
```

若在一些文件（如头文件）中，只用到对象的声明，而没有具体实现，可以使用前置声明的方式代替包含头文件，例如：

```
class QFont
class QLCDNumber
class QPushButton
class QSlider
class QVBoxLayout
class QWidget
```

（3）Qt中的类

为了适合用户的需求，Qt 的 API 提供了几百个 C++ 类，该类大多数用于专门的 GUI。Qt 还提供了基于模板的初始化、文件和通用的 I/O 设备、目录管理、日期/时间类、常用表达式解析等。目的是利用这些类，建立或生成不同的功能，用它们来实现 Qt 的通用化。下面通过本实验涉及的 4 个类（QWidget、QPushButton、QVBoxLayout、QApplication）使大家对 Qt 中类的特点和用法有个大致的了解。程序中还用到了 QLCDNumber 和 QSlider 这两个类，但是由于它们是和 QPushButton 类似的窗口部件类，故不重复说明，相应的内容可以查阅相关帮助文档。

1) QWidget。QWidget 是所有窗口部件（widget）类的基类，属于 QtGui 模块，继承自 QObject 和 QPaintDevice。它能够接收鼠标、键盘和其他事件。窗口部件都是矩形的，如果未嵌入父窗口部件，就被称为窗口（window）。

按钮、菜单、滚动条和框架都是窗口部件。窗口部件也可以包含其他窗口部件。例如，应用程序窗口这个部件，通常会包含一个 QMenuBar、几个 QToolBar、一个 QStatusBar 等其他窗口部件。

在本实验中，自定义了一个窗口部件类 MyWidget，它的一个对象 widget 作为应用程序的窗口，代码如下：

```
class MyWidget : public QWidget
{
    Q_OBJECT //宏
public:
    explicit MyWidget(QWidget *parent = 0); //构造函数,默认值为没有父对象
};
```

这个自定义的 MyWidget 类以公有方式继承了 QWidget。在开始时，声明了 Q_OBJECT



宏。然后，声明了一个构造函数，参数为父对象的指针，默认值为没有父对象。构造函数实现于另一个文件。在构造函数中，声明了 3 个部件对象：quit（QPushButton 类）、lcd（QLCDNumber 类）和 slider（QSlider 类），分别是退出按钮、液晶数字显示框和滑块。然后，通过信号与槽机制建立 lcd 和 slider 的连接。最后，声明了一个布局对象 layout（QVBoxLayout 类），对几个部件的布局进行设置。

2) QPushButton。QPushButton 属于 QtGui 模块，继承自 QAbstractButton，而 QAbstractButton 又继承自 QWidget，其功能是提供一个命令按钮。这也许是在图形用户接口中最常用的窗口部件，用户通过按下或者单击按钮来命令计算机应答或执行操作。

在本实验中，QPushButton 的使用如下：

```
QPushButton *quit=new QPushButton(tr("Quit")); //tr 用来翻译字符串成其他语言
quit->setFont(QFont("Times",18,QFont::Bold)); //设置字体
```

第一行声明了一个 QPushButton 类的指针 quit，并且调用了其构造函数，创建对象。该函数的原型如下：

```
QPushButton::QPushButton ( const QString & text, QWidget * parent = 0 );
```

第一个参数是一个字符串，是在命令按钮上显示的文字；第二个参数是父对象，默认值为 0，即没有父对象。

程序中，在传递字符串“Quit”时调用了 tr()函数，这是把字符串翻译成其他语言的标记。在每个 QObject 对象以及包含有 Q_OBJECT 宏的子类中都含有这个函数的声明。也许当前没有准备将程序立刻翻译成其他语言，但是对每个用户可见的字符串使用 tr()函数会提高程序的可扩展性，给以后的国际化带来便利。

第二行通过成员函数设置了字体。该成员函数由 QPushButton 从父类 QWidget 继承而来，函数原型如下：

```
void setFont(const QFont&)
```

其中，QFont 是 QtGui 模块中的一个类，它的一个常用构造函数的原型如下：

```
QFont ( const QString & family, int pointSize = -1, int weight = -1, bool italic = false )
```

第一个参数 family 为字体类型；第二个参数 pointSize 为字号，当为 0 或负数时，使用系统默认字体大小；第三个参数 weight 为字体的粗细，范围为 0~99；第四个参数 italic，是 bool 类型，若为 true，则为斜体，否则为默认值。在本实验中，使用 Times 字体，18 号，粗体，非斜体。

3) QVBoxLayout。VBoxLayout 类将窗口部件垂直排列，属于 QtGui 模块，继承自 QVBoxLayout，使用方法如下：

```
VBoxLayout *layout=new QVBoxLayout; //布局管理器
layout->addWidget(quit); //依次置入 3 个部件
layout->addWidget(lcd);
```



```
layout->addWidget(slider);
setLayout(layout); //在窗口安装布局管理器
```

首先，创建布局对象，然后将窗口部件依次加入布局对象，最后调用 `setLayout()` 函数将该布局对象 `QVBoxLayout` 安装到窗口部件 `MyWidget`。此时，布局对象中加入的窗口对象将自动以 `MyWidget` 为父对象。

4) `QApplication`。`QApplication` 类属于 `QtGui` 模块，继承自 `QCoreApplication`（属于 `QtCore` 模块）。`QApplication` 类管理 GUI 应用程序的控制流和主要设置。它包含主事件循环，同时处理应用程序的初始化和结束，提供会话管理。此外，`QApplication` 处理大多数的系统和应用程序的设置。

对于任何 GUI 程序，有且只能有一个 `QApplication` 对象。对于非 GUI 的 Qt 程序，使用 `QCoreApplication` 来替代，因为它不依赖 `QtGui` 库。

在本实验中，该对象的声明和使用如下：

```
QApplication app(argc,argv); //声明一个 QApplication 对象
MyWidget widget; //程序的主窗口
widget.show(); //显示主窗口
return app.exec(); //调用 QApplication 对象的成员函数 exec()
```

最后一行的 `exec()` 是 `QApplication` 的静态成员函数，调用后，进入主事件循环，等待用户的操作直到 `exit()` 被调用。主事件循环接收来自视窗系统的事件并发送给应用部件。用户的动作会让相应的响应程序生成一些事件，为了开始事件处理，必须要调用该函数。

(4) 信号和槽

信号和槽机制是 Qt 编程的基础。它可以让应用程序编程人员把这些互不了解的对象绑定在一起。槽和普通的 C++ 成员函数几乎是一样的——可以是虚函数，可以被重载，可以是公有的、保护的或者私有的，并且也可以被其他 C++ 成员函数直接调用，它们的参数可以是任意类型的。唯一不同的是：槽还可以和信号连接在一起，在这种情况下，每当发射这个信号时，就会自动调用这个槽。

在 Qt 中，通过调用 `QObject` 对象的 `connect()` 函数来将某个对象的信号与另外一个对象的槽函数相关联。

`connect()` 语句形式如下：

```
connect(sender, SIGNAL(signal),receiver, SLOT(slot));
```

这个函数建立了一个从 `sender` 对象的信号到 `receiver` 对象的槽的连接。

下面的例子建立了从 `slider` 的 `valueChanged()` 信号到 `lcd` 的 `display` 槽的连接：

```
QLCDNumber *lcd=new QLCDNumber(2); //显示 LCD 数字的部件
QSlider *slider=new QSlider(Qt::Horizontal); //水平滑动条
connect(slider,SIGNAL(valueChanged(int)),
        lcd,SLOT(display(int))); //数字随滑块改变而改变
```

Qt 的窗口部件通过发射信号来表明一个用户动作已经发生了或者是一个状态已经改变了。当用户滑动滑块 (`Slider`) 时，该滑块就会发射一个 `valueChanged()` 信号，连接着的



QLCDNumber 对象的 display()槽自动执行，显示改变后的数值。

(5) 程序的实现

本实验的 Qt 程序包含 3 个文件：mywidget.h、mywidget.cpp 和 main.cpp。

1) mywidget.h 声明了作为应用程序的窗口的类 MyWidget。

2) mywidget.cpp 实现了 mywidget.h 中声明的 MyWidget 类。在类的构造函数中，声明了 3 个部件对象 quit、lcd、slider，分别是退出按钮、液晶数字显示框、滑块，并且进行了相应的设置。lcd 和 slider 通过信号与槽的机制相联系。滑动滑块后，slider 发射信号 valueChanged()，lcd 的 display()槽自动执行，显示当前滑块位置对应的数字。退出按钮 quit 的功能也由该机制实现。在 3 个部件对象的声明与设置完成之后，声明了一个布局对象 layout。layout 依次加入部件对象后，在主窗口安装布局管理器 layout，完成布局设置。相应的代码已经在上面介绍，这里不再赘述。

3) main.cpp 中为 main()函数声明了一个 QApplication 对象和一个 MyWidget 对象，显示主窗口，通过调用 exec()函数将应用程序的控制权传递给 Qt，使程序进入事件循环状态，等候用户的操作。相应的代码在 QApplication 类的介绍中已经给出。

4. 实验内容

浏览 Qt 的官方网站 <http://qt.nokia.com/>，了解 Qt 的特点和应用。

下载 Nokia Qt SDK，并进行安装。阅读手册中的“Creating a Mobile Application with Nokia Qt SDK”及其他帮助文档，了解建立手机应用程序的步骤。

浏览 http://www.forum.nokia.com/Develop/Qt/Getting_started/，了解开发环境的基本使用方法，并阅读相关的 Qt 书籍或文档，了解 Qt 的编程方法和过程。

分析程序结构，通过 Qt Assistant 学习使用到的 QWidget、QPushButton、QVBoxLayout、QLCDNumber、QSlider、QApplication 等类，了解各自的用法、成员函数等。学习 Qt 的信号与槽机制，了解将用户的动作和一段代码连接起来的方法。编写 Qt 程序，在仿真器上查看。

5. 实验步骤

1) 安装开发包 Nokia Qt SDK。

2) 启动 Qt Creator。

3) File→New File or Project→Qt C++ Project→Mobile Qt Application。单击“Choose”按钮。

4) Location 设置。在 D 盘根目录下新建一个 projects 文件夹，用来保存新建的项目。在“Name”文本框中输入“mywidget”，Create in 通过“Browse”按钮找到 projects 文件夹，单击“Next”按钮。

5) Qt Versions 设置。确保 Qt Simulator 复选框是选中的。单击“Next”按钮。

6) Details 设置。在“Class Name”文本框中输入“MyWidget”，Base Class 选择“QWidget”，表示类 MyWidget 从 QWidget 继承。取消选中“Generate Form”复选框。单击“Next”按钮。

7) Summary。显示生成的工程中所包含的文件：main.cpp、mywidget.cpp、mywidget.h、mywidget.pro。如果没有问题，单击“Finish”按钮；否则，可以返回修改。

8) 单击左侧工具栏中的“Edit”按钮，进入编辑模式，可以查看和编辑工程中的各文件。

9) 单击左侧下方工具栏中的“Build All”按钮，如果有错误对程序进行调试。



10) 单击左侧工具栏中的手机形状的图标，将编译目标选择为“Qt Simulator”。

11) 单击左侧工具栏中绿色的“Run”按钮，程序在仿真器上运行，弹出如图 2-12a 所示的窗口。

12) 滑块在最左边时，显示的是数值 0，向右滑动，数值逐渐变大，到最右边时，显示的是数值 99。单击“Quit”按钮，则退出程序，同时弹出“Qt Simulator Control”对话框，可以进行一些相应的设置，如 View 下的设备选择、旋转、缩放等操作。图 2-12b 是将 Device 选项由 Maemo Fremantle 改为 Symbian^3，并单击 Rotate Device 后的效果。

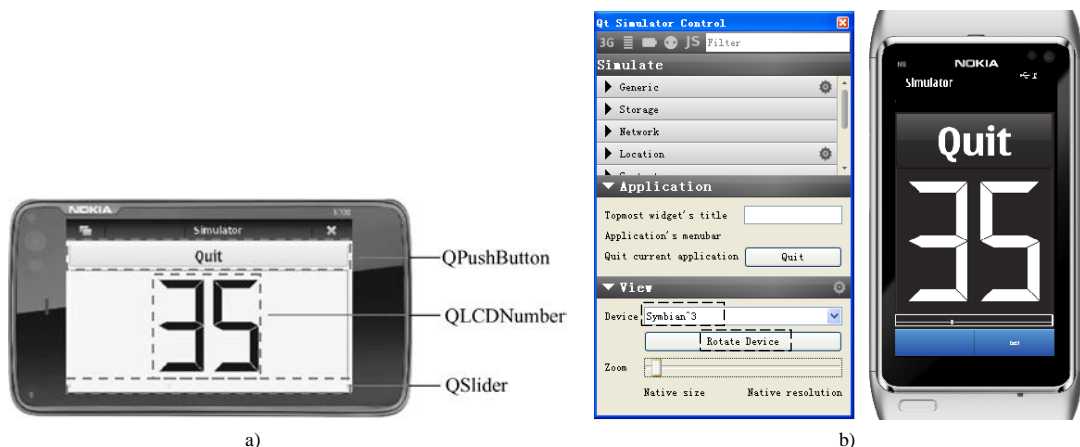


图 2-12 Qt Simulator 实验结果

a) Simulator 窗口 b) Qt Simulator Control 对话框

6. 思考题

- 1) QtCreator、QtDesigner、QtAssistant 等各有什么用途？
- 2) 如何使用 QtDesigner 实现图形界面设计？
- 3) 如何在命令行下使用 qmake 编译？
- 4) 信号与槽机制和普通的函数调用有什么联系和区别？

7. 补充阅读

- [1] 布兰切特, 萨摩菲尔德. C++ GUI Qt 4 编程[M]. 2 版. 闫锋欣, 等译. 北京: 电子工业出版社, 2008.
- [2] 丁林松, 黄丽琴. Qt 4 图形设计与嵌入式开发[M]. 北京: 人民邮电出版社, 2009.
- [3] Qt Reference Documentation. <http://doc.trolltech.com/4.6/>.

第3章 物联网感知与标识实验

感知和标识是物联网与物理世界的主要界面。通过感知技术，物理世界的信息变成了电气信号，并进一步转化为数字信息，通过物联网的信息标识，将这些物理的信息变成在整个物联网中唯一识别的数据。物联网的感知和标识主要以传感器技术与 RFID 技术为基础。感知和标识是物联网规模化、行业化的重要基础。本章将以单片机平台为主介绍感知与标识实验的原理及方法。当然，这里所有的实验都可以在嵌入式平台（有两个实验在不同平台上分别实现，以供选修），甚至可以在智能终端平台上完成，读者完全可以自行完成。

3.1 感知与标识实验概述

感知和标识是物联网深入物理世界的第一步，是物联网中海量信息的源头，也是物联网感知互动层的最主要的技术。目前，我国在传感器核心技术、芯片、标准等方面缺少自主知识产权，其高端产品及市场也一直由国外企业所控制，种类繁多的传感器在工艺、形态、性能、价格等方面缺少统一规范，不利于物联网行业间的互联互通，甚至由于缺乏统一管理而影响到国家安全。因此，很多科研机构和企业已致力于这方面的开发。

传感器是物联网感知的核心内容之一，物联网标识技术是以 RFID 标识为基础的。在这一层次的主要难点是研发高精度、低成本、低功耗、稳定可靠的智能数字传感器以及物联网规模化发展所需的物体标识和通信标识系统，进一步融合及适当兼容现有国内外各种传感器和标识方法，构建物联网的标准化传感器、智能感知、物体标识、通信标识和应用标识，构建我国自主的物联网的统一传感器行业规范和标识解析体系。

下面结合目前较常用的几类传感器和 RFID 方案介绍物联网感知与标识层的实验。实验本身难度不大，希望读者能够将本章实验扩充到更复杂的传感器系统中，并深入各类 RFID 系统的协议内部，这对设计实际系统将有很大的帮助。

3.2 典型传感器感知实验

传感器是通过敏感元件，把外部物理世界的信息按当量比例转换成电信号的一种重要的电子部件。传感器种类繁多，按照其敏感机理、敏感材料、工艺设备和计测技术都会衍生出不同种类的传感器，而不同行业对同类传感器的性能要求也是千差万别。我国将传感器分为 10 大类、42 小类近 6000 个品种，而国外品种更多（美国约有 1.7 万种）。本节结合常用的传感器介绍传感器的实验原理和方法，主要包括高精度温湿度传感器实验、运动感知实验、图像采集实验、步进电动机控制实验、键盘与显示屏交互实验等，至于其他种类的传感器，完全可以根据这些实验举一反三。



3.2.1 实验四A 单片机平台高精度温湿度传感器实验

1. 实验目的

本实验使用单片机 ATmega128 来控制高精度温湿度传感器 SHT10 获取周围环境的温度和湿度，并将温湿度数据发送到计算机。通过实验，使读者初步掌握 ATmega128 的基本编程方法，并掌握高精度温湿度传感器 SHT10 的使用方法。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- 串口调试助手。
- WinAVR20050214。
- AVRStudio4.18。

3. 实验原理

本实验基于 ATmega128 实现一个高精度温湿度数据采集程序，主要硬件平台为单片机实验母板 (IOT-SCMMB)，以及相关的传感器和外围模块。程序主要完成的工作包括：首先，单片机发送命令给高精度传感器，高精度传感器再根据单片机命令，执行温度或湿度的采集；然后，单片机读出相应的温度或湿度的数据，并把相关数据通过单片机的串口发送到 PC；最后，通过在 PC 上的串口调试助手，获取采集到的温度和湿度数据，并计算转化为具体的温湿度值。为了便于理解，将实验内容分为 3 个部分：各模块的初始化，单片机控制传感器采集温度和湿度，把采集的数据通过串口发送到 PC 上。下面分别对以上内容进行介绍。

(1) 各模块的初始化

由于串口的初始化在实验一已经介绍，这里主要介绍高精度传感器的初始化。因为高精度温湿度传感器只需要将 DATA 引脚和 CLK 引脚与单片机相连，所以使用单片机的 PC6 和 PC7 引脚分别连接传感器的 CLK 和 DATA 口，并且只要单片机给 PC6 和 PC7 发送相应的时序就能控制高精度温湿度传感器。单片机对传感器 I/O 口的初始化代码如下：

```
void initIO_sensor(void)
{
    DDRC = (1<<7) | (1<<6); //PC6 用做 CLK 信号的发送, PC7 用做 DATA 信号的发送
    PORTC = (1<<7) | (1<<6); //单片机的 PC6 和 PC7 用做输出引脚, 且为高电平
}
```

(2) 高精度温度和湿度数据的采集

数字高精度温湿度传感器 SHT10 有 4 个引脚：GND、DATA、SCK、VDD。图 3-1 为 SHT10 与单片机 (即所使用的 ATmega128) 的连接方式。

在本实验中，SHT10 的 SCK 和 DATA 引脚与 ATmega128 的 PC6 和 PC7 引脚相连。因此，只要通过单片机的 PC6 和 PC7 引脚向 SHT10 发送相应的时序，就能驱动 SHT10 进行采



样，并返回采样数据。注意如图 3-1 所示的 DATA 引脚要接上拉电阻。

要驱动 SHT10 进行采样必须发送如下命令：首先，单片机向 SHT10 发送“启动传输”时序，完成数据传输的初始化。如图 3-2 所示，时序包括当 SCK 时钟高电平时 DATA 翻转为低电平，紧接着 SCK 变为低电平，随后在 SCK 时钟高电平时 DATA 翻转为高电平。初始化之后，单片机便可以向 SHT10 发送命令。通常的命令包含 3 个地址位（目前只支持“000”）和 5 个命令位，具体将在后面结合代码进行介绍。SHT10 会以下述方式表示已正确地接收到指令：在第 8 个 SCK 时钟的下降沿之后，将 DATA 下拉为低电平，并且在第 9 个 SCK 时钟的下降沿之后，将 DATA 位恢复为高电平。

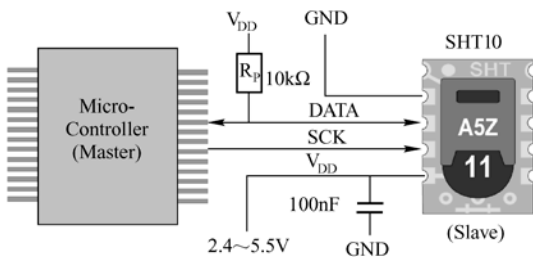


图 3-1 SHT10 与单片机连接方式

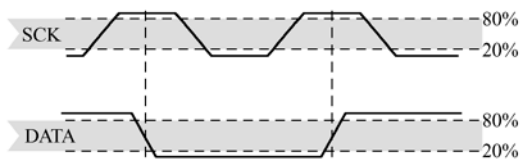


图 3-2 SHT10 启动传输时序图

启动传输程序如下：

```
void s_transstart(void)
{
    PORTC |= (1<<7);           //用做 DATA 引脚，初始状态置为高电平
    delay();
    PORTC &= ~(1<<6);         //用做 SCK 引脚，初始状态置为低电平
    delay();
    PORTC |= (1<<6);          //置 SCK 为高电平
    delay();
    PORTC &= ~(1<<7);         //当 SCK 为高电平时，DATA 翻转为低电平
    delay();
    PORTC &= ~(1<<6);         //置 SCK 为低电平
    delay();
    PORTC |= (1<<6);          //置 SCK 为高电平
    delay();
    PORTC |= (1<<7);          //当 SCK 为高电平时，DATA 从低电平翻转为高电平
    delay();
    PORTC &= ~(1<<6);         //置 SCK 为低电平
    delay();
}
```

启动时序完成之后，SHT10 便会以串行数据方式与单片机进行通信，时序图如图 3-3 所示。DATA 三态门用于数据的读取，DATA 在 SCK 时钟下降沿之后改变状态，并仅在 SCK 时钟上升沿有效。数据传输期间，在 SCK 时钟高电平时，DATA 必须保持稳定。为避免信号冲突，单片机应驱动 DATA 在低电平。

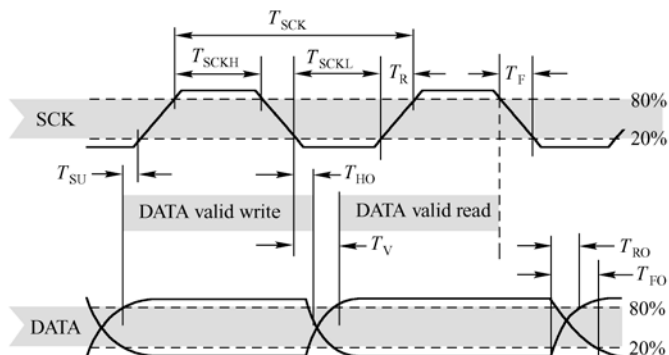


图 3-3 SHT10 与单片机通信时序图

给传感器发送数据需要注意时序，在上升沿之前把数据写入，上升沿时数据有效，在下降沿时把数据发送给传感器。数据发送的相关代码如下：

```
char s_write_byte(unsigned char value)
{
    unsigned char i,error = 0;
    for (i = 0x80; i > 0; i /= 2)           //传送 8B 的数据
    {
        if (i & value)                     //判断要发送的数据是 1 还是 0
        {
            PORTC |= (1<<7);               //发送数据 1
        }
        else
        {
            PORTC &= ~(1<<7);              //发送数据 0
        }
        //置 SCK 位为高电平，上升沿有效，此时数据已准备好
        PORTC |= (1<<6);
        delay();
        //置 SCK 位为低电平，下降沿打入数据，此时 I/O 口的数据发送到传感器
        PORTC &= ~(1<<6);
        delay();
    }
    PORTC |= (1<<7);                         //数据发送完毕，置 DATA 位为高电平
    delay();
    PORTC |= (1<<6);                         //置 SCK 位为高电平，准备读取数据
    DDRC &= ~(1<<7);                         //改变 DATA 的引脚为输入引脚，读取数据
    delay();
    error = PINC >> 7;                       //检查 DATA 引脚是否收到应答
    DDRC |= (1<<7);                         //置 DATA 位为输出引脚
    PORTC &= ~(1<<6);                       //置 SCK 为低电平
    delay();
    return error;
}
```



}

从传感器读数据，同样也要注意时序，只有在上升沿数据有效之后才能读。读数据的相关代码如下：

```
char s_read_byte(unsigned char ack)
{
    unsigned char i,val=0;
    PORTC |= (1<<7);           //置 DATA 位为高电平，释放 DATA 位
    DDRC &= ~(1<<7);          //改变 DATA 的引脚为输入引脚
    for (i = 0x80; i > 0; i /= 2)
    {
        PORTC |= (1<<6);      //DATA 在上升沿时读出数据
        if (PINC >> 7)
        {
            val=(val | i);    //读 DATA 位
        }
        PORTC &= ~(1<<6);     //置 SCK 位为 0，在下降沿时芯片给单片机打入数据
    }
    if (ack)                   //判断是否需要发送应答
    {
        PORTC &= ~(1<<7);     //发送应答
    }
    else
    {
        PORTC |= (1<<7);      //不需要发送应答
    }
    PORTC |= (1<<6);           //置 SCK 位为高电平
    DDRC |= (1<<7);           //改变 DATA 的引脚为输入引脚
    PORTC &= ~(1<<7);         //置 DATA 位为低电平，发送应答
    delay();
    PORTC &= ~(1<<6);         //置 SCK 为低电平，发送数据
    DDRC |= (1<<7);           //置 DATA 位为输出引脚
    PORTC |= (1<<7);          //置 DATA 位为高电平
    return val;
}
```

温湿度测量的过程如下：单片机发布一组测量命令后（“00000101”表示测量相对湿度，“00000011”表示测量温度），传感器开始采集数据。对应 8/12/14bit 3 种不同的测量，这个过程分别需要大约 20/80/320ms。SHT10 通过下拉 DATA 至低电平并进入空闲模式，表示测量的结束。单片机在触发 SCK 时钟来读出数据前，必须等待这个“数据备妥”信号。检测数据可以先被存储，这样单片机可以继续执行其他任务，在需要时再读出数据。在收到“数据备妥”信号之后，传输 2B 的测量数据和 1B 的 CRC 奇偶校验。单片机需要通过下拉 DATA 为低电平，以确认每字节。在测量和通信结束后，SHT10 自动转入休眠模式。温湿度测量的相关代码如下：



```

char s_measure(unsigned char *p_value, unsigned char *p_checksum,
               unsigned char mode)
{
    unsigned error=0;
    s_transstart();           //启动传输
    switch(mode)              //判断发送的是温度还是湿度的测量命令
    {
        case TEMP:
            error += s_write_byte(MEASURE_TEMP); //发送测量温度命令给传感器
            break;
        case HUMI:
            error += s_write_byte(MEASURE_HUMI); //发送测量湿度命令给传感器
            break;
        default:
            break;
    }
    DDRC &= ~(1<<7);         //改变 DATA 引脚为输入引脚
    while(1)
    {
        if(!(PINC >> 7))     //等待测量完成
        {
            break;
        }
    }
    if(PINC >> 7)             //DATA 为高电平, 说明有错误发生
    {
        error += 1;         //error 变量累加, 用于统计发生多少错误
    }
    DDRC |= (1<<7);         //改变 DATA 引脚为输出引脚
    //读测量值
    *(p_value) = s_read_byte(ACK);
    *(p_value + 1) = s_read_byte(ACK);
    *p_checksum = s_read_byte(noACK); //读校验码
    return error;
}

```

(3) 把采集的数据通过串口发送到PC上

单片机在获取了温湿度数据之后, 通过串口将数据发送给 PC。从串口获取数据的函数为:

```

unsigned char getc_uart0(void)
{
    while( !(UCSR0A & (1<<RXC0)) ); //接收完成
    return UDR0;
}

```

将数据发送到串口的函数为:



```
void putc_uart0(unsigned char c)
{
    while( !(UCSROA & (1<<UDRE0)) );           //数据发送已准备好
    UDR0=c;
}
```

以上是此实验的各个模块的分析，下面介绍实验的流程。首先，主函数初始化各模块；然后，单片机给高精度传感器发送命令，高精度传感器把采集的数据发送给单片机；最后，单片机通过串口把数据发送到 PC。下面是 main()函数的主要代码：

```
int main(void)
{
    init();                                     //初始化各模块
    delay();
    while(1)
    {
        s_connectionreset();
        s_measure(temp2,&status,0);           //发送温度采集命令
        temp=temp2[1];                         //保存采集的温度数据
        temp+=(temp2[0]<<8);
        s_measure(humid2,&status,1);         //发送湿度采集命令
        humid=humid2[1];                       //保存采集的湿度数据
        humid+=(humid2[0]<<8);
        printf("read digital tmep is %d \n",temp); //打印温度数据
        printf("read digital humid is %d \n",humid); //打印湿度数据
        for(a = 20;a > 0;a--)
            mydelay();
    }
}
```

4. 实验内容

阅读高精度温湿度传感器 **STH10** 芯片的数据手册，学会阅读并分析芯片的时序图，并能够依照时序图，编写相关的驱动程序，读出传感器的温度和湿度数据，然后通过串口把数据发送到 PC。

5. 实验步骤

- 1) 编写实验代码，在 WinAVR 环境下编译得到 .hex 文件。
- 2) 使用 AVR Studio 将 .hex 文件下载到开发板上。
- 3) 把开发板的串口与 PC 相连，使用串口调试助手获得节点发回的温湿度数据。波特率 9600bit/s，串口调试助手会显示：

```
read digital temp is X
read digital humid is Y
```

其中的 X 和 Y 分别为传感器的采样数据，这个数据必须通过芯片数据手册上的公式进行计算才能获得最后的温湿度值。



计算温度的公式为:

$$\text{Temperature}=d_1+d_2\times X$$

式中, d_1 , d_2 为参数; X 为传感器采集的温度数据。

对于本实验采用 3.5V电压、14 位采样精度, 查芯片手册可知: $d_1=-39.67$, $d_2=0.01$ 。代入计算即可得到以摄氏度为单位的温度值。

计算湿度的公式为:

$$\text{Humidity}=c_1+c_2\times Y+c_3\times Y^2$$

式中, c_1 , c_2 , c_3 为参数; Y 为传感器采集的湿度数据。

对于本实验采用 12 位采样精度, 查芯片手册可知: $c_1=-4$, $c_2=0.0405$, $c_3=-2.8\times 10^{-6}$ 。代入计算即可得到湿度百分比。

6. 思考题

本程序采样精度为 8 位温度, 14 位湿度, 如何修改程序能够使精度变为 8 位湿度、12 位温度。

7. 补充阅读

[1] 徐勇军, 安竹林, 等. 无线传感器网络实验教程[M]. 北京: 北京理工大学出版社, 2007.

[2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.

[3] SHT10 Datasheet. <http://www.sensirion.com/>. SENSIRION, 2009.

3.2.2 实验四B 嵌入式平台高精度温湿度传感器实验

1. 实验目的

本实验介绍在 μ Clinux 环境下基于 General Purpose Input/Output (GPIO) 驱动程序编写嵌入式应用程序。程序使用 LPC2220 微控制器来控制高精度温湿度传感器 SHT10 获取周围环境的温度和湿度, 并将温湿度数据通过串口传送到 PC 的超级终端上进行显示。通过实验, 使读者初步掌握 μ Clinux 应用程序的编写方法, 并掌握 GPIO 驱动和高精度温湿度传感器 SHT10 的使用方法。

2. 实验设备

(1) 硬件

- ARM 实验母板 (IOT-ARMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- PC (含串口) 1 台。

(2) 软件

- Linux 和 Windows 操作系统。
- H-JTAG。
- μ Clinux 2.4x 发行包和 arm-elf-gcc 交叉编译环境。

3. 实验原理

本实验基于 LPC2220 嵌入式微控制器实现一个高精度温湿度数据采集程序, 主要硬件为 ARM 实验母板 (IOT-ARMMB) 及传感器和其他外围模块。与本实验教程实验四 A 中通过单片机实现的高精度温湿度实验不同, 本实验基于 Linux 的 GPIO 驱动程序。有别于单片机需要将程序下载到 Flash 的烦琐过程, 本实验基于 NFS (网络文件系统) 建立的交叉编译



环境，可以将主机中的程序读取到实验板的 RAM 中运行。NFS 是 FreeBSD 支持的文件系统中的一种，其允许一个系统在网络上与其他系统共享目录和文件。通过网络接口将主机与实验板连接，NFS 使用户和程序可以像访问本地文件一样访问远端系统上的文件。这样，在程序的修改调试中，就不需要反复的下载过程。实验板可以随时通过 NFS 运行主机上最新版本的程序。另外，读者可以通过本实验进一步熟悉 Linux 下字符型设备的驱动方式。

程序主要完成的工作包括：首先，微控制器发送命令给高精度温湿度传感器，高精度温湿度传感器根据微控制器的命令，执行温度或湿度的采集；然后，微控制器读出相应的温度或湿度的数据，并把相关数据发送到 PC 的串口上；最后，通过 PC 上的超级终端，显示采集到的温度和湿度数据。为了便于理解，将实验内容分为 3 个部分：初始化，微控制器采集温度和湿度，将采集到的数值转换为实际温湿度值并将其通过串口发送到 PC 的超级终端显示。下面分别对以上的内容进行介绍。

(1) 初始化

这部分主要是对微控制器和高精度温湿度传感器的连接引脚初始化。因为高精度温湿度传感器只通过 DATA 和 SCK 这两个引脚与微控制器通信，所以使用微控制器的 P0.25 和 P2.20 引脚分别连接传感器的 DATA 和 SCK 口。由于 μ Clinux 将所有的设备都当做文件进行处理，因此对微控制器的 P0.25 和 P2.20 引脚的处理是使用 GPIO 的相关 I/O 函数进行操作（详见 GPIO 驱动程序）。初始化代码如下：

```
void initIO_sensor(void)
{
    fd1 = open("/dev/P0",O_RDONLY);           //以只读方式打开 P0 设备文件
    if(fd1 == -1)
    {
        printf("\nCan't open P0!\n");
        exit(-1);
    }
    fd2 = open("/dev/P2",O_RDONLY);           //以只读方式打开 P2 设备文件
    if(fd2 == -1)
    {
        printf("\nCan't open P2!\n");
        exit(-1);
    }
    ioctl(fd1, GPIO_SET_PIN_OUT, 25);        //设置 P0.25 为输出
    ioctl(fd2, GPIO_SET_PIN_OUT, 20);        //设置 P2.20 为输出
    ioctl(fd1, GPIO_SET_PIN, 25);           //设置 P0.25 为高电平
    ioctl(fd2, GPIO_SET_PIN, 20);           //设置 P2.20 为高电平
}
```

(2) 高精度温度和湿度数据的采集

数字高精度温湿度传感器 SHT10 有 4 个引脚：GND、DATA、SCK、VDD。图 3-4 所示为 SHT10 与微控制器的连接方式。

在本实验中，SHT10 的 DATA 和 SCK 引脚与微控制器的 P0.25 和 P2.20 引脚相连。因此，只要通过微控制器的 P0.25 和 P2.20 引脚向 SHT10 发送相应的时序，就能驱动 SHT10



进行采样，并返回采样数据。注意 DATA 引脚要接上拉电阻。

要驱动 SHT10 进行采样必须发送如下命令：首先，微控制器向 SHT10 发送“启动传输”时序，来完成数据传输的初始化。如图 3-5 所示，时序包括当 SCK 时钟高电平时 DATA 翻转为低电平，紧接着 SCK 变为低电平，随后在 SCK 时钟高电平时 DATA 翻转为高电平。初始化之后，微控制器便可以向 SHT10 发送命令。通常的命令包含 3 个地址位（目前只支持“000”）和 5 个命令位，具体将在后面结合代码进行介绍。SHT10 会以下述方式表示已正确地接收到指令：在第 8 个 SCK 时钟的下降沿之后，将 DATA 下拉为低电平，并且在第 9 个 SCK 时钟的下降沿之后，将 DATA 位恢复为高电平。

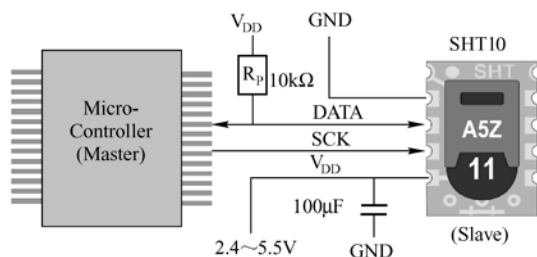


图 3-4 SHT10 与微控制器的连接方式

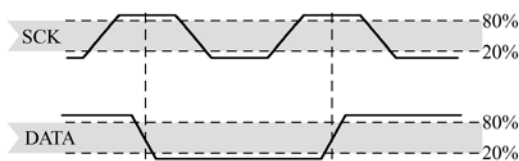


图 3-5 SHT10 启动传输时序图

启动传输程序代码如下：

```
void s_transstart(void)
{
    ioctl(fd1, GPIO_SET_PIN, 25); //用做 DATA 引脚，初始状态置为高电平
    delay();
    ioctl(fd2, GPIO_CLR_PIN, 20); //用做 SCK 引脚，初始状态置为低电平
    delay();
    ioctl(fd2, GPIO_SET_PIN, 20); //置 SCK 为高电平
    delay();
    ioctl(fd1, GPIO_CLR_PIN, 25); //当 SCK 为高电平时，DATA 翻转为低电平
    delay();
    ioctl(fd2, GPIO_CLR_PIN, 20); //置 SCK 为低电平
    delay();
    ioctl(fd2, GPIO_SET_PIN, 20); //置 SCK 为高电平
    delay();
    ioctl(fd1, GPIO_SET_PIN, 25); //当 SCK 为高电平时，DATA 从低电平翻转为高电平
    delay();
    ioctl(fd2, GPIO_CLR_PIN, 20); //置 SCK 为低电平
    delay();
}
```

启动时序完成之后，SHT10 便会以串行数据方式与微控制器进行通信，时序图如图 3-6 所示。DATA 三态门用于数据的读取，DATA 在 SCK 时钟下降沿之后改变状态，并仅在 SCK 时钟上升沿有效。数据传输期间，在 SCK 时钟高电平时，DATA 必须保持稳定。为避免信号冲突，微处理器应保持 DATA 在低电平。

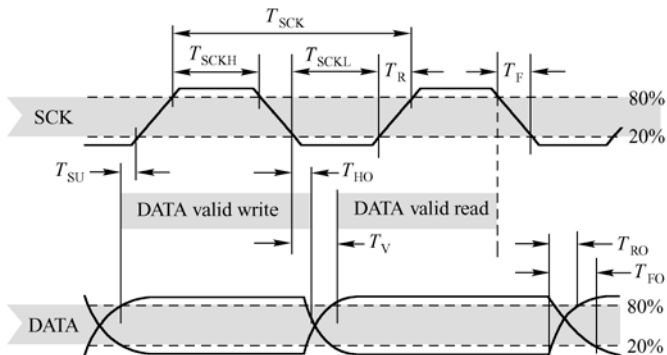


图 3-6 SHT10 与微控制器通信时序图

给传感器发送数据需要注意时序，在上升沿之前把数据写入，上升沿时数据有效，在下降沿时把数据传给传感器。数据发送的相关代码如下：

```
char s_write_byte(unsigned char value)
{
    unsigned char i;
    uint32 error;
    for (i = 0x80; i > 0; i /= 2)           //传送 8 字节的数据
    {
        if (i & value)                       //判断要发送的数据是 1 还是 0
        {
            ioctl(fd1, GPIO_SET_PIN, 25);    //发送数据 1
        }
        else
        {
            ioctl(fd1, GPIO_CLR_PIN, 25);    //发送数据 0
        }
        //置 SCK 位为高电平，上升沿有效，此时数据已准备好
        ioctl(fd2, GPIO_SET_PIN, 20);
        delay();
        //置 SCK 位为低电平，下降沿打入数据，此时 I/O 口的数据发送到传感器
        ioctl(fd2, GPIO_CLR_PIN, 20);
        delay();
    }
    ioctl(fd1, GPIO_SET_PIN, 25);           //数据发送完毕，置 DATA 位为高电平
    delay();
    ioctl(fd2, GPIO_SET_PIN, 20);          //置 SCK 位为高电平，准备读取数据
    ioctl(fd1, GPIO_SET_PIN_IN, 25);       //改变 DATA 的引脚为输入引脚，读取数据
    delay();
    error=25;                               //将要读取值的引脚号
    ioctl(fd1, GPIO_READ_PIN, &error);     //读取 ACK 值保存到 error 变量中
    ioctl(fd1, GPIO_SET_PIN_OUT, 25);      //置 DATA 位为输出引脚
    ioctl(fd2, GPIO_CLR_PIN, 20);          //置 SCK 为低电平
}
```



```

delay();
return error;           //返回 ACK 的值，若为 1 则无响应，为 0 则有
}

```

从传感器读数据，同样也要注意时序，只有在上升沿数据有效之后才能读。读数据的相关代码如下：

```

char s_read_byte(unsigned char ack)
{
    unsigned char i,val = 0;
    uint32 tempor;
    ioctl(fd1, GPIO_SET_PIN, 25);           //置 DATA 位为高电平，释放 DATA 位
    ioctl(fd1, GPIO_SET_PIN_IN, 25);       //改变 DATA 的引脚为输入引脚
    for (i = 0x80; i > 0; i /= 2)
        ioctl(fd2, GPIO_SET_PIN, 20);      //DATA 在上升沿时读出数据
    tempor=25;
    ioctl(fd1, GPIO_READ_PIN, &tempor);    //读 DATA 位保存到 tempor 变量中
    if (tempor)
    {
        val=(val | i);
    }
    ioctl(fd2, GPIO_CLR_PIN, 20);          //置 SCK 位为 0
    }
    if (ack)                                //判断是否需要发送应答
    {
        ioctl(fd1, GPIO_CLR_PIN, 25);      //发送应答
    }
    else
    {
        ioctl(fd1, GPIO_SET_PIN, 25);      //不需要发送应答
    }
    ioctl(fd2, GPIO_SET_PIN, 20);          //置 SCK 位为高电平
    ioctl(fd1, GPIO_SET_PIN_OUT, 25);      //改变 DATA 的引脚为输出引脚
    ioctl(fd1, GPIO_CLR_PIN, 25);          //置 DATA 位为低电平，发送应答
    delay();
    ioctl(fd2, GPIO_CLR_PIN, 20);          //置 SCK 为低电平，发送数据
    ioctl(fd1, GPIO_SET_PIN_OUT, 25);      //置 DATA 位为输出引脚
    ioctl(fd1, GPIO_SET_PIN, 25);          //置 DATA 位为高电平
    return val;                             //返回采集的值
}

```

温湿度测量的过程如下，发布一组测量命令后（“00000101”表示相对湿度，“00000011”表示温度），传感器开始采集数据。对应 8/12/14 位 3 种不同的测量，这个过程分别需要大约 20/80/320ms。SHT10 通过下拉 DATA 至低电平并进入空闲模式，表示测量的结束。微控制器在触发 SCK 时钟来读出数据前，必须等待这个“数据备妥”信号。检测数据可以先被存储，这样微控制器可以继续执行其他任务，在需要时再读出数据。在收到“数



据备妥”信号之后，传输 2 字节的测量数据和 1 字节的 CRC 奇偶校验。微控制器需要通过下拉 DATA 为低电平，以确认每个字节。在测量和通信结束后，SHT10 自动转入休眠模式。温湿度测量的相关代码：

```
char s_measure(unsigned char *p_value, unsigned char *p_checksum, unsigned
char mode)
{
    unsigned error=0;
    uint32 k;
    s_transstart();           //启动传输
    switch(mode)              //判断发送的是温度还是湿度的测量
    {
        case TEMP:
            error + = s_write_byte(MEASURE_TEMP);    //发送测量温度命令给传感器
            break;
        case HUMI:
            error += s_write_byte(MEASURE_HUMI);    //发送测量湿度命令给传感器
            break;
        default:
            break;
    }
    ioctl(fd1, GPIO_SET_PIN_IN, 25);    //改变 DATA 的引脚为输入引脚
    while(1)
    {
        k=25;
        ioctl(fd1, GPIO_READ_PIN, &k);
        if(!k)                        //等待测量完成
        {
            break;
        }
    }
    if(k)                              //DATA 为高电平，说明有错误发生
    {
        error += 1;
    }
    ioctl(fd1, GPIO_SET_PIN_OUT, 25);   //改变 DATA 的引脚为输出引脚
    *(p_value) = s_read_byte(ACK);      //读高字节数据
    *(p_value + 1) = s_read_byte(ACK);  //读低字节数据
    *p_checksum = s_read_byte(noACK);   //读校验码
    return error;
}
```

(3) 转换采集到的数据为具体的温度和湿度值并将数据通过串口发送到PC的超级终端显示

微控制器从传感器获取的数据并不是实际的温湿度值，要得到实际的温湿度值必须将从传感器获得的数据通过 SHT10 数据手册中给出的公式进行相应的转换。在程序中使用转换



函数 `calc_sht10` 实现。该函数的两个参数分别为指向采集的湿度和温度数据的指针变量。

温度转换公式为：

$$\text{Temperature} = d_1 + d_2 \times X$$

式中， X 为传感器采样的温度数据； d_1 、 d_2 为系数。

对于本实验采用 3.5V 电压、14 位温度采样精度，查数据手册可知该公式的系数分别为： $d_1 = -39.67$ ， $d_2 = 0.01$ ）。

湿度转换公式为：

$$\text{Humidity} = c_1 + c_2 \times Y + c_3 \times Y^2$$

式中， Y 为传感器采样的湿度数据； c_1 ， c_2 ， c_3 为系数。

对于本实验采用 12 位湿度采样精度，查数据手册可知该公式的系数分别为： $c_1 = -4$ ， $c_2 = 0.0405$ ， $c_3 = -2.8 \times 10^{-6}$ ）。

转换函数 `calc_sht10` 的代码如下：

```
void calc_sht10(float *p_humidity ,float *p_temperature)
{
    const float D1 = -39.67;           //对应于 3.5v 电压
    const float D2 = 0.01;             //对应于 12 位湿度采样精度

    const float C1 = -4.0;             //对应于 12 位湿度采样精度
    const float C2 = 0.0405;           //对应于 12 位湿度采样精度
    const float C3 = -0.00000028;     //对应于 12 位湿度采样精度

    float humi = *p_humidity;
    float temp = *p_temperature;
    float temperature;
    float humidity;

    temperature = D1 + D2* temp;       //温度转换公式
    humidity = C3 * humi * humi + C2 * humi + C1; //湿度转换公式

    *p_temperature = temperature;      //返回转换后的温度值
    *p_humidity = humidity;            //返回转换后的湿度百分比
}
```

在获取并转换了温湿度数据之后，使用 `printf()` 函数通过串口将读取的数据以及转换后的数据发送给 PC。

以上是本实验的各个模块的分析，下面介绍实验的流程。首先，主函数初始化各模块，然后微控制器给高精度传感器发送命令，高精度传感器把采集的数据发送给微控制器，最后微控制器通过串口把数据发送到电脑上去。下面是 `main()` 函数的主要代码：

```
int main(void)
{
    unsigned int a;
    unsigned char temp2[2],humid2[2],status;
```



```
unsigned int temp=0,humid=0;
float p_temperature,p_humidity;
initIO_sensor(); //初始化
delay();
while(1)
{
    s_connectionreset();
    s_measure(temp2,&status,0); //发送温度采集命令
    //保存采集的温度数据
    temp=temp2[1];
    temp+=(temp2[0]<<8);
    s_measure(humid2,&status,1); //发送湿度采集命令
    //保存采集的湿度数据
    humid=humid2[1];
    humid+=(humid2[0]<<8);
    p_temperature=(float)temp;
    p_humidity=(float)humid;
    calc_sht10(&p_humidity ,&p_temperature); //转换读取的数据
    printf("temperature value is %d \n",temp); //输出读取的温度数据
    printf("humidity value is %d\n",humid); //输出读取的湿度数据
    printf("temperature is %.2f \n",p_temperature); //输出转换后的温度值
    printf("humidity is %.2f%%\n\n",p_humidity); //输出转换后的湿度百分比
    temp=0;
    humid=0;
    for(a = 20;a > 0;a--)
        mydelay();
}
}
```

4. 实验内容

阅读 STH10 芯片的数据手册，学会分析芯片的时序图。阅读 GPIO 驱动程序，能够依照前述数据手册上的时序图，使用 GPIO 驱动程序编写一个简单的 μ Linux 应用程序，读出传感器的温度和湿度数据，使用数据手册中的公式转换之后，通过串口把数据发送到 PC。

5. 实验步骤

1) 打开终端或者进入虚拟控制台，进入 arm 实验目录，为本实验新建工作目录 temp-hum。

```
# cd /home/armwork
# mkdir temp-hum
```

- 2) 将源代码文件 temp-hum.c 和 Makefile 以及 config.h 文件复制到工作目录。
- 3) 将光盘提供的 gpio 文件夹复制到 nfs 共享目录下的 header 目录下。
- 4) 编译程序，生成可执行代码 temp-hum。

```
# make
```



5) 启动 μ CLinux, 进行 NFS 连接, 进入 GPIO 驱动所在目录, 加载 GPIO 驱动。

```
/> ./loadgpio
```

6) 进入 cat1025 目录, 运行实验程序。

```
/> ./temp-hum
```

之后可以在超级终端上看到程序运行的结果, 如图 3-7 所示。

```

μCLinux - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
/mnt/header> cd ..
/mnt> cd temp-hum
/mnt/temp-hum> ./temp-hum
temperature value is 6276
humidity value is 474
temperature is 23.09
humidity is 15.13%

temperature value is 6271
humidity value is 478
temperature is 23.04
humidity is 15.30%

temperature value is 6270
humidity value is 482
temperature is 23.03
humidity is 15.46%

temperature value is 6269
humidity value is 470
temperature is 23.02
humidity is 14.97%
  
```

图 3-7 嵌入式平台高精度温湿度传感器实验结果

其中每一组前两行为从传感器获得的数据, 后两行为转换后的温湿度数据。

6. 思考题

- 1) 本程序的采样精度为 14 位温度, 12 位湿度, 如何修改程序能够使精度变为 12 位温度、8 位湿度。
- 2) 怎样修改程序使得采样频率变小?

7. 补充阅读

- [1] 徐勇军, 安竹林, 蒋文丰, 姜鹏. 无线传感器网络实验教程[M]. 北京: 北京理工大学出版社, 2007.
- [2] 周立功, 等. ARM 嵌入式系统实验教程[M]. 北京: 北京航空航天大学出版社, 2005.
- [3] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.
- [4] SHT10 Datasheet. <http://www.sensirion.com/>. SENSIRION, 2009.

3.2.3 实验五A 单片机平台运动感知实验

1. 实验目的

本实验介绍了用单片机和运动传感器来检测物体运动的实验。通过实验, 使读者能学会如何用角速度传感器 ENC-03 测量物体运动的角速度, 用加速度传感器 ADXL330 测量加速度, 并用模数转换器 LTC2351 把 ENC-03 和 ADXL330 采样的模拟量转换为数字量。同时,



培养读者阅读芯片手册和编写驱动的能力。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- 串口调试助手。
- WinAVR20050214。
- AVRStudio4.18。

3. 实验原理

本实验基于 ATmega128 实现一个感知物体运动的程序，本实验的硬件平台为单片机实验母板 (IOT-SCMMB) 及相配套的传感器和外围模块。实验程序使用 ADXL330 测量物体运动 X、Y、Z 三个方向上加速度的模拟量，并用 3 个 ENC-03 分别测量物体运动的 X、Y、Z 三个方向的角速度。ADXL330 的 3 个输出引脚直接与模数转换器 LTC2351 的 3 个引脚相连，3 个 ENC-03 经过电压放大器后，再与 LTC2351 的 3 个引脚相连。通过 LTC2351 采样，将模拟量转换为数字量。最后，把上述 6 个数字量通过单片机发送到 PC 的串口上去，通过串口调试助手就可以看到相关的数字量了。

本实验程序的关键在于如何把通过传感器得到的模拟量转换为数字量并通过单片机的串口发送到 PC，并显示出来。传感器采样得到的模拟量是通过 LTC2351 芯片转换成数字量的。所以，在实验之前应该阅读 LTC2351 芯片的数据手册，学习如何驱动 LTC2351 芯片；通过 LTC2351 的采样，读出采样的数字量，并将其换算成为相应的电压；通过对电压的计算，得到物体运动的加速度和角速度。图 3-8 所示为 LTC2351 时序图的第 1~第 33 个周期，依时序图，就可以读出数字量。

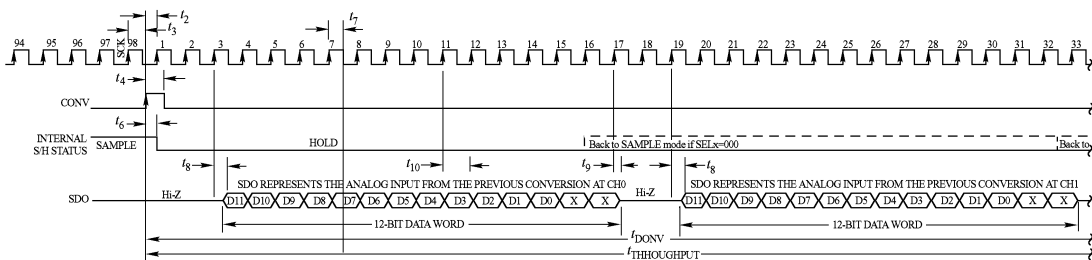


图 3-8 LTC2351 时序图 (第 1~第 33 个周期)

单片机 LTC2351 读出采样数据的过程如下：

在第 1 个时钟周期的时候，CONV 信号由高变为低，第 3 个周期的时候便开始读数据了，一直读 12 位的数据，最后两个时钟周期没有读数据。由于后面其他的 5 个数据的读取不要控制 CONV，所以后 5 个数据的读取是一样的，可以使用一个循环进行读取。

相关的程序如下：



```

/*****

```

函数功能：单片机从 LTC2351 读采样的数据，保存在 AX, AY, AZ, BX, BY, BZ

入口参数：无

出口参数：无

说明：这个函数的关键点在于在读数据时，DATA 在 SCK 变低以后有效，且维持到下一个 SCK 的下降沿，因此在 SCK 变低以后才能读

```

*****/

```

```

void GetData_LTC2351(void)
{
    unsigned int i,j;
    unsigned int data=0;
    PORTA&=~(_BV(PA0));
    PORTA|=_BV(PA2);
    PORTA|=_BV(PA0);
    //以下是第1个周期
    PORTA&=~(_BV(PA2));           //CONV 信号由高变低，首次读数据时要设置
    PORTA&=~(_BV(PA0));
    PORTA|=_BV(PA0);
    PORTA&=~(_BV(PA0));           //第2个周期
    asm("nop");
    asm("nop");
    for(j=12;j>0;j--)           //从第3~第14个周期接收数据
    {
        PORTA|=_BV(PA0);           //SCK 置为高电平
        PORTA&=~(_BV(PA0));       //SCK 置为低电平，就可以读取采样的数据了
        if(PINA&0x02)             //读数据
        {
            data|=(1<<(j-1));
        }
    }
    AZ=data;                     //将读到的数据保存到AZ中
    data=0;
    PORTA|=_BV(PA0);             //第15个周期
    PORTA&=~(_BV(PA0));
    PORTA|=_BV(PA0);
    PORTA&=~(_BV(PA0));           //第16个周期
    //其他5位数据的读取，不要CONV信号，所以可以用一个FOR循环来实现
    for(i = 1;i <= 5; i++)
    {
        PORTA|=_BV(PA0);           //时序图中的两个x，这个时序要，但读不到数据
        PORTA&=~(_BV(PA0));
        PORTA|=_BV(PA0);
        PORTA&=~(_BV(PA0));
        for(j = 12;j > 0;j--)       //开始读取数据
        {
            PORTA|=_BV(PA0);       //SCK 置为高电平

```



```
        PORTA&=~(_BV(PA0));        //SCK 置为低电平, 读取采样的数据了
        if (PINA&0x02)            //读数据
        {
            data|=(1<<(j-1));
        }
    }
    PORTA|=_BV(PA0);
    PORTA&=~(_BV(PA0));        //第 15 个周期
    PORTA|=_BV(PA0);
    PORTA&=~(_BV(PA0));        //第 16 个周期

    //根据读取数据的类型, 保存到相应的变量中
    if(i==1)
        AY=data;
    if(i==2)
        AX=data;
    if(i==3)
        BX=data;
    if(i==4)
        BY=data;
    if(i==5)
        BZ=data;
    data=0;
}
PORTA|=_BV(PA0);
PORTA&=~(_BV(PA0));        //第 97 个周期;
PORTA|=_BV(PA0);
PORTA&=~(_BV(PA0));        //第 98 个周期;
}
```

实验程序的主函数首先初始化相关的接口, 接口包括串口以及 LTC2351 与单片机连接的 I/O 口, 然后调用上述采样函数发送相关的时序以获得传感器数据。

```
int main( void )
{
    unsigned int i,j,k;
    DDRA|=_BV(PA0)|_BV(PA2);        //PA0 连接 SCK, PA2 连接 CONV

    //初始化 LTC2351 与单片机相连的 I/O 口
    DDRA&=~(_BV(PA1));
    PORTA&=0xfa;
    PORTA|=0x02;
    uart_init();                    //串口 0 的初始化
    sei();
    while(1)
    {
        delay();
    }
}
```



```

GetData_LTC2351();           //获取相关的数据
uart_xyz();                   //把数据发送到串口
}
}

```

4. 实验内容

阅读硬件的原理图，了解加速度、角速度芯片如何与单片机进行连接，学习如何通过单片机控制芯片工作，并分析芯片的数据手册，按照时序编写程序控制芯片采集数据。

5. 实验步骤

- 1) 编写实验代码，使用 WinAVR 编译得到 .hex 文件。
- 2) 使用 AVR Studio 将 .hex 文件下载到开发板上。
- 3) 将串口与 PC 相连，使用串口调试助手波特率为 57600bit/s，校验位 NONE，数据位 8 位，停止位 1 位。另外，还要用十六进制显示所得到的数据，通常的结果如下：

8E 08 16 08 30 0A 03 08 74 08 76 08 79 7E（此时是静止）

数据包以 8E 开始，以 7E 结束，前两位数据 08 16 是 AZ 的十六进制数值，其中 08 是高 8 位，16 是低 8 位，后面的 10 位分别为 AY、AX、BX、BY、BZ 的传感器数据。

计算所得到的电压值方法如下。

加速度的计算方式：

$$2070 \times 2.5V / 4096 = 1.263V$$

式中，2070 是 08 16 的十进制形式；4096 对应于 12 位采样精度；2.5 是 LTC2351 采样的参考电压。

由上式得到静止时 AZ 的输出电压 $AZ_{out=0} = 1.236V$ 。阅读芯片手册可知，任意时刻电压计算公式 $AZ_{out} = AZ_{out=0} + 0.3/g \times G$ (G 为此时的加速度， g 为自由落体加速度)，因此只要得到此时的电压，依此公式就可以得到此时的各方向的加速度。

角速度的计算方式：

$$2164 \times 2.5V / 4096 = 1.32V$$

式中，2164 是 08 74 的十进制形式；4096 对应于 12 位采样精度；2.5 是 LTC2351 采样的参考电压。

由上式得到静止时 BX 的输出电压 $BX_{out=0} = 1.32V$ 。阅读芯片手册可知，任意时刻电压计算公式为 $V_{out} = V_0 + S_v \omega_{[V]} P$ (V_0 为静止时的输出电压；角速度比例系数 $S_v = 0.67mV/deg/s$ ； $\omega_{[V]}$ 为角速度； P 为运放电压的放大系数，为常数 5.7)。因此，只要得到测量电压的数字量，就可以通过以上的计算方法，得到相关方向的角速度。

6. 思考题

为什么只需要一个加速度传感器就可以测量 3 个方向的加速度，而角速度传感器必须要使用 3 个？

7. 补充阅读

[1] 徐勇军，安竹林，等. 无线传感器网络实验教程[M]. 北京：北京理工大学出版社，2007.

[2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.

[3] LTC2351 Datasheet. <http://www.linear.com/>. LINEAR, 2007.



[4] ENC-03 Datasheet. <http://www.murata.com/>.Murata,2006.

[5] ADXL330 Datasheet. <http://www.siphec.com>. Analog Devices, Inc,2006.

3.2.4 实验五B 嵌入式平台运动感知实验

1. 实验目的

本实验介绍了用 ARM 实验母板和运动传感器来检测物体运动的实验。通过实验，使读者能学会如何用角速度传感器 ENC-03 测量物体运动的角速度，用加速度传感器 ADXL330 测量加速度，并通过模数转换器 LTC2351 把 ENC-03 和 ADXL330 采样的模拟量转换为数字量。同时，培养读者阅读芯片手册和基于 μ CLinux 开发嵌入式应用程序的能力。

2. 实验设备

(1) 硬件

- ARM 实验母板 (IOT-ARMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- PC (含串口) 1 台。

(2) 软件

- Linux 操作系统。
- H-JTAG。
- μ CLinux 2.4x 发行包和 arm-elf-gcc 交叉编译环境。

3. 实验原理

本实验基于 IOT-ARMMB 开发母板搭载的 LPC2200 ARM 处理器和传感器板上的 LTC2351 模数转换芯片，对速度传感器输出的模拟量进行数字化采样，最终实现对运动物体的感知。实验使用 ADXL330 测量物体运动 X、Y、Z 三个方向上加速度的模拟量，并用 3 个 ENC-03 分别测量物体运动的 X、Y、Z 三个方向的角速度。ADXL330 的 3 个输出引脚直接与模数转换器 LTC2351 的 3 个引脚相连，3 个 ENC-03 经过电压放大器后，再与 LTC2351 的 3 个引脚相连。上述 6 个模拟量将通过 LTC2351 的采样转换为数字量，并通过 GPIO 接口发送到 ARM 处理器，再通过网络接口发送到 PC。通过交叉编译环境就可以在 PC 屏幕上实时观测到加速度的测量结果。

运动传感器采样得到的模拟量是通过 LTC2351 芯片转换成数字量的。因此，编写 LTC2351 的驱动程序是本实验的核心内容。在实验之前，应该阅读 LTC2351 芯片的数据手册，学习 LTC2351 的工作时序。图 3-9 所示为 LTC2351 时序图的第 1~第 33 个周期。

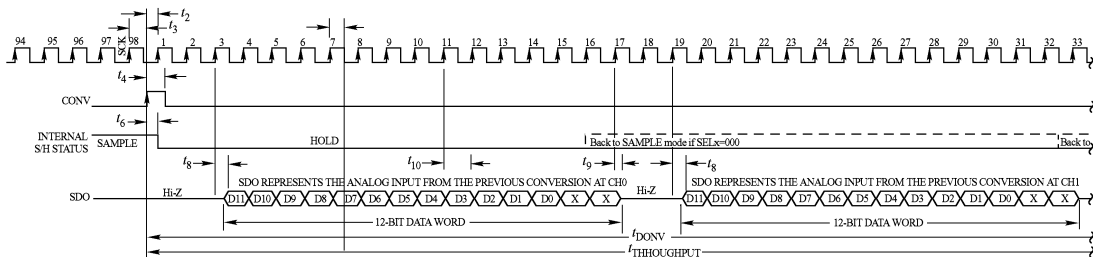


图 3-9 LTC2351 时序图 (第 1~第 33 个周期)



LTC2351 读出采样数据的过程如下:

在第 1 个时钟周期, CONV 信号由高变低。第 3 个周期起开始读取数据, 连续读 12 位, 最后两个时钟周期没有读数据。由于后面其他的 5 个数据的读取不须改变 CONV 的状态, 所以以后 5 个数据的读取过程是相同的, 可以使用一个循环进行读取。

相关的程序如下:

```
#define SCK 16 //16 号口对应 SCK 时钟信号
#define SDO 17 //17 号口对应 SDO 数据
#define CONV 18 //18 号口对应 CONV 信号

int ax,ay,az,wx,wy,wz;
int get_data_ltc2351(void)
{
    unsigned int i,j;
    unsigned int data=0;
    int fd;
    int pin;

    fd = open("/dev/P1",O_RDONLY);
    if(fd == -1)
    {
        printf("\nCan't open P1!\n");
        return -1;
    }

    ioctl(fd,GPIO_SET_PIN_OUT,SCK); //SCK 设为输出口
    ioctl(fd,GPIO_SET_PIN_OUT,CONV); //CONV 设为输出口
    ioctl(fd,GPIO_SET_PIN_IN,SDO); //SDO 设为输入口

    ioctl(fd,GPIO_CLR_PIN,SCK); //SCK 拉低
    ioctl(fd,GPIO_SET_PIN,CONV); //CONV 拉高
    ioctl(fd,GPIO_SET_PIN,SCK); //SCK 拉高
    ioctl(fd,GPIO_CLR_PIN,CONV); //CONV 拉低
    ioctl(fd,GPIO_CLR_PIN,SCK); //SCK 拉低
    ioctl(fd,GPIO_SET_PIN,SCK); //SCK 拉高
    ioctl(fd,GPIO_CLR_PIN,SCK); //SCK 拉低
    asm("nop");
    asm("nop");
    for(j=12;j>0;j--)
    {
        ioctl(fd,GPIO_SET_PIN,SCK); //SCK 拉高
        ioctl(fd,GPIO_CLR_PIN,SCK); //SCK 拉低
        pin = SDO;
        ioctl(fd, GPIO_READ_PIN, &pin); //开始读数据
        if(pin)
        {
```



```
        data|=(1<<(j-1));
    }
}
ax=data;
data=0;
ioctl(fd,GPIO_SET_PIN,SCK);           //SCK 拉高
ioctl(fd,GPIO_CLR_PIN,SCK);           //SCK 拉低

ioctl(fd,GPIO_SET_PIN,SCK);           //SCK 拉高
ioctl(fd,GPIO_CLR_PIN,SCK);           //SCK 拉低

for(i = 1;i <= 5; i++)                 //循环读取第 2 组到第 6 组数据
{
    ioctl(fd,GPIO_SET_PIN,SCK);         //SCK 拉高
    ioctl(fd,GPIO_CLR_PIN,SCK);         //SCK 拉低
    ioctl(fd,GPIO_SET_PIN,SCK);         //SCK 拉高
    ioctl(fd,GPIO_CLR_PIN,SCK);         //SCK 拉低
    for(j=12;j>0;j--)
    {
        ioctl(fd,GPIO_SET_PIN,SCK);     //SCK 拉高
        ioctl(fd,GPIO_CLR_PIN,SCK);     //SCK 拉低
        pin = SDO;
        ioctl(fd, GPIO_READ_PIN, &pin);
        if(pin)
        {
            data|=(1<<(j-1));
        }
    }
    ioctl(fd,GPIO_SET_PIN,SCK);         //SCK 拉高
    ioctl(fd,GPIO_CLR_PIN,SCK);         //SCK 拉低
    ioctl(fd,GPIO_SET_PIN,SCK);         //SCK 拉高
    ioctl(fd,GPIO_CLR_PIN,SCK);         //SCK 拉低

    if(i==1)
        ay=data;

    if(i==2)
        az=data;

    if(i==3)
        wx=data;

    if(i==4)
        wy=data;

    if(i==5)
```



```
wz=data;

data=0;

}
ioctl(fd,GPIO_SET_PIN,SCK);           //SCK 拉高
ioctl(fd,GPIO_CLR_PIN,SCK);          //SCK 拉低
ioctl(fd,GPIO_SET_PIN,SCK);         //SCK 拉高
ioctl(fd,GPIO_CLR_PIN,SCK);         //SCK 拉低
}
```

实验程序的主函数以一定的时间间隔调用上述采样函数，并将结果发送到屏幕上。

```
int main(void)
{
    unsigned int i,j,k;

    while(1)
    {
        ax=0;ay=0;az=0;wx=0;wy=0;wz=0;
        for(i=0;i<5000;i++)
        {
            for(j=0;j<2000;j++)
            {
                k=0;
            }
            get_data_ltc2351();
            printf("ax = %d,ay = %d,az = %d,
                    wx = %d,wy = %d,wz = %d\n",az,ay,az,wx,wy,wz);
        }
    }
    exit(0);
}
```

4. 实验内容

阅读硬件的原理图，了解加速度、角速度芯片如何与 LPC2200 ARM 处理器进行连接，分析芯片的数据手册，按照时序编写程序控制芯片采集数据。

5. 实验步骤

1) 打开终端或者进入虚拟控制台，进入 arm 实验目录/home/armwork/，为本实验新建工作目录 ltc。

```
# cd /home/armwork
# mkdir ltc
# mkdir header
```

2) 将源代码文件 ltc.c 和 Makefile 以及 config.h 文件复制到工作目录。



3) 将光盘提供的 `gpio` 文件夹复制到 `nfs` 共享目录下的 `header` 目录下。

```
#include "../header/gpio/gpio.h"
```

4) 编译程序, 生成可执行代码 `ltc`。

```
# make
```

5) 启动 `μCLinux`, 进行 `NFS` 连接, 进入 `gpio` 驱动所在目录, 加载 `gpio` 驱动。

```
/mnt/header/gpio> ./loadgpio
```

6) 进入 `ltc` 目录, 运行实验程序。

```
/mnt/ltc> ./ltc
```

运行结果如图 3-10 所示

```
/mnt/ltc> ./ltc
ax = 2981,ay = 2576,az = 2981,wx = 2165,wy = 2168,wz = 2170
ax = 2999,ay = 2559,az = 2999,wx = 2151,wy = 2166,wz = 2168
ax = 3051,ay = 2470,az = 3051,wx = 2142,wy = 2167,wz = 2169
ax = 2777,ay = 2544,az = 2777,wx = 2223,wy = 2167,wz = 2165
ax = 2989,ay = 2497,az = 2989,wx = 2158,wy = 2169,wz = 2169
ax = 2974,ay = 2643,az = 2974,wx = 2162,wy = 2170,wz = 2171
```

图 3-10 运动感知实验结果

下面对第一组数据进行计算。

加速度的计算方式:

$$2981 \times 2.5V / 4096 = 1.819V$$

式中, 4096 对应于 12 位采样精度; 2.5V 是 LTC2351 采样的参考电压。

由公式可得到静止时 `ax` 的输出电压 $ax_{out=0} = 1.819V$ 。阅读芯片数据手册可知任意时刻电压计算公式为 $ax_{out} = ax_{out=0} + 0.3/g \times G$ (G 为此时的加速度, g 为自由落体加速度), 因此只要得到此时的电压, 根据公式就可以得到此时的各方向的加速度。

角速度的计算方式:

$$2165 \times 2.5V / 4096 = 1.32V$$

式中, 4096 对应于 12 位采样精度; 2.5V 是 LTC2351 采样的参考电压。

由上式可得到静止时 `wx` 的输出电压 $wx_{out=0} = 1.32V$ 。阅读芯片数据手册可知, 任意时刻电压计算公式为 $V_{out} = V_0 + S_v \times w_{[V]} \times P$ (V_0 静止时的输出电压; $S_v = 0.67mV/deg/s$; $w_{[V]}$ 为角速度; P 为运放电压的放大系数, 为常数 5.7)。因此, 只要得到测量电压的数字量, 就可以通过以上的计算方法, 得到相关方向的角速度。

6. 思考题

同使用单片机控制运动传感器相比, 使用嵌入式方式有什么相同点和不同点?

7. 补充阅读

[1] Sreekrishnan Venkateswaran. 精通 Linux 设备驱动程序开发[M]. 北京: 人民邮电出版社, 2010.



[2] LTC2351 Datasheet. <http://www.linear.com/>. LINEAR, 2007.

[3] ENC-03 Datasheet. <http://www.murata.com/>. Murata, 2006.

[4] ADXL330 Datasheet. <http://www.siphec.com/>. Analog Devices, Inc, 2006.

3.2.5 实验六 图像采集实验

1. 实验目的

本实验介绍如何使用单片机 ATmega128 控制串口摄像模块 C328 进行拍照。通过实验使读者掌握 C328 的使用方法, 并进一步熟练 ATmega128 的程序设计。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- 串口调试助手。
- WinAVR20050214。
- AVRStudio4.18。

3. 实验原理

本实验基于 ATmega128 实现一个串口摄像的程序, 硬件平台是基于单片机实验母板 (IOT-SCMMB) 以及相应的传感器及外围模块。实验中用到了两个单片机 U4、U5, 单片机 U5 实现了以下的功能: U5 通过串口 1 发送命令给串口摄像模块, 串口摄像模块把相片数据通过串口 1 发回给 U5, U5 再把数据通过它的串口 0 发送给另外一个单片机 U4 的串口 1 的扩展芯片 GM8123, 最后单片机 U4 通过串口 0 把数据发送到 PC, 在 PC 上打开串口调试助手, 就可以获得相片的数据。

实验各模块的关系如图 3-11 所示。

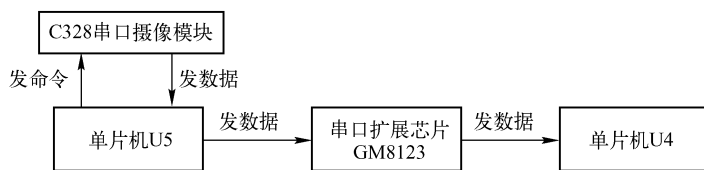


图 3-11 实验各模块的关系

为了便于理解, 实验内容分为 3 部分: 串口的接收和发送程序, C328 串口摄像模块的控制程序, 单片机 U4 和 U5 的双机通信程序。下面分别对各部分进行介绍。

(1) 串口的接收和发送程序

因为在实验一中已经介绍过串口的使用方法, 串口的初始化以及基本的数据收发已经为大家所熟悉, 所以这里不再详细描述, 着重关注本实验所增加的基于缓冲区的串口数据收发过程。在本实验中, 串口 1 的发送和接收过程都使用了缓冲区, 即首先把要发送的数据保存



到缓冲区中，如果单片机的串口空闲，数据就会直接被发送出去；如果串口正在发送或者接收数据，则数据被保存在缓冲区中，待串口空闲时再将其发送出去。这里串口 1 的发送和接收都是用中断实现的。

串口 1 的发送程序首先把要发送的数据保存到缓冲区，然后打开串口中断（即使能 UDR1E）。此时由于没有数据发送，中断处理程序会立即被执行，其判断缓冲区中是否有数据。如果有，就把数据发送出去；如果没有，则关闭串口中断（关闭 UDR1E 使能），等待下一次的数据发送。

```
void TransmitByte(unsigned char data)
{
    unsigned char tmphead;
    tmphead = (UART_TxHead + 1)&UART_TX_BUFFER_MASK; //获得缓冲区指针的空闲位置
    while(tmphead==UART_TxTail); //如果缓冲区满了就保持等待
    UART_TxBuf[UART_TxHead] = data; //把数据放到缓冲区
    UART_TxHead = tmphead; //移动缓冲区指针到下一个空闲位置
    UCSR1B|=(1<<UDR1E); //打开中断使能，使能中断处理程序
}
```

串口 1 的中断发送处理函数主要负责数据的发送。当中断使能时，如果没有数据正在发送或者接收，则会发生中断。中断处理函数会把缓冲区中的数据发送出去，如果没有数据，则关闭中断使能。待下次串口 1 有数据需要发送时，再把中断打开，进行数据发送。

```
SIGNAL(SIG_UART1_DATA)
{
    unsigned char tmptail;
    if(UART_TxHead != UART_TxTail) //判断缓冲区是否有数据要发送
    {
        tmptail = (UART_TxTail+1)&UART_TX_BUFFER_MASK;
        UDR1 = UART_TxBuf[UART_TxTail];
        UART_TxTail = tmptail; //移动缓冲区的指针
    }
    else
        UCSR1B&=~(1<<UDR1E); //如果没有数据，则关闭中断使能
}
```

串口 1 的接收程序中首先判断接收缓冲区是否有数据。如果有数据，取出数据；如果没有，则等待。

```
unsigned char ReceiveByte(void)
{
    unsigned char tmptail;
    unsigned char data;
    while(UART_RxHead == UART_RxTail) //如果没有数据，则等待
        ;
    data = UART_RxBuf[UART_RxTail]; //取出数据
    tmptail = (UART_RxTail+1)&UART_RX_BUFFER_MASK; //移动缓冲区的指针
}
```



```

UART_RxTail = tmptail;
return data;
}

```

单片机接收到数据会发生接收处理中断，触发串口 1 的中断接收处理函数，把接收到的数据发送到发送缓冲区，并移动发送缓冲区指针。

```

SIGNAL(SIG_UART1_RECV)
{
    unsigned char data;
    unsigned char tmphead;
    if(first == 1)           //标识是否是第一次收到数据，在与 C328 同步时使用
        first = 0;
    data = UDR1;
    tmphead = (UART_RxHead+1)&UART_RX_BUFFER_MASK; //移动接收缓冲区指针
    if(tmphead == UART_RxTail) //如果缓冲区满了，则退出
    {
        return;
    }
    UART_RxBuf[UART_RxHead] = data; //把数据保存在缓冲区中去
    UART_RxHead=tmphead; //移动缓冲区指针到下一个位置
}

```

(2) C328 串口摄像模块的控制程序

C328 串口摄像模块支持标准串口输出 JPEG 图像数据。该模块有 4 个引脚：GND、RX、TX、VDD，其中 TX 和 RX 引脚分别与单片机 U5 串口 1 的 RX 和 TX 引脚相连，其他两个引脚分别接地和电源。单片机 U5 通过串口 1 向 C328 发送命令，C328 收到命令后，给单片机发送应答（ACK）。

单片机控制 C328，通常需要用到如下几个命令。

1) 同步命令。在使用 C328 之前，需要给 C328 发送同步命令。该命令调节单片机串口与 C328 串口的波特率，并唤醒 C328。当发送命令时，需要多次发送唤醒命令，且发送命令之间要有一定的时间间隔，直到 C328 发送应答信号给单片机，则 C328 被唤醒，完成同步。相关的程序如下：

```

void sync1(void)
{
    unsigned char i=0,b=1;
    createCommand( CMD_SYNC, 0, 0, 0, 0 ); //构建同步命令
    while(i<100 && first) //最多发100次，通过 first 来判断是否第一次收到应答
    {
        sendCommand();
        delay1();
        i++;
    }
    if(waitForResponse((char *)ack,CMD_SYNC)) //等待 C328 的应答

```



```
    ;
    while(b)
    {
        if(waitForCommand((char *)sync,CMD_SYNC)) //等待 C328 发送应答给单片机
        {
            b=0;
            sendAck(CMD_SYNC); //发送应答
        }
        delay();
    }
}
```

2) 初始化命令。单片机发送初始化命令给 C328，设置 C328 照相色彩类型、预览方式，发完命令后还要等待 C328 的应答。

```
void initial( unsigned char colorType, unsigned char previewResolution,
              unsigned char jpegResolution )
{
    unsigned char flag = 1;
    //构建发送命令
    createCommand(CMD_INITIAL, 0, colorType, previewResolution,
                 jpegResolution);
    while(flag)
    {
        sendCommand(); //发送命令
        if(waitForResponse((char *)ack, CMD_INITIAL)) //等待应答
        {
            flag = 0;
        }
    }
}
```

下文中设置数据包大小的命令和发送照相命令这两个命令的实现过程与初始化命令类似，只是构建命令的参数不同。这里只给出构建发送命令的语句，具体请参考实验代码。

3) 设置数据包大小命令。此命令用来设置数据包的大小，当 C328 给单片机传送相片数据时，必须按这个大小进行传送。

```
createCommand( CMD_PACKAGESIZE, 0x08, (unsigned char )(size & 0xFF),
              (unsigned char)(size >> 8), 0 );
```

4) 发送照相操作命令。单片机给 C328 发送照相命令，C328 执行照相操作，把相片数据放到缓冲区中。

```
createCommand( CMD_SNAPSHOT, snapshotType,
              (unsigned char)(skipFrames & 0xFF),(unsigned char)(skipFrames>>8),0 );
```

5) 发送获取已照相片数据大小命令。发送此命令给 C328，C328 会返回在其缓冲区的



相片数据的大小，单片机通过这个命令来确定相片数据的大小。

```
int getPictureSize(unsigned char pictureType, unsigned long *pictureSize)
{
    unsigned char flag = 1;
    *pictureSize = 0;
    createCommand( CMD_GETPICTURE, pictureType, 0, 0, 0 );
    while(flag)
    {
        sendCommand();
        delay1();
        if(waitForResponse((char *)ack, CMD_GETPICTURE))//等待响应
        {
            flag = 0;
        }
    }
    flag = 1;
    delay1();
    while(flag)
    {
        if(waitForCommand((char *)datacmd,CMD_DATA)) //等待 C328 给单片机发命令
            flag = 0;
    }
    if(datacmd[1] == CMD_DATA&&datacmd[2]==pictureType)
    {
        *pictureSize *= ((unsigned int *)&datacmd[3]);
        return 1;
    }
    return 0;
}
```

6) 获得相片数据命令。C328 收到单片机发送的这个命令，会把缓冲区的相片数据发送给单片机，然后单片机通过串口 0 把数据发送给单片机 U4。

```
int getJPEGPicture(unsigned char pictureType,unsigned int packageSize)
{
    unsigned long pictureSize = 0;
    unsigned int next=1;
    unsigned int packageSuccess=0;
    unsigned int packageCount=0,errorCount=0;
    if( !getPictureSize( pictureType, &pictureSize ) )//获得相片数据的大小
        return 0;
    while(next)
    {
        sendDataACK( 0, packageCount); //发送应答
        delay();
        packageSuccess = waitForData(&next); //把数据发到其他的串口
    }
}
```



```
    if(packageSuccess)
    {
        packageCount++;
    }
    else
    {
        errorCount++;
    }
}
sendDataACK( 0, LAST_JPEG_ACK );
if( errorCount < MAX_ERRORS )
{
    return 1;
}
else
{
    return 0;
}
}
```

在上一个命令中，包含了单片机把获得的数据通过其他串口发送出去的函数 `waitForData`。这个函数从 C328 获得数据，然后通过串口 0 发送给与单片机 U4 的串口 1 相连的串口扩展芯片 GM8123 上去。

```
int waitForData(unsigned int *next) //通过这个 next 变量判断是否还有数据要发送
{
    unsigned int i;
    unsigned char j;
    unsigned int k=0;
    unsigned char length[4]={0};
    for(i=0; i<4; i++) //得到数据包的大小
    {
        j=ReceiveByte();
        length[i]=j;
    }
    k=((unsigned int *)&length[2]);
    for(i=0; i<k; i++) //获得相片数据
    {
        j=ReceiveByte();
        putc_uart0(j); //把获得的数据通过串口 0 发送到扩展芯片上去
    }
    if(k<506)
    {
        *next=0;
    }
    j=ReceiveByte(); //除去校验码
}
```



```

j=ReceiveByte();
return 1;
}

```

(3) 单片机U4和U5的双机通信程序

单片机 U5 的串口 0 与单片机 U4 的串口扩展芯片 GM8123 的扩展口 3 相连，所以如果 U5 要给 U4 发送数据，需要让串口扩展芯片 GM8123 工作在串口 3 的单模式下。这样，U4 可以从 GM8123 收到数据，然后通过单片机 U4 的串口 0 发送到 PC 上，通过串口调试助手就可以得到相关的数据。

GM8123 的初始化程序是让 GM8123 工作在单通道模式下。单片机通过这个扩展串口接收来自单片机 U5 发送过来的数据。

```

void gm8123_single(void)
{
    PORTB|= BV(PB7);           //MS=1 设置 GM8125 工作在单通道模式下
    PORTE|=_BV(PE4);          //设定 STADD1, STADD0 为 11
    PORTE|=_BV(PE5);
    PORTB&=~(_BV(PB5));       //设定 SRADD1, SRADD0 为 01
    PORTB|=( _BV(PB6));
}

```

单片机 U4 的串口扩展芯片工作在单通道的工作方式下，等待单片机 U5 给其发送数据，相关的代码如下：

```

int main(void)
{
    char data;
    UBRR1L=47;                //uart 初始化，波特率为 9600
    UCSR1B=_BV(RXEN1)|_BV(TXEN1); //使能发送和接受允许
    uart0_init();
    gm8123_single();
    while(1)
    {
        data=get_uart1();     //从串口 1 得到数据
        putc(data);           //从串口 0 发送出去
    }
    return 0;
}

```

单片机 U5 初始化 C328，并且给 C328 发送相关命令，C328 执行完照相命令后，把数据发送给单片机 U5，然后 U5 把相片数据发送串口扩展芯片 GM8123，相关的代码如下：

```

int main(void)
{
    uart0_init(9600);         //串口 0 初始化
}

```



```
uart1_init(9600);           //串口 1 初始化
sei();
sync1();
initial(0x07,0x00,0x05);    //C328 初始化
setPackageSize(0x200);     //设置包的大小
snapshot(0x00,0x00);       //照相
delay1();
getJPEGPicture(0x01,0x200); //得到相片数据
powerOff();
}
```

4. 实验内容

阅读 C328 串口摄像模块的文档，掌握如何编写其相关驱动，并通过单片机控制 C328 照相，以得到相片的数据。

掌握 ATmega128 用中断实现串口通信的过程，阅读串口扩展芯片 GM8123 的数据手册，参照原理图，实现单片机双机通信。

5. 实验步骤

1) 建立两个文件夹，一个是 U4，一个是 U5，使用 WinAVR 分别写相关的代码，并编译得到.hex 文件。

2) 使用 AVR Studio 分别将.hex 文件下载到单片机 U4 和 U5 上。

3) 将单片机 U4 串口与 PC 相连，打开串口调试助手，波特率为 9600bit/s，校验位 NONE，数据位 8 位，停止位 1 位，且用十六进制显示。再按单片机的复位键，过 5s，就可以在串口调试助手上收到数据，把这个数据保存起来，命名为 src.txt，最后运行 convert.exe 就可以得到 src.jpg 图像文件。

6. 思考题

1) 本程序中，单片机 U5 串口 1 使用中断方式接收数据，可否使用查询方式？程序如何实现？

2) 如何编写从串口读取图像数据并直接显示 JPG 文件的上位机软件。

7. 补充阅读

[1] 徐勇军，安竹林，等. 无线传感器网络实验教程[M]. 北京：北京理工大学出版社，2007.

[2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.

[3] C328 Datasheet. <http://www.sparkfun.com/>. 2009.

[4] GM8123 Datasheet. <http://www.sia.cn/>. 2004.

3.2.6 实验七 步进电动机实验

1. 实验目的

本实验介绍了用单片机 ATmega128 控制步进电动机，使步进电动机每隔 7.5s 反转一次。通过这个实验，使读者能掌握使用单片机控制步进电动机进行转动的方法。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。



- 传感器及外围模块 (IOT-MODULES) 1 套。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- WinAVR 20050214。
- AVRStudio4.18。

3. 实验原理

本实验使用 IOT-SCMMB 单片机实验母板上的 ATmega128 实现一个步进电动机转动的实验。实验中使用单片机的 4 个 I/O 口给步进电动机发送相关的时序，控制步进电动机的转动，并且通过单片机的定时器控制步进电动机正反向转动。

两相步进电动机有 4 根引线，分为 A、 \bar{A} 、B、 \bar{B} ，单极两相步进电动机的控制过程如图 3-12 所示。

如果信号的时序是 $AB - \bar{B}\bar{A} - \bar{A}\bar{B} - \bar{B}A$ ，步进电动机就顺时针转动；如果按照相反的时序 ($\bar{B}A - \bar{A}\bar{B} - \bar{B}\bar{A} - AB$) 给信号，步进电动机就会反向转动。

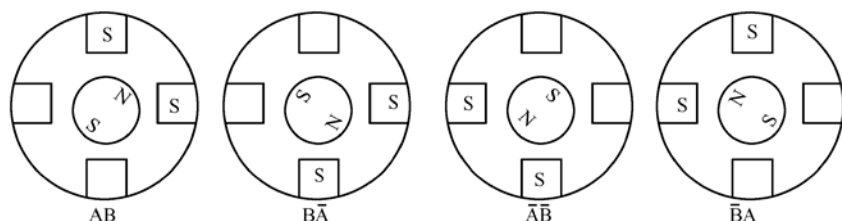


图 3-12 单极两相步进电动机的控制过程

单片机与步进电动机的接口连接方式是这样的：单片机的 PB4 与 A 相连，PB5 与 \bar{A} 相连，PB6 与 B 相连，PB7 与 \bar{B} 相连。要产生 AB 信号，仅当 PB4 为高电平，PB5 为低电平，并且 PB6 为高电平，PB7 为低电平。依此类推，产生 $\bar{B}\bar{A} - \bar{A}\bar{B} - \bar{B}A$ 信号。A、 \bar{A} 、B、 \bar{B} 的时序图如图 3-13 所示，注意，两条虚线之间就是产生 $AB - \bar{B}\bar{A} - \bar{A}\bar{B} - \bar{B}A$ 信号的时序图。

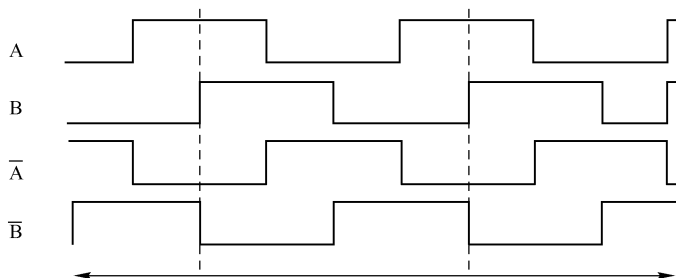


图 3-13 单极两相步进电动机时序图

下面结合代码进行分析。要产生上述时序，需要定义如下数组：

```
/******
```

```
*说明：two_phase 的参数意义
```



```
0x55 是高 4 位输出, 5 =0101  AB (AB)
0x66 是高 4 位输出, 6 =0110  B-A(B 负 A)
0xaa 是高 4 位输出, a =1100  -B-A(负 B 负 A)
0x99 是高 4 位输出, 9 =1001  A-B(A 负 B)
*****/
const unsigned char two_phase[]={0x55,0x66,0xaa,0x99};
```

按顺序将上述数组发送出去, 就会产生相应的时序, 即:

```
/******
*名称 go_step_2phase(unsigned int time)
*功能: 步进电动机转动
*入口参数: time 是时延参数
*出口参数: 无
*****/
void go_step_2phase(unsigned int time)
{
    if(!dir)
        PORTB=two_phase[phase_pnt++]; //PORTB 高 4 位是输出口, 与步进电动机引脚相连
    else
        PORTB=two_phase[phase_pnt--]; //如果是反转, 数组指针就减 1
    phase_pnt&=0x3; //若数组指针 phase_pnt 越界, 则 phase_pnt=0
    delay(time,100);
}
```

实验时使用定时器, 每隔 7.5s 改变步进电动机的转动方向。使用定时器改变转动的方向的定时器处理函数如下 (其中 `dir=~dir` 改变步进电动机的转动方向):

```
/******
*名称 SIGNAL()
*功能: 定时器/计数器 0 溢出中断
*入口参数: SIG_OVERFLOW0
*出口参数: 无
*****/
SIGNAL(SIG_OVERFLOW0)
{
    //晶振频率实验中的晶振频率为 7.3728MHz, 采用 64 分频和 8 位计数器 (溢出值 256)
    //产生中断周期 T = 256 * (7.3728M / 64), 因此 T*3375 等于 7.5s
    if(++g_bCount > 3375)
    {
        dir=~dir;
        g_bCount=0;
    }
}
```

实验过程如下:

首先, 单片机向连接了步进电动机的 I/O 口发送相应的数据, 便产生上述的时序, 控制



步进电动机进行转动。每过 7.5s，定时器中断处理程序执行以改变步进电动机的旋转方向。

下面是主函数代码（其中，`go_step_2phase(1000)`为调用步进电动机转动函数，控制步进电动机进行转动，其参数用来控制步进电动机转动的快慢，当这个参数变大时，步进电动机转速变慢；参数变小时，步进电动机转速变快）：

```
int main(void)
{
    DDRB=0xf0;           //端口 B 的高 4 位为输出，低 4 位为输入
    PORTB=0x11;         //初始化 I/O 口
    phase_pnt=1;        //指向数组的下标
    dir=0;
    init_timer0();      //初始化计数器
    while(1)
    {
        go_step_2phase(1000);
    }
}
```

4. 实验内容

阅读相关的书籍，学习步进电动机的转动原理和通过单片机控制步进电动机转动的方法。查阅 ATmega128 的芯片手册，学习单片机定时器的使用方法，并使用其控制步进电动机的转动方向。

5. 实验步骤

- 1) 编写实验代码，并编译得到 .hex 文件。
- 2) 使用 AVR Studio 将 .hex 文件下载到节点上。
- 3) 在实验板上会看到步进电动机正向转动，过了 7.5s 后步进电动机会反向转动，如此往复运行。

6. 思考题

- 1) 本程序中，步进电动机是两相的，如何实现一相、一二相的步进电动机的转动。
- 2) 如何修改程序能使步进电动机每隔 5s 转向一次。

7. 补充阅读

- [1] 徐勇军，安竹林，等. 无线传感器网络实验教程[M]. 北京：北京理工大学出版社，2007.
- [2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.
- [3] Xiaohua Zhang, Bingji Xu. Research on Stepper Motor Control Based on Single Chip and Serial Communication. World Congress on Intelligent Control and Automation, 2010, 3019-3023.

3.2.7 实验八 键盘与显示屏实验

1. 实验目的

本实验介绍了通过 ATmega128 的 I/O 口控制 4×4 矩阵键盘以及 SMG12864G2-ZK 液晶显示模块的方法。通过本实验，使读者掌握 4×4 矩阵键盘的驱动方法和 SMG12864G2-ZK 液晶显示模块的显示原理，以及 AVR 单片机 I/O 口的基本控制。



2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- SMG12864G2-ZK 液晶显示模块 1 块。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- WinAVR20050214。
- AVRStudio4.18。

3. 实验原理

本实验实现了将 4×4 矩阵键盘的按键值通过液晶显示模块进行显示，即通过按键模块获得按键值，并把按键值作为液晶显示模块显示函数的参数值进行显示。

(1) 按键模块

键盘原理图如图 3-14 所示，KB0~KB7 分别接 ATmega128 单片机的 PC0~PC7，其中 PC0~PC3 配置为输出，PC4~PC7 配置为输入。检测哪个按键被按下的方法为：从键盘 KB4 这一行开始每行依次输出 0，检查列线是否非全高，如果列线非全高，则说明该行有按键按下，然后根据第几列线读入为 0，即可判断在当前行的哪一列位置有按键按下。即先第 4 行输出 0(即 PC4 设置为 0)，检查列线是否非全高(即 PC0~PC3 是否全是 1)；全高则继续检查第 3 行、第 2 行、第 1 行直到某一行对应的列线非全高。另外，当按下按键时，无论速度有多快，都会出现接触一断开一接触一断开……这种不稳定的过程叫“抖动”。松开按键也是如此。为了防止此不稳定状态，在进入列检测时加入了延时程序用来防抖动。

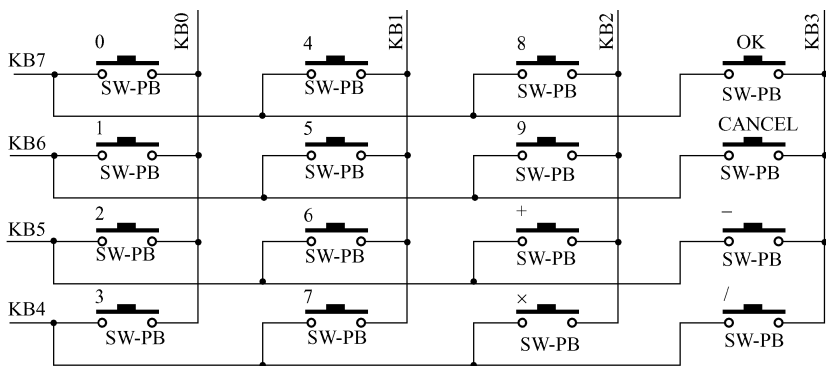


图 3-14 4×4 矩阵键盘原理图

以上过程对应的程序代码为：

```
void Check_Key(void)
{
    unsigned char row,col,tmp1,tmp2;
    //tmp1 用来设置 PC 口的输出，取反后使 PC4~PC7 中有一个为 0
```



```

tmp1 = 0x10;
for(row=0; row<4; row++)          //行检测
{
    PORTC = ~tmp1;                //使当前行对应的单片机端口输出 0
    tmp1 *= 2;                    //tmp1 左移一位
    //检测 PC0~PC3 中是否有一位为 0, 只要有, 则说明此行有键按下, 进入列检测
    if((PINC&0x0F) < 0x0F)
    {
        delay_ms(0.5);           //防抖动
        if((PINC&0x0F) < 0x0F)    //PC0~PC3 中有一个读入为 0
        {
            m = m + 1;
            tmp2 = 0x01;          //tmp2 用于检测 PC0~PC3 中哪一个读入为 0
            for(col=0; col<4; col++) //列检测
            {
                //如果当前列为低电平, 则可以判定此列读入为 0
                if((PINC&tmp2) == 0x00)
                {
                    //获取键值, 识别按键; key_Map 为按键的定义表数组
                    key_val = key_Map[row*4 +col];
                    return;        //返回
                }
                tmp2 *= 2;         //tmp2 左移一位
            }
        }
    }
}
//结束

```

其中 key_val 为保存按下按键的按键值所对应的字符串指针地址。

(2) 液晶显示模块

SMG12864G3-ZK 液晶显示模块与 ATmega128 单片机采用并行接口的方式进行硬件连接。其中 SMG12864G3-ZK 液晶显示模块与 ATmega128 单片机之间加入了 SN74LVCC3245ADW 双向缓冲器, 单片机用 PF3 端口控制该缓冲器的输入输出方向。

该模块的功能主要包括 3 个部分: 初始化部分、写指令/读写数据部分, 以及数字和字符的显示部分。初始化部分是液晶显示模块进行显示之前要进行的必要工作, 主要包括设置接口模式, 指令集选择, 清屏等工作, 相关代码如下:

```

void lcdinit(void)                //液晶显示控制器初始化子程序
{
    DDRE = _BV(PE6) | _BV(PE7);
    PORTE = _BV(PE6) | _BV(PE7);
    lcdwc(0x33);                  //接口模式设置
    delay3ms();                   //延时 3ms
    lcdwc(0x30);                  //基本指令集
}

```



```
delay3ms(); //延时 3ms
lcdwc(0x30); //重复送基本指令集
delay3ms(); //延时 3ms
lcdwc(0x01); //清屏控制字
delay3ms(); //延时 3ms
lcdon(); //开始显示
}
```

写指令、读写数据部分是液晶显示模块驱动程序的关键部分，所有对液晶显示模块的操作都是通过调用这部分程序实现的。这部分的时序需要严格按照时序图设置（时序图见 SMG12864G3-ZK 数据手册第 3 页）。其中在向液晶显示控制器写数据和指令以及从液晶显示控制器读数据前必须先调用 `lcdwaitidle()` 函数来确认液晶显示控制器内部处于非忙状态。上述 3 个函数的代码如下：

```
//向液晶显示控制器写数据函数
void lcdwd(unsigned char d)
{
    unsigned char i,j;
    lcdwaitidle(); //ST7920 液晶显示控制器忙检测
    //设置双向缓冲器为输出
    DDRF = _BV(PF3);
    PORTF = _BV(PF3);
    DDRA = 0XFF; //设置单片机 A 口为输出口
    //设置单片机与液晶显示模块以及双向缓冲器的控制信号线相连的端口为输出，分别为 CS(PB0)
    //液晶显示模块数据命令选择端、R/W(PB1)液晶显示模块读写控制信号、SCLK(PB2)液晶显示
    //模块使能信号、DIR_CTL(PB3)双向缓冲器使能信号线
    DDRB = _BV(PB0)|_BV(PB1)|_BV(PB2)|_BV(PB3);
    PORTB = _BV(PB3); //使能双向缓冲器
    PORTB |= 0X00; //初始化各控制信号线
    PORTB |= _BV(PB0); //设置为向液晶显示模块输出数据
    PORTB &= ~_BV(PB1); //设置为写模式
    //延时约 1628ns，此延时必须大于或等于数据手册规定延时，详见 SMG12864G2-ZK 数据手册
    for(i=0; i<12; i++)
    {
        asm("nop");
    }
    PORTB |= _BV(PB2); //使能液晶显示模块
    PORTA=d; //写入 A 端口要发送的数据
    //延时约 1628ns，此延时必须大于或等于数据手册规定延时，详见 SMG12864G2-ZK 数据手册
    for(j=0; j<12; j++)
    {
        asm("nop");
    }
    PORTB &= ~_BV(PB2); //结束液晶显示模块使能
}
```



```

//向液晶显示控制器送指令函数
void lcdwc(unsigned char c)
{
    unsigned char i,j;
    lcdwaitidle(); //ST7920 液晶显示控制器忙检测
    //设置双向缓冲器为输出
    DDRF = _BV(PF3);
    PORTF = _BV(PF3);
    DDRA = 0xFF; //设置单片机 A 口为输出输出数据
    //设置单片机与液晶显示模块以及双向缓冲器的控制信号线相连的端口为输出, 分别为 CS(PB0)
    //液晶显示模块数据命令选择端、R/W(PB1)液晶显示模块读写控制信号、SCLK(PB2)液晶显示模
    //块使能信号、DIR_CTL(PB3)双向缓冲器使能信号线
    DDRB = _BV(PB0)|_BV(PB1)|_BV(PB2)|_BV(PB3);
    PORTB = _BV(PB3); //使能双向缓冲器
    PORTB |= 0X01; //初始化各控制信号线
    PORTB &= ~_BV(PB0); //设置为向液晶显示模块输出命令
    PORTB &= ~_BV(PB1); //设置为写模式
    //延时约 1628ns, 此延时必须大于或等于数据手册规定延时, 详见 SMG12864G2-ZK 数据手册
    for(i=0; i<12; i++)
    {
        asm("nop");
    }
    PORTB |= _BV(PB2); //使能液晶显示模块
    PORTA =c ; //写入 A 端口要发送的命令
    //延时约 1628ns, 此延时必须大于或等于数据手册规定延时, 详见 SMG12864G2-ZK 数据手册
    for(j=0; j<12; j++)
    {
        asm("nop");
    }
    PORTB &= ~_BV(PB2); //结束液晶显示模块使能

//从液晶显示控制器读数据函数
unsigned char lcdrd(void)
{
    unsigned char i,j,k,d;
    lcdwaitidle(); //ST7920 液晶显示控制器忙检测
    DDRF = _BV(PF3);
    PORTF &= ~_BV(PF3); //设置双向缓冲器为输入
    DDRA = 0X00; //设置单片机 A 口为输入输入数据
    //设置单片机与液晶显示模块以及双向缓冲器的控制信号线相连的端口为输出, 分别为 CS(PB0)
    //液晶显示模块数据命令选择端、R/W(PB1)液晶显示模块读写控制信号、SCLK(PB2)液晶显示
    //模块使能信号、DIR_CTL(PB3)双向缓冲器使能信号线
    DDRB = _BV(PB0)|_BV(PB1)|_BV(PB2)|_BV(PB3);
    PORTB = _BV(PB3); //使能双向缓冲器

```



```
PORTB |= 0X00; //初始化各控制信号线
PORTB |= _BV(PB0); //设置从液晶显示模块输入数据
PORTB |= _BV(PB1); //设置为读模式
//延时约 1628ns, 此延时必须大于或等于数据手册规定延时, 详见 SMG12864G2-ZK 数据手册
for(i=0;i<12;i++)
{
    asm("nop");
}
PORTB |= _BV(PB2); //使能液晶显示模块
//延时约 1628ns, 此延时必须大于或等于数据手册规定延时, 详见 SMG12864G2-ZK 数据手册
for(j=0; j<12; j++)
{
    asm("nop");
}
PORTA = 0XFF;
d = PINA; //从 A 口读入数据
//延时约 2170ns, 此延时必须大于或等于数据手册规定延时, 详见 SMG12864G2-ZK 数据手册
for(k=0;k<16;k++)
{
    asm("nop");
}
PORTB &= ~_BV(PB2); //结束液晶显示模块使能
return d; //返回从液晶显示屏读入的数据
}
```

通过调用前面所述的初始化和读写函数就可以实现数字和字符的显示, 在实验中首先使用了 `putchar()` 函数对上述 3 个基本读写函数进行了封装, 然后使用 `putstr()` 提供基于字符串的读写, 最后使用了 `putstrxy()` 提供指定位置读写的接口。该函数的参数 `cx`、`cy` 为显示内容要写到 LCD 显示屏上的坐标, `s` 为要显示的字符串首地址。`putstrxy()` 的实现如下:

```
//在(cx,cy)字符位置写字符串子程序
void putstrxy(unsigned char cx,unsigned char cy,unsigned char *s)
{
    CXPOS=cx; //置当前 X 位置为 cx
    CYPOS=cy; //置当前 Y 位置为 cy
    putstr(s);
}
```

(3) 按键模块和液晶显示模块的结合

把按键检测值所对应的存放地址作为液晶显示函数的参数进行按键检测值的显示。如主函数中 `putstrxy(6,1,key_val)` 函数, 其中参数 6 和 1 为显示内容首地址坐标值, `key_val` 为得到的按键检测值所对应的存放地址。

```
int main(void)
{
```



```
//单片机与键盘连接端口初始化
DDRC = 0xF0;
PORTC = 0xFF;
lcdinit(); //LCD 初始化
while(1)
{
    charlcdfill(' '); //显示清屏
    Check_Key(); //按键值检测函数
    if(key_val != m) //防止重复循环同一个按键
    {
        putstrxy(6,1,key_val); //液晶显示函数
        m = key_val; //保存当前按键值地址
        delay(300); //延时约 300ms
    }
}
```

4. 实验内容

阅读 ATmega128 数据手册的 I/O 口的控制原理，能够根据 4×4 矩阵键盘的原理图编写 4×4 矩阵键盘的按键检测与防抖动程序。

阅读 SMG12864G3-ZK 数据手册，能够使用通用 LCD 显示模块进行数字、字符的显示，并能够结合按键程序进行按键值的显示。

5. 实验步骤

1) 启动 WinAVR，建立一个新的工程，编写代码，建立 Makefile 文件，并编译得到.hex 文件。

2) 使用 AVR Studio 将.hex 文件下载到节点上。

3) 按下矩阵键盘上的某个键，LCD 液晶显示屏上会显示该按键的值。

6. 思考题

1) 如果键盘的第 1 行第 2 列和第 3 行第 4 列两个键值改为@和&，应如何修改程序？

2) 如何利用 SMG12864G2-ZK 液晶显示模块显示图形？

7. 补充阅读

[1] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel,2010.

[2] SMG12864G3-ZK Datasheet. <http://download.sunman.cn/>. 2010.

3.2.8 实验九 嵌入式平台基于MiniGUI的简易编辑器实验

1. 实验目的

本实验在 ARM 开发母板 (IOT-ARMMB) 上使用键盘、TFT 液晶屏实现了一个基于 MiniGUI 的简单编辑器程序。该实验首先在 μClinux 下实现了 MiniGUI 的移植，简易编辑器程序利用 MiniGUI 创建主窗口，在窗口过程函数中处理字符消息，实现一个简易编辑器的功能。通过该演示例子，使读者对开发母板的键盘和 TFT 液晶屏有更进一步的了解，并掌握在 μClinux 下移植 MiniGUI 和交叉编译 MiniGUI 应用程序的操作方法。



2. 实验设备

(1) 硬件

- ARM 实验母板 (IOT-ARMMB) 1 套。
- 传感器及外围模块 (IOT-MODULES) 1 套。
- PC (含串口) 1 台。

(2) 软件

- Linux 操作系统。
- μ Clinux 2.4x 发行包和 arm-elf-gcc 交叉编译环境。

3. 实验原理

本实验的演示程序除使用了开发板上的基本部件之外, 为了进一步丰富演示效果, 还使用了 TFT 液晶屏, 实物照片如图 3-15 所示。



图 3-15 TFT 液晶屏

MiniGUI 是一款面向实时嵌入式系统的图形用户界面支持系统, 其为应用程序提供了一组轻量级的窗口和图形设备接口。利用这些接口, 每个应用程序可以建立多个主窗口, 然后在这些主窗口中创建按键和编辑框等控件。MiniGUI 还为用户提供了丰富的图形功能, 以显示各种格式的位图, 还可以在窗口中绘制复杂图形。MiniGUI 与操作系统的关系如图 3-16 所示。

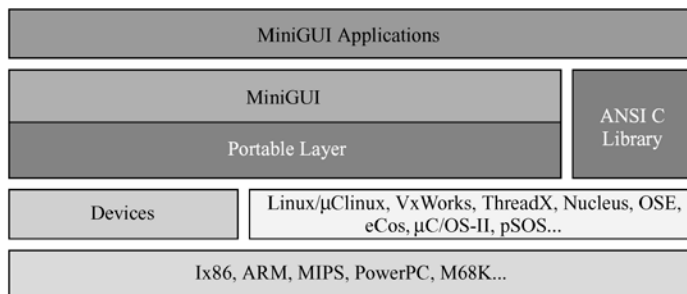


图 3-16 MiniGUI 与操作系统的关系

像 μ Clinux、 μ C/OS-II、eCos 等操作系统, 通常运行在没有 MMU (内存管理单元, 用于提供虚拟内存支持) 的 CPU 上; 这时, 往往就没有进程的概念, 而只有线程或者任务的概



念，这样 MiniGUI 的运行环境也就大相径庭。因此，为了适合不同的操作系统环境，可将 MiniGUI 配置成 3 种不同的运行模式：MiniGUI-Threads、MiniGUI-Processes、MiniGUI-Standalone。

MiniGUI-Threads 为多线程的应用模式，运行在 MiniGUI-Threads 上的程序可以在不同线程中建立多个窗口，但所有窗口在同一个进程或者地址空间中运行。这种运行模式主要用来支持大多数传统意义上的嵌入式操作系统，如 $\mu\text{C}/\text{OS-II}$ 、eCos 等。当然，在 Linux 和 μClinux 上，MiniGUI 也能以 MiniGUI-Threads 的模式运行。

MiniGUI-Processes 为多进程的应用模式，MiniGUI-Processes 上的每个程序是单独的进程，每个进程也可以建立多个窗口，并且实现了多进程窗口系统。MiniGUI-Processes 适合于具有完整 UNIX 特性的嵌入式操作系统，比如嵌入式 Linux。

MiniGUI-Standalone 为独立进程的应用模式，这种运行模式下，MiniGUI 可以以独立任务的方式运行，既不需要多线程也不需要多进程的支持，这种运行模式适合功能单一的应用场合。比如在一些使用 μClinux 的嵌入式产品中，因为各种原因而缺少线程支持，这时，就可以使用 MiniGUI-Standalone 来开发应用软件。MiniGUI-Standalone 模式的适应面最广，支持几乎所有的操作系统，甚至包括 DOS。但不论采用哪种运行模式，MiniGUI 为上层应用软件提供了最大程度上的一致性；只有少数几个涉及初始化的接口在不同运行模式上有所不同。

要在开发板上的 μClinux 操作系统上运行 MiniGUI 应用程序，首先要在宿主机（即 PC）上对 MiniGUI 的源代码进行编译（交叉编译），生成 MiniGUI 静态链接库文件（ μClinux 不支持共享库，故 MiniGUI 要使用静态链接库）。然后 MiniGUI 应用程序也在宿主主机上进行交叉编译，并与 MiniGUI 静态链接生成目标板可运行的 FLAT 格式文件，交叉编译示意图如图 3-17 所示。最后，在目标板的 μClinux 操作系统上搭建 MiniGUI 运行环境，即安装 MiniGUI 的资源文件和运行时的配置文件，再将 MiniGUI 应用程序（FLAT 格式文件）下载和运行。

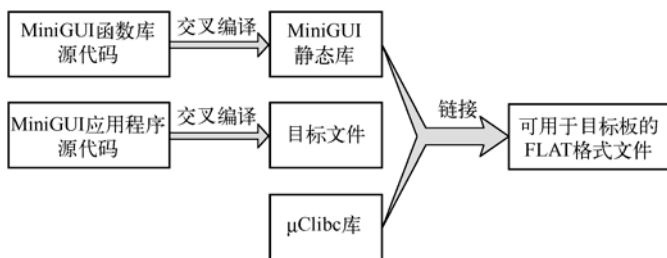


图 3-17 交叉编译示意图

由于不同目标机的图形显示设备与输入设备不一致，所以用户需要为 MiniGUI 编写相应的图形引擎和输入引擎（即驱动程序），再与 MiniGUI 的源代码一起交叉编译生成 MiniGUI 静态链接库文件。光盘提供的 MiniGUI-STR 压缩包已经加入了适用于 IOT-ARMMB 开发母板的图形引擎和输入引擎。IOT-ARMMB 开发母板上的 16 个按键定义如图 3-18 所示。



TAB (S1)	ENTER (S2)	ESC (S3)	BACK (S4)
9 (S5)	0 (S6)	- (S7)	SPACE (S8)
5 (S10)	6 (S11)	7 (S12)	8 (S13)
1 (S14)	2 (S15)	3 (S16)	4 (S17)

图 3-18 开发母板上的 16 个按键定义

在 MiniGUI 程序的入口点中，首先调用 `InitMainWindow()` 建立主窗口，然后调用 `ShowWindow()` 显示主窗口。紧接着进入消息循环，不停执行 `TranslateMessage()` 和 `DispatchMessage()` 函数来处理消息。相关代码如下：

```

/*****
*名称: MiniGUIMain()
*功能: MiniGUI 程序入口点
*入口参数: argc      参数个数
*          argv      参数字符串指针
*出口函数: 返回 0
*****/

int MiniGUIMain(int argc, const char *argv[])
{
    MSG Msg;
#ifdef _LITE_VERSION
    SetDesktopRect(0,0, 800,600);
#endif
    InitMainWindow();           //初始化主窗口
    ShowWindow(hMainWnd, SW_SHOWNORMAL); //显示窗口
    //进入消息循环处理消息
    while (GetMessage(&Msg, hMainWnd))
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    MainWindowThreadCleanup (hMainWnd);

    return(0);
}

```

键盘是基本的输入设备之一，MiniGUI 程序通过处理窗口的键盘事件来接收按键输入，键盘事件有 `MSG_KEYDOWN`（按键按下）、`MSG_KEYUP`（键释放）、`MSG_SYSKEYDOWN`（系统击键消息，即〈Alt〉键）和 `MSG_SYSKEYUP`（系统击键消息）等。

一般窗口用户程序不直接处理键盘事件，而是处理 `MSG_CHAR` 和 `MSG_SYSCHAR` 事件，即字符消息。字符消息由消息循环代码中的 `TranslateMessage` 函数产生，`TranslateMessage` 函数将键盘消息 `MSG_KEYDOWN` 和 `MSG_SYSKEYDOWN` 转换成字符消



息，同时也将按键扫描码转换成相应的字符编码（字符消息中的 wParam 参数）。相关的代码如下：

```
/******  
*名称: WinProc()  
*功能: 主窗口过程函数  
* 处理MSG_PAINT消息, 在窗口中显示相关内容  
*入口参数: hWnd 窗口句柄  
*          message 消息  
*          wParam 消息附加参数 1  
*          lParam 消息附加参数 2  
*****/  
static int WinProc(HWND hWnd, int message, WPARAM wParam, LPARAM lParam)  
{  
    HDC hdc;  
    static char str_buf[1000]; //待输出的字符串  
    static int no; //当前行中已输出字符的个数  
    static int y; //当前行中已输出字符的个数  
    static int char_w, char_h;  
    //根据消息不同来调用不同的消息处理代码。  
    switch(message)  
    {  
        case MSG_CREATE:  
            CreateCaret(hWnd, NULL, 2, GetSysCharHeight()); //创建光标  
            SetCaretBlinkTime(hWnd, 500); //指定光标闪烁频率  
            char_w = GetSysCharWidth(); //获取系统字符宽度  
            char_h = GetSysCharHeight(); //获取系统字符高度  
            sprintf(str_buf, "");  
            no = 0;  
            y = 0;  
            break;  
        case MSG_PAINT:  
            hdc = BeginPaint(hWnd); //处理MSG_PAINT绘制消息  
            //启动绘制  
            TextOut(hdc, 0, y, str_buf); //输出文本内容  
            EndPaint(hWnd, hdc); //结束绘制  
            SetCaretPos(hWnd, 0,0);  
            break;  
        case MSG_SETFOCUS:  
            ShowCaret(hWnd); //处理MSG_SETFOCUS聚焦消息  
            //显示光标  
            break;  
        case MSG_KILLFOCUS:  
            HideCaret(hWnd); //处理MSG_KILLFOCUS撤销聚焦消息  
            //隐藏光标  
            break;  
        case MSG_CLOSE:  
            DestroyCaret(hWnd); //处理MSG_CLOSE关闭消息  
            //释放插入标记  
            DestroyMainWindow(hWnd); //释放主窗口
```



```
PostQuitMessage(hWnd); //发送退出消息
break;
case MSG_CHAR: //处理 MSG_CHAR 字符消息
    if(wParam==127) //判断是不是 BackSpace 键
    {
        if(no>0) //若是则删除前一个字符
        {
            no--;
            str_buf[no] = ' ';
        }
    }
    else //若不是 BackSpace 键，则添加字符
    {
        str_buf[no++] = wParam; //取得字符，并加入 str_buf 字符串
        str_buf[no] = '\0';
    }
    //以 30 个字符为一行，若遇回车键，则换行
    if( (no%30) == 0 || (wParam == 0x0D ) )
    {
        if(no>0)
        {
            y = y + char_h;
            no = 0;
            sprintf(str_buf, "");
        }
    }
    hdc = BeginPaint(hWnd); //启动绘制
    TextOut(hdc, 0, y, str_buf); //显示当前字符
    EndPaint(hWnd, hdc); //结束绘制
    SetCaretPos(hWnd, no * char_w, y); //设置插入字符位置，即光标位置
    break;
default:
    return(DefaultMainWinProc(hWnd, message, wParam, lParam));
}
return(0);
}
```

4. 实验内容

在 ARM 下的 μ Clinux 开发平台构建 MiniGUI 开发环境，初步掌握移植 MiniGUI for μ Clinux 的基本方法和应用配置的一般步骤。

初步掌握 MiniGUI 开发的一般步骤，能够基于提供的图形引擎和输入引擎设计实现简单的图形用户界面应用程序。

5. 实验步骤

(1) 在PC上安装MiniGUI并移植图形引擎和输入引擎

启动 Linux 系统，以 root 用户登录系统。在/home 目录下新建一个名为 minigui 的文件



夹，从光盘中将 MiniGUI-STR 中的 libminigui-str-1.6.2.tar.gz、minigui-res-str-1.6.tar.gz 和 mg-samples-str-1.6.2.tar.gz 文件复制到此文件夹中，并解压到当前文件夹中。

说明：由于交叉编译 MiniGUI 需要用到 Autoconf 和 Automake 工具，若在安装 Linux 操作系统时没有选择这两个工具，可以使用系统的添加/删除应用程序来安装。

由于光盘提供的 MiniGUI-STR 压缩包已经完成了适用于 IOT-ARMMB 开发母板的图形引擎和输入引擎的移植，在这里略过，有兴趣的读者可以参阅 MiniGUI 图形引擎和输入引擎移植的相关资料。

(2) 交叉编译MiniGUI

打开终端窗口，使用以下指令执行 MiniGUI-STR 的编译和库文件的安装。操作结束后，在/home/minigui_lib 目录下会有 MiniGUI 的静态库文件，这样 MiniGUI 只需要指定链接/home/minigui_lib 目录下的 MiniGUI 库文件，即可生成在 IOT-ARMMB 开发板上运行的 MiniGUI 程序文件（FLAT 格式文件）。

```
#cd /home/minigui/libminigui-str-1.6.2
#./autogen.sh
#./buildlib-wlarmmb7
#make
#make install
```

(3) 交叉编译MiniGUI应用程序

在 mg-samples-str-1.6.2/src 目录下建立源文件 sedit.c，编写实验程序代码。在 mg-samples-str-1.6.2 中编写一个配置脚本 build-uclinux-wlarmmb7，内容如下：

```
rm config.cache config.status -f

UCLINUX_DIR=/usr/src/uClinux-dist

CC=arm-elf-gcc \
CFLAGS="-Os -g -Dlinux -D_linux__ -Dunix -D_uClinux__ -DEMBED -
I${UCLINUX_DIR}/lib/uClibc/include -I${UCLINUX_DIR}/lib/libm -I${UCLINUX_DIR}/lib/
libcrypt_old -I${UCLINUX_DIR} -fno-builtin -nostartfiles -D_PIC__ -fpic -msingle-
pic-base -I${UCLINUX_DIR}/linux-2.4.x/include -I/home/minigui_lib/include " \
LDFLAGS="-Wl,-elf2flt -static ${UCLINUX_DIR}/lib/uClibc/lib/crt0.o ${UCL
INUX_DIR}/lib/uClibc/lib/crti.o ${UCLINUX_DIR}/lib/uClibc/lib/crtn.o -L${UCLINUX_
DIR}/lib/uClibc/. -L${UCLINUX_DIR}/lib/uClibc/lib -L/${UCLINUX_DIR}/lib/libm -
L/${UCLINUX_DIR}/lib/libnet -L/${UCLINUX_DIR}/lib/libdes -L/${UCLINUX_DIR}/lib/
libaes -L/${UCLINUX_DIR}/lib/libcap -L/${UCLINUX_DIR}/lib/libcrypt_old -
L/${UCLINUX_DIR}/lib/libssl -L/${UCLINUX_DIR}/lib/zlib -L/home/minigui_lib/lib -lc " \
./configure \
--build=i386-linux \
--host=arm-elf-linux \
--target=arm-elf-linux
```

说明：可以从 mg-samples-str-1.6.2 目录下复制 build-uclinux-arm3000 文件，然后更改文



件名为 build-uclinux-wlarmmb7，修改文件内容。

修改 mg-samples-str-1.6.2/src 目录下的 Makefile.am 文件，设置编译 sedit.c 文件，代码如下：

```
COMMON_PROGS=helloworld mycontrol dialogbox simplekey \  
    static button listbox edit combobox menubutton progressbar \  
    trackbar newtoolbar propsheet \  
    scrollbar painter capture bitblt stretchblt loadbmp drawicon \  
    createicon caretdemo cursordemo input bmpbkgnd sedit \  
.....  
noinst_PROGRAMS=$(COMMON_PROGS) $(LITE_PROGS)  
  
sedit_SOURCES=sedit.c  
helloworld_SOURCES=helloworld.c  
mycontrol_SOURCES=mycontrol.c  
dialogbox_SOURCES=dialogbox.c  
.....
```

打开终端窗口，执行以下指令编译 MiniGUI 应用程序。编译成功通过后，在 mg-samples-str-1.6.2/src 目录下会生成 sedit 文件（FLAT 格式文件）。

```
#cd /home/minigui/mg-samples-str-1.6.2  
#./build-uclinux-wlarmmb7  
#make
```

继续执行以下指令安装资源文件。执行以下指令之后，在 /usr/local/lib/minigui 目录下会产生一个 res 目录，该目录下保存有 MiniGUI 的资源文件(如图片、图标和字体等)。

```
#cd /home/minigui/minigui-res-str-1.6  
#make install
```

MiniGUI 运行时需要使用 MiniGUI.cfg（即 MiniGUI 运行时配置文件），主要用来配置使用的图形引擎、输入引擎和窗口字体等。修改 /home/minigui_lib/etc 下的 MiniGUI.cfg 文件，使用 commlcd 图形引擎和 comm 输入引擎，代码如下：

```
.....  
[system]  
# GAL engine  
gal_engine=commlcd  
  
# IAL engine  
ial_engine=comm  
  
mdev=/dev/mouse  
mtype=IMPS2  
  
.....
```



(4) 在目标板的 μ Clinux操作系统建立MiniGUI运行环境和执行MiniGUI应用程序至此, MiniGUI 的资源文件、运行时配置文件和应用程序(FLAT 格式文件)都已经产生了, 接着在目标板的 μ Clinux 操作系统建立 MiniGUI 运行环境。

连接好 IOT-ARMMB 开发母板的串口线、网线和电源线, 并将 TFT 液晶模块安装到 WLARMMB7 开发母板上。启动开发板上的 μ Clinux, 设置和重启 NFS 服务(具体步骤详见实验二)。

将 PC 端/home/minigui_lib/etc/MiniGUI.cfg, /home/minigui/mg-samples-str-1.6.2/src/sedit 这两个文件和/usr/local/lib/minigui/res 文件夹复制到 NFS 的共享目录/home/armwork(此处 在实验二中设置为/home/armwork)。将光盘中的 rescp.sh 文件复制到/home/armwork/res 目录下。由于输入引擎需要用到 I²C 驱动, 所以还要将光盘中的 i2c 文件复制到/home/armwork 下。

在开发板的 μ Clinux 文件系统中建立与 PC 端相对应的目录及子目录, 目录及子目录列表如下:

```
/usr/pro
/usr/pro/i2c
/usr/local/lib/minigui/res/bmp
/usr/local/lib/minigui/res/cursor
/usr/local/lib/minigui/res/font
/usr/local/lib/minigui/res/icon
/usr/local/lib/minigui/res/imetab
```

在 PC 端使用 minicom 软件操作开发板, 挂载 PC 端的 NFS(挂载方法详见实验二), 并执行以下命令复制相关文件到开发板目录。

```
/>cd /mnt
/mnt>cp MiniGUI.cfg /usr/pro
/mnt>cp sedit /usr/pro
/mnt>cd i2c
/mnt/i2c>cp i2c /usr/pro/i2c
/mnt/i2c>cp loadi2c /usr/pro/i2c
/mnt/i2c>cd ..
/mnt>cd res
/mnt/res>./cpres.sh
```

紧接着执行以下命令运行 MiniGUI 程序。

```
/>cd /usr/pro/i2c
/usr/pro/i2c >./loadi2c
/usr/pro/i2c >cd ..
/usr/pro>./sedit
```

正确运行 sedit 程序之后, 在开发板的 TFT 液晶屏上将会显示一个简易编辑窗口, 使用开发板上的功能键〈0〉~〈9〉输入数字, 使用功能键〈Back〉删除已经输入的字符, 使用功能键〈Enter〉换行。程序运行结果如图 3-19 所示。

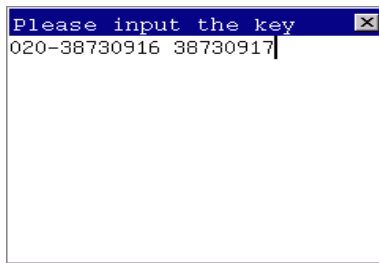


图 3-19 程序运行结果

6. 思考题

- 1) 如何进行 MiniGUI 针对不同开发板图形引擎和输入引擎的移植?
- 2) 在本实验中在开发板目录建立 MiniGUI 运行环境都需要哪些文件的支持?
- 3) 如何利用消息处理实现一个 MiniGUI 应用程序, 根据按键输入显示不同的图片?

7. 补充阅读

- [1] MiniGUI Technology White Paper. <http://www.minigui.com/>.
- [2] Datasheet for MiniGUI V2.0.4/V1.6.10. <http://www.minigui.org/>.
- [3] MiniGUI User Manual Version2.0.<http://www.minigui.org/>.
- [4] MiniGUI Programming Guide Version2.0.<http://www.minigui.org/>.

3.3 条形码与RFID标识实验

条形码和 RFID 是目前使用较为广泛的两种标识技术。条形码由于使用方便、价格低廉, 已经获得了大范围使用; RFID 可以实现较远距离、非接触式批量读写、更高的安全性, 也已经在很多场合使用。本节介绍条形码和 RFID 标识实验的原理及实验。由于它们仅在读写装置的前端有所差异, 其后端处理有很大相似之处, 本节仅以高频及超高频无源 RFID 实验进行介绍, 有源 RFID 部分的实验原理类似于单跳的通信, 将在第 4 章进行阐述。

3.3.1 条形码及RFID技术概述

条形码和 RFID 技术是物联网感知层中非常重要的数据标识技术, 可以用来快速准确地识别目标对象并获取相关数据, 目前均有较多应用, 下面分别进行介绍。

(1) 条形码技术

条形码是将宽度不等的多个黑条和空白, 按照一定的编码规则排列, 用以表达一组信息的图形标识符。条形码起源于 20 世纪 20 年代, 最初的目的是用来表达信封上收件人的地址。条形码技术发展到现在已经可以标出物品的各种信息, 如生产日期、商品名称、生产国家、制造厂家、图书分类号、邮件起止地点、类别、日期等, 因而在商品流通、图书管理、邮政管理等许多领域都得到了广泛的应用。条形码技术和人们的生活息息相关。比如, 目前中国几乎所有的超市都使用条形码进行货品的管理。

条形码的信息读取一般是通过光扫描完成的, 通俗来说, 就是必须是可视的。其原理是物体的颜色是由其反射光的类型决定的, 白色物体能反射各种波长的可见光, 黑色物体则吸



收各种波长的可见光，所以当条形码扫描器光源发出的光在条形码上反射后，反射光照射到条形码扫描器内部的光电转换器上，光电转换器根据强弱不同的反射光信号，转换成相应的电信号。电信号输出到条形码扫描器的放大电路增强信号之后，再送到整形电路将模拟信号转换成数字信号。白条、黑条的宽度不同，相应的电信号持续时间长短也不同。然后译码器通过测量脉冲数字电信号 0 与 1 的数目来判别条和空的数目，再根据对应的编码规则（如 UPC 码，EAN 码）将条形符号转换成相应的数字、字符信息。

条形码按照维度来划分，可分为一维码，二维码。目前，在中国应用最广泛的是一维码。一维码简便易用的特点决定了它能够被大范围的使用，但是随着社会的进步，产业的发展，一维码记录的信息有限且简单，逐渐不能满足人们记录和标识物品信息的需求，所以二维码应运而生了。二维条形码增加了一定面积上的条形码信息密度和信息量。二维条形码在水平和垂直方向都表达信息。其也有很多码制，主要可分为堆叠式二维条形码和矩阵式二维条形码两种类型。

1) 堆叠式二维条形码又称行排式二维条形码、堆积式二维条形码或层排式二维条形码，其编码原理是建立在一维条形码基础之上，按需要堆积成两行或多行。它在编码设计、校验原理、识读方式等方面继承了一维条形码的一些特点，识读设备和条形码印刷与一维条形码技术兼容，但由于行数的增加，需要对行进行判定，其译码算法与软件也不完全相同于一维条形码。堆叠式条形码主要的码制有 PDF417、Code16K、Code49。

2) 矩阵式二维条形码又称棋盘式二维条形码，它是在一个矩形空间通过黑、白像素在矩阵中的不同分布进行编码。在矩阵相应元素位置上，用点（方点、圆点或其他形状）的出现表示二进制“1”，点的不出现在表示二进制的“0”，点的排列组合确定了矩阵式二维条形码所代表的意义。矩阵式二维条形码是建立在计算机图像处理技术、组合编码原理等基础上的一种新型图形符号自动识读处理码制。其主要的码制有 Data Matrix、Aztec、Maxi Code、QR Code 等。

由于受信息容量的限制，一维条形码用于对物体进行标识，而不是对物体描述。这样要是不通过链接数据库查询物体的信息，这个标识就没有意义。而二维条形码的信息容量增大，可存放 1K 字符，可以用于对物体进行描述，而且无需链接数据库就能用扫描仪直接读取内容，这在物流中非常有用。由于可以加密，二维条形码还具有保密性高的特点。当安全级别很高时，即使污损 50% 仍可读取完整信息。因此，二维条形码具有如下特点。

1) 高密度编码，信息容量大：可容纳多达 1850 个大写字母或 2710 个数字或 1108 字节的数据，或 500 多个汉字，比普通条形码的信息容量约高几十倍。

2) 编码范围广：该条形码可以对图片、声音、文字、签字、指纹等可以数字化的信息进行编码并用条形码表示出来，可以表示多种语言文字，也可表示图像数据。

3) 容错能力强，具有纠错功能：这使得二维条形码因穿孔、污损等引起局部损坏时，照样可以正确得到识读，损毁面积达 50% 仍可恢复信息。

4) 译码可靠性高：它比普通条形码的译码错误率（百万分之二）要低得多，误码率不超过千万分之一。

5) 可引入加密措施：保密性、防伪性好。

6) 成本低，易制作，持久耐用。

7) 条形码符号形状、尺寸大小比例可变。



二维条形码的这些优点,使它在邮政、铁路、机场、码头等物流上被采用,实现了货物运输全过程跟踪,加快了运输过程中数据处理的速度。但是 RFID 技术的出现就使得条形码相形见绌了。首先,RFID 标签存储信息的容量是二维条形码的几百倍;其次,RFID 标签的 ID 唯一性使得它能唯一的标识世界上每一件物品,而条形码往往都只能标识某一类的物品;再次,RFID 标识系统能同时识别大量的带标签的物品,这样在实际应用中大大提高了速度;最后,RFID 标识系统的识别范围能达到几米甚至几十米,而条形码都只能近距离的扫描;另外,条形码的内容无法随意地修改,而 RFID 标签可以反复重新改写内容。因此,由于 RFID 的技术优势将使它的发展空间比条形码更广阔。

(2) RFID 技术

RFID 技术是一种利用射频信号自动识别目标对象并获取相关信息的无线通信技术,又称电子标签 (E-Tag),已经有数十年的应用历史,目前已在一些领域开始取代传统的条形码技术。标签 (Tag) 信息可在多个对象与读卡器 (Reader) 间同时传输,在一定范围内不受物理障碍、距离的限制。

RFID 的基本技术原理起源于二战时期,最初盟军利用无线电数据技术来识别敌我双方的飞机和军舰。战后,由于较高的成本,该技术一直主要应用于军事领域,并未很快在民用领域得到推广应用。直到 20 世纪八九十年代,随着芯片和电子技术的提高和普及,欧洲开始率先将 RFID 技术应用到公路收费等民用领域。到 21 世纪初,RFID 迎来了一个崭新的发展时期,其在民用领域的价值开始得到世界各国的广泛关注,特别是在西方发达国家,RFID 技术大量应用于生产自动化、门禁、公路收费、停车场管理、身份识别、货物跟踪等民用领域中。目前,中国国内大多数公司的考勤都采用 RFID 技术,其新的应用范围还在不断扩展,层出不穷。

RFID 技术的基本工作原理是:标签进入磁场后,接收读卡器发出的射频信号,凭借感应电流所获得的能量发送出存储在芯片中的产品信息 (Passive Tag, 无源标签或被动标签),或者主动发送某一频率的信号 (Active Tag, 有源标签或主动标签);读卡器读取信息并解码后,送至后台控制端进行有关信息数据处理。

一套完整的 RFID 系统,由 Reader 与 Tag,也就是所谓的应答器 (Transponder) 及应用软件系统 3 部分所组成。其工作原理是 Reader 发射一特定频率的无线电波能量给 Transponder,用以驱动 Transponder 电路将内部的数据送出,此时 Reader 便依序接收解读数据,送给应用程序做相应的处理。

RFID 按照能源的供给方式分为无源 RFID、有源 RFID,以及半有源 RFID。无源 RFID 读写距离近,价格低;有源 RFID 可以提供更远的读写距离,但是需要电池供电,成本要更高一些,适用于远距离读写的应用场合。

RFID 按应用频率的不同分为低频 (LF)、高频 (HF)、超高频 (UHF)、微波 (MW),相对应的代表性频率分别为:低频 135kHz 以下、高频 13.56MHz、超高频 860~960MHz、微波 2.4GHz, 5.8GHz。电子标签的工作频率是其最重要的特点之一。电子标签的工作频率不仅决定着射频识别系统工作原理 (电感耦合还是电磁耦合)、识别距离,还决定着电子标签及读写器实现的难易程度和设备的成本。下面对高频和超高频以及有源 RFID 进行介绍。

(3) 高频 RFID

13.56MHz 的高频波段是世界范围内有效的国际科学和医学波段。高频 RFID 标签一般



以无源为主, 射频的距离一般小于 1m。工作时, 标签必须位于读卡器天线辐射的近场区内才能和读写器交换数据。高频 RFID 系统的这种近场感应耦合减少了无线电干扰, 因此使得它具有极强的抗干扰能力。

高频标签多半是无源的, 因此成本较低, 可以做成卡状、薄皮状等。高频系统的特点是: 成本较低, 穿透非金属物体的能力强, 工作频率受无线电频率的干扰低, 大多应用在证件防伪、电子支付(如电子车票)、门禁系统中。

高频技术相比下面即将介绍的超高频技术成熟很多, 已经得到广泛的应用, 目前市场的标签主要以高频标签为主。高频技术也得到了用户的好评, 很多医药行业、图书馆等都青睐高频标签。高频标签非常适合应用在单品极方面, 因为单品极标签需要体积足够小, 抗液体、金属干扰力强等要求, 都正好符合高频标签的特点。

在标准方面, 由于 13.56MHz 的 RFID 技术发展较早, 相关标准也较为成熟, 主要的国际标准有 ISO/IEC 14443 和 ISO/IEC 15693 两种, 国内 13.56MHz 的 RFID 标准也主要源自于这两个国际标准。13.56MHz 的 RFID 技术, 已在国内得到广泛的应用, 主要集中于身份识别、公共交通管理等领域。

- 各地公交卡、校园卡, 主要基于 ISO/IEC 14443-A 标准的 13.56MHz 电子标签。
- 中国第二代居民身份证, 基于 ISO/IEC 14443-B 标准的 13.56MHz 电子标签。
- 教育部学生购票优惠卡, 基于 ISO/IEC 15693 标准的 13.56MHz 电子标签。
- 国内许多城市(如上海), 使用基于 ISO/IEC 14443-A 标准的 13.56MHz 电子标签作为地铁、出租、公交的电子车票。

(4) 超高频RFID

超高频 RFID 系统的工作频率在世界范围内没有统一, 各个国家的频率段标准不一样。欧盟指定的是 865~868MHz, 美国是 902~928MHz, 日本是 952~954MHz。而我国的这些频段已经被公共通信、数字通信等占用, 现在还没有指定一个合适的频段给超高频使用。

超高频 RFID 系统的标签也包括有源和无源两种, 标签及读卡器的成本较高。标签只要在读写器天线辐射场的远场区内就可以工作, 标签和读卡器之间的耦合方式为电磁耦合方式。超高频系统的特点是标签存储的数据量大、识别距离远(几米到几十米)、适应高速运动的物体、传送数据速度快、可靠性和寿命高、耐受户外恶劣环境。这些特点使它受到世界各国的重视和青睐。超高频的应用已逐步转到物流中的集装箱识别、交通管理中的铁路车辆自动识别、电子商务以及军事管理等领域。但是超高频有其弱点, 其能耗较大、穿透力较弱, 容易受无线电频率的干扰, 成本较高。

在我国, 超高频技术的应用还在起步阶段, 技术水平还不成熟。同时, 由于超高频的成本较高, 因此超高频在市场的占有率还较低。然而, 随着技术的进步, 标签成本的下降, 超高频将成为市场的主流。

(5) 有源 RFID

有源 RFID 标签使用标签内电池的能量, 因此在识别距离和安全性上做得更好。但它的成本高, 且寿命有限, 而且由于带有电池, 在体积上较大。

根据标签的数据调制方式的不同, 可以将标签分为主动式、被动式和半主动式。有源 RFID 标签一般是主动式, 无源的一般是被动式。主动式标签使用自身的射频能量主动向读



写器发送数据。被动式必须利用读写器的载波得到能量发送数据，常用在门禁系统和交通应用中。主动式由于射频距离远，适合应用在有障碍物的应用中。

在实际的应用中，有源标签可以通过对电池设计不同的寿命来控制标签的读写次数。这在需要控制标签的使用次数的应用中非常有用。有源标签的主要缺点是电池的寿命有限，特别是随着电池能量的消耗，标签射频能量减弱，射频的距离会减少，就会影响系统的正常工作。有源 RFID 具有识别距离长、传输数据量大、安全性好、低发射功率等特点，因此被广泛地应用在港口货运管理、交通管理中。

目前，有源 RFID 的标准比较混乱，没有比较统一的标准，从 433MHz~2.4GHz 都有人应用。不过，目前也逐渐开始规范，在 433MHz 工作频率上 ISO 18000-7 逐渐成为事实上的标准，组成 ISO 18000-7 的多项知识产权都属于美国塞维公司（Savi Technology）。

3.3.2 实验十 高频RFID读/写实验

1. 实验目的

本实验介绍如何使用高频手持机对符合 ISO 14443 Type A 标准的 13.56MHz 的高频 RFID 卡片进行信息写入和信息读出的操作，并通过一个电子钱包实例的讲解使读者对高频 RFID 有一个初步的了解，熟悉和掌握高频 RFID 卡片的特点，为今后开发基于高频 RFID 的应用做准备。

2. 实验设备

(1) 硬件

- RFID 实验硬件（IOT-RFID）1 套。
- ISO 14443 A 标准的电子标签 1 个。

(2) 软件

- 高频手持机 RFID 读卡与电子钱包程序。

3. 实验原理

ISO 14443 Type A 标准的卡片类型较多，本实验以飞利浦 Mifare One S50 非接触式卡片为例，演示如何对该卡片进行查询、读卡、写卡等基本操作。在进行实验前，建议先阅读 ISO 14443 Type A 标准，对其原理和过程有大致地了解。

在进行写卡操作前，需要了解所要操作卡的卡结构，否则容易造成锁卡，导致部分扇区无法使用。飞利浦 Mifare One S5 卡片有如下结构：

1) 卡片共分 16 个扇区，扇区编号为 0~15，每个扇区分 4 块存储区，块号为 0~3。其中用户可操作区域为扇区 0 中的 1~3 块，扇区 1~15 中的 0~3 块。

2) 在卡中每块可以保存 16 字节数据，在扇区 0 中数据可以保存在 1、2 块中，共保存 2×16 字节数据，扇区 1~15 中数据可以保存在 0~2 中，共保存 3×16 字节数据。

3) 每个扇区中有 3 块用于保存该扇区的密钥信息。

在操作飞利浦 Mifare One S5 卡片时，有如下注意事项：

1) 在扇区 0 中 0 块为只读块保存卡片基本信息，不可进行写操作。

2) 每个扇区中有 3 块用于保存该扇区的密钥信息，如果更改内容请将更改后的扇区密钥记录，如果遗忘扇区密钥，会导致该扇区不能使用，无法读取扇区中的数据以及对该扇区进行写数据操作。



下面结合一个电子钱包的代码实例,说明如何实现卡片的查询和读写操作。电子钱包是一种常见的基于 RFID 的电子支付手段,通常用于校园、交通等小额支付场景。本实验实现一个基本的电子钱包的演示程序。该程序具有卡片查询、初始化、余额查询、充值、消费等基本功能。下面对这 5 个函数接口进行介绍。

卡片的查询是 RFID 卡片的一项基本操作,是指查询手持机读卡范围内有无卡片,有什么类型的卡片,具体函数如下:

```
HANDHELD_API int Isol443A_QueryCard(UINT8 bMode, UINT8 *pszData);
```

相关参数如下:

bMode 为查询卡的模式; x52 为 QUERY_ALL; x26 为 QUERY_IDLE;

pszData 为返回查询到的卡数据,第 1 个字节为状态, x00 表示有卡, x01 表示无卡; 第 2 个字节为卡类型, x01 Mifare Light; x04 Mifare Ultra Light; x08 Mifare Standard S50; x10 Mifare Plus S70; 第 3、4 字节为卡的 ATQ; 第 5~8 字节为查到的卡号。

该函数成功返回数据的长度,失败则返回 0。

电子钱包金额初始化函数以 A 密钥认证方式初始化块金额。该函数是将要初始化的金额数据进行转换,存储到指定块的某一字节中,取值的时候再转换回来。而写卡是存储指定块的指定字节,这是两者的区别。UpdateData 函数的功能是更新控件关联的变量。相关函数原型如下:

```
HANDHELD_API int Isol443A_InitValue_A(int nSector, int nBlock,
    UINT8* pszKey, int iKeyLen, UINT8* pszData, int iDataLen);
```

该函数的参数如下:

nSector 为扇区号;

nBlock 为块号;

pszKey 为扇区密钥;

iKeyLen 为扇区密钥长度;

pszData 为初始化钱包里的金额;

iDataLen 为金额数据的长度。

该函数成功返回 1,失败则返回 0。电子钱包充值、扣款以及余额查询的函数参数与初始化函数类似,其原型分别为:

```
HANDHELD_API int Isol443A_IncrmentValue_A(int nSector, int nBlock,
    UINT8* pszKey, int iKeyLen, UINT8* pszData, int iDataLen);
```

```
HANDHELD_API int Isol443A_DecrmentValue_A(int nSector, int nBlock,
    UINT8* pszKey, int iKeyLen, UINT8* pszData, int iDataLen);
```

```
HANDHELD_API int Isol443A_ReadValue_A(int nSector, int nBlock,
    UINT8* pszKey, int iKeyLen, UINT8* pszData);
```

下面结合手持机上后台程序的代码,简单介绍如何使用上述函数操作 RFID 卡片。



卡片查询是进行卡片其他操作的前提，主要是查询卡片是否存在，并获得卡片序列号（CSN）。主要查询过程如下：

```
//查询卡，并显示 CSN
void CRFID_ISO14443ADlg::OnBnClickedBtuCsn()
{
    UINT8 recBuff[300];
    int recLength;
    CString strRecv = L"";
    CString strOldRecv = L"";

    //取得文本框控件指针
    CEdit *pEdtRecvMsg = (CEdit*)GetDlgItem(IDC_EDIT_ID);
    //文本框清空
    pEdtRecvMsg->SetWindowTextW(strOldRecv);
    //查询卡
    recLength = Iso1443A_QueryCard(0x52, recBuff);
    //判断是否返回数据长度
    if (recLength>0)
    {
        //判断是否读取失败
        if(recBuff[4]!=0x04)
        {
            //格式化查询到的数据到字符串中
            for (int i=4; i<8; i++)
            {
                //格式化字符串
                strRecv.Format(L"%02X ", recBuff[i]);
                //连接字符串
                strOldRecv = strOldRecv + strRecv;
                m_rzt=L"Query card success";
                UpdateData(FALSE);
            }
            //显示 CSN
            pEdtRecvMsg->SetWindowTextW(strOldRecv);
        }
        else
        {
            UpdateData(TRUE);
            //若读取失败，则提示查询卡失败
            m_rzt=L"Query card failure";
            UpdateData(FALSE);
        }
    }
    else
    {
```



```
UpdateData(TRUE);
//若返回长度为0,则提示查询卡失败
m_rzt=L"Query card failure";
UpdateData(FALSE);
}
}
```

电子钱包初始化主要完成钱包金额的设定,即将变量 `m_money` 所保存的初始化金额进行格式转换并写入卡片,相关的代码如下:

```
//电子钱包初始化
void CCWriteMoney::OnBnClickedButInit()
{
    UpdateData(TRUE);
    m_money;
    if(m_money!=0)
    {
        int ret;
        UINT8 bLocalArr[4];
        //转换要初始化的金额
        for (int i = 0; i < 4; i++)
        {
            bLocalArr[3 - i] = ((m_money >> (8 * i)) & 0xFF);
        }
        //电子钱包初始化
        ret=Isol443A_InitValue_A(m_Sectorx,m_blockx,pszKeyw,6,bLocalArr,4);
    }
}
```

电子钱包充值增加钱包的金额,具体过程与初始化类似,代码如下:

```
//电子钱包充值
void CCWriteMoney::OnBnClickedButIncr()
{
    UpdateData(TRUE);
    m_money;
    if (m_money!=0)
    {
        CString strc=L"";
        int ret;
        UINT8 bLocalArr[4];
        //转换要写入的数据
        for (int i = 0; i < 4; i++)
        {
            bLocalArr[3 - i] = ((m_money >> (8 * i)) & 0xFF);
        }
    }
}
```



```
        //电子钱包充值
        ret=Iso1443A_IncrmentValue_A(m_Sectorx,m_blockx,pszKeyw,6,
bLocalArr,4);
    }
}
```

电子钱包的消费过程与充值过程相反，即减去钱包中相应的金额，相关代码如下：

```
//电子钱包消费
void CCWriteMoney::OnBnClickedButDecr()
{
    UpdateData(TRUE);
    m_money;
    if (m_money!=0)
    {
        CString strc=L"";
        int ret;

        UINT8 bLocalArr[4];
        //转换要消费的数额
        for (int i = 0; i < 4; i++)
        {
            bLocalArr[3 - i] =((m_money>> (8 * i)) & 0xFF);
        }
        //电子钱包扣值
        ret=Iso1443A_DecrmentValue_A(m_Sectorx,m_blockx, pszKeyw,6,
                                     bLocalArr, 4);
    }
}
```

余额查询是最常用的一个操作，即读取 RFID 卡片中的金额数目，相关的代码如下：

```
//电子钱包余额查询
void CCWriteMoney::OnBnClickedButRead()
{
    UINT8 recBuff[4];
    CString str=L"";
    char * buf = new char[4];
    int ret;

    //余额查询
    ret=Iso1443A_ReadValue_A(m_Sectorx,m_blockx,pszKeyw,6,recBuff);
    buf=(char *)recBuff;

    //转换余额数据
```



```

int iOutcome=ByteArrayToInt(buf);
str.Format(L"%d",iOutcome);
CEdit * p_edit=(CEdit*)GetDlgItem(IDC_EDIT_MO);
p_edit->SetWindowTextW(str);
}

```

4. 实验内容

熟悉 ISO 14443 Type A 标准，了解 RFID 相关基础知识和 RFID 卡片构造，以及存储和访问方式。

掌握手持机上高频手持机 RFID 读卡程序和电子钱包程序的操作，并能够使用手持机对 13.56MHz 卡片进行操作。

5. 实验步骤

1) 启动手持机上的 RFID 读卡与电子钱包程序。

2) 将卡放置读卡区（设备背板后面）。在“Key”文本框中输入 14443A 密码。单击“Sector”下拉列表框的下拉按钮，可以选择扇区号（0~15），共有 16 个扇区。

3) 选择完扇区后单击“Read”按钮可以进行 14443 A 读卡操作，读到的数据会显示在下面的列表框中，读到的卡数据以十六进制显示，Block 列显示块号，Data 列显示读取到块中的数据，每个扇区有 4 个块，这里只读（0~2）3 块。双击要进行电子钱包操作的块号（这里选择 Block3）就可以打开电子钱包操作程序界面——14443A_WRITE_MONEY 对话框，如图 3-20 所示。

4) 钱包初始化。在“Purse”文本框中输入要初始化的金额。单击“Initialize”按钮进行初始化操作，操作成功在下面的状态栏会提示初始化成功。

5) 余额查询。单击“Readval”按钮，查询成功后余额会显示在“Purse”文本框中。

6) 钱包充值。进入 14443A_WRITE_MONEY 对话框，先在“Purse”文本框中输入充值金额。单击“Increment”按钮，充值成功后在窗口下方的状态显示区会提示充值成功，充值完成后可以单击“Readval”按钮查询余额。

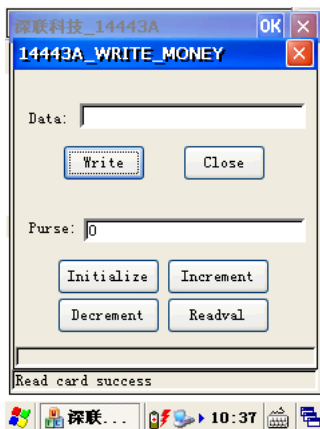


图 3-20 电子钱包操作程序界面

7) 钱包消费。在“Purse”文本框中输入要扣值的数额，单击“Decrement”按钮进行扣



值操作，扣值成功后在窗口下面的状态显示区会提示扣值成功，单击“Readval”按钮余额查询，可以看到钱包内容已经改变。

6. 思考题

- 1) 卡片为何要紧贴手持机?
- 2) 通过学习本实验，能否使用一张 RFID 卡片既作为饭卡，又作为公交卡?

7. 补充阅读

[1] ISO/IEC 14443-1:2008 Identification cards—Contactless integrated circuit cards—Proximity cards—Part 1: Physical characteristics.

[2] ISO/IEC 14443-2:2001 Identification cards—Contactless integrated circuit(s) cards—Proximity cards—Part 2: Radio frequency power and signal interface.

[3] ISO/IEC 14443-3:2001 Identification cards—Contactless integrated circuit(s) cards—Proximity cards—Part 3: Initialization and anticollision.

[4] ISO/IEC 14443-4:2008 Identification cards—Contactless integrated circuit cards—Proximity cards—Part 4: Transmission protocol.

3.3.3 实验十一 超高频RFID读/写实验

1. 实验目的

通过本实验了解 IOT-RFID-UHF 读写器（915MHz）的工作方式、后台配置程序、演示程序的操作方式，能够正确地操作读写器对标签进行读写。感受 UHF 频段的 RFID 技术远距离读写的能力，并能够使用 IOT-RFID-UHF 读写器进行相关工作和工程的实现。

2. 实验设备

(1) 硬件

- IOT-RFID-UHF 读写器 1 台。
- RS232 串口通信线（交叉线）1 条。
- +5V/1A 电源适配器 1 个。
- 天线 1 副。
- 915MHz 无源 RFID 卡片若干。

(2) 软件

- 工作参数配置程序 SetReaderMemo.exe。
- 读写器演示程序 ReaderMemo.exe。

3. 实验原理

本实验通过 PC 后台与 IOT-RFID 超高频读写器的串口通信，实现对读写器的工作方式和各参数的配置，并实现 UHF 读写器对于标签的读取、写内存、单标签读取、多标签读取、天线选择配置等功能定义。PC 后台和读写器的通信采用面向字节的异步通信协议数据格式。规定主机发给读写器的数据帧为命令，读写器返回给主机的数据帧为响应。命令或响应数据帧是变长字节数，采用组包方法并用 CRC 校验方法进行后向检错。

通信协议采用如图 3-21 所示的层次结构，包括物理层、

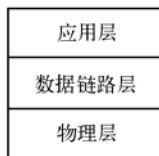


图 3-21 IOT-RFID 超高频读写器通信协议结构图



数据链路层和应用层。本协议对物理层和数据链路层进行了定义。

物理层完成信号的比特数据发送与接收，IOT-RFID 读写器在物理层有多种连接方式，包括 RS-232、RS-485、USB、韦根接口和网络接口。

其中 RS-232 接口具体设计要求如下：

- 1 位起始位、8 位数据位、1 位停止位、无奇偶校验。
- 通信波特率设计为 9600bit/s、19200bit/s、38400bit/s、57600bit/s、115200bit/s 可选。读写器上电或复位后初始波特率为 9600bit/s，可由主机发送命令改变读写器通信波特率。当主机与读写器传输发生错误时，读写器波特率恢复为 9600bit/s。

数据链路层具体规定命令帧和响应帧的类型与数据格式。帧类型分为命令帧、响应帧。

下面详细介绍数据链路层通信协议格式和部分设置命令。

(1) 命令帧格式

命令帧是主机操作读写器的数据帧，格式如表 3-1 所示。

表 3-1 IOT-RFID 超高频读写器命令帧格式

Start	Length	Reader ID	Flags	Command	Parameter	CRC
1 字节	2 字节	1 字节	1 字节	1 字节	<i>n</i> 字节	2 字节

	强制域，表示此域必须出现在命令帧中
	可选域，取决于命令码

其中：

Start 为帧头域，固定为 0xE0；

Length 是包长域，表示 Length 域后帧中字节数，长度 2 字节；

Reader ID 是站地址域，在总线网络中，表明读写器的唯一身份，0xFF 代表任意站，0x00 代表广播地址，0x01~0xFE 代表可独立寻址的站；

Flags 是数据区的标志域，0x00 表示 Parameter 域不存在，0x01 表示有 Parameter 域；

Command 是命令码域，一个字节；

Parameter 是命令帧中的参数域，该域为可选域，取决于命令码；

CRC 是 CRC-16 循环冗余校验域，校验范围是从帧头域到参数域最后一个字节为止，采用 CRC_CCITT 多项式 $(x^{16} + x^{12} + x^5 + 1)$ ，初始值为 0xFFFF。

(2) 响应帧格式

响应帧是读写器返回给主机的数据帧，响应帧包含了读写器需要采集的数据，其格式定义如表 3-2 所示。

表 3-2 IOT-RFID 超高频读写器响应帧格式

Start	Length	Reader ID	Flags	Response Code	Status	Response Data	CRC
1 字节	2 字节	1 字节	1 字节	1 字节	1 字节	<i>n</i> 字节	2 字节

	强制域，表示此域必须出现在响应帧中
	可选域，取决于命令码



其中:

- Start 为帧头域, 固定为 0x0E;
- Length 是包长域, 表示 Length 域后帧中字节数, 长度 2 字节;
- Reader ID 是站地址域, 返回读写器的 ID;
- Flags 是数据区的标志域, 0x00 返回帧中不存在数据域, 0x01 表示存在;
- Response Code 是响应码域, 取值为所响应的命令帧的命令码;
- Status 表示命令的执行结果, 0 表示正确执行, 其他值表示有错误发生;
- Response Data 是响应帧中的参数域;
- CRC 是 CRC-16 循环冗余校验域, 校验数据与命令帧相同。

(3) 读写器常用命令

1) Set Baud Rate: 该命令设置读写器 RS232 的通信波特率, 命令的格式如表 3-3 所示。

表 3-3 设置读写器 RS232 通信波特率命令格式

Length	Reader ID	Flags	Command Code	Command Data	CRC-16
6	0xFF	0x01	0x14	New Baud Rate	Crc

New Baud Rate 为需要设置的波特率参数。具体参数含义为: 0x00, 9600bit/s; 0x01, 19200bit/s; 0x02, 38400bit/s; 0x03, 57600bit/s; 0x04, 115200bit/s。

读写器接收到此命令帧后, 以原来波特率返回命令完成帧后修改读写器参数以新波特率进行通信。后台接收到相应操作时调用该函数来对读写器进行处理设置。PC 后台设置读写器 RS232 通信波特率的相关代码如下:

```
//波特率设置按钮单击事件
procedure TForm1.Set_BaudRate_BtnClick(Sender: TObject);
var strTime:string;
    ReaderID:byte;
    TickTime:DWord;
    ErrCode,i,Res:integer;
begin
    if (length(trim(RzEdit10.Text))<>2) then //检查 ReaderID 的长度是否合法
    begin
        RzMemol.Lines.Add('ReadID 错误。请输入有效的 ReadID'); //错误警告提示
        exit;
    end;

    ReaderID:=strtoint('$'+trim(RzEdit10.Text)); //ReaderID 赋值
    TickTime:=GetTickCount; //获取系统计数时间
    Res:=SetBaudRate(ReaderID,RzComboBox11.ItemIndex,ErrCode); //调用串口配置函数
    TickTime:=GetTickCount-TickTime; //获取系统计数时间
    i:=trunc(TickTime/1000); //计算串口配置函数执行时间
    strTime:='耗时 '+inttostr(i)+'秒 '+inttostr(TickTime-1000*i)+'毫秒';

    if Res<>ErrCodeFault then //检查串口函数反馈是否正确
    begin
```



```

RzMemo1.Lines.Add('设置波特率成功。'+strTime);    //打印成功消息和系统执行耗时
Sleep(500);                                       //休眠
PortStartBtn.Click;                              //产生串口打开单击事件
RzComboBox2.ItemIndex:=RzComboBox11.ItemIndex; //复制文本框内的波特率
PortStartBtn.Click;                              //再次产生串口打开单击事件
end
else
begin
    RzMemo1.Lines.Add('设置波特率失败。错误码为: '+ErrCodeToString (ErrCode)+'。
        '+strTime);                               //串口函数反馈错误, 打印错误码
end;
end;
end;

```

2) Reset Reader: 该命令对读写器进行复位, 命令的格式如表 3-4 所示。

表 3-4 读写器复位命令格式

Length	Reader ID	Flags	Command Code	CRC-16
5	0xFF	0x00	0x15	Crc

读写器收到此命令帧后, 先返回命令完成帧, 然后对读写器进行复位。在实验中, 后台接收到相应按钮操作后, 会调用该函数对读写器进行命令操作, 发送相应的命令帧, 以实现读写器的复位, 相关的 PC 后台代码如下:

```

//复位 Reader 按钮单击事件
procedure TForm1.Reset_Reader_BtnClick(Sender: TObject);
var strTime:string;
    ReaderID:byte;
    TickTime:DWord;
    ErrCode,i,Res:integer;
begin
    if (length(trim(RzEdit10.Text))<>2) then //检查 ReaderID 的长度是否合法

begin
    RzMemo1.Lines.Add('ReadID 错误。请输入有效的 ReadID'); //错误警告提示
    exit;
end;

ReaderID:=strtoint('$'+trim(RzEdit10.Text)); //ReaderID 赋值
TickTime:=GetTickCount; //获取系统计数时间
Res:=ResetReader(ReaderID,ErrCode); //调用 Reader 复位函数
TickTime:=GetTickCount-TickTime; //获取系统计数时间
i:=trunc(TickTime/1000); //计算串口配置函数执行时间

strTime:='耗时 '+inttostr(i)+'秒 '+inttostr(TickTime-1000*i)+'毫秒';

if Res<>ErrCodeFault then //检查 Reader 复位反馈是否正确

```



```
begin
    RzMemo1.Lines.Add('复位成功。'+strTime); //打印成功消息和系统执行耗时
end
else
begin
    RzMemo1.Lines.Add('复位失败。错误码为: '+ErrCodeToString(ErrCode)+' 。
        '+strTime); //Reader 复位函数反馈错误, 打印错误码
end;
end;
```

3) Set Reader ID: 该命令设置读写器 ID, 命令格式如表 3-5 所示。

表 3-5 设置读写器 ID 命令格式

Length	Reader ID	Flags	Command Code	Command Data	CRC-16
6	0xFF	1 byte	0x22	New Reader ID	Crc

New Reader ID 为修改后的读写器 ID, 读写器接收到此命令帧后, 修改读写器 ID 值, 并以新读写器 ID 返回如表 3-6 所示格式的命令完成帧。

表 3-6 读写器 ID 设置完成命令格式

Length	Reader ID	Flags	Response Code	Status	CRC-16
6	New Reader ID	1 byte	0x22	1 byte	Crc

PC 后台程序设置读写器 ID 的相关代码如下:

```
//设置 ReaderID 按钮单击事件
procedure TForm1.Set_ReadID_BtnClick(Sender: TObject);
var strTime:string;
    ReaderID,nReaderID:byte;
    TickTime:DWord;
    ErrCode,i,Res:integer;
begin
    if (length(trim(RzEdit10.Text))<>2) then //检查 ReaderID 的长度是否合法
    begin
        RzMemo1.Lines.Add('ReadID 错误。请输入有效的 ReadID'); //错误警告提示
        exit;
    end;
    if (length(trim(RzEdit42.Text))<>2) then//检查新输入 ReaderID 的长度是否合法
    begin
        RzMemo1.Lines.Add('新 ReadID 错误。请输入有效的新 ReadID');
        exit;
    end;

    ReaderID:=strtoint('$'+trim(RzEdit10.Text)); //旧 ReaderID 赋值
    nReaderID:=strtoint('$'+trim(RzEdit42.Text)); //新 ReaderID 赋值
```



```
TickTime:=GetTickCount; //获取系统计数时间
Res:=SetReaderID(ReaderID,nReaderID,ErrCode); //调用 ReaderID 设置函数
TickTime:=GetTickCount-TickTime; //获取系统计数时间
i:=trunc(TickTime/1000); //计算串口配置函数执行时间
strTime:='耗时 '+inttostr(i)+'秒 '+inttostr(TickTime-1000*i)+'毫秒';

if Res<>ErrCodeFault then //ReaderID 设置函数反馈正确
begin
RzMemo1.Lines.Add('设置读写器 ID 成功。'+strTime); //显示设置成功
RzEdit15.Text:=RzEdit42.Text; //文本框赋值
RzEdit4.Text:=RzEdit15.Text;
RzEdit6.Text:=RzEdit15.Text;
RzEdit18.Text:=RzEdit15.Text;
RzEdit10.Text:=RzEdit15.Text;
end
else
begin
RzMemo1.Lines.Add('设置读写器 ID 失败。错误码'+ErrCodeToString(ErrCode)+'。
'+strTime); //ReaderID 设置函数反馈错误，打印错误码

end;
end;
```

4. 实验内容

了解 ISO 18000-6C 相关标准，并熟悉 IOT-RFID-UHF 读写器的正确操作方法。通过对 PC 后台程序的阅读，对超高频 RFID 有一个直观的认识和了解。

5. 实验步骤

1) 把 RS-232 交叉串口线母头一端可靠连接在 PC 的 COM1 端，另一端公头可靠连接在读写器的串口端。

2) 选择将要工作的天线号 ANT1，将天线的 MAC 端连接到读写器上对应的天线接口 ANT1 上。

3) 将 5V 的电源适配器输入端连接到交流电源的插座上，直流输出端插入读写器电源插孔，给读写器供电，可以观察到读写器上红色 LED 灯被点亮，系统即完成初始化过程，进入待机状态，同时读写器会自动检测天线接口上的天线，安装有天线的 ANT1 对应的 LED 将被点亮。如果未被点亮，请检查电源和读写器。（在读写器上电之前必须确保已连接上了天线）。接通读写器后，读写器进入待机状态，考虑到用户需要，读写器在出厂时已经配置了基本参数，以满足基本的操作要求。如果不需要对参数进行设置，请直接进入步骤 5) 进行相关操作。当用户需要对读写器的工作参数进行个性化配置时，请进入下面的操作步骤。

4) 在 PC 上打开工作参数配置程序 SetReaderMemo.exe，选择串口 COM1，与连接 PC 的串口端匹配；波特率可以根据要求在下拉菜单中选择设置，ReaderID 为“FF”，单击“连接”按钮，观察状态信息栏的输出信息，如果读写器与 PC 连接成功，信息栏中会显示相关



信息；如果信息框出现“串口打开失败”等信息，表明读写器与 PC 连接失败，请检查读写器和串口线。

单击“参数设置”按钮，可以查询和设置读写器相关查询，参数配置分为工作方式设置、读写器参数设置、协议参数设置和功率设置 4 个模块。在工作方式模块中，可设置读写器的工作方式，分别可选主从工作方式、定时工作方式或触发工作方式；同时可设置通信方式，分别可选 RS-232、RS-485 等方式；可配置蜂鸣器的工作方式和掉电波特率保存等设置，在选择完相关信息后，单击右边的“设置参数”按钮，可将相关设置配置到读写器中，单击“查询设置”按钮可查询读写器现有的配置参数。

单击“默认参数”可将读写器参数恢复到出厂默认状态。如果设置成功，信息栏中将显示相应信息。在读写器参数模块中可设置读写器串口波特率、当前工作天线和工作天线、天线检测、读写器工作频段，跳频频点、跳频时间间隔、读写器相关 ID 和 IP 号等，在协议参数模块中可设置支持协议及相关参数，主要为协议标准选择，功率设置可分别配置 1~4 号天线的工作功率，并可查询现有的天线工作功率。

单击“EPC C1G2 标签读写命令”，可以对 C1G2 标签进行设置和读取地址的命令设置，此设置主要针对不同的用户区进行设置，通常情况下无需使用，请慎重设置。设置完所有的读写器参数后。

单击“更新数据”，将所有参数更新到读写器中，同时读写器会自动和 SetReaderMemo.exe 断开连接。

在 PC 上打开读写器演示程序 ReaderMemo.exe，选择串口 COM1，与连接 PC 的串口端匹配；波特率可以根据要求在下拉菜单中选择设置，ReaderID “FF”，单击“连接”按钮，观察状态信息栏的输出信息，如果读写器与 PC 连接成功，信息栏中会显示相关信息；如果信息框出现“串口打开失败”等信息，表明读写器与 PC 连接失败，请检查读写器和串口线。

在读写器演示程序中，分为标签 ID 识别命令、ISO 18000 标签读写命令、EPC C1G2 标签读写命令、读写器参数设置命令 4 个模块。

5) 将对应的 UHF 标签放于天线的正前方，角度尽量和天线表面平行，然后单击“标签 ID 识别命令”按钮，可选择识别命令、识别的标签类型，读取次数和读取的时间间隔，同时也可以选择连续读取功能，在选择完后，单击“开始”按钮读写器开始工作，读取相应标签，信息栏中将显示读取到的标签 ID 号和次数，此时对应的 COM 口和 RF 口 LED 灯将会连续闪亮，等读取到相关标签时，READ 的 LED 灯闪亮，同时蜂鸣器鸣叫（蜂鸣器鸣叫已被选择）。单击“连续开始”按钮，信息栏中将显示读取命令次数、读取成功数和对应的标签 ID 号，再次单击“连续开始”按钮，连续读取结束，按钮右边的显示框中将自动显示出指令次数、成功次数和成功率统计。

在 ISO 18000 标签读写命令、EPC C1G2 标签读写命令中，可分别对 ISO18000-6B 和 6C 的标签进行相关的用户区数据读写，以及 LOCK 操作。在相关操作后信息栏中将显示对应的信息。

在读写器参数设置命令中，可以对读写器的部分参数进行设置，如波特率、当前工作天线、跳频方式、工作天线的功率等，在选择完相关参数后，需要单击其右方的“设置”按钮使参数设置生效，其中的循环标志是指对所选择的天线进行循环工作读取。



6. 思考题

- 1) 实验中用到了天线 ANT1, 能否通过实验的学习, 手动改为其他天线接口?
- 2) 本套系统能否增加通信距离, 手段都有哪些?
- 3) 如何实现多标签读取?

7. 补充阅读

[1] ISO/IEC 18000-6:2004 Information technology—Radio frequency identification for item management—Part 6: Parameters for air interface communications at 860 MHz to 960 MHz.

第4章 物联网通信与网络实验

通信和网络让海量的感知信息网络化，并为全面智能化提供了可能。本章集中介绍与通信和网络相关的实验，包括短距离通信实验、无线自组网、IEEE 802.15.4 及 ZigBee 网络等实验。考虑到互联网和移动通信网的底层协议内容较多都有专门的课程讲述，并且难以在实验室环境下进行实验，本章并未涉及骨干网络相关的通信协议。相信通过本章对物联网前端这些基础通信实验的学习和实践，骨干网的相关通信实验原理也可以触类旁通。

4.1 通信与网络实验概述

互联网和移动通信网（包括移动互联网）是物联网应用和发展的核心基础设施，而传感器网络、RFID 等短距离网络又是其重要组成部分，这些网络的融合是物联网发展的关键内容之一。物联网的通信和网络技术的目标是构建物联网发展所需的开放、分层、可扩展的网络体系结构，解决物联网中 IP 地址、网络标识、通信、存储及计算等基础资源管理与服务技术。未来的网络是属于异构异种网络融合网络，已经不再严格区别究竟是互联网还是通信网，它是基于多种网络大融合的，具备虚拟化、自治计算、高可信、移动泛在、高效节能和高可扩展的物联网网络和接入平台。

4.2 短距离无线通信实验

本节以短距离无线通信网络实验为例，介绍无线通信方面的实验原理与方法，主要考虑到实验环境简单、易重现、成本低等优点，非常适合在实验室中完成而不失一般性。实际上，无线传感器网络出现后，短距离通信技术获得较多重视，目前也有很多备选方案可供选择，包括 IEEE 802.15.4/ZigBee、蓝牙、超宽带、低功耗 WiFi、低功耗蓝牙（Bluetooth Low Energy）等。下面对这几种技术进行简单介绍。

4.2.1 短距离无线通信规范简介

随着集成电路和无线通信技术的飞速发展以及无线传感器网络发展的推动，近几年在短距离通信领域的发展日新月异。目前在短距离无线通信方面的规范较多，从物理层和媒体控制层来看，包括 IEEE 802.15.4、蓝牙、超宽带、红外、低功耗 Wifi 等，在网络层也有 ZigBee、IEEE 1451、6LowPan 等。下面选取部分内容进行简单介绍。

(1) IEEE 802.15.4

IEEE 802.15.4 标准主要针对低速无线个域网（Low-Rate Wireless Personal Area Network, LR-WPAN）而制定。该标准把低能量消耗、低速率传输、低成本作为重点目标



(这和无线传感器网络一致),旨在为个人或者家庭范围内不同设备之间的低速互连提供统一接口。由于 IEEE 802.15.4 定义的 LR-WPAN 网络的特性和无线传感器网络的簇内通信有众多相似之处,很多研究机构把它作为传感器网络节点的物理及链路层通信标准。

IEEE 802.15.4b 标准拥有多个变种,包括了低速超宽带的 IEEE 802.15.4a、最近中国正在着力推进的 IEEE 802.15.4c 和 IEEE 802.15.4e,以及日本主要推动的 IEEE 802.15.4d,这里不深入讨论。

(2) 蓝牙

1998年5月,就在 IEEE 802.15 无线个域网工作组成立不久,5家世界著名的 IT 公司——爱立信、IBM、英特尔、诺基亚和东芝联合宣布了一项叫做“蓝牙 (Bluetooth)”的研发计划。1999年7月蓝牙工作组推出了蓝牙协议 1.0 版,2001年更新为版本 1.1,即我们熟知的 IEEE 802.15.1 协议。该协议旨在设计通用的无线空中接口 (Radio Air Interface) 及其软件的国际标准,使通信和计算机进一步结合,让不同厂家生产的便携式设备具有在没有电缆的情况下实现近距离范围内互通的能力。计划一经公布,就得到了包括摩托罗拉、朗讯、康柏、西门子、3COM、TDK 以及微软等大公司在内的近 2000 家厂商的广泛支持和采纳。

蓝牙技术也是工作在 2.4GHz 的 ISM 频段,采用快速跳频和短包技术减少同频干扰,保证物理层传输的可靠性和安全性,具有一定的组网能力,支持 64kbit/s 的实时语音。蓝牙技术日益普及,市场上的相关产品也在不断增多,但随着超宽带技术、无线局域网及 ZigBee 技术的出现,特别是其安全性、价格、功耗等方面的问题日益显现,其竞争优势开始下降。2004年蓝牙工作组推出 2.0 版本,带宽提高 3 倍,且功耗降低一半,在一定程度上重建了产业界信心。由于蓝牙技术与 ZigBee 技术存在一定的共性,所以它们经常被应用于无线传感器网络中。

(3) 超宽带

超宽带 (Ultra Wide-Band, UWB) 技术起源于 20 世纪 50 年代末,是一项使用从几赫兹到几吉赫兹的宽带电波信号的技术,通过发射极短暂的脉冲,并接收和分析反射回来的信号,就可以得到检测对象的信息。UWB 因为使用了极高的带宽,故其功率谱密度非常平坦,表现为在任何频点的输出功率都非常小,甚至低于普通设备放射的噪声,故其具有很好的抗干扰性和安全性。超宽带技术最初主要作为军事技术在雷达探测和定位等应用领域中使用,美国 FCC (联邦通信委员会) 于 2002 年 2 月准许该技术进入民用领域。除了低功耗外,超宽带技术的传输速率轻易可达 100Mbit/s 以上,其第二代产品可望达到 500Mbit/s 以上,仅这一项指标就让其他众多技术望尘莫及。

由于其功耗低、带宽高、抗干扰能力强,超宽带技术无疑具有梦幻般的发展前景,但超宽带芯片产品却迟迟未曾面市,这无疑留给我们一个大大的遗憾。近年来开始出现相关产品的报道,不过这项底蕴极深的技术还需要整个产业界的共同推动。目前超宽带技术可谓初露锋芒,相信它属于大器晚成、老而弥坚的类型,在无线传感器网络应用中必会大有作为。

(4) 红外

红外技术是一种利用红外线进行点对点通信的技术,是由成立于 1993 年的非营利性组织红外线数据标准协会 (Infrared Data Association, IrDA) 负责推进的。该协会致力于建立无线传播连接的世界标准,目前拥有 130 个以上的正式企业会员。红外技术的传输速率已经从最初 FIR 的 4Mbit/s 上升为现在 VFIR 的 16Mbit/s,接收角度也由最初的 30° 扩展到 120°。



由于它仅用于点对点通信，且具有一定方向性，故数据传输所受的干扰较少。由于产品体积小、成本低、功耗低、不需要频率申请等优势，红外技术从诞生到现在一直被广泛应用，可谓无线个域网领域的一棵常青树。经过多年的发展，其硬件与配套的软件技术都已相当成熟，目前全世界有至少 5000 万台设备采用红外技术，并且仍然以年递增 50% 的速度在增长。遥控设备（电视机、空调、数字产品等）更是普遍采用红外技术。

但是红外是一种视距传输技术，核心部件红外线 LED 也不是十分耐用，更无法构建长时间运行的稳定网络，造成红外技术终究没能成为无线个域网的物理层标准技术，仅在极少数无线传感器网络应用中进行过尝试（如定位跟踪），并且是与其他无线技术配合使用的。

（5）ZigBee 协议规范

ZigBee 联盟成立于 2001 年 8 月，最初成员包括霍尼韦尔、Invensys、三菱、摩托罗拉和飞利浦公司等，目前拥有超过 200 多个会员。ZigBee 1.0 (Revision 7) 规格正式于 2004 年 12 月推出，2006 年 12 月，推出了 ZigBee 2006 (Revision 13)，即 1.1 版，2007 年又推出了 ZigBee 2007 Pro，2008 年春天又有一定的更新。ZigBee 技术具有功耗低、成本低、网络容量大、时延短、安全可靠、工作频段灵活等诸多优点，目前是被普遍看好的无线个域网解决方案，也被很多人视为无线传感器网络的事实标准。ZigBee 协议在无线传感器网络中应用较多，后文将结合实验进行介绍。

（6）IEEE 1451.5 标准

除了以上通用规范以外，在无线传感器网络的不同应用领域，也正在酝酿着特定行业的专用标准，如电力、水力、工业控制、消费电子、智能家居等。这里以工控领域为例简单讨论一下 IEEE 1451.x。当然，工业标准纷繁复杂，最近正在制定专门面向工业自动化应用的无线技术标准 ISA SP100，有很多中国工业及学术界的同仁努力参与了该标准的制定工作。

IEEE 1451 标准族是通过定义一套通用的通信接口，以使工业变送器（传感器+执行器）能够独立于通信网络，并与现有的微处理器系统、仪表仪器和现场总线网络相连，解决不同网络之间的兼容性问题，并最终能够实现变送器到网络的互换性与互操作性。IEEE 1451 标准族定义了变送器的软硬件接口，将传感器分成两层模块结构。第一层用来运行网络协议和应用硬件，称为网络适配器（Network Capable Application Processor, NCAP）；第二层为智能变送器接口模块（Smart Transducer Interface Module, STIM），其中包括变送器和电子数据表格 TEDS。IEEE 1451 工作组先后提出了 5 项标准提案（IEEE 1451.1~IEEE 1451.5），分别针对不同的工业应用现场需求，其中 IEEE 1451.5 为无线传感通信接口标准。

（7）6LowPan 草案

无线传感器网络从诞生开始就与下一代互联网相关联，6LowPAN（IP v6 over Low Power Wireless Personal Area Network）就是结合这两个领域的标准草案。该草案的目标是制定如何在 LowPAN（低功率个域网）上传输 IP v6 报文。当前，LowPAN 采用的开放协议主要指前面提到的 IEEE 802.15.4 介质访问控制层标准，在上层并没有一个真正开放的标准支持路由等功能。由于 IP v6 是下一代互联网标准，在技术上趋于成熟，并且在 LowPAN 上采用 IP v6 协议可以与 IP v6 网络实现无缝连接，因此互联网工程任务组（Internet Engineering Task Force, IETF）成立了专门的工作组制定如何在 802.15.4 协议上发送和接收 IP v6 报文等相关技术标准。6LowPAN 将无线传感器网络和下一代互联网的距离进一步拉近了。

本章后续的实验都是基于短距离通信展开的。实际上，由于在实验室里操作的要求，本



书的几乎所有实验（除公网接入实验）都是基于不同种类的短距离通信的，包括规范和规范的。通过这些实验的讲述，相信读者会对无线通信有一个较为清晰的认识。

4.2.2 实验十二 433MHz无线通信实验

1. 实验目的

本实验介绍了基于 ATmega128 和 CC1000 实现 433MHz 传感网节点的点到点无线通信方法。通过实验，使读者理解和掌握如何使用 ATmega128 控制 CC1000 进行点对点无线通信，并进一步了解如何实现自己的无线通信协议。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。
- IOT-NODE433 1 个。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- WinAVR20050214。
- AVRStudio4.18。
- 串口调试助手。

3. 实验原理

433MHz 无线通信实验是以在无线传感器网络中常用的 CC1000 芯片为基础完成的。该芯片具有良好的可配置性，非常适合于在实验环境下展示无线通信的原理。CC1000 是根据 Chipcon 公司的 SmartRF 技术，在 0.35 μ m CMOS 工艺下制造的一种理想的超高频单片收发通信芯片。它的工作频带在 315MHz、433MHz、868MHz 及 915MHz，但也很容易通过编程使其工作在 300~1000MHz 范围内。它具有低电压 (2.3~3.6V)，极低的功耗，可编程输出功率 (-20~10dBm)，高灵敏度 (一般为 -109dBm)，小尺寸 (TSSOP-28 封装)，集成了位同步器等特点。其 FSK 数据传输率可达 72.8kbit/s，具有 250Hz 步长可编程频率能力，适用于跳频协议；主要工作参数能通过串行总线接口编程改变，使用非常灵活。

当调制数据时，CC1000 能被设置成 3 种不同的数据形式，分别为同步 NRZ 模式、同步曼彻斯特码模式、异步传输 (UART) 模式。为了满足电池供电情况下严格的电源损耗要求，CC1000 提供了十分方便的电源管理方法。通过 MAIN 寄存器控制低电平模式，有单独的位控制接收部分、发射部分、频率合成以及晶振。这种独立控制可用来优化在某个应用中最低可能达到的电流损耗。CC1000 优良的性能使它应用于工业、科学及医疗等各个行业。

实验需要两个节点，433MHz 无线传感器网络节点 (IOT-NODE433) 作为发送节点，开发套件作为接收节点。开始时两个节点将 CC1000 的状态设置为接收状态，发送节点通过时钟中断，定期地将 CC1000 的状态设置为发送状态，同时发送一个预定义的数据包 (包括包头、包尾、有效载荷，有效载荷为 main.c 中定义的 buff[4])，发送完毕转换为接收状态。

通信的初始化以及参数设置在 CC1000.c 中实现，主要包括：

- 1) 通过 CC1000HPLInit 函数配置 ATmega128 单片机和 CC1000 交互的引脚方向和状态。
- 2) CC1000 的主要工作参数的配置。图 4-1 为 CC1000 与 ATmega128 的连接方式。其



中，单片机使用 3 个输出端与 CC1000 的 3 线串行接口（PDATA、PALE 和 PCLK）相连，配置 CC1000 的工作参数；使用 SPI 接口与 CC1000 进行数据发送和接收；另外，单片机能从引脚 CHP_OUT 监视频率的锁定状态，从引脚 RSSI 获取信号强度的输出。其中 CC1000 的主要工作参数配置过程为：当芯片供电时，首先进行复位（通过清零 MAIN 寄存器的 RESET_N 位）；然后对需要配置的寄存器进行编程。程序中各个寄存器的设置方法请参见 CC1000 的数据手册。本实验中写寄存器是通过函数 CC1000HPLWrite 实现的。其参数 `addr` 为要写入数据的寄存器的地址；`data` 为要写入寄存器的数据。该函数先通过 CC1000 的 PDATA 引脚写入地址和读/写位，在此期间，PALE 必须保持低电平，然后 PALE 置高，通过 PDATA 逐位地写入数据，PDATA 在 PCLK 下降沿有效。

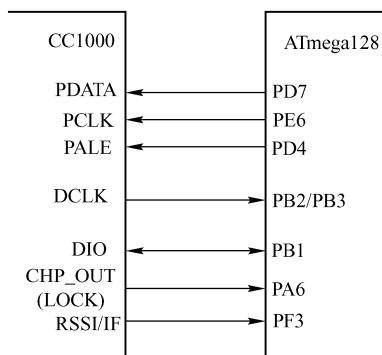


图 4-1 CC1000 与 ATmega128 的连接方式

函数实现代码如下：

```

result_t CC1000HPLWrite(uint8_t addr, uint8_t data)
{
    char cnt=0;
    addr<<=1;
    OSH_CLR_CC_PALE_PIN(); //设置 CC1000 的 PALE 引脚为低电平，发送地址
    for(cnt=0;cnt<7;cnt++) //设置发送次数，发送 7 位地址位
    {
        //如果地址位为高，则设置 CC1000 的 PDATA 引脚为高电平
        if(addr&0x80)
        {
            OSH_SET_CC_PDATA_PIN();
        }
        //如果地址位为低，则设置 CC1000 的 PDATA 引脚为低电平
        else
        {
            OSH_CLR_CC_PDATA_PIN();
        }
        //PCLK 已初始化为高电平，此时 PCLK 设置为低电平产生下降沿，PDATA 在 PCLK 下降
        //沿有效，数据写入对应的信号线
        OSH_CLR_CC_PCLK_PIN();
    }
}
  
```



```

    OSH_SET_CC_PCLK_PIN(); //设置 PCLK 为高电平, 为下次产生 PCLK 下降沿做准备
    addr<<=1;              //地址位左移 1 位, 为下一个地址位传递做准备
}

OSH_SET_CC_PDATA_PIN(); //设置 CC1000 的 PDATA 为高电平, 即为写模式
OSH_CLR_CC_PCLK_PIN();  //设置 PCLK 为低, 产生下降沿, 数据写入对应的信号线
OSH_SET_CC_PCLK_PIN();  //设置 PCLK 为高电平, 为下次产生 PCLK 下降沿做准备
OSH_SET_CC_PALE_PIN();  //设置 CC1000 的 PALE 引脚为高电平, 发送数据
for(cnt=0; cnt<8; cnt++) //按位发送 8 位数据位
{
    //如果数据位为高, 则设置 CC1000 的 PDATA 引脚为高电平
    if(data&0x80)
    {
        OSH_SET_CC_PDATA_PIN();
    }
    //如果地址位为低, 则设置 CC1000 的 PDATA 引脚为低电平
    else
    {
        OSH_CLR_CC_PDATA_PIN();
    }
    OSH_SET_CC_PCLK_PIN(); //设置 PCLK 为高电平, 为下次产生 PCLK 下降沿做准备
    data<<=1;              //数据位左移 1 位, 为下一个数据位传递做准备
}

//以下为恢复信号线状态为初始状态
OSH_SET_CC_PALE_PIN();
OSH_SET_CC_PDATA_PIN();
OSH_SET_CC_PCLK_PIN();
return SUCCESS;
}

```

3) CC1000 发送或接收模式的设置也是通过向 CC1000 的相关寄存器写入所要求的值而实现的, 如何设置请参见 CC1000 数据手册。

发送部分主要使用定时器来定时, 并调用 `physical.c` 中提供的接口进行数据的发送, 其主要流程如图 4-2 所示。

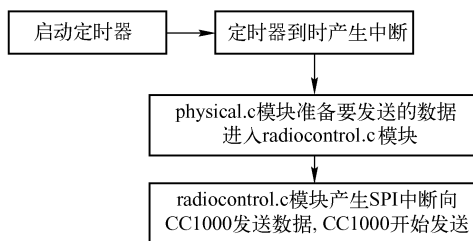


图 4-2 CC1000 数据发送的主要流程

当定时器溢出时, 在其回调函数中调用物理层的 `physical.c` 模块进行数据发送。回调函



数的代码如下:

```
void Timer0_Fired(void)
{
    PhysicalTxPkt(buff,4);           //调用低层函数进行数据发送
    LedRedToggle();                 //用红色 LED 表示数据发送
}
```

回调函数中调用的 `physical.c` 中的 `PhysicalTxPkt` 函数参数 `packet` 为要发送的数据包的地址, `length` 为物理层要发送的字节数。该函数使物理层变为传送状态, 调用 `radiocontrol.c` 的 `RadiocontrolStartTx()` 函数进行数据包发送。其实现代码如下:

```
result_t PhysicalTxPkt(void* packet, uint8_t length)
{
    state = TRANSMITTING;           //物理层状态变为发送状态
    RadiocontrolStartTx();          //调用 radiocontrol.c 模块发送函数
    sendPtr = (char*)packet;        //要发送包的首地址
    pktLength = length;             //要发送包的长度
    txCount = 0;                    //要发送的字节数
    return SUCCESS;
}
```

`RadiocontrolStartTx()` 函数首先关闭 SPI 中断, 再判断 CC1000 的状态, 若为睡眠则将其唤醒; 然后将待发送的数据 (包括导频字节和两个同步字节以及负载) 写入 SPI 的 SPDR 寄存器; 再打开 SPI 中断, 通过 SPI 中断处理程序 `__attribute((signal)) __vector_17()` 把 SPDR 寄存器的数据传入 CC1000; 最后 CC1000 在发送模式下将要发送的数据发送出去。该函数的代码如下:

```
result_t RadiocontrolStartTx()
{
    char temp;
    *(volatile unsigned char *) (0x0D+0x20)=0x00; //关闭 SPI 中断
    if(state==SLEEP) //如果 CC1000 为睡眠状态, 则将其唤醒
    {
        CC1000ControlStdControlStart(); //唤醒 CC1000
    }
    nextByte = 0xaa; //缓存要发送的导频字节
    txCount = 2; //导频字节数
    temp = *(volatile unsigned char *) (0x0E+0x20); //清空 SPSR 寄存器
    //将要发送的第 1 字节写入 SPDR
    *(volatile unsigned char *) (0x0F+0x20)=0xaa;
    CC1000ControlTxMode(); //进入发送模式
    OSH_MAKE_MISO_OUTPUT(); //将 SPI 的 MISO 引脚设置为输出模式
    OSH_MAKE_MOSI_OUTPUT(); //将 SPI 的 MOSI 引脚设置为输出模式
    *(volatile unsigned char *) (0x0D+0x20)=0xc0; //打开 SPI 中断
    state = TRANSMIT;
```



```

return SUCCESS;
}

```

CC1000 接收到数据之后，触发单片机的中断进行数据的接收。接收部分的主要流程如图 4-3 所示。

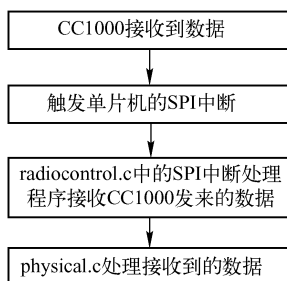


图 4-3 CC1000 接收部分的主要流程

CC1000 接收到数据就会产生由 CC1000 的时钟信号引发的 SPI 中断，使得单片机与 CC1000 同步，再根据 CC1000 状态进行相应的数据的发送和接收。SPI 中断处理函数 `__attribute__((signal)) __vector_17()` 在检测到起始符号且同步完成之后，使射频控制进入接收状态，通过调用 `PhysicalStartSymDetected` 函数判断如果物理层处于空闲状态，则使物理层也转为接收状态。接下来根据偏移量取出字节，并调用 `PhysicalRxByteDone` 函数通知物理层处理接收到的字节。

```

void __attribute__((signal)) __vector_17(void)
{
    uint8_t data;
    data=(volatile unsigned char *) (0x0F+0x20); //从 SPDR 读入字节
    if(bInvertRxData) data=~data; //判断数据是否需要倒置
    //如果处于空闲状态
    if(state == IDLE)
    {
        //判断 CHP_OUT 位是否为 0，不是 0 则为频率锁定状态
        bManchesterBad=(volatile unsigned char *) (0x1A+0x20)&0x40;
        if((!bManchesterBad)&&(data==0xaa | data==0x55))
        { //不处于频率锁定状态且接收的数据为 0xaa 或 0x55，则为接收到导频字节
            PreambleCount++; //导频字节数加 1
            if(PreambleCount>VALID_PRECURSOR)
            { //接收到足够的导频字节
                state=SYNC_START; //状态变为同步
            }
        }
    }
    else
    {
        PreambleCount=0; //导频计数置 0
    }
}

```



```
}
else if(state==SYNC_START)
{ //接收同步字
  uint8_t i;
  if(data==0xaa | data==0x55)
  {
    SOFCount=0; //再收到导频字节则丢弃
  }
  else
  { //进行同步
    uint8_t usTmp;
    SOFCount++; //同步偏移数加 1
    switch(SOFCount)
    {
      case 1: //同步偏移数为 1
        RxShiftBuf.MSB=data; //接收的数据给移位缓存高位字节
        break;
      case 2: //同步偏移数为 2
        RxShiftBuf.LSB=data; //接收的数据给移位缓存低位字节
        //移位缓存字为同步字进入接收态并通知上层检测到起始符号且同步完成
        if(RxShiftBuf.W==SYNC_WORD)
        {
          state=RECEIVE;
          RxBitOffset=0; //同步结果，即偏移的比特数
          //物理层不处于空闲状态，则 Radiocontrol.c 模块变为空闲状态
          if(PhysicalStartSymDetected()==FAIL)
          {
            RadiocontrolIdle();
          }
        }
        break;
      case 3: //同步偏移数为 3
        usTmp=data;
        for(i=0;i<8;i++)
        {
          RxShiftBuf.W<<=1; //移位缓存字左移 1 位
          if(usTmp&0x80) RxShiftBuf.W|=0x1;
          usTmp<<=1;
          if(RxShiftBuf.W==SYNC_WORD)
          {
            state = RECEIVE;
            RxBitOffset=7-i; //同步结果，即偏移的比特数
            RxShiftBuf.LSB=data;
            //物理层不处于空闲状态，则 Radiocontrol.c 模块变为空闲状态
            if(PhysicalStartSymDetected()==FAIL)
            {
```



```
        RadiocontrolIdle();
    }
    break;
}
}
break;
default:
RadiocontrolIdle();    //Radiocontrol.c 模块变为空闲态
break;
}
}
}
//如果处于接收状态
else if(state==RECEIVE)
{ //根据偏移量接收数据
    char Byte;
    RxShiftBuf.W<<=8;           //移去已经接收的字节
    RxShiftBuf.LSB=data;       //存入刚刚收到的字节到低8位
    Byte=(RxShiftBuf.W>>RxBitOffset); //根据偏移量取出字节
    PhysicalRxByteDone(Byte);   //通知上层处理接收到的字节
}
}
```

4. 实验内容

阅读 CC1000 数据手册，能够根据数据手册进行 CC1000 的配置并掌握 CC1000 数据收发程序的编写方法。

阅读 ATmega128 数据手册，学习 ATmega128 单片机 SPI 接口的使用方法，并能够使用 ATmega128 控制 CC1000 进行数据的收发。

5. 实验步骤

1) 启动 WinAVR，建立一个 base 节点工程，编写代码，建立 Makefile 文件，并编译得到接收节点的.hex 文件。

2) 建立一个发送节点工程，编写代码，建立 Makefile 文件，并编译得到发送节点的.hex 文件。

3) 将接收和发送节点的.hex 文件分别下载到开发板上和 IOT-NODE433 节点上。

4) 将接收节点通过串口线和 PC 主机连接起来，PC 主机运行串口调试助手（波特率设为 57600bit/s），打开接收节点和发送节点，这时可以看到发送节点红色 LED 闪烁表示正在进行数据发送，接收节点蓝色 LED 闪烁表示正在进行数据接收。串口调试助手显示收到的类似“8E 04 FF EE DD 7E”格式的数据包。其中，8E 为包头，7E 为包尾，04 FF EE DD 为 main.c 中定义的负载数据 uint8_t buff[4]。

6. 思考题

- 1) 如何实现 CRC-16 校验？
- 2) 简述使用 AVR 单片机 SPI 的步骤。



7. 补充阅读

[1] 徐勇军, 安竹林, 等. 无线传感器网络实验教程[M]. 北京: 北京理工大学出版社, 2007.

[2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel,2010.

[3] CC1000 Datasheet. <http://www.ti.com/>. Texas Instruments, 2009.

4.2.3 实验十三 2.4GHz无线通信实验

1. 实验目的

本实验介绍了使用 ATmega128 单片机控制 CC2420 实现的 2.4GHz 点到点无线通信技术。通过实验, 使读者理解和掌握如何使用 CC2420 进行点到点无线通信以及 ATmega128 单片机 SPI 口的使用。

2. 实验设备

(1) 硬件

- 单片机实验母板 (IOT-SCMMB) 1 套。
- IOT- NODE24 1 个。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- WinAVR20050214。
- AVRStudio4.18。
- 串口调试助手。

3. 实验原理

本实验需要两个节点, 2.4GHz 无线传感器网络节点 (IOT-NODE24) 作为发送节点, 单片机实验母板 (IOT-SCMMB) 作为接收节点。开始时两个节点将 CC2420 的状态设置为接收状态, 发送节点通过延时函数, 定期地将 CC2420 的状态设置为发送状态, 并立即发送一个预定义的数据包 (数据包的格式在 project.h 中定义)。CC2420 和 ATmega128 通过 SPI 总线连接, 其连接图如图 4-4 所示。

如图 4-4 所示, 单片机可以通过 4 线 SPI 总线 (CSn、SI、SO、SCLK) 设置芯片的工作模式, 并实现读/写缓存数据, 读/写状态寄存器等。CC2420 的 CCA、SFD、FIFO 和 FIFOP 这 4 个引脚向单片机返回收发数据的状态。FIFO 和 FIFOP 引脚标识接收 FIFO 缓存区的状态, 如果接收 FIFO 缓存区有数据, 则 FIFO 引脚为高, 相反则低; FIFOP 引脚在接收 FIFO 缓存区的数据超过某一个临界值时或 CC2420

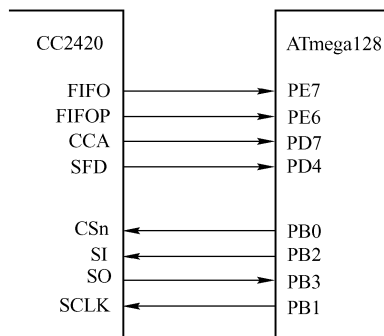


图 4-4 CC2420 和 ATmega128 的连接

收到一个完整的帧时输出高电平, 临界值可通过 CC2420 的寄存器设置。CCA 引脚在信道有信号时输出高电平, 其仅在接收状态下有效, 并且在 CC2420 进入接收状态至少 8 个符号周期后, 才会在 CCA 引脚上输出有效的信道状态信息。SFD 引脚状态的表示是否可以控制时



钟/定时信息的输入，CC2420 收到 SFD 字段后，会在 SFD 引脚输出高电平，直到接收完为止；如果启用了地址辨识，地址辨识失败，SFD 引脚会立即输出低电平。单片机将与该 4 个引脚相连的端口配置为输入。

要使用 CC2420 进行数据收发，首先需要对其进行配置和初始化。初始化过程包括 CC2420 初始化以及发送和接收模式的设置，下面分别进行介绍。

1) CC2420 初始化包括 ATmega128 单片机 SPI 的初始化、打开电压调节器、重新启动芯片、打开晶振、初始化寄存器值、设置寄存器值等。相关代码如下：

```
void cc2420_init(void)
{
    cc2420_pin_init();           //配置单片机和 CC2420 的引脚
    SPI_init();                 //SPI 初始化
    cc2420_VREFOn();           //打开电压调节器
    OS_CLR_CC_RSTN_PIN();      //关闭芯片
    OS_uwait(10);
    OS_SET_CC_RSTN_PIN();      //重新启动芯片
    OS_uwait(10);
    cc2420_oscon();            //打开晶振
    cc2420_reginit();          //初始化 CC2420 寄存器值
    cc2420_setreg();           //设置 CC2420 寄存器值
    cc2420_rxbuf_now = 0;
    cc2420_rxlength_now = 0;
    cc2420_rxflag = 0;
    cc2420_rxbufuse = 0;
    rfpackernum = 0;
}
```

其中：

SPI_init 为 SPI 初始化函数，具体过程请参考 ATmega128 数据手册；

cc2420_oscon 为 CC2420 晶振开启函数，其主要是通过 cc2420_sendcmd 函数访问 CC2420 的 SXOSCON 命令寄存器；

cc2420_reginit 为寄存器初始化值设置函数，其主要完成 CC2420 寄存器的预设参数（具体请参见 CC2420 数据手册），再通过 cc2420_setreg 函数调用 cc2420_writereg 函数把这些参数预设值写入对应的寄存器。

cc2420_writereg 的实现代码如下：

```
uint8_t cc2420_writereg(uint8_t addr, uint16_t data)
{
    uint8_t status;
    {
        os_atomic_t_nesc_atomic=os_atomic_start(); //原子操作开始函数
        {
            cc2420_bramspi = FALSE;
            //单片机的 SPI 必须将 CS 引脚设为低电平才能输出 SPDR 寄存器的值
        }
    }
}
```



```
OS_CLR_CC_CS_PIN();
//单片机要向 CC2420 写入数据的寄存器地址
SPDR = addr;
while(!(SPSR&0x80)) //判断 SPI 串行发送是否完成, 否则等待, 是则继续执行
;
status = SPDR;      //CC2420 返回给单片机的状态位
if(addr > 0x0E)     //该寄存器地址必须为非命令寄存器地址, 方可写入数据
{
    SPDR = data>>8; //传送数据高 8 位
    while(!(SPSR&0x80))//判断 SPI 串行发送是否完成, 否则等待, 是则继续执行
        ;
    SPDR = data&0xff; //传送数据低 8 位
    while(!(SPSR&0x80))//判断 SPI 串行发送是否完成, 否则等待, 是则继续执行
        ;
}
cc2420_bramspi = TRUE;
}
os_atomic_end(_nesc_atomic); //原子操作结束函数
}
OS_SET_CC_CS_PIN(); //数据传输结束, CS 设为低电平 (进一步确定是否为低电平)
return status;
}
```

该函数的参数 `addr` 表示单片机要向 CC2420 写入数据的寄存器的地址；`data` 表示要写入寄存器的数据。单片机通过 SPI 总线写 CC2420 寄存器，若 SPSR 中的最高位置位，则写 SPDR 寄存器会启动数据传输。需要注意的是，写 SPDR 寄存器时要保持 SPI 的 CS 信号线为低电平，先将 `addr` 写入 SPI 的 SPDR 寄存器输出给 CC2420，CC2420 收到地址的每一位都会同时向单片机返回 CC2420 的状态位。然后按照上述方式，再将 `data` 写入 SPDR 寄存器输出给 CC2420。

2) 设置发送和接收模式。发送模式和接收模式的设置是通过 `cc2420_sendcmd` 函数分别访问 CC2420 的 STXON 和 SRXON 这两个命令寄存器完成的。相关代码如下：

```
uint8_t cc2420_settxmode(void)
{
    cc2420_sendcmd(cc2420_STXON);
    return SUCCESS;
}

uint8_t cc2420_setrxmode(void)
{
    cc2420_sendcmd(cc2420_SRXON);
    return SUCCESS;
}
```

CC2420 初始化之后，即可以进行数据的发送了。在实验中，使用一个 `for` 循环进行发送，每发送完毕一个数据包之后，插入 255ms 的延时，之后调用 `cc2420_writetxfifo` 函数进



行无线发送。发送的相关代码如下：

```
for(uint16_t i=0; i<5000; i++)
{
    for(uint16_t j=0; j<255; j++) //延时 255ms
    {
        _delay_loop_2(2000);
    }
    packet_number++;
    printf("第%d个包", packet_number);
    //开始发送
    cc2420_writetxfifo(((demopacket->length)-1), (uint8_t *)demopacket);
}
}
```

无线发送的启动是通过调用 `cc2420_writetxfifo` 向 CC2420 的发送寄存器 TXFIFO 写数据实现的，`cc2420_writetxfifo` 函数向 CC2420 发送寄存器 TXFIFO 写入要发送数据包，该数据包的长度为 `(demopacket->length)-1`，数据包的存放首地址为 `demopacket` 的首地址，* `demopacket` 为指向结构体的指针变量，该结构体的内容为要发送的数据帧的内容（数据帧格式请参见 CC2420 数据手册）。其中，写 TXFIFO 的方法请参考 CC2420 初始化过程中的写寄存器方法。

CC2420 在接收到数据包之后触发单片机的 FIFOP 中断，单片机在该中断处理函数中读入 CC2420 的 RXFIFO 缓存区中的数据。FIFOP 中断处理程序如下：

```
uint8_t cc2420_fifopintr(void)
{
    if(!OS_READ_CC_FIFO_PIN()) //FIFO 引脚为低，则 RXFIFO 缓存溢出
    {
        cc2420_readreg(0x3F); //从 RXFIFO 缓存读数据
        rfpackernum=0;
        cc2420_sendcmd(0x08); //发送 SFLUSHRX 命令清空接收区
        cc2420_sendcmd(0x08);
        return SUCCESS;
    }
    rfpackernum++;
    if(cc2420_rxbufuse>1) //判断正确，则没有可用 RAM 空间
    {
        return SUCCESS;
    }
    else //判断不正确，则有可用的存储空间
    {
        cc2420_rxbufuse++;
        OS_post(&cc2420_readrxfifo); //开始读寄存器任务
    }
    return SUCCESS;
}
```



在 FIFOP 中断处理程序中，FIFOP 引脚在 RXFIFO 缓存区的数据超过某一个临界值（RXFIFO 缓存区在一次接收中，只可以保存 128 字节，只要数据小于 128 字节，它可以是多帧数据，如果大于该字节，则发生溢出 FIFOP 引脚为高，FIFO 引脚为低）时或 CC2420 收到一个完整的帧时，输出高电平。通过 FIFO 引脚可以获得该中断的类型，如果 FIFO 引脚为低，则 FIFOP 中断是由 RXFIFO 缓存区的数据超过临界值而引起的，此时已接收到的数据仍可以读出。继续接收数据，则必须使用 SFLUSHRX 命令清空接收区，并且 SFLUSHRX 需发送两次。以确保 SFD 无效，不再继续接收数据。如果 FIFO 引脚为高，则证明 RXFIFO 缓存区有数据。该 FIFOP 中断是由 CC2420 收到一个完整的帧引起的，此时要使用 cc2420_rxbufuse 变量判断是否有可用的 RAM 空间：如果有可用的存储空间，则启动读 RXFIFO 任务进行数据包的接收；如果没有可用的存储空间，则等待。读 RXFIFO 的相关代码如下：

```
void cc2420_readrxfifo()
{
    uint8_t status;
    uint8_t i;
    {
        os_atomic_t_nesc_atomic=os_atomic_start(); //原子操作开始函数
        {
            cc2420fifo_bspi=FALSE;
            //单片机的 SPI 要从 CC2420 寄存器输入数据到 SPDR 寄存器须将 CS 引脚设为低电平
            OS_CLR_CC_CS_PIN();
            //先送一个字节到 SPDR，其中该字节低 7 位为 RXFIFO 寄存器地址，
            //最高位为读写模式位，此时为读设为 0
            SPDR=0x3F|0x40;
            while(!(SPSR&0x80)) //判断 SPI 串行发送是否完成，否则等待，是则继续执行
                ;
            status=SPDR; //返回 CC2420 状态位
            SPDR=0; //清空 SPDR
            while(!(SPSR&0x80)) //判断 SPI 串行发送是否完成，否则等待，是则继续执行
                ;
            //首先读入单片机的是数据包的长度
            cc2420_rxlenght=SPDR;
            //根据数据包的长度分配存储空间
            cc2420_rxbuf[cc2420_rxflag]=malloc(cc2420_rxlenght+1);
            if(cc2420_rxlenght>0)
            {
                //数据包的长度存入缓存
                cc2420_rxbuf[cc2420_rxflag][0]=cc2420_rxlenght;
                cc2420_rxlenght++;
                for(i=1;i<cc2420_rxlenght;i++)
                {
                    SPDR=0; //清空 SPDR
                    //判断 SPI 串行发送是否完成，否则等待，是则执行下一步
                    while(!(SPSR&0x80))
```



```
        ;
        //读入 SPDR 的数据到缓存
        cc2420_rxbuf[cc2420_rxflag][i]=SPDR;
    }
}
//各变量恢复初始状态
cc2420fifo_bspi=TRUE;
rfpackernum--;
cc2420_rxflag++;
cc2420_rxflag&=0x01;
}
os_atomic_end(_nesc_atomic); //原子操作结束函数
}
OS_SET_CC_CS_PIN(); //数据传输结束，CS 设为低电平
OS_post(&demo_handlepacket); //开始数据包处理任务
}
```

单片机通过 SPI 总线接收 CC2420 的 RXFIFO 寄存器中的数据并保存到 SPDR 寄存器中。读 RXFIFO 寄存器时要保持 SPI 的 CS 信号线为低电平，先将 RXFIFO 寄存器的地址写入 SPI 的 SPDR 寄存器，若 SPSR 中的最高位置位，单片机就会将 SPDR 寄存器的值发送给 CC2420，CC2420 会返回给单片机状态位，接下来再逐字节的读入 RXFIFO 寄存器的值到 SPDR，并将 SPDR 寄存器的值存入所开辟的缓存区。通过 cc2420.c 中的 demo_handlepacket 函数将接收到的数据处理并通过串口发送到上位机上进行显示。

4. 实验内容

阅读 CC2420 数据手册，能够根据数据手册进行 CC2420 的配置并掌握 CC2420 初始化以及数据收发方法。

阅读 ATmega128 数据手册，能够根据数据手册并结合 CC2420 数据收发使用 AVR 单片机的 SPI 端口。

5. 实验步骤

1) 启动 WinAVR，建立一个接收节点工程，编写代码，建立 Makefile 文件，并编译得到接收节点的.hex 文件。

2) 建立一个发送节点工程，编写代码，建立 Makefile 文件，并编译得到发送节点的.hex 文件。

3) 将接收和发送节点的.hex 文件分别下载到开发板和 IOT-NODE24 节点上。

4) 将接收节点和计算机通过串口连接起来，打开接收节点和发送节点。发送节点上电之后 3 个 LED 同时点亮，当有数据发送时会同时闪烁红色和绿色 LED；接收节点上电后所有 LED 均熄灭，当收到数据时黄色 LED 闪烁。通过串口调试助手可观察到类似如下格式的数据包：

```
7E 42 21 02 03 01 01 E3 00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00 01 01 7E
```

该数据包内容按顺序如下：

7E 42 21 02 03 为包头，01 为数据包源地址，01 为规定串口输出，E3 为数据包序列号，00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00 为发送节点发送的有效载荷数



据, 01 01 为 CRC 校验值, 7E 为包尾。

6. 思考题

- 1) CC2420 无线发送原理与前文 CC1000 有何区别?
- 2) 在本实验中调用了 AVR-GCC 库中的延时函数, 如果使用定时器以每 640ms 发一个帧的速度来发送, 则需要如何修改程序? (提示: 使用定时来控制发送, 注意每次发送一个帧时 dsn 要加 1)

7. 补充阅读

[1] 徐勇军, 安竹林, 等. 无线传感器网络实验教程[M]. 北京: 北京理工大学出版社, 2007.

[2] ATmega128/L Datasheet. <http://www.atmel.com/>. Atmel, 2010.

[3] CC2420 Datasheet. <http://www.ti.com/>. Texas Instruments, 2010.

4.3 无线自组网实验

自组织网络在传感器网络和未来的骨干网络中, 都扮演着重要的角色, 这也是新型网络中很重要的特色技术。本节以短距离无线自组织网络为例, 介绍技术原理与实验方法。

4.3.1 无线自组网技术简介

无线自组网(无线自组织网络)是一种不同于传统无线通信网络的技术。传统的无线蜂窝通信网络, 需要固定的网络设备(如基站)的支持, 进行数据的转发和客户服务控制。而无线自组织网络不需要固定设备支持, 各节点(即用户终端)自行组网, 通信时, 由其他用户节点进行数据的转发。这种网络形式突破了传统无线蜂窝网络的地理局限性, 能够更加快速、便捷、高效地部署, 适合于一些紧急场合的通信需要, 如战场的单兵通信系统。但无线自组织网络也存在网络带宽受限、对实时性业务支持较差、安全性不高的弊端。自组织网络的路由方式有指定路由、动态路由或者两者的结合。

(1) 无线自组网的特点

无线自组网有以下几个特点:

1) 无中心网络。网络中所有的节点都是平等的, 严格来说没有中心节点这种说法, 网络中的任何节点都可以是中心, 也都可以不是中心。换句话说, 节点在网络中的功能可以根据网络的需求来改变, 任何节点可以是终端也可以是路由器。

2) 网络拓扑是动态的。网络中节点的随机移动, 节点的开机、关机, 环境的变化, 信道之间的干扰, 地形影响或者其他不可预知的情况都会导致网络的拓扑状态发生变化。

3) 自组织性。网络具有自发现、自动配置、自愈性。无线自组织网络具有快速发现其他节点的能力, 并且按照某种分布式的协议、算法或自组织原则, 协调彼此的行为。任何节点的故障都不会影响整个网络的运行, 具有很强的抗摧毁性和健壮性。

4) 节点能源和处理能力受限。在无线自组织网络中, 节点都被要求小巧、轻便、便于携带。节点的能源一般都选择电池。出于体积和成本方面的考虑, 处理器的 CPU 速度一般较慢、内存较低、处理能力较弱。

5) 多跳通信。由于网络中的节点大多受到的能源限制, 所以单个节点的通信距离有



限,和目的节点间的通信往往要经过其他中间节点转发,这种通信方式就叫做多跳。需要注意的是,转发功能是由网络中的普通节点完成的,而不是由传统意义上的网络中心完成的。

6) 传输带宽有限。由于无线通信本身的物理特性和节点处理器能力较弱,再加上为了维持网络本身的拓扑结构和应对各种可能出现的意外情况,节点已经浪费了很多带宽资源,所以能够利用的有效带宽十分有限。

7) 安全性较差。由于网络中采用无线信道、有限电源、分布式控制等技术,网络更加容易受到被动窃听、主动入侵、拒绝服务、剥夺“睡眠”等网络攻击。信道加密、抗干扰、用户认证和其他安全措施都需要特别考虑。

(2) 无线自组网研究的重点

无线自组网是无线传感器网络研究的一项核心内容,目前其研究的重点主要集中在以下几个方面。

1) MAC 协议的研究。一个无线自组织网络中,所有节点使用同一无线信道,由于各节点发送的随机性,为了减少碰撞,必须由 MAC 层协议来建立共享信道的访问机制。

2) 路由协议的研究。由于无线自组织网络的拓扑是动态变化的,能源是受限的,一个好的路由协议可以保证网络通信的可靠性和稳定性,提高网络的传输效率。节省网络的能源开销,平均各个节点的能源消耗,提高网络的使用寿命。

3) 网络安全。因为无线自组织网络的安全性较差,攻击者可以很容易地对网络进行侵犯和攻击,在目前的情况下还没有很好的方案来抵御这些手段,所以网络安全性的研究一直是无线自组织网络的研究热点。

4) 网络融合。因为无线自组织网络具有动态拓扑、多跳等功能,所以目前大多应用于一些特殊场合,如军事、野外勘测等,这就限制了自组织网络的应用和推广。目前,最好的方法就是把无线自组织网络和传统的网络结合起来,如 Internet、GPRS 等,作为传统网络的扩展和延伸。

4.3.2 实验十四 433MHz低功耗自组网实验

1. 实验目的

本实验介绍了使用 IOT-NODE433 的 MAC 库实现一个基于简单的泛洪协议的多跳路由网络的程序。通过实验操作,使读者了解无线自组织网络的特点及它与传统网络的区别,并对无线自组网有一个直观的印象,激发对于无线自组织网络的兴趣和爱好。同时,使读者初步掌握利用 MAC 库编程的方法和步骤,并能够通过 MAC 库开发高级通信协议。

2. 实验设备

(1) 硬件

- 433MHz 节点 (IOT-NODE433) 3 个。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- WINAVR4.07。
- AVRStudio。
- 多跳库文件 libmac433.a。



- 可视化软件 Snamp。

3. 实验原理

该协议的实现基于泛洪协议的思想，也就是产生数据的节点向全网广播该数据包，收到该数据包的节点也采用广播的方式向全网广播接收到的数据包，直到数据包到达发送节点，发送节点接收到该数据包后将其发送给后台 PC。

协议的开发加入了一些避免循环路由和广播风暴的机制。广播风暴是由广播引起的，以图 4-5 所示的网络为例，假设 A 为数据源，A 通过广播给 B 传输了一个数据包，那么 B 要转发该数据包，在广播方式下 A, C, D 都会收到 B 点转发的数据包，那么 A, C, D 又要转发该数据包，结果造成数据包在网络中无限的被广播，并最终导致网络瘫痪。可以采用数据包登记的机制来避免这种广播效应：每个节点接收到一个数据包后，先要检查自己的登记表，看是否收到过相同的数据包，如果有相同的就不转发，如果没有相同的就转发并且将该数据包登记。数据包的标示由数据包的源节点号以及数据包的包号组成，由于数据包中的 mac 头部分和网络层部分的参数在数据包传输的过程中是可变的，因此只能提取应用层部分的数据，选择 SensorMsg 中的 src 和 sensorseq 作为判断一个数据包的标识。

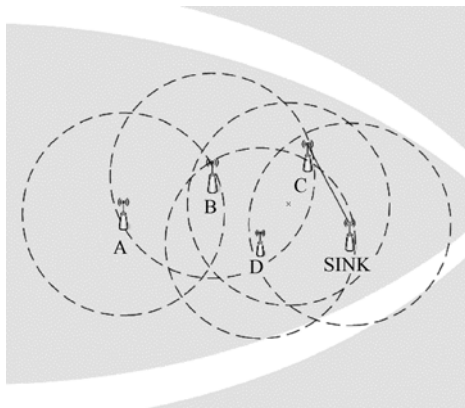


图 4-5 无线自组织网络示意图

```
typedef struct SensorMsg {
    uint8_t type; //传感器数据的类型
    uint16_t sensorseq; //数据包序列号
    uint16_t reading; //传感器数值
    uint8_t src; //数据包的源节点号
    uint8_t passnode[MAXPASS]; //数据包经过的节点号
} SensorMsg;
```

(1) IOT-NODE433 的 433MAC库介绍

IOT-NODE433 试验开发板最主要的功能是实现半双工的无线通信。为了方便读者的开发学习，提供了 MAC 层协议库，相关的函数以功能库的形式提供。MAC 层协议库能够提供的功能包括 ACK 机制、射频睡眠机制以及透明的传输数据包的能力，采用了 CSMA/CA 技术，使用二进制退避算法实现冲突避免，利用 CRC-16 算法进行检校。

(2) MAC数据包格式说明

下面是 message.h 中的部分代码，相关数据包的格式在该文件中定义。

```
#define MAX_PKT_LEN 100
#define MIN_PKT_LEN 7
#define MAC_HEADER_LEN 7
#define OS_DEFAULT_GROUP 1
#define MAC_CTRL_LEN 7
typedef struct {
```



```

uint8_t length;           //包总长度
uint8_t type;            //包类型
uint8_t toAddr;         //目的地址
uint8_t fromAddr;       //源地址
uint8_t group;          //群号
int8_t data[MAX_PKT_LEN - MAC_HEADER_LEN]; //MAC包数据部分
int16_t crc;            //CRC16
} OSMACMsg;
typedef OSMACMsg* OSMACMsgPtr; // MAC 数据包类型

```

length, type, toAddr, fromAddr, group 这 5 字节是数据包的 MAC 头。其中 length 是比较重要的部分，它表示包的长度。由#define MAC_HEADER_LEN 7 这段代码可以看出，在设计的时候 crc 的 2 字节是计算在 MAC 头中的，对于 length 的计算需要先说明一下 data 数组。data[MAX_PKT_LEN - MAC_HEADER_LEN]才是用户可以自由发挥的地方，在这里用户可以定义自己的数据包格式，MAC 库对数据包是透明的，数据包的解析和填充过程是在外部完成的，不过数据包要放在 data 数组中。比如下面定义的一个数据包格式：

```

#define BUFFER_LEN 10
typedef struct{
    uint8_t seqNo;
    int16_t data[BUFFER_LEN];
}SensorMsg;

```

下面是填充 data 数组的代码：

```

SensorMsg *pack;
pack = (SensorMsg *)LLCmsg[LLCcurrentMsg].data; //LLCmsg 是 OSMACMsg 类型
pack->seqNo = LLCpacketNumber;
pack->data[LLCpacketReadingNumber++] = data;

```

对于这个数据包来说，length 的计算应该是 sizeof(SensorMsg) + MAC_HEADER_LEN。这里有一点要强调的就是，length 不是一次就算出来的，只有在 MAC 层中才会加入 MAC_HEADER_LEN，这会在后面介绍 MAC 库的时候介绍。

type 字节表示数据包的类型，这都是在 MAC 库中定义的，分为下面几种，在库中用枚举类型进行了定义：enum { RTS_PKT, CTS_PKT, DATA_PKT, ACK_PKT }; toAddr 和 fromAddr 字段表示源地址和目的地址；group 字段表示组号，MAC 中设定为 0x01，一般来说 group 字段是用不到的，可以不管；crc 字段是用来做检校用的。

(3) MAC库接口函数介绍

IOT-NODE433 的 MAC 库给用户提供了丰富的接口，便于用户自己开发。MAC 库包含了 3 个模块，分别是 mac, physical 和 radiocontrol，对外接口主要由 mac 模块提供。radiocontrol 模块在初始化的时候会调用 CC1000 模块完成射频芯片的一些配置工作，在 TX 和 RX 切换的时候也会调用 CC1000 模块（为了方便，CC1000 模块也被封装在库中）。还有一点需要说明的是，radiocontrol 模块在进行载波监听的时候需要使用 ADC 转换部件。ADC 转换结束的时候会回调 radiocontrol 模块中的 RadiocontrolRSSIADCDataReady 函数，该函数



会根据 ADC 转换获得的数据来判断信道是否空闲。实现这个功能需要占用硬件上的 ADC0 通道。下面分两部分对相关的接口函数进行简单的介绍。

1) 入口函数。入口函数也就是可以供用户调用的功能接口，主要包含两个函数，分别是支持单播的函数 `MACUnicastMsg` 和支持广播的函数 `MACBroadcastMsg`。另外，还有 MAC 层的初始化函数 `MACInit`。本实验中主要使用广播函数 `MACBroadcastMsg`，其原型和实现如下：

`MACBroadcastMsg(void* data, uint8_t length)`

```
/******  
*功能描述：发送广播数据包的接口函数  
*参数说明：包地址，包长度(MAC包数据部分的长度)  
*返回值：  
*****/  
result_t MACBroadcastMsg(void* data, uint8_t length)  
{  
    char intEnabled;  
  
    //错误的发送方式  
    if (data == 0 || length == 0 || length > MAX_PKT_LEN - MAC_HEADER_LEN) {  
        return FAIL;  
    }  
    if(state != IDLE ) return FAIL;  
  
    //设置发送请求标识  
    intEnabled = inp(SREG) & 0x80;  
    cli();  
    if ((txRequest == 0) && (AckWaitTime == 0)) {  
        txRequest = 1;  
        if (intEnabled) sei();  
    }  
    else {  
        if (intEnabled) sei();  
        return FAIL;  
    }  
  
    dataPkt = (OSMACMsgPtr)data;  
    txPktLen = length + MAC_HEADER_LEN;  
  
    //设置数据包包头  
    dataPkt->type = DATA_PKT;  
    dataPkt->toAddr = OS_BCAST_ADDR;  
    dataPkt->fromAddr = OS_LOCAL_ADDRESS;  
    dataPkt->group = 0x01;  
  
    tryToSend();  
}
```



```
return SUCCESS;  
}
```

2) 回调函数。回调函数是 MAC 库对外的信息接口, 通过回调函数外部模块可以被动地获得 MAC 层的一些运行状态。需要注意的是, 回调函数需要用户来实现, 回调函数中最主要的几个函数是 `ReceiveDone`, `TransmitDone`, `SendFail`, `GetFreeQueueLength`。由于这几个函数需要用户来实现, 这里对其功能进行简单介绍。

```
result_t ReceiveDone(OSMACMsgPtr packet);
```

函数的参数是接收到的数据包指针。该函数是 MAC 层正确接收到一个完整的数据包后向上层提交数据包的一个响应, 能通过 MAC 层的数据包要么是广播数据包, 要么是指定发往本节点的数据包, 其他非发往本节点的单播数据包会被 MAC 层过滤掉, 上层看不见这些数据包。

```
result_t TransmitDone(OSMACMsgPtr msg);
```

函数的参数是发送的数据包指针。该函数是 MAC 层完成一个数据包的传输后通知上层的函数。对于广播和单播, 它们的意义有所不同: 在广播方式下, MAC 层只要将数据包发送出去了, 就表示数据发送完成了, 可以调用 `TransmitDone` 函数来通知上层数据发送完成; 在单播方式下, 如果采取不需要 ACK 的方式, 那么其工作的机理和广播一样, 但在有 ACK 方式下, MAC 层只有在收到 ACK 信息以后, 才会调用 `TransmitDone` 函数来通知上层数据发送完成。

```
void SendFail(OSMACMsgPtr receiveMsg);
```

该函数的参数是发送失败的数据包指针, 用于通知上层发送失败, 它只有在采用单播并且同时使用 ACK 机制的时候才有效, 在 ACK 等待超时的情况下 MAC 层会回调该函数, 通知上层数据包发送失败。

(4) 泛洪协议实现过程

协议的实现在代码 `app.c` 中。主要过程是, 实验中 3 号节点为数据源, 每 2s, 3 号节点产生一个数据包。该数据包通过广播的方式发送出去, 其他节点 (非发送节点) 接收到该数据包后, 也是通过广播的方式进行转发。下面分发送和接收两部分分别进行介绍和说明。

数据的发送过程主要负责采集光传感器数据, 并将此数据发送出去, 其主要过程为: 主程序在启动之后, 启动一个计数器。每次计数器溢出时, 调用采集光传感器数据的函数 `SensorExternalPhotoADCGetData`。采集到数据之后系统触发 `SensordataReady` 事件, 在其处理函数中, 调用 `SendSensorData` 函数。该函数又使用 `MACBroadcastMsg` 函数进行发送, 最后系统触发 `TransmitDone` 事件。各函数的调用关系如图 4-6 所示。

下面分别对图中的主要函数进行说明。

光传感器数据采集完毕事件响应函数 `SensordataReady`: 该函数判断节点号, 如果是数据源节点就把采集到的数据发送出去。

```
result_t SensordataReady(uint16_t data) {  
    //如果是数据源节点, 采集好数据以后, 执行发送数据任务
```

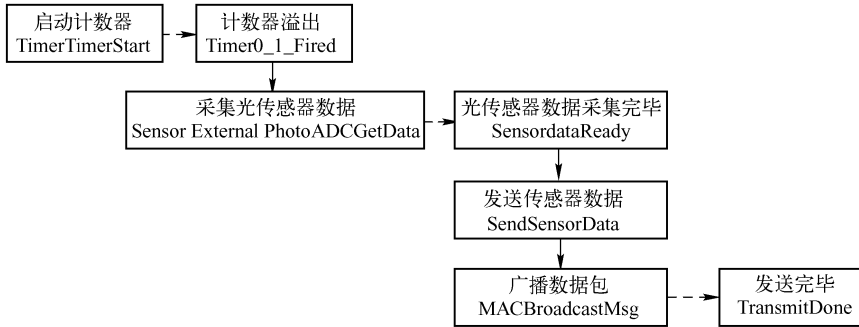


图 4-6 各函数调用关系

```

if( 3 == OS_LOCAL_ADDRESS ) {
    SendSensorData(data);
}
return SUCCESS;
}
  
```

数据采集和广播函数 **SendSensorData**: 在数据源节点中, 函数由 ADC 中断响应函数调用, 其功能是将由 ADC 采样得到的数据组合成数据包, 通过 **MACBroadcastMsg** 函数, 将数据包广播出去。

```

/*****
*功能描述: 将由 ADC 采样得到的数据采用广播的方式发送出去
*参数说明: ADC 采样数据
*返回值:
*****/
static void SendSensorData(uint16_t localsensordata) {
    //发送采集到的数据
    SensorMsg *sensordata;
    SHopMsgPtr sHopMsg;
    uint8_t i;
    uint8_t length;

    //设置数据包信息和数据
    sHopMsg = (SHopMsgPtr )routesensormsgptr->data;
    sHopMsg->type = SENSOR;
    sHopMsg->seq = networkseq;
    sHopMsg->lifetime = 0;

    sensordata = (SensorMsg *) (sHopMsg->data);
    sensordata->type = LIGHT;
    sensordata->sensorseq = sensorseq;
    sensordata->reading = localsensordata;
    sensordata->src = OS_LOCAL_ADDRESS;
  
```



```

//清空数据包中记录路径的数组
for (i = 0; i < MAXPASS; i++) {
    sensordata->passnode[i] = 0;
}
sensorseq++; //应用层数据包号自增
sensordata->passnode[0] = OS_LOCAL_ADDRESS; //记录当前节点号到路径数组中

length = SHOP_HEADER_LEN + sizeof(SensorMsg);

//数据包登记
InRecord(sensordata->src, sensordata->sensorseq);

MACBroadcastMsg(routesensormsgptr, length);
}

```

上述函数中调用了数据包登记函数 `InRecord`。该函数的功能是登记一个数据包到本地的数据包登记表，采用数据包登记机制的目的是避免广播风暴。其代码如下：

```

/*****
*功能描述：将新的数据包登记到本地的登记表中
*参数说明：数据包的源节点号，数据包的序号
*返回值：
*****/
static void InRecord(uint8_t nodeid, uint16_t sensorseq){
    record[recordflag].nodeid = nodeid;
    record[recordflag].sensorseq = sensorseq;
    recordflag = (recordflag + 1)%MAXPACKET;
}

```

数据发送完毕，`TransmitDone` 事件被触发，在其处理函数中将网络层数据包序列号递加，并改变 LED 灯的状态。

```

/*****
*功能描述：mac 层传输数据完成后的回调函数，通知上层数据包发送完成
*参数说明：被发送的数据包的指针
*返回值： 1
*****/
result_t TransmitDone(OSMACMsgPtr msg) {
    networkseq++;
    LedYellowToggle();
    return 1;
}

```

接收到数据包，系统触发 `ReceiveDone` 函数。该函数调用 `routehandlepkt` 对数据包类型进行处理。如果是传感器数据包，再调用 `routeReceiveSENSOR` 根据节点的类型进行转发或者发送到串口。主要函数调用关系如图 4-7 所示。

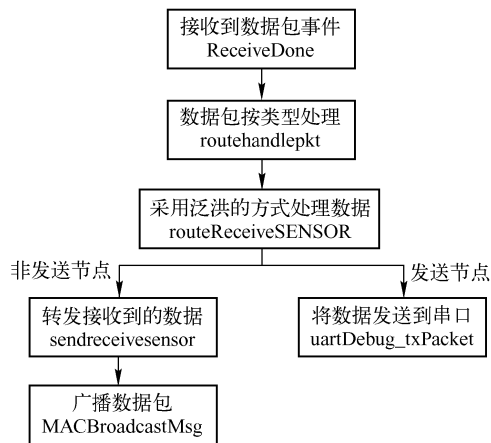


图 4-7 主要函数调用关系

下面对图中主要函数进行说明。

ReceiveDone 函数判断：如果是发送给本节点的数据包，则调用 **routehandlepkt** 进行处理。

```

/*****
*功能描述: mac 层接收到一个完整的数据包后的回调函数，通知上层接收到一个数据包
*参数说明: 被正确接收的数据包的指针
*返回值: 1
*****/
result_t ReceiveDone(OSMACMsgPtr packet) { //与底层接口函数
    uint8_t type;
    SHopMsgPtr sHopMsg;
    sHopMsg = (SHopMsgPtr )packet->data;

    type = sHopMsg->type;
    if( packet->group == 0x01)
    {
        routehandlepkt(type, packet);
    }

    return 1;
}

```

routehandlepkt 函数根据数据包网络层类型判断：如果是传感器数据包，再调用 **routeReceiveSENSOR** 根据节点的类型进行转发或者发送到串口。

```

/*****
*功能描述: 在网络层的概念上区分接收到的数据包，以区分不同的处理方式
*参数说明: 网络层上的数据包类型，被接收的数据包的指针
*返回值:
*****/
static void routehandlepkt(uint8_t type, OSMACMsgPtr receivemsg)

```



```

{
    switch (type) {
        case BROADCAST:
            break;

        case SENSOR:
            routeReceiveSENSOR(receivemsg);
            break;
    }
}

```

`routeReceiveSENSOR` 函数首先登记接收到的数据包，然后判断自己的节点类型对数据包进行处理。如果是非发送节点先要检查自己的登记表，看是否收到过相同的数据包，如果有相同的就不转发，如果没有相同的就转发并且将该数据包登记；如果是发送节点则把数据打印至串口显示，完成整个传输过程。下面是相关的代码：

```

/*****
*功能描述：非发送节点缓存接收到的数据，并调用发送任务
            发送节点将数据包发往串口
*参数说明：被发送的数据包的指针
*返回值：
*****/
static void routeReceiveSENSOR(OSMACMsgPtr receivemsg)
{
    SHopMsgPtr sHopMsg;
    SensorMsg *sensordata;
    uint8_t i;
    uint8_t j;
    sHopMsg = (SHopMsgPtr )receivemsg->data;
    sensordata = (SensorMsg *) (sHopMsg->data);
    //判断接收到的数据包是否在登记表中登记过
    if(CheckInRecord(sensordata->src,sensordata->sensorseq) == 1)
    {
        return ;
    }
    else //接收到的数据包不在登记表中，那么在登记表中将该数据包登记
    {
        InRecord(sensordata->src,sensordata->sensorseq);
    }

    uartMsg = *receivemsg;

    if (OS_LOCAL_ADDRESS != SINKNODE)
    {
        sHopMsg = (SHopMsgPtr )receivemsg->data;
        sensordata = (SensorMsg *) (sHopMsg->data);
    }
}

```



```
LifeTime = sHopMsg->lifetime;

//将收到数据包的信息保存到待发送的数据包中
forwardsensor.type = sensordata->type;
forwardsensor.sensorseq = sensordata->sensorseq;
forwardsensor.reading = sensordata->reading;
forwardsensor.src = sensordata->src;
for (i = 0; i < MAXPASS; i++)
{
    forwardsensor.passnode[i] = sensordata->passnode[i];
}
j = 0;

for (i = 0; i < MAXPASS; i++)
{
    if (sensordata->passnode[i] != 0)
    {
        j = j+1;
    }
    else
    {
        break;
    }
}
if( j < MAXPASS ) //越界检查
{
    forwardsensor.passnode[j] = OS_LOCAL_ADDRESS;
}
LedGreenToggle();
OSPostTask(sendreceivesensor);
}
else //如果是发送节点，则写到后台
{
    LedRedToggle(); //红灯亮，表示基站接收到传感数据包
    uartDebug_txPacket(&uartMsg);
}
}
```

接收数据转发函数 `sendreceivesensor` 调用 MAC 库提供的广播函数把收到的数据发送出去。用于数据的转发，使节点具有路由功能。

```
/******
*功能描述：发送缓存的数据包
*参数说明：
*返回值：
*****/
static void sendreceivesensor(void)
```



```
{
    SHopMsgPtr sHopMsg;
    SensorMsg *sensordata;
    uint8_t length;

    sHopMsg = (SHopMsgPtr )routesensormsgptr->data;
    sHopMsg->type = SENSOR;
    sHopMsg->seq = networkseq;
    sHopMsg->lifetime = LifeTime;
    sensordata = (SensorMsg *) (sHopMsg->data);
    *sensordata = forwardsensor;

    length = SHOP_HEADER_LEN + sizeof(SensorMsg);
    MACBroadcastMsg(routesensormsgptr, length);
}
```

4. 实验内容

阅读 MAC 库的介绍，学习 MAC 库的功能，了解主要接口函数。

掌握使用 MAC 库设计通信程序的方法，了解 MAC 库的功能架构，通过使用该库文件可以实现简单的路由协议。对 IOT-NODE433 节点进行实际操作，包括程序烧入，在线仿真等，了解 IOT-NODE433 节点的硬件结构。

在 PC 上通过 WinAVR 编译提供的实验代码，使用 AVRStudio 和仿真器把编译好的或者我们提供的现成的 .hex 文件烧入开发板，并在 PC 上通过可视化软件观察实验结果

5. 实验步骤

1) 启动 WinAVR，编写实验代码，并建立 Makefile 文件。建立 Makefile 文件时，在文件中添加对 MAC 库文件的引用，即添加如下代码：

```
LD_FLAGS += -LE:/433mac -libmac433
```

其中“E:/433mac”为存放 MAC 库文件的位置，这里假设库放在 E 盘的 433mac 目录下，库的名字为 libmac433.a。

2) 在 app.c 中的 Init 函数中修改 OS_LOCAL_ADDRESS 的值，给其赋值 1（即发送节点），2，3，分别编译后获得 3 个 .hex 文件，切记每次编译后生成的文件名均为 main.hex，所以编译后需要把生成的 main.hex 文件改名，以防下次编译把它覆盖，比如把 OS_LOCAL_ADDRESS 的值设为 1，编译后生成的 main.hex 文件，需要改成 1.hex，以此类推。

3) 将 .hex 文件下载到节点中。

4) 通过串口线连接 1 号节点（即烧入 1.hex 的节点）到 PC，打开所有节点。

5) 启动 SNAMP 可视化后台软件，通过 SNAMP 可以观察到 3 号节点的通信状况，如果节点之间位置摆放正确，就会看到 3 号节点每次都通过 2 号节点转发数据包到发送节点。

6) 注意节点之间的摆放位置，3 个节点摆放的很近，不容易出现多跳效果，为了最快出现自组织网络多跳效果，可以把 3 个节点摆放成一条直线，最好使 1 号和 2 号节点可以互相通信，2 号和 3 号可以互相通信，而 1 号和 3 号之间不能直接通信。



6. 思考题

- 1) 为什么需要自组织网络?
- 2) 为什么实验中节点的摆放位置会影响多跳效果?
- 3) 如何将实验中演示的 3 个节点组网, 改成 4 个或者更多个节点?

7. 补充阅读

- [1] 李晓维, 徐勇军, 等. 无线传感器网络技术[M]. 北京: 北京理工大学出版社, 2007.
- [2] 徐勇军, 安竹林, 等. 无线传感器网络实验教程[M]. 北京: 北京理工大学出版社,

2007.

4.4 IEEE 802.15.4 实验

IEEE 802.15.4 协议是无线传感器网络领域的常用标准协议之一, 也是到目前为止应用最多的传感器网络专用协议。本小节将对 IEEE 802.15.4 及其相关知识进行简单介绍, 然后基于常用的 TinyOS 安排专门实验展示协议内容, 供读者深入学习进行参考。

4.4.1 IEEE 802.15.4 协议以及TinyOS简介

1. IEEE 802.15.4 协议简介

IEEE 802 协议族是 IEEE 标准中关于局域网和城域网的一系列标准, 像常见的以太网、令牌环网、无线局域网协议都是 IEEE 802 协议族中的子标准, 其中的子标准 IEEE 802.15 主要关注无线个域网 (Personal Area Network, PAN)。无线个域网是指在个人周围空间形成的无线网络, 通常指覆盖范围在 10m 半径以内的短距离无线网络, 尤其指能在便携式消费者电器和通信设备之间进行短距离特别连接的自组织网。而 IEEE 802.15.4 则主要针对低速无线个域网, 规定了其物理层和介质访问控制层 (Media Access Control, MAC)。

IEEE 802.15.4 为无线个域网的设备间提供低速通信的基础底层协议, 其重点在于强调在没有辅助基础设施的情况下进行低成本和低功耗的通信。协议能够在 10m 的通信范围内提供 250kbit/s 速率的通信, 通过多物理层定义, 还可以支持如 20kbit/s、40kbit/s 等更低速率的通信, 当然功耗也会随之降低。另外, 协议还支持 CSMA/CA、安全通信、电源管理等特性。

IEEE 802.15.4 标准基于 OSI 模型, 但是只定义了物理层和介质访问控制层, 物理层提供原始的数据传输服务, 其直接管理射频收发器并进行信道选择、能量和信号管理, 主要特性包括激活和休眠射频发射器、信道能量检测、链路状态指示、信道选择、信道空闲评估以及在物理介质上收发数据。物理层提供 868MHz, 915MHz, 2.4GHz ISM 频段上的共 27 个信道。其中:

- 信道 0, 868~868.6MHz, 中心频率 868.3Hz。提供 20kbit/s 的数据通路。
- 信道 1~10, 中心频率=906MHz+2×(信道号-1)。每信道提供 40kbit/s 的数据通路。
- 信道 11~26, 中心频率=2405MHz+5×(信道号-11)。每信道提供 250kbit/s 的数据通路。

MAC 层通过使用物理信道进行 MAC 帧的传输, 除了数据服务, MAC 层还提供了 MAC 层管理服务, 即维护 MAC 子层协议的状态相关信息。其主要的工作包括: 信标管理以及基于信标的时间同步、支持 CSMA/CA 的信道访问、时槽保障管理、帧验证以及确认 (ACK) 帧传输、PAN 网络协调器的关联和取消关联。IEEE 802.15.4 的 MAC 层规定了 4 种



帧，分别是信标帧、数据帧、确认帧、命令帧。各帧的功能如下。

- 信标帧主要用来建立和维护一个 PAN，并进行时间同步。
- 数据帧用来承载上层发给 MAC 层的数据，数据传送至 MAC 子层时，被称为 MAC 服务数据单元，作为 MAC 层的负载，其首尾被分别附加了 MHR 头信息和 MFR 尾信息后，就构成了 MAC 帧。
- 确认帧用来使发送方确认数据正确到达。如果设备收到目的地址为其自身的数据帧或命令帧，并且帧的控制信息字段的确认请求位被置 1，设备需要回应一个确认帧。确认帧的序列号应该与被确认帧的序列号相同，并且负载长度应该为零。确认帧紧接着被确认帧发送，不需要使用 CSMA-CA 机制竞争信道。
- 命令帧用于组建 PAN 网络，传输同步数据等。其主要完成 3 方面的功能：把设备关联到 PAN 网络，与协调器交换数据，分配 GTS。

另外，标准通过定义逻辑链路控制（Logical Link Control）子层为上层提供服务。链路控制子层只是 IEEE 802.15.4 标准可能的上层协议，并不在 IEEE 802.15.4 标准的定义范围之内。为 IEEE 802.15.4 的 MAC 层接入 IEEE 802.2 标准中定义的 LLC 子层提供聚合服务。

IEEE 802.15.4 标准定义了两种类型的网络设备：全功能设备（Full Function Device，FFD）和精简功能设备（Reduced Function Device，RFD）。其中，FFD 具有全部的通信能力；RFD 具有部分的通信能力。FFD 设备之间以及 FFD 设备与 RFD 设备之间都可以直接通信，但 RFD 设备之间不能直接通信，只能与 FFD 设备通信，或者通过以一个 FFD 设备向外传输数据，这个与 RFD 相关联的 FFD 设备成为该 RFD 的协调器。IEEE 802.15.4 标准还定义了 3 种角色设备：协调器（Coordinator）、路由器（Router）以及端设备（End Device）。其中，协调器和路由器为 FFD，端设备为 FFD 或 RFD 均可。在 IEEE 802.15.4 网络中，有一个称为 PAN 协调器的 FFD 设备，是 LR-WPAN 网络中的主控制器。PAN 协调器除了直接参与应用外，还要完成成员身份管理、链路状态信息管理以及分组转发等任务。其他两种设备，路由器主要负责数据的转发，而端设备则负责数据的获取。

IEEE 802.15.4 支持点到点网络和星形网络两种拓扑结构，但是两种网络中均必须存在一个 FFD 作为协调器。在星形网络中，所有设备都与中心设备 PAN 协调器通信。星形网络适合家庭自动化、PC 的外设以及个人健康护理等小范围的室内应用。点对点网络也需要协调器，负责链路状态信息管理、设备身份认证等功能。点对点网络模式可以支持 AD HOC 网络，允许通过多跳路由的方式在网络中传输数据。

2. TinyOS 程序设计简介

TinyOS 是由美国加州大学伯利克分校专门为 WSN 开发的一种微型操作系统。TinyOS 虽然名字里有 OS，但并不是传统意义上的操作系统。确切地说，它是一个适用于网络化嵌入式系统的编程框架，通过在这个框架内链接一组必要的组件，就能方便地编译出面向特定应用的操作系统。这对于资源非常有限的嵌入式系统来说是非常重要的。针对 WSN 包含节点多，以及并发操作的工作方式，该操作系统采用了事件驱动的体系结构。由于 WSN 既有多样化的上层应用，又强调系统的低功耗要求，为此，系统采用既便于上层应用的开发，也有利于程序的快速执行的模块化设计方案。

TinyOS 使用 nesC 作为程序设计语言。nesC 在 C 语言的基础上进行了扩充。使用 nesC 编写的应用程序是通过把一个或多个组件连接起来从而组成一个完整的可执行程序。nesC



有两种类型的组件：模块（module）和配件（configuration）。组件的定义格式请见图 4-8。模块提供应用程序代码，实现一个或多个接口；配件则是用来将其他组件连接起来的组件。组件定义了两个作用域，一个是形式说明（specification）作用域，在该作用域内声明了组件提供和使用的接口；另一个是组件实现（implementation）作用域，该作用域对于模块来说是程序功能的代码实现部分；而对于配件来说是将接口的使用组件和提供组件连接起来组成一个程序。组件内部还可以定义任务。

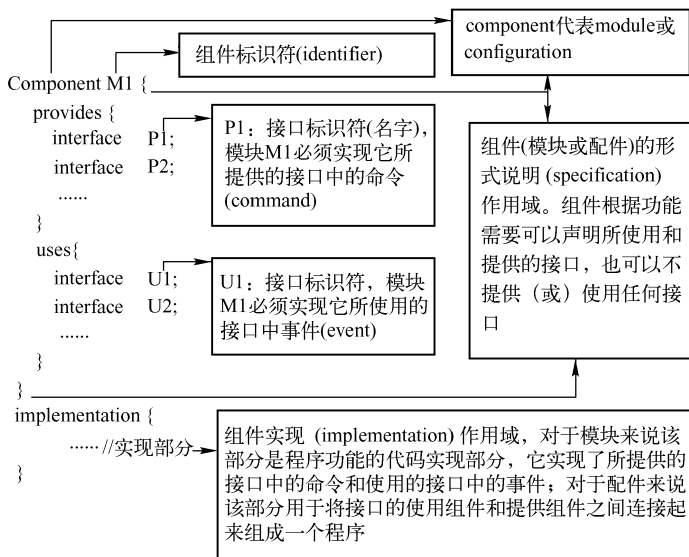


图 4-8 组件的定义格式

组件可以提供（provide）和使用（use）接口，接口是一组相关函数的集合，它是双向的并且是组件间的唯一访问点。接口声明了一组函数，称为命令（command），接口的提供者必须实现它们；接口还声明了另外一组函数，称为事件（event），接口的使用者必须实现它们。图 4-9 是 send 接口的定义。

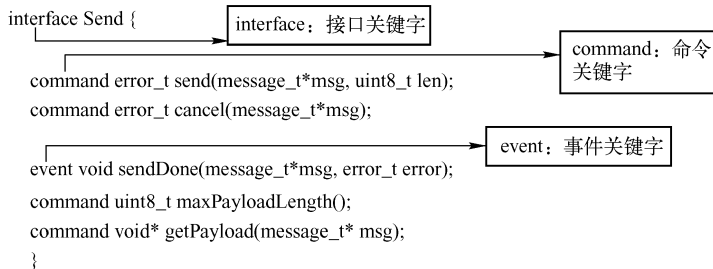


图 4-9 send 接口的定义

send 接口定义了 4 个命令：send, cancel, maxPayloadLength, getPayload。其中，前两个命令的返回值都是 error_t 类型的；而后两个命令的返回值分别是 uint8_t 和 void*。它还定义了一个事件 sendDone，该事件无返回值。

对于一个组件而言，如果它要使用某个接口中的命令，那么它还必须实现这个接口的事



件。一个组件可以使用或提供多个接口以及同一个接口的多个实例。

将组件所使用的接口与组件所提供的接口连接在一起，这种行为称为连接（Wiring）。每个 nesC 应用程序都由一个顶级配件所描述，其内容就是将该应用程序所用到的所有组件连接起来，形成一个有机整体。nesC 的所有源文件，包括接口、模块和配件，其文件扩展名都是“.nc”。

4.4.2 实验十五 IEEE 802.15.4 协议通信实验

1. 实验目的

本实验介绍了使用 TinyOS 实现一个基于 IEEE 802.15.4 协议的物联网节点的点到点通信程序。通过实验，使读者初步掌握 TinyOS 编程的方法和步骤，并能够使用 TinyOS 开发简单实用的物联网应用程序。

2. 实验设备

(1) 硬件

- 2.4GHz 节点 (IOT-NODE24) 2 个。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- TinyOS 2.x。
- AVRStudio4.18。
- 串口调试工具。

3. 实验原理

本实验的硬件平台是 2.4GHz 无线传感器网络节点硬件 (IOT-NODE24)，其核心芯片是 TI-Chipcon 公司推出的 CC2420。该芯片是首款符合 2.4GHz IEEE 802.15.4 标准的射频收发器，包括众多额外功能，是第一款适用于 ZigBee 产品的 RF 器件。它基于 Chipcon 公司的 SmartRF 03 技术，以 0.18 μm CMOS 工艺制成，只需极少外部元器件，性能稳定，成本低。CC2420 的选择性和敏感性指数超过了 IEEE 802.15.4 标准的要求，可确保短距离通信的有效性和可靠性。利用此芯片开发的无线通信设备支持的数据传输速率高达 250kbit/s，可以实现多点对多点的快速组网。其 MAC 层和物理层协议都符合 IEEE 802.15.4 规范，工作于免执照的 2.4GHz 频段。此芯片采用低电压 (2.1~3.6V) 供电，同时支持休眠模式，且从休眠模式被激活的时延短，因此可以分配更多的时间处于休眠状态，而处于休眠状态时芯片的能耗极低，从而可大大减少能耗，适于电池长期供电。

此外，CC2420 芯片还具有硬件加密、安全可靠、组网灵活、抗毁性强等特点，为家庭自动控制、工业监控、传感网络、消费电子、智能玩具等提供了理想的解决方案。此外，芯片已经集成了 CRC 和数据完整性检查等功能，这就相对减少了程序员编程的工作量，而且硬件处理速度一般都快于软件处理速度，因此加快了通信的速度，减少了能量消耗。同时，芯片还采用了 CSMA/CA 技术来避免数据发送时的竞争和冲突，减少了不必要的能量消耗。

在软件平台方面，采用了研究领域使用较多的 TinyOS 系统，它对 CC2420 也有着较好的支持。本实验基于 TinyOS 实现一个无线传感器网络节点之间的点到点通信程序。每个节点的程序功能包括：从节点上的光传感器获得传感器数据，然后建立数据包



并将其广播出去；同时，当节点收到数据包时，将数据包通过串口传到 PC 上。

为了使实验过程清晰，并且便于理解，实验内容分为 4 部分：程序框架的建立以及计数器的使用，IEEE 802.15.4 协议点对点通信，串口数据包的发送，光传感器数据的获取（各主要组件关系如图 4-10 所示）。下面对这 4 部分内容分别进行介绍。

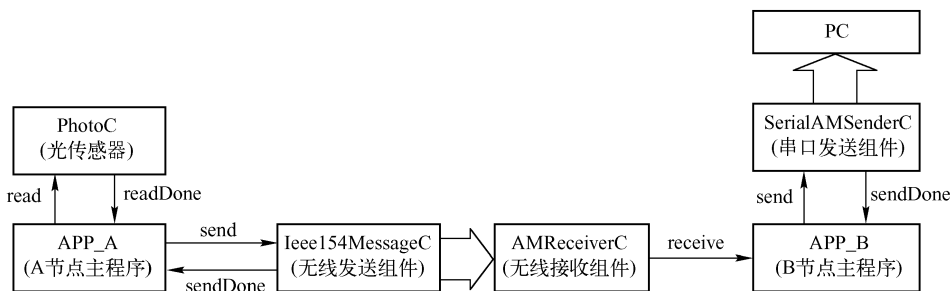


图 4-10 IEEE 802.15.4 协议点对点通信实验主要组件关系

(1) 程序框架的建立以及计数器的使用

正如上面所述的 TinyOS 的程序构成方式，一个最简单的 TinyOS 程序通常包括两个源程序，其分别对应于模块（module）和配置（configure）。对于本实验的程序，不妨命名为 SensingToRadioAppC.nc 和 SensingToRadioC.nc。SensingToRadioAppC.nc 为配置文件，主要描述了 SensingToRadioC.nc 中模块如何与系统模块进行连接；SensingToRadioC.nc 为模块文件，包括对系统模块的调用和对系统事件的响应。

计数器是 TinyOS 最重要的基本组件之一，其基本的函数和事件如下：

```

interface Timer<precision_tag>
{
    //basic interface
    command void startPeriodic( uint32_t dt );
    command void startOneShot( uint32_t dt );
    command void stop();
    event void fired();
}
    
```

其中，<precision_tag>表示计数器的计数精度，可选择<TMilli>，<T32khz>以及<TMicro>，它们分别表示计数器一秒钟计数 1024，32768 和 1048576 次。Timer 接口提供了计数器的两种启动方式 startPeriodic 和 startOneShot。使用 startPeriodic 启动计数器时，计数器到达溢出值之后，触发 fired 事件，并立即置零重新开始下一轮计数；而使用 startOneShot 启动计数器时，计数器只溢出一次。fired 事件为计数器溢出时触发的事件，用户可以在 fired 事件的相应函数中编写自己要执行的代码。

基于上面的介绍，分别建立如下两个文件。

SensingToRadioAppC.nc:

```

#include <Timer.h>
#include "SensingToRadio.h"
    
```



```
configuration SensingToRadioAppC {
}
implementation
{
    //基本控制组件
    components MainC, LedsC;
    components new TimerMilliC() as Timer;
    components SensingToRadioC as App;
    //基本控制组件连接
    App.Boot -> MainC;
    App.Leds -> LedsC;
    App.Timer -> Timer;
}
```

SensingToRadioC.nc:

```
#include <Timer.h>
#include "SensingToRadio.h"

module SensingToRadioC
{
    //基本控制接口
    uses interface Boot;
    uses interface Leds;
    uses interface Timer<TMilli> as Timer;
}
implementation {
    //系统启动后触发事件
    event void Boot.booted()
    {
        call Timer.startPeriodic(TIMER_PERIOD);
    }

    //计数器溢出触发事件
    event void Timer.fired()
    {
        call Leds.led0Toggle();
    }
}
```

SensingToRadioAppC.nc 文件包含了配置信息，主要是将程序的启动、LED、计数器与相应组件的接口进行连接。**SensingToRadioC.nc** 文件包含了对接口函数的调用和对事件的响应，主要是在系统启动之后启动计数器，然后在计数器溢出时改变 LED 的状态，从而实现定时闪烁的功能。另外，上述两个文件均包含了定时器周期的定义头文件 **Sensing To Radio.h**，其内容如下：



```
#ifndef SENSINGTORADIO_H
#define SENSINGTORADIO_H

enum
{
    TIMER_PERIOD = 1000
};

#endif
```

(2) IEEE 802.15.4 协议点对点通信

实现 IEEE 802.15.4 协议点对点通信主要用到 `Ieee154Send`，`Receive` 以及 `CC2420PacketBody` 3 个接口，分别负责数据的发送、接收和数据包解析。`Ieee154Send` 接口提供符合 IEEE 802.15.4 协议的发送函数。其主要函数和事件如下：

```
interface Ieee154Send
{
    command error_t send(ieee154_saddr_t addr,message_t* msg,uint8_t len);
    event void sendDone(message_t* msg, error_t error);
    command void* getPayload(message_t* msg, uint8_t len);
}
```

其中，`send` 发送一个数据包，3 个参数分别为地址、数据包指针和数据包长度；`getPayload` 提供从数据包里获得的负载指针，以使用户加入自己要发送的内容；`sendDone` 事件当发送完毕之后被触发。

`Receive` 接口提供如下数据包的接收函数。

```
interface Receive
{
    event message_t* receive(message_t* msg, void* payload, uint8_t len);
}
```

`receive` 函数提供接收到的数据包指针 `msg`、负载指针 `payload` 和数据包长度 `len`。

`CC2420PacketBody` 接口提供从 IEEE 802.15.4 协议数据包中获得的包头以及 `metadata` 的函数：

```
interface CC2420PacketBody
{
    async command cc2420_header_t * ONE getHeader(message_t * ONE msg);
    async command cc2420_metadata_t * ONE getMetadata(message_t * ONE msg);
}
```

下面介绍如何使用上述接口进行 IEEE 802.15.4 协议点对点通信。首先，定义无线数据报的格式，以及无线信道的标识。按如下内容修改 `SensingToRadio.h`：

```
#ifndef SENSINGTORADIO_H
#define SENSINGTORADIO_H
```



```
enum
{
    AM_SENSINGTORADIO = 6,
    TIMER_PERIOD = 1000
};

//无线数据包格式
typedef nx_struct SensingToRadioMsg
{
    nx_uint8_t nodeId;
    nx_uint16_t counter;
    nx_uint16_t sensorData;
} SensingToRadioMsg;

#endif
```

然后，要在 `SensingToRadioAppC.nc` 中将无线发送的相关接口连接到相应的组件上，添加如下代码：

```
//无线数据收发相关的组件
components ActiveMessageC;
components new AMReceiverC(AM_SENSINGTORADIO);
components Ieee154MessageC;
components CC2420PacketC;

//无线数据收发相关组件的连接
App.RadioControl -> ActiveMessageC;
App.Ieee154Send -> Ieee154MessageC;
App.Receive -> Ieee154MessageC;
App.CC2420PacketBody-> CC2420PacketC;
```

最后，在 `SensingToRadioC.nc` 中编写数据发送以及接收的响应代码。首先，要添加射频启动的控制，使用 `SplitControl` 方式启动：

```
//系统启动后触发事件
event void Boot.booted()
{
    call Timer.startPeriodic(TIMER_PERIOD); //启动计数器
    call RadioControl.start(); //启动无线数据收发的 Split Control
}

//无线数据收发启动后触发事件
event void RadioControl.startDone(error_t err)
{
    if (err == SUCCESS) {
    }
}
```



```
else
{
    call RadioControl.start();
}
}

//无线数据收发停止后触发事件
event void RadioControl.stopDone(error_t err) {
}
```

由于数据的发送为一个相对独立的过程，并且可以与主程序异步并发的执行，将数据的发送作为一个 task 来执行。每当计数器溢出时，post 数据发送的 task:

```
//计数器溢出触发事件
event void Timer.fired()
{
    post RadioSend();
}
```

RadioSend 的实现如下:

```
//无线数据发送任务
task void RadioSend()
{
    if (!radioBusy)
    {
        //获得数据负载指针
        SensingToRadioMsg* btrpkt = (SensingToRadioMsg*)
            (call Ieee154Send.getPayload(&pkt, sizeof(SensingToRadioMsg)));
        if (btrpkt == NULL)
        {
            return;
        }

        //写入数据负载内容
        btrpkt->nodeId = TOS_NODE_ID;
        btrpkt->counter = counter;

        //发送数据
        if (call Ieee154Send.send(IEEE154_BROADCAST_ADDR, &pkt,
            sizeof(SensingToRadioMsg)) == SUCCESS)
        {
            radioBusy = TRUE;
            call Leds.led0Toggle(); //发送时改变 LED 灯的状态
        }
    }
}
```



发送过程分为 3 步：首先，指向获得待发送数据包的指针；然后，通过这个指针将待发送的数据写入数据包中；最后，调用发送函数进行数据发送。其中，`radioBusy` 为一个布尔型控制变量，当其为 `TRUE` 时，表明正在执行发送的 `task`。在发送之前将其置为 `TRUE`，而在发送完毕将其改为 `FALSE`：

```
//发送完毕触发事件
event void Ieee154Send.sendDone(message_t* msg, error_t err)
{
    if (&pkt == msg)
    {
        radioBusy = FALSE;
    }
}
```

节点接收到数据包之后，触发 `receive` 事件，在事件响应函数中，将收到的数据包的包头和负载保存到本地的缓冲区 `ieee154Header` 和 `ieee154Payload` 中备用：

```
//接收到数据包触发事件
event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len)
{
    cc2420_header_t* hdr;
    if (len == sizeof(SensingToRadioMsg)) {
        SensingToRadioMsg* btrpkt = (SensingToRadioMsg*)payload;
        memcpy(ieee154Payload, btrpkt, sizeof(SensingToRadioMsg));

        hdr = call CC2420PacketBody.getHeader(msg);
        memcpy(ieee154Header, hdr, sizeof(cc2420_header_t));

        call Leds.led1Toggle();
    }
    return msg;
}
```

`ieee154Header` 和 `ieee154Payload` 的声明如下：

```
uint8_t ieee154Header[sizeof(cc2420_header_t)]; //802.15.4 包头本地缓冲区
uint8_t ieee154Payload[sizeof(SensingToRadioMsg)]; //802.15.4 负载本地缓冲区
```

(3) 串口数据包的发送

串口数据包的发送跟无线数据包的发送类似，由于没有接收，因此只用到接口 `AMSend`：

```
interface AMSend
{
    command error_t send(am_addr_t addr, message_t* msg, uint8_t len);
    event void sendDone(message_t* msg, error_t error);
    command void* getPayload(message_t* msg, uint8_t len);
}
```



各个参数与 Ieee154Send 接口类似，这里不再赘述。

要把收到的无线数据包传到串口，首先还是需要定义串口数据包格式，在 SensingToRadio.h 中添加：

```
//串口数据包格式
typedef nx_struct SerialPortMsg
{
    nx_uint16_t seqNo;
    nx_uint8_t nodeId;
    nx_uint8_t ieee154Header[sizeof(cc2420_header_t)];
    nx_uint8_t ieee154Payload[sizeof(SensingToRadioMsg)];
} SerialPortMsg;
```

然后，在 SensingToRadioAppC.nc 中添加串口数据发送的相应接口连接关系：

```
//串口数据收发相关的组件
components SerialActiveMessageC;
components new SerialAMSenderC(AM_SENSINGTORADIO);

//串口数据收发相关组件的连接
App.SerialAMSend -> SerialAMSenderC;
App.SerialControl -> SerialActiveMessageC;
```

串口的启动方式跟无线数据收发类似，也是需要使用 Splitcontrol，数据的发送略有不同：

```
//串口数据发送任务
task void SerialPortSend()
{
    if (!serialBusy)
    {
        SerialPortMsg* serialpkt = (SerialPortMsg*)
        (call SerialAMSend.getPayload(&pktSerial, sizeof(SerialPortMsg)));
        serialpkt->nodeId = TOS_NODE_ID;
        serialpkt->seqNo = serialSeq++;

        memcpy(serialpkt->ieee154Header, ieee154Header,
        sizeof(cc2420_header_t));

        memcpy(serialpkt->ieee154Payload, ieee154Payload,
        sizeof(SensingToRadioMsg));

        if (call SerialAMSend.send(AM_BROADCAST_ADDR, &pktSerial,
        sizeof(SerialPortMsg)) == SUCCESS)
        {
            serialBusy = TRUE;
            call Leds.led1On();
        }
    }
}
```



```
    }  
}  
  
//串口数据发送完毕触发事件  
event void SerialAMSend.sendDone(message_t* msg, error_t error)  
{  
    if (&pktSerial == msg)  
    {  
        serialBusy = FALSE;  
        call Leds.led1Off();  
    }  
}
```

上述串口发送的 task 在收到无线数据包的 receive 事件响应函数里进行 post，即在闪灯的代码后面添加：

```
post SerialPortSend();
```

(4) 光传感器数据的获取

本实验使用的 IOT-NODE24 节点上提供了光传感器组件 PhotoC。该组件提供了 Read 接口，通过该接口可以直接获得光传感器的传感值。接口定义如下：

```
interface Read<val_t>  
{  
    command error_t read();  
    event void readDone( error_t result, val_t val );  
}
```

Read 接口提供了一个方法和一个事件。通过调用 read 方法，可以发起一个读传感器数据的操作，当数据获取完毕之后，readDone 事件被触发，在事件响应函数里可以对获得的传感器数据进行处理。另外，参数<val_t>表示读取数据的类型，可以为<uint8_t>、<uint16_t>等。

要从传感器获得数据，首先需要连接提供 Read 的组件，在 SensingToRadioAppC.nc 中添加：

```
//光传感器数据获取组件  
components new PhotoC() as Sensor;  
  
//读取传感器数据相关组件的连接  
App.Read -> Sensor;
```

修改计数器溢出事件响应函数，改为读传感器数据。

```
//计数器溢出触发事件  
event void Timer.fired()  
{  
    call Read.read();  
}
```



然后在 readDone 事件处理函数里保存获得的传感器数据，并 post 无线数据发送过程：

```
//获得传感器数据后触发事件
event void Read.readDone(error_t result, uint16_t data)
{
    sensorData = data;
    post RadioSend();
}
```

4. 实验内容

阅读 IEEE 802.15.4 协议，学习协议的运行机制以及协议中各层的数据包格式。

初步掌握 TinyOS 程序设计方法，了解 nesC 的编程过程以及程序结构，能够编写简单的 TinyOS 程序

对 IOT- NODE24 节点编程、运行过程进行实际的操作，了解物联网前端感知节点的工作机理。

5. 实验步骤

1) 编写实验代码，启动 Cygwin 环境，在实验代码所在文件夹下使用“make micaz”进行编译。编译通过之后，在实验代码文件夹的“build\micaz”子文件夹下会生成 main.ihex 文件。

2) 将节点与 PC 相连，并使用 AvrStudio 将步骤 1) 编译得到的 ihex 分别烧写到两个节点中去。

3) 将其中一个节点与 PC 的串口相连。

4) 启动串口调试助手，设置串口号为与节点连接的串口号，波特率为 57600bit/s，校验位 NONE，数据位 8 位，停止位 1 位。另外，为了分析接收到的数据，还需设置十六进制显示以及自动换行。

5) 先打开未与 PC 连接的节点开关，此时节点红色灯闪烁，表明正在广播数据。再打开与 PC 连接的节点，绿色灯闪烁表明接收到数据。同时串口调试助手会显示收到的数据包。IEEE 802.15.4 协议点对点通信实验串口数据包格式如表 4-1 所示。

表 4-1 IEEE 802.15.4 协议点对点通信实验串口数据包格式

数据类别	数据	含义
串口帧头	7E	同步字
	45	串口数据包类型（无应答）
串口数据包包头	00	表示该数据包为 AM 数据包
	FF FF	串口目的地址
	00 00	串口链路源地址
	14	数据包长度
	00	组 ID
	06	AM 类型
串口负载	00 02	串口数据包序号（seqNo）
	01	节点号（nodeId）
IEEE 802.15.4 MAC 层数据包头 (串口负载的一部分)	10	IEEE 802.15.4 MAC 层数据包长度除去本位的长度 1
	41 88	IEEE 802.15.4 帧控制域



(续)

数据类别	数据	含义
IEEE 802.15.4 MAC 层数据包头 (串口负载的一部分)	02	数据序列号
	22 00	目的 PAN 的 ID
	FF FF	目的地址
	01 00	源地址
	01	TinyOS 网络 ID
	00	AM 类型
IEEE 802.15.4 MAC 层负载	01	节点号 (nodeId)
	00 02	数据包序号 (counter)
	03 C8	光传感器数据 (sensorData)
串口帧尾	6F D7	CRC 校验码
	7E	同步字

6) 用手遮住光传感器, 观察数据包中 IEEE 802.15.4 MAC 层负载的光传感器数据 (sensorData) 位变化情况。

6. 思考题

- 1) 如果想降低传感器数据的采样频率为每 5s 一次, 如何修改程序?
- 2) 目前实验中两个节点都是同构的, 即都进行数据的采集、发送和接收。如果想采用主从模式, 即一个主节点采集传感器数据并通过广播数据包, 一个从节点接收数据包并通过串口送到 PC, 如何修改程序?
- 3) 如果增加节点的数目, 与 PC 相连的节点收到的数据会有什么变化?

7. 补充阅读

[1] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE 802.15.4.

[2] Philip Levis, David Gay TinyOS Programming. Cambridge : Cambridge University Press, 2009.

4.5 ZigBee网络实验

ZigBee 技术是一种短距离、低速率、低功耗、低成本的短距离无线通信技术。它和 IEEE 802.15.4 一起构成无线传感器网络的标准通信协议, 适合于承载数据流量小、需要数据采集或监控的网点多、地形复杂、低成本和可靠性高的业务中, 是近几年研究和应用较多的协议, 在无线传感器网络领域占有重要地位。

4.5.1 ZigBee协议简介

ZigBee 协议栈是在 IEEE 802.15.4 标准基础上建立的, IEEE 802.15.4 标准定义了协议的 MAC 层和物理层, ZigBee 协议栈定义了网络层、应用层和安全服务层的标准。

ZigBee 应用层由应用支持 (APS)、应用框架 (AF)、ZigBee 设备对象 (ZDO, 包含 ZDO 管理平台) 和制造商定义的应用对象组成。APS 子层连接应用层和网络层, 负责维护设备间的绑定表 (Binding Table), 两个服务和需求相互匹配的设备, 彼此绑定后根据绑定表



进行通信。AF 是应用对象的环境，应用对象在 AF 里，通过 APS 接收和发送数据。控制和管理应用对象由 ZDO 负责，其负责定义设备在网络中的角色，比如是作为协调器 (Coordinator)、路由器 (Router) 或者终端设备 (End Device)，同时负责初始化 APS、网络层 (NWK)、安全服务规范 (SSS)。另外，在网络中 AF 还负责设备发现 (Device Discovery)、服务发现 (Service Discovery) 等。当一个 ZigBee 设备加入网络后，ZDO 将发起一系列初始化动作，通过 APS 进行设备发现和服务发现，然后根据事先定义的描述信息 (Description) 来接收和发送数据。如果应用层发起了绑定请求，则根据描述信息将与自己相关的服务和要求绑定的设备对象记录在 APS 中的绑定表，然后这个设备的所有通信，都将通过这个绑定表来查询。一旦两个设备绑定了，APS 将可以把从一个绑定设备接收的信息传送给另一个设备。

ZigBee 网络层负责设备到设备的通信，并负责网络拓扑的搭建、设备寻址、消息路由和网络发现，属于通用的网络层功能范畴。

ZigBee 设备以应用对象 (Application Objects) 的形式实现，一个设备可以包含多个应用对象。例如，在环境监测实验中，监测温度和湿度的应用就是一个应用对象。端点是应用对象的事例，每个端点就是一个应用对象，每个应用对象根据端点地址的不同得到区分。ZigBee 联盟规定端点地址的取值范围为 0~255。但端点 0 和端点 255 是两个特殊端点。端点 0 用于 ZigBee 设备对象配置和管理设备，应用程序可以通过端点 0 与其他层通信。端点 255 用于向所有端点广播。另外，端点 241~254 是保留端点，不能被用户使用。因此在 ZigBee 网络中，应用对象的地址由设备的网络地址和应用对象端点地址决定，每个应用对象通过一个端点连接到 ZigBee 协议栈的其他部分。

设备间的通信就是端点和端点之间的通信。在通信之前必须指定设备应用 Profile，应用 Profile 是为了多个设备、各个厂家的设备能互连、互通而制定的约束。它是一组统一的消息格式和处理方法。在 ZigBee 网络中两个设备之间通信的关键是统一一个 Profile。Profile 可以是公有的，也可以是私人制定的。Profile 由 Profile 标识符指定，每一个 Profile 标识符定义了设备描述和簇 (Cluster) 标识符。端点之间的通信是通过数据结构 Cluster 实现的，设备的数据输入输出和它有直接的联系。Cluster 是应用对象之间共享信息所需的全部属性的容器，端点间根据 Cluster 进行通信，具体传输的数据就是属性值。每个端点都有用于输入或输出的 Cluster，发送数据时根据指定的 Cluster 的属性发送数据，接收数据时的 Cluster 必须与发送方相对应。

ZigBee 网络节点包含 3 种类型：Coordinator、Router 和 End device。

所有 ZigBee 网络中都必须有一个并且只能有一个 Coordinator 节点。在网络层上，Coordinator 通常只用来进行系统初始化。在一些应用中，网络初始化完成后，即便是关闭了 Coordinator 节点，网络仍然可以正常工作。但是如果 Coordinator 还负责提供路由路径等工作，比如在星形网络拓扑中，Coordinator 转发消息的功能就不能被关闭，而必须持续的处于工作状态。同样，如果 Coordinator 在应用层提供一些服务，也必须持续地处于工作状态。

Coordinator 在网络层应完成如下任务。

- 选择网络所使用的频率通道，通常是最安静的频率通道。
- 启动网络。
- 将其他节点加入网络。



- Coordinator 通常还会提供信息路由、安全管理和其他的服

如果 ZigBee 网络采用了树形和 Mesh 拓扑结构就需要用到 Router 这种类型的节点。

Router 类型节点具有以下主要功能。

- 在节点之间转发信息。
- 容许子节点通过它加入网络。
- 需要注意的是，通常 Router 节点不能休眠。

End Device 节点的主要任务就是发送和接收信息。通常，End Device 节点是由电池供电的，并且当它不在数据收发状态时它通常都处于休眠状态以节省电能。End Device 节点不能转发信息，也不能让其他节点加入网络。

上述 3 种节点在 ZigBee 网络中担当了不同的角色。协调器节点负责启动网络，它首先搜寻一个合适的频率通道（通常是最安静的通道），然后启动网络，并且设定网络的 PAN ID，最后准备相应其他节点加入网络的请求。PAN ID 为该个域网的一个唯一的标识。

网络启动之后，路由器（Router）和终端（End Device）节点加入到由协调器节点启动的网络。路由器和协调器节点都能允许其他节点加入网络。加入的过程如下：

- 新的要加入网络的节点首先扫描所有可用的通道找到正在运行的网络并识别要加入的网络。一个通道可能有多个网络运行，各个网络以其他 PAN ID 区分。
- 新节点可以选择连接到哪个节点。一个个域网中或有一个协调器同时还可能存在多个路由器，新节点可以自由选择，选择的标准通常是根据信号强度。
- 新节点向选择的节点发送加入网络的信息。
- 被选择的节点检测新节点是否是合法的，同时是否允许新节点加入（由该节点当前是否允许其他节点连接和它是否有足够的地址空间决定）。如果条件满足，被选择的节点将允许新节点加入并给新节点分配一个地址。

一般的，可以配置路由器或协调器节点在哪一段时间内允许其他节点的加入，当然也可以设置成一个无限期的时间，这样在任何时刻都能允许其他节点的加入。

ZigBee 标准支持 3 种主要的网络拓扑结构：星形结构、树形结构和网状结构。星形结构是最简单的拓扑结构，由协调器和终端节点组成。每个终端节点只能和协调器通信，如果终端节点间相互通信，必须通过协调器作为中转节点。树形结构由协调器、路由器和终端节点组成。树的根节点为协调器节点，它可以有多个路由器和终端节点作为它的孩子，每个路由器也可以有多个路由器和终端节点作为它的孩子。每个孩子节点只能直接与它的父节点通信；每个中间节点只能与它的孩子节点和它的父节点直接通信；两个不能直接通信的节点要想通信，消息必须从源节点传到两个节点最近的祖先节点，然后由祖先节点传递到目的节点。网状结构由协调器、路由器和终端节点组成。网络结构和树形结构一样，但通信更灵活，路由器可以和它相邻的所有节点直接通信。所以当有一条通信路由拥塞或不可用时，能灵活地寻找替代的路由。

ZigBee 具有如下技术特点。

1) 设备功耗低。ZigBee 技术采用了很多省电的模式，设备在使用电池供电情况下，可以使用最短数月最长数年。

2) 可靠度高。ZigBee 采用了 CSMA/CA 防碰撞机制，大幅提高信息传输的可靠度。

3) 网络自组织、自愈能力强。ZigBee 网络节点能感知其他节点的存在，自动确定



相互关系，自组织成网络。而当网络部分节点出现故障，网络能够自我修复，重新调整网络拓扑结构。

4) 网络可扩充性高。一个 ZigBee 网络最多可以有 254 个从设备和一个主设备，而通过使用网络协调器，一个区域内可以同时存在 65000 多个网络节点。

5) 成本低。ZigBee 设备使用标准的普通电池，不需要重换电池，而且 ZigBee 协议是免专利费的，这也有效地降低了成本。

6) 数据安全。ZigBee 提供了一套基于 128 位高级加密标准的安全类软件，并集成了 IEEE 802.15.4 的安全元素，同时可以为各应用提供灵活的安全属性。

4.5.2 实验十六 ZigBee协议组网通信实验

1. 实验目的

本实验介绍了使用 IOT-ZBJ 节点实现一个基于 ZigBee 协议的环境检测程序。通过本实验的学习，读者能了解 ZigBee 协议的基本原理，掌握 IOT-ZBJ 节点的程序设计方法，并将二者结合实现一个简单的监测环境的光强度、温湿度的程序。

2. 实验设备

(1) 硬件

- ZigBee 协议实验节点 (IOT-ZBJ) 10 个。
- PC 1 台。

(2) 软件

- Code::Blocks 软件。
- Jennic Flash Programmer 烧写程序软件。

3. 实验原理

本实验实现了一个简单的环境监测程序，即由终端设备定时地收集温湿度等环境监测数据，并通过 ZigBee 协议建立网络，将监测数据汇集到协调器。为了对实验有更深入的了解，下面首先介绍 Jennic 提供的 ZigBee 协议栈的接口，包括实验中所用到的计时器、传感器数据采集接口、数据的发送以及串口等；然后对如何使用 Jennic 开发环境开发基于 ZigBee 的 IOT-ZBJ 节点应用程序进行简单的说明。

(1) Jennic开发环境介绍

1) ZigBee 协议栈的开发接口 API。任何一个 Jennic ZigBee 应用程序，都必须定义下面这些函数，即使有些函数用不到。首先是程序的入口函数 `AppColdStart()`、`AppWarmStart()`，以及 `JZA_boAppStart()`、`JZA_vStackEvent()`、`JZA_vPeripheralEvent()`、`JZA_vAppEventHandler()`、`JZA_vAppDefineTasks()`、`JZA_bAfKvpObject()`、`JZA_vAfKvpResponse()`、`JZA_bAfMsgObject()`和 `JZA_vZdpResponse()` 9 个函数。

还有两个必须在上述函数中调用用来分别初始化协议栈和启动协议栈的两个函数 `JZS_u32InitSystem()`和 `JZS_vStartStack()`。下面对每个函数做简单说明。

`AppColdStart()`：用户应用程序的入口。设备上电或从睡眠状态苏醒后应用程序从这个函数开始运行。一般在这个函数中应该调用协议栈和 BOS (Jennic 的一个简单调度器。它采用非抢先式调度机制，且所有任务具有相同的优先权) 的初始化函数。

`AppWarmStart()`：当设备从内存供电的休眠模式唤醒的时候将进入这个函数重新启动用



户应用程序。在这种情况下，所有内存数据都不会丢失。如果设备不需要休眠唤醒，这个函数可以为空。

JZS_u32InitSystem(): 用于初始化 Jennic 的 ZigBee 协议栈。

JZS_vStartStack(): 用于启动协议栈，并决定设备启动时作为协调器、路由器、终端中的哪一种启动。设备的网络角色在 `makefile` 中指定。如果是协调器将试着启动网络，是路由器或终端则试着找寻并加入网络。

JZA_boAppStart(): 允许用户应用程序在协议栈启动前注册端点的描述信息。一般在这个函数中调用 `JZS_vStartStack()` 启动协议栈。

JZA_vStackEvent(): 协议栈通过这个回调函数反馈网络层的一些网络事件，比如网络启动成功或者节点加入成功，或者数据发送完成等。

JZA_vPeripheralEvent(): 主要用来处理外部的硬件（如时钟或者串口等）中断。

JZA_vAppEventHandler(): 用户的应用函数，不需要任何特殊事件的触发而被 BOS 规则地调用。可以用来处理用户的主要任务或处理外部的硬件中断。此函数只能在短时间内处理后就必须返回，这样使 BOS 有时间处理其他的事件。因此在此函数中不要处理需要长时间等待的事件。

JZA_vAppDefineTasks(): 向 BOS 注册用户任务。很少被用到。在应用开始时只被调用一次。

JZA_bAfKvpObject(): 用来接收其他节点发送过来的 KVP 数据。函数返回的数据被 ZigBee 协议栈用来构造一个 KVP 回应帧。

JZA_bAfMsgObject(): 用来接收其他节点发送过来的 MSG 数据。当远程节点发送数据时根据具体的需求选择发送 KVP 数据还是 MSG 数据。KVP 和 MSG 分别是 ZigBee 协议定义两种不同的数据格式。其中，KVP (Key Value Pair) 是 ZigBee 协议中规定的一种特殊的数据传输机制，是专用的规范的信息格式，采用键值对的形式，也即按一种规定的格式进行数据传输，主要用于传输较简单的变量值格式；MSG (Message Service Type) 是 ZigBee 协议中规定的一种特殊的数据传输机制，没有具体格式上的规定，通常用于多信息、复杂信息的传输，在实践中一般采用 MSG 模式。

JZA_vAfKvpResponse(): 接收发送出去的 KVP 包后的回应消息，该回应消息由远程节点发出。

JZA_vZdpResponse(): 接收发送出去的 ZDP 请求的回应消息，如 Binding。

2) 使用 BOS 计时器。在 JN5121 和 JN513x 设备上，由于协议栈使用了 `tick-timer`，用户应用程序如果使用 `tick-timer` 将会使 BOS 或协议栈功能异常。因此，用户必须使用协议栈提供的 BOS 计时器来进行相关定时操作。BOS 计时器通过软件实现，其应用通过函数调用创建。计数器启动函数的原型为：

```
Bool_t bBosCreateTimer(void (*pfvTimerHandler)(void *, uint8),
                      void *pvMsg,
                      uint8 u8MsgSize,
                      uint32 u32Delay,
                      uint8 *pu8TimerId);
```



相关的参数为：

*pvTimerHandler 表示指向当计时器规定时间到后，被 BOS 调用的回调函数；

*pvMsg 表示指向传给计时器的消息；

u8MsgSize 表示消息长度；

u32Delay 表示计时器工作时间，其值乘以 10ms 就是计时器工作时间，例如定时 50ms，其值就为 5；

*pu8TimerId 表示指向接收创建计时器的 ID 的地址。

BOS 计时器不能重复、连续工作，即当计时器规定的时间到了之后，指定的相应事件只发生一次，之后计时器失效。如果想使指定的事件每隔一段时间就发生的话，必须在计时器的回调函数中再次重新启动计时器。例如：

```
PRIVATE void vTimer(void *vpMsg, uint8 u8Param) {
    //此处为执行预定的计时器任务相关代码
    bBosCreateTimer(vTimer, NULL, 0, 100, &u8TimerId); //重新启动计时器
}
```

3) 温湿度数据采集。首先初始化湿度和温度联合感应器，接口函数为 vHtsReset ()，必须在使用湿度和温度联合感应器前调用这个初始化函数。函数原型为：

```
PUBLIC void vHtsReset (void);
```

之后就可以调用温湿度的读取函数 vHtsStartReadTemp()和 vHtsStartReadHumidity()获取温度。其函数原型分别为：

```
PUBLIC void vHtsStartReadTemp(void);
PUBLIC void vHtsStartReadHumidity(void);
```

这两个函数只是发起读取的操作，在温湿度数据获取完毕之后，系统会调用相应的回调函数 i16HtsReadTempResult()和 u16HtsReadHumidityResult()返回数据。在发起读取操作之后而读取尚未完成之前，系统会一直阻塞程序，这样 BOS 就不能处理其他事件。因此读取前需要使用 if ((u32AHI_DioReadInput() & HTS_DATA_DIO_BIT_MASK) == 0)语句判断读取是否已经完成。u32AHI_DioReadInput()为读取每个输入引脚值的函数。读取完成后可以读取到温度或湿度的值。其函数原型分别为：

```
PUBLIC int16 i16HtsReadTempResult(void);
```

函数返回-40~124 度之间的值。

```
PUBLIC uint16 u16HtsReadHumidityResult(void);
```

函数返回 0%~160%之间的值。

温湿度的读取是使用了简单的状态机实现的，相关代码如下：

```
PRIVATE void vReadTempHumidity(void)
{
```



```
switch(sTempHumiditySensor.eState)
{
    //读取湿度
    case E_STATE_READ_TEMP_HUMID_IDLE:
        vHTSstartReadHumidity();
        sTempHumiditySensor.eState = E_STATE_READ_HUMID_RUNNING;
        break;
    //判断湿度是否读取完毕
    case E_STATE_READ_HUMID_RUNNING:
        if ((u32AHI_DioReadInput() & HTS_DATA_DIO_BIT_MASK) == 0)
        {
            sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_COMPLETE;
        }
        break;
    //读取完毕就存储读取的湿度值
    case E_STATE_READ_TEMP_HUMID_COMPLETE:
        sTempHumiditySensor.u16HumidReading = u16HTSreadHumidityResult();
        sTempHumiditySensor.eState = E_STATE_READ_TEMP_START;
        break;
    //开始读取温度
    case E_STATE_READ_TEMP_START:
        vHTSstartReadTemp();
        sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_RUNNING;
        break;
    //判断读取温度是否完成
    case E_STATE_READ_TEMP_HUMID_RUNNING:
        if ((u32AHI_DioReadInput() & HTS_DATA_DIO_BIT_MASK) == 0)
        {
            sTempHumiditySensor.eState = E_STATE_READ_TEMP_COMPLETE;
        }
        break;
    //存储读取的温度值
    case E_STATE_READ_TEMP_COMPLETE:
        sTempHumiditySensor.u16TempReading = u16HTSreadTempResult();
        sTempHumiditySensor.eState = E_STATE_READ_TEMP_HUMID_READY;
        break;

    case E_STATE_READ_TEMP_HUMID_READY:
        break;

    default:
        break;
}
}
```



4) 光传感数据采集。同温湿度传感器类似, 读取光传感器的值首先也需要初始化光传感器, 初始化接口函数原型为:

```
PUBLIC void vAlsReset(void);
```

然后初始化读取光, 初始化时需要选择一个读取的通道。函数原型为:

```
PUBLIC void vAlsStartReadChannel(uint8 u8Channel);
```

输入: `u8Channel` 0 表示通道既可以感知可见光, 又可以感知红外线光;
1 表示通道只感知红外线光。

调用了这个函数后设备会连续启动进行采集, 因此不需要重复调用这个函数。之后就可以读取特定通道的光值了, 函数 `u16AlsReadChannelResult()` 就是读取最近读取的光级值。函数原型为:

```
PUBLIC uint16 u16AlsReadChannelResult(void);
```

返回范围为 0~4015 的光级值。
读取两个通道的代码如下:

```
PRIVATE void vReadAls0(void)
{
    //通道 0
    vALSstartReadChannel(0);
    u16AlsReading0 = u16ALSreadChannelResult();
}

PRIVATE void vReadAls1(void)
{
    //通道 1
    vALSstartReadChannel(1);
    u16AlsReading1 = u16ALSreadChannelResult();
}
```

5) 发送数据。在 Jennic ZigBee 协议栈 API 中节点发送数据的函数为 `afdeDataRequest()`。其函数原型请参考 JN-RM-2018。

使用函数 `afdeDataRequest()` 发送数据之前, 先将要发送的数据存储在 `AF_Transaction_s` 类型的结构中。`afdeDataRequest()` 主要指定了目的地址、目的端口、源端口、发送的消息类型、存储数据的 `AF_Transaction_s` 类型变量。

```
PRIVATE void vSendData(void)
{
    AF_Transaction_s asTransaction[1];

    //读自身的 MAC 地址
    MAC_ExtAddr_s sIeeeAddr;
```



```
sIeeeAddr = *(MAC_ExtAddr_s *)pvAppApiGetMacAddrLocation();
//sIeeeAddr 是 64 位的地址，因此用 4 个 16 位的变量记录其高位和低位地址
uint16 sIeeeAddr32HL = sIeeeAddr.u32L;
uint16 sIeeeAddr32HH = sIeeeAddr.u32L >> 16;

uint16 sIeeeAddr32LL = sIeeeAddr.u32H;
uint16 sIeeeAddr32LH = sIeeeAddr.u32H >> 16;
//发送的数据存储在 asTransaction[0]结构中
asTransaction[0].u8SequenceNum = u8AfGetTransactionSequence(TRUE);
asTransaction[0].uFrame.sMsg.u8TransactionDataLen = 18;
//存储电压值
asTransaction[0].uFrame.sMsg
    .au8TransactionData[0] = sBattSensor.u16Reading;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[1] = sBattSensor.u16Reading >> 8;
//存储温度值
asTransaction[0].uFrame.sMsg
    .au8TransactionData[2] = sTempHumiditySensor.u16TempReading;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[3] = sTempHumiditySensor.u16TempReading >> 8;
//存储湿度值
asTransaction[0].uFrame.sMsg
    .au8TransactionData[4] = sTempHumiditySensor.u16HumidReading;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[5] = sTempHumiditySensor.u16HumidReading >> 8;
//存储两个光级值
asTransaction[0].uFrame.sMsg
    .au8TransactionData[6] = u16AlsReading0;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[7] = u16AlsReading0>>8;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[8] = u16AlsReading1;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[9] = u16AlsReading1>>8;
//存储地址
asTransaction[0].uFrame.sMsg
    .au8TransactionData[10] = sIeeeAddr32HH;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[11] = sIeeeAddr32HH >> 8;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[12] = sIeeeAddr32HL;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[13] = sIeeeAddr32HL >> 8;
asTransaction[0].uFrame.sMsg
    .au8TransactionData[14] = sIeeeAddr32LH;
asTransaction[0].uFrame.sMsg
```



```

        .au8TransactionData[15] = sIeeeAddr32LH >> 8;
asTransaction[0].uFrame.sMsg
        .au8TransactionData[16] = sIeeeAddr32LL;
asTransaction[0].uFrame.sMsg
        .au8TransactionData[17] = sIeeeAddr32LL >> 8;
//调用 afdeDataRequest 函数将数据发送出去
(void)afdeDataRequest(APS_ADDRMODE_SHORT, //地址类型
                    0x0000, //目的地址
                    WSN_DATA_SINK_ENDPOINT, //目的端口
                    WSN_DATA_SOURCE_ENDPOINT, //源端口
                    WSN_PROFILE_ID, //Profile 号
                    WSN_CID_SENSOR_READINGS, //Cluster 号
                    AF_MSG, //消息类型
                    1, //Transactions
                    asTransaction, //Transaction 数据
                    APS_TXOPTION_NONE, //Transmit options
                    SUPPRESS_ROUTE_DISCOVERY, //路由模式
                    0); //Radius count
}

```

6) 使用串口与 PC 通信。Jennic 的 ZigBee 协议栈提供了类似 printf 的函数来向串口输出信息，该函数封装在 printf.c、printf.h 中，将这两个文件加入到工程，然后在主程序的头部加入头文件#include "..\..\..\Chip\Common\Include\Printf.h"，即可调用相关的函数向串口发送信息了。

在程序中，首先需要初始化串口，在代码的初始化设备的函数里加入 vUART_printInit()，待初始化完成之后，可以使用 VPrintf 来向串口输出信息。其函数原型为：

```
PUBLIC void vPrintf(const char *fmt,...)
```

例如：

```
vPrintf("Temp= %d \r\n",u16Temp);
```

也可以使用函数 vPutC 向串口发送一个 8 位的信息。函数原型为：

```
PUBLIC void vPutC(unsigned char c);
```

(2) 环境监测程序的实现

实验需要完成 3 个设备 Coordinator、Router 和 End Device 节点的程序。Coordinator 负责发起网络，等待 Router 和 End Device 节点的加入，它本身不采集任何数据，只接收其他节点发送的数据，并将这些数据定期发送给 PC 的串口中。数据主要包括发送帧的源设备 16 位短地址，设备采集的传感器的数据。Router 负责加入网络并采集传感器数据，定期将这些数据发送给 Coordinator，并可以转发其他节点的数据。End Device 负责加入网络，并采集数据，将采集的数据发送出去完成后进入休眠期，经过固定的时间后苏醒，开始采集数据，如此循环。End Device 只具有收发数据的功能，不能转发数据。



实验通过这3种节点组成 Mesh 网络, 由于存在 Router 节点, 因此可以实现多跳的功能。

1) 程序包含以下4个文件。

- **WSN_Profile.h**: 此头文件主要定义了网络和设备的一些参数设置, 比如 PAN ID、网络所在的通道、设备端点、端点使用的 Cluster。
- **WSN_Coordinator.c**: 此文件为节点 Coordinator 的源代码。
- **WSN_Router.c**: 此文件为节点 Router 的源代码。
- **WSN_EndDevice.c**: 此文件为节点 End Device 的源代码。

2) 程序的主要函数。Jennic 程序由 boot loader 来启动和引导, 引导完成后将自动调用 AppColdStart 函数, 因此 AppColdStart 是所有 Jennic 程序的入口函数, 相当于 C 语言中的函数 main()。主要用来设置网络信息 PAN ID 和网络的通道。调用函数 vInit()完成其他初始化设置。

在 WSN_Coordinator.c 中, vInit()用来初始化 ZigBee 协议栈、初始化指示灯、初始化 Uart、启动 BOS。

在 WSN_Router.c 和 WSN_EndDevice.c 中, vInit()用来初始化 ZigBee 协议栈、初始化指示灯、调用函数 vInitSensors()初始化传感器、启动 BOS。

其他函数的功能为:

PRIVATE void vTxSerialDataFrame()用来向 Uart 发送数据的节点的 16 位短地址、消息序号、接收到的温度值、接收到的湿度值、节点的电压值、通道 0、通道 1 的光级值和发送数据的节点的 IEEE 地址。函数中用到的函数 vPutC(), 用来向 Uart 发送一个 8 位的数据, 因此若发送一个 16 位的数据, 则将数据分为高 8 位、低 8 位分别发送。例如:

```
uint16 u16NodeId;
vPutC(u16NodeId >> 8);
vPutC(u16NodeId);
```

PRIVATE void vInitSensors()初始化了温湿度、光传感器, 初始化并配置了 ADC。

PRIVATE void vAppTick()为 WSN_Router.c 和 WSN_EndDevice.c 程序的核心内容, 完成了整个程序的主要工作, 包括: 读取各个传感器的值、电压值, 控制指示灯的闪烁, 最后将数据发送到协调器。这个函数是一个时钟周期性调用的函数, 在 BOS 计时器失效后, 函数在后来又创建一个 BOS 计时器, 以便在固定的周期后调用自己。因此, 这个函数被周期性地调用。

PRIVATE void vReadBatteryVoltage()用来读取电压值。关于如何使用 ADC 采样电压值前面已经做了介绍。

PRIVATE void vReadTempHumidity()用来读取传感器温度、湿度值。

PRIVATE void vReadAls0()用来读取通道 0 的光级值。

PRIVATE void vReadAls1()用来读取通道 1 的光级值。

PRIVATE void vSendData()将读取到的数据通过数据包发送给协调器。

3) WSN_Coordinator.c 流程:

- 进入 AppColdStart(), 初始化串口、协议栈、启动 BOS。
- BOS 调用函数 JZA_boAppStart(), 启动协议栈。



- 设备试着创建网络，产生 `JZS_EVENT_NWK_STARTED` 协议栈事件。
- 协议栈调用 `JZA_vStackEvent()`，为端点注册简单描述。
- BOS 启动后，规则的调用函数 `JZA_vAppEventHandler()` 周期性的闪烁指示灯，表示节点正在工作。
- 只要收到 `MSG` 消息，协议栈调用函数 `JZA_bAfMsgObject()` 接收数据，并将收到的数据发送到串口。

4) WSN_Router.c 流程:

- 进入 `AppColdStart()`，初始化传感器、ADC、协议栈、启动 BOS。
- BOS 调用函数 `JZA_boAppStart()`，启动协议栈。
- 设备试着加入网络，产生 `JZS_EVENT_NWK_JOINED_AS_ROUTER` 协议栈事件。
- 协议栈调用函数 `JZA_vStackEvent()`。
- BOS 启动后，规则的调用函数 `JZA_vAppEventHandler()`，从而周期性地调用函数 `vAppTick()`，即周期性执行以下操作：读取温湿度传感器值；读取节点电压值；读取通道 0 和通道 1 的光级值；当读取全部完成时，闪烁指示灯提示节点正常工作。
- 调用函数 `vSendData()` 生产数据包，将读取的值发送给协调器。

5) WSN_EndDevice.c 流程:

其所有流程和 `WSN_Router.c` 的一样，只是在调用函数 `vSendData()` 发送数据后，产生 `JZS_EVENT_APS_DATA_CONFIRM` 事件后，调用函数 `JZA_vStackEvent()`，定义一个计时器，30s 后将休眠的节点唤醒。节点进入休眠，30s 后节点被唤醒，将从函数 `vAppWarmStart()` 重新启动程序。

4. 实验内容

阅读 ZigBee 协议，学习协议的运行机制以及协议中各层的数据包格式。

初步掌握 Jennic ZigBee 协议栈的使用方法，能够结合 IOT-ZBJ 节点进行简单的基于 ZigBee 的应用的开发。

5. 实验步骤

1) 启动软件 Code::Blocks，在 `C:\Jennic\cygwin\Jennic\SDK\Application` 路径下新建文件夹 `ZigBeeWSN`。在此文件夹中新建工程 `WSN_Coordinator.cbp`，工程类型选择 `ZigBee Coordinator`。

2) 在新建工程中的 `JN51xx_JZ_Coord.c` 文件里编辑代码或直接将此文件从工程中删除，然后在工程中加入 `WSN_Coordinator.c`、`printf.c` 和 `WSN_Profile.h`。这些文件在所附光盘中可以找到。然后编译源程序。

3) 目标文件 `WSN_Coordinator.bin` 生成在目录 `C:\Jennic\cygwin\Jennic\SDK\Application\ZigBeeWSN\JN5139_Build\Release` 下。

4) 重复上面的步骤，将步骤 1) 中的工程类型改为 `ZigBee Router`，新建 `WSN_Router.cbp` 工程，步骤 2) 中只需要加入文件 `WSN_Router.c` 和 `WSN_Profile.h`，然后编译，在工程文件 `Realse` 目录下产生 `WSN_Router.bin` 文件。

5) 同样，将步骤 1) 中的工程类型改为 `ZigBee End Device`，新建 `WSN_EndDevice.cbp` 工程，步骤 2) 中只需要加入文件 `WSN_EndDevice.c` 和 `WSN_Profile.h`，然后编译，生成 `WSN_EndDevice.bin` 文件。



6) 将串口线连接到节点, 打开 Jennic Flash Programmer 软件, 然后给节点上电, 下载程序到节点上, 不同的设备下载不同的目标程序。注意: 烧写程序时, 一定要在打开 Jennic Flash Programmer 软件后, 才能给节点上电。

7) 关闭 Jennic Flash Programmer 软件, 重启节点。先给 Coordinator 节点上电, Coordinator 节点上的 LED1 闪烁, 表示发起网络成功。给 Router 节点和 End device 节点上电, 加入网络后, 它们的 LED1 闪烁。其中 End Device 节点的 LED1 闪烁一下后熄灭, 经过 30s 后 LED1 再亮一次, 如此重复。

8) 将串口线连接到 Coordinator 节点, 打开串口调试助手, 波特率设置为 115200bit/s, 可以收到节点发来的数据。下面是一个典型的数据包:

```
00 16 00 15 8D 00 00 09 10 1D 00 01 00 0F 00 1D 00 DB 00 39 0A 3E FF FF
```

其中, “00 16” 为包序号; “00 15 8D 00 00 09 10 1D” 为发送节点的长地址; “00 01” 为 Coordinator 分配的临时短地址; “00 0F” 为湿度值; “00 1D” 为温度值; “00 DB” 为通道 0 的光强值; “00 39” 为通道 1 的光强值; “0A 3E” 为节点电压值; “FF FF” 为包尾。此时用手按住某 End Device 或者 Router 的温度传感器, 可以发现收到的数据包中温湿度值会相应地发生变化。

6. 思考题

- 1) 如何将 End Device 节点休眠的时间改为 10s?
- 2) 如何改变节点发送数据的周期?

7. 补充阅读

[1] 徐勇军, 刘峰, 等. 低速无线个域网实验教程[M]. 北京: 北京理工大学出版社, 2008.

[2] ZigBee Application Framework API Reference Manual. Jennic, 2008.

第5章 物联网接入与处理实验

借助于通信和网络技术，物联网可以将海量的感知信息完整地呈现给用户，真正做到无处不在的感知。但正如互联网上的海量信息一样，普通用户无法直接消化这些信息，只有通过进一步的加工处理，这些数据才能为用户所用。本章设计了物联网的接入与处理实验，主要包括公网接入实验、网络安全通信实验、室内实时定位实验、自组网时间同步实验、海量数据搜索实验等。通过一系列的处理和加工，物联网可以给用户提供内容丰富的服务。本章实验是物联网进行行业应用的技术支撑，属于综合性实验。

5.1 接入与处理实验概述

物联网连接的网络具有很强的异构性，感知信息系统也具有很强的多样性。为了实现异构信息系统之间的互联、互通与互操作，未来的物联网需要以一个开放的、分层的、可扩展的网络体系结构为框架，实现异种异构网络能够与骨干网络实现无缝连接，并提供相应的服务质量保证，这就是网络接入的目标。基于物联网的下一代网络演进技术包括未来适应的网络体系架构、业务架构、安全、QoS 等诸多方面，重点技术涵盖多网融合、虚拟化、自治计算、高可信、移动泛在、高效节能和高可扩展等关键技术，这给现有的接入技术及网络技术提出了新的课题。以传感器网络为代表的末梢网络在规模化应用后同样面临与骨干网的接入问题，并且需要与骨干网络进行充分协同，这些都面临着巨大的挑战。

当海量的感知数据被推送给用户后，这些海量感知信息计算与处理的技术将成为瓶颈。如何面向行业及用户提供方便定制的物联网云计算服务，是物联网应用大规模发展后面临的巨大技术挑战，这需要用户能够按需要构建包括硬件设备平台、核心软件平台、智能信息处理平台及服务平台在内的面向物联网应用的“云计算”平台，重点突破海量感知信息的数据融合、高效存储、语义集成、并行处理、知识发现和数据挖掘等关键技术，攻克物联网“云计算”中的虚拟化、网格计算、服务化和智能化等一系列核心技术，实现安全可靠、高可用、易扩展、低成本、界面友好、用户可定制的物联网计算与服务平台。

物联网的接入与处理工作异常艰巨，它将从很大程度上约束物联网应用的广度与深度，并且直接影响到物联网的规模化，在这一层几乎所有的工作都是很大的工程，要想用实验的方式来一一展示难度很大。本章在消化吸收相关知识的前提下，将从物联网的接入和处理技术进行切入，介绍公网接入实验、网络安全通信实验、室内实时定位实验、自组网时间同步实验、海量数据搜索实验。当然，在实际系统中这一层次的技术远远不止这些，还需要读者进一步挖掘，并根据需求开发应用。



5.2 公网接入实验

物联网的发展和应用离不开互联网和移动通信网的基础设施。目前, 这些信息高速公路已经足够承载物联网的这部分应用。将传感器网络、RFID 等短距离网络与这些骨干网络进行接入, 并完成互联互通互操作是十分重要的。本实验通过以手机为代表的智能终端平台, 很方便地完成传感器网络与互联网和移动通信网的融合, 与实验三呼应。

5.2.1 通信网接入简介

手机的发展史不只代表着科技的进步, 也证明了人类文明的发展。手机的产生给我们带来了很大的方便。第一代(1G)模拟制式手机只能进行语音通信, 无线带宽利用不足; 第二代(2G) GSM、CDMA 等数字制式手机除了可以进行语音通信外, 还可以收发短信以及接收数据(如浏览网页和收发邮件)等。尤其是 2009 年诞生的 3G, 对于手机、移动通信和移动互联网都是革命性的变革; 其功能让人眼花缭乱, 比如高速的无线带宽上网、无线搜索、视频通话、在线手机音乐等。

无线网络的发展速度更是异常迅猛, 当第三代通信技术建立起来以后, 无线网络的带宽得到了巨大的提升, 可靠性更高。有了无线网络的支持, 就可以随时随地进行数据交换, 浏览万维网、WAP 网。随着互联需求的不断提高, 人们开始通过移动网络来接入 Internet, 比如浏览网页、手机银行、手机证券等, 因此使手机接入移动网络成为其发展的目标和方向。

Android 平台出自 Google 公司, 而 Google 公司又是网络界的巨头, 因此 Android 与 Internet 的接轨与应用, 自然成为必然。当人们提到 Internet 时首先想到的就是 HTTP Protocol 的支持能力, 在 Android SDK 中, 已经集成了 Apache 的 HttpClient 模块, 通过这个模块, 开发者可以轻而易举地实现 Android 接入 Internet。Android 平台是智能终端环境中的新宠儿, 拥有巨大的发展潜力, 这在前面进行过详细的介绍。下面通过一个具体的基于 Android 平台的互联网接入实验使大家对移动互联技术有一个初步的了解。

5.2.2 实验十七 Android系统下互联网接入实验

1. 实验目的

本实验介绍了使用 Android 手机接入互联网。通过实验, 可以使读者初步掌握 Android 平台手机接入互联网的编程方法和步骤, 并能够开发简单的互联网接入的应用程序。

2. 实验设备

(1) 硬件

- Android 手机 1 个。
- PC 1 台。

(2) 软件

- Java SE Development Kit 6u22 for Windows。
- Eclipse IDE for Java EE Developers。
- Android Development Tools 0.9.9。



3. 实验原理

Android 平台手机可以通过 GSM、CDMA、3G 网络和 WiFi 等无线网络接入 Internet。Android 平台手机主要是通过 HTTP 请求和发送数据到服务器端，使用 HTTP 请求和发送数据有两种方式，一种是 Get 方法；一种是 Post 方法。Get 是用来从服务器上获得数据；而 Post 是用来向服务器传递数据。Get 方式不安全，因为传输过程中，数据请求放在请求的 URL 中；而 Post 的所有操作对用户来说都是不可见的，因此安全性较高。另外，受 URL 长度的限制，Get 传输的数据量较小；而 Post 则可以传输大量的数据。在编写程序实现上，Get 方法相对来说简单一点；Post 方法相对于 Get 方法稍显复杂。在本实验中采用 Post 方法。图 5-1 为 Android 通过 HTTP 接入网络的示意图。

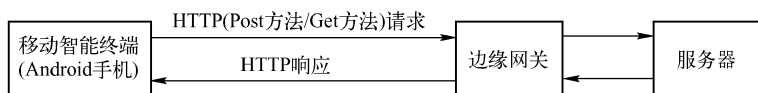


图 5-1 Android 通过 HTTP 接入网络示意图

基于 Linux 内核的 Android 操作系统包含了一组非常优秀的联网功能。目前，Android 平台有 3 种网络接口可以使用，它们分别是：标准 Java 接口 (java.net.*)、Apache 接口 (org.apache) 和 Android 网络接口 (android.net.*)。

(1) 标准Java接口

java.net.*下面提供了访问 HTTP 服务的基本功能和接口，包括流和数据包套接字、Internet 协议、常见 HTTP 处理。使用这些接口的基本操作主要包括：创建 URL 以及 URLConnection/HttpURLConnection 对象，设置链接参数，连接到服务器，向服务器写数据，从服务器读取数据。

下面是常见的使用 java.net.*包的 HTTP 例子，代码清单如下：

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.*
//.....
try{
//创建一个 URL 对象
    URL url = new URL("http://www.google.com");
    //创建一个 URL 链接
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    //得到链接状态
    int state_RC = http.getResponseCode(){
    //向服务器发送数据并获取应答
    InputStream in = conn.getInputStream();
    }
}catch(Exception e){
}
```

(2) Apache接口

Android 提供的 Apache HttpClient 是一个开源项目，弥补了标准 Java 接口(java.net.*)灵



活性不足的缺点，功能更加完善，为客户端的 HTTP 编程提供了高效、最新、功能丰富的工具包支持。使用这部分接口的基本操作与 `java.net.*` 基本类似，主要包括：创建 `HttpClient` 以及 `GetMethod/PostMethod`，`HttpRequest` 等对象，设置连接参数，执行 HTTP 操作，处理服务器返回结果。

下面是使用 `android.net.http.*` 包的例子，代码清单如下：

```
import android.net.http.*;
//.....
try{
    //创建一个 HttpClient 实例
    //使用 HttpClient 是 Commons HttpClient 中的用法，在 Android 1.5 中需要使用 Apache 的
    //默认实现 DefaultHttpClient
    HttpClient httpClient = new HttpClient();
    //创建 HttpGet 实例，该方法会自动处理 URL 地址的重定向
    HttpGet httpGet = new HttpGet("http://www.google.com");
    //链接
    HttpResponse response = httpClient.execute(httpGet);
    If(response.getStatusLine().getStatusCode() != HttpStatus.SC_OK){
        InputStream is = response.getEntity().getContent();
        //处理数据.....
    }
} catch(Exception e){
}
//.....
```

(3) Android 网络接口

`android.net.*` 包实际上是通过在 Apache 中 `HttpClient` 的封装来实现的一个 HTTP 编程接口，同时还提供了 HTTP 请求队列管理以及 HTTP 连接池管理，以提高并发请求情况下的处理效率。除此之外，还有网络状态监视等接口、网络访问的 `Socket`、常用的 `URI` 类以及与 WiFi 相关的类等。

下面是一个通过 `AndroidHttpClient` 访问服务器的简单例子，代码清单如下：

```
import android.net.http.AndroidHttpClient;
//.....
try{
    AndroidHttpClient client = AndroidHttpClient.newInstance("agent");
    //创建 HttpGet 方法，该方法会自动处理 URL 地址的重定向
    HttpGet httpGet = new HttpGet("http://www.google.com");
    HttpResponse response = client.execute(httpGet);
    If(response.getStatusLine().getStatusCode() != HttpStatus.SC_OK){
        InputStream is = response.getEntity().getContent();
        //数据处理.....
    }
} catch(Exception e){
}
}
```



以上简单地介绍了网络访问的 3 种方式，但在实际开发应用中，可能会用到更加复杂的联网操作。在本实验中以第二种方式为例。首先让我们来了解使用 `HttpClient` 所用到的一些类。

1) `ClientConnectionManager` 接口。`ClientConnectionManager` 是客户端连接管理器接口，它提供了几个抽象方法，如表 5-1 所示。

表 5-1 `ClientConnectionManager` 的抽象方法

方 法	说 明
<code>ClientConnectionManager</code>	关闭所有无效、超时的连接
<code>closeIdleConnection</code>	关闭空闲的连接
<code>releaseConnection</code>	释放一个连接
<code>requestConnection</code>	请求一个新的连接
<code>shutdown</code>	关闭管理器并释放资源

2) `DefaultHttpClient`。`DefaultHttpClient` 是默认的一个 HTTP 客户端，可以使用它创建一个 HTTP 连接。

```
HttpClient httpClient = new DefaultHttpClient();
```

3) `HttpResponse`。`HttpResponse` 是一个 HTTP 连接响应，当执行一个 HTTP 连接后，就会返回一个 `HttpResponse`，可以通过 `HttpResponse` 获得一些相应信息。请求一个 HTTP 连接并获得连接请求是否成功的代码如下：

```
HttpResponse httpResponse = httpClient.execute(httpRequest);
if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
    //连接成功
}
```

本实验中采用 `Post` 方式来请求网络数据。使用 `Post` 方法首先要使用 `URL` 创建一个 `HttpPost` 对象，代码如下：

```
String url = "http://10.10.0.102:8080/httpptest.jsp";
//使用 url 创建 HttpPost 对象
HttpPost request = new HttpPost(url);
```

使用 `Post` 方式，需要使用 `NameValuePair` 来保存将要传递的参数，这里使用 `BasicNameValuePair` 来构造一个要被传递的参数 `params`，然后通过 `add` 方法添加这个参数到 `NameValuePair` 中，代码如下：

```
//使用 NameValuePair 来保存要传递的参数，必须是 NameValuePair
List<NameValuePair> params = new ArrayList<NameValuePair>();
//将要传递的参数添加到 params 中
Params.add(new BasicNameValuePair("param", "Hello Network!"));
```

`Post` 方式还需要设置所使用的字符集，最后通过 `HttpClient` 来请求这个连接，返回响应并处理，代码如下：



```
//设置字符集, 如果与服务器交互, 则需要与服务器编码方式一致
HttpEntity httpEntity = new UrlEncodedFormEntity(params, "UTF-8");
//请求 request
request.setEntity(httpEntity);
//取得默认的 HttpClient
HttpClient httpClient = new DefaultHttpClient();
//取得 HttpResponse
HttpResponse httpResponse = httpClient.execute(request);
//判断连接状态
if(httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
    //对返回的数据进行处理
}
```

代码编写完毕, 为了使程序拥有访问网络的权限, 要在 `AndroidManifest.xml` 中添加如下内容:

```
<uses-permission android:name="android.permission.INTERNET" />
```

服务器端代码的编写, 在后台服务器只需要编写一个能够接收客户端传递过来数据的 `Servlet` 就可以了。Servlet 的 `doPost` 方法代码如下:

```
Public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    //取得客户端传递过来的参数
    String str = request.getParameter("params");
    System.out.println(nameString);
}
```

与一般的 Java 项目一样, `src` 文件夹是项目的包及源文件, `res` 文件夹则包含了项目中的所有资源。项目下的 `gen` 文件夹下的 `R.java` 文件是自动生成的, 不需要对其进行修改。`AndroidManifest.xml` 文件包含了该项目中所使用的 `Activity`、`Service`、`Receiver`。程序启动时显示的第一个界面是由 `AndroidManifest.xml` 确定的。

主程序 `HttpActivity` 类继承自 `Activity` 类, 重写了 `void onCreate (Bundle savedInstanceState)` 方法。在 `onCreate` 方法通过 `setContentView(R.layout.main)` 设置 `Activity` 要显示的布局文件, 该方法也是程序的入口。上面介绍的使用 `Post` 方法访问 `Internet` 的代码均包含在该函数中。其代码如下:

```
import ...
public class HttpActivity extends Activity{
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btn_post = (Button) findViewById(R.id.upload);
        txt_show = (TextView) findViewById(R.id.text);

        //给 Button 设置触发事件
        btn_post.setOnClickListener(new OnClickListener() {
```



```
@Override
public void onClick(View v) {
    // TODO Auto-generated method stub
    txt_show.append("Hello Network");
    String httpUrl = "http://10.20.0.102:8080/server";
    //使用 url 创建 HttpPost 对象
    HttpPost httpRequest = new HttpPost(httpUrl);
    //使用 NameValuePair 保存要传递的参数, 必须是 NameValuePair
    List<NameValuePair> params = new ArrayList<NameValuePair>();
    //将要传递的参数添加到 params 中
    params.add(new BasicNameValuePair("par", "Hello Network"));
    try{
        //设置字符集, 如果与服务器交互, 则需要与服务器编码方式一致
        HttpEntity httpEntity = new UrlEncodedFormEntity(params, "UTF-8");
        //请求 request
        request.setEntity(httpEntity);
        //取得默认的 HttpClient
        HttpClient httpClient = new DefaultHttpClient();
        //取得 HttpResponse
        HttpResponse httpResponse = httpClient.execute(request);
        //判断连接状态
        if(httpResponse.getStatusLine().getStatusCode() ==
            HttpStatus.SC_OK){ //取得返回的字符串
            String str = EntityUtils.toString(httpResponse.getEntity());
            txt_show.setText(str);
        }
        else{
            txt_show.setText("Error!");
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}
});
}
```

4. 实验内容

学习 Android 平台下基于 HTTP 访问互联网的原理和方法。

初步掌握 Android 平台下的网络编程方法, 学习在 Android 环境下访问 Internet 的 3 种方法, 并能够写出简单的网络连接程序。

5. 实验步骤

1) 在 Eclipse 下配置 Android 开发环境。

- 下载 ADT 使用本地安装, 或者通过在线安装。
- 安装完毕, 在 Eclipse 中选择 Windows→Preferences→Android, 选择 Android SDK 的



路径。

- 如果使用模拟器调试程序，选择 Android SDK and AVD Manager，在里面新建一个 AVD。具体的配置步骤详见 Android 官方网站 <http://developer.android.com.nyud.net>，也可以参考本书配套光盘中相应的文档。

2) 创建 Android 工程。

- 选择 File→New→Android Project。

- 使用 droiddraw 创建页面，当界面绘制完毕后，单击 Generate，生成.xml 文件，将生成的内容复制到工程下的 res 下的 layout 文件夹下的 main.xml 文件中，保存。

- 在 src 文件夹下编写 Java 代码。

3) 编写服务器端 Servlet，新建一个 Servlet，命名为 Receive.java。建议在 Eclipse 中安装 MyEclipse 插件，在 Receive.java 中编写修改 doPost 方法。然后将其部署到 Apache Tomcat 下，并启动 Apache 和服务器，在安装有 MyEclipse 的 Eclipse 中直接右击 Receive.java 文件，选择 Run as→MyEclipse Server Application。

4) 运行 Android 程序。在 Eclipse 中右击工程目录，选择 Run as→Android Application。

5) 如果服务器端在 Eclipse 下的 Console 窗口下显示“Hello Network”，则联网成功。

6. 思考题

1) 如何使用 Get 方式请求数据？

2) 考虑使用另外两种方式实现网络接入。

7. 补充阅读

[1] 杨丰盛. Android 应用开发解密[M]. 北京：机械工业出版社，2010.

[2] 余志龙，陈昱勋，郑名杰，等，王世江，改编. Google Android SDK 开发范例大全[M]. 2 版. 北京：人民邮电出版社，2010.

[3] Android 开发者指南. <http://www.developer.android.com>. 2010.

5.3 网络安全通信实验

网络与信息的安全是物联网发展的重要瓶颈，也是目前物联网领域的研究者所面临的重要课题。安全问题不是简单地哪个层面的独立技术，而是贯穿于整个物联网的一条主线，安全可信的物联网网络架构是物联网发展的重要保障。这需要对各个应用的总体安全需求进行总体评估和规划，研究基于普适、异构环境下的可信物联网的安全功能和需求，以保证物联网构件多层次相互间协同运作的整体安全性和可靠性。

在骨干网络中，已经部署大量的安全机制，由于其计算、存储及网络资源相对丰富，其安全性已经相当成功。对于功能简单、网络脆弱的终端网络，在目前的物联网领域显得尤其突出。本节给出了一种基于椭圆曲线及群组密钥协商的感知网络安全实验。

5.3.1 椭圆曲线及群组密钥协商方法简介

1. 定义

给定一个域 F ，方程

$$y^2 + a_1xy + a_2 = x^3 + a_3x^2 + a_4x + a_5 \quad (a_1, a_2, a_3, a_4, a_5 \in F)$$



所确定的平面曲线外加一个无穷远点共同构成 F 上的椭圆曲线，通常用 E 表示，该方程称作 Weierstrass 方程。若 F 的特征大于 3，则通过坐标变换可将上述方程化简为

$$y^2 = x^3 + ax + b$$

2. 运算法则

设 P, Q, R 为椭圆曲线上的点， ∞ 为无穷远点， $E(F)$ 为椭圆曲线上所有点的集合，有限域 F 上的椭圆曲线满足以下运算法则：

若 $P(x, y) \in E(F)$ ，则 $P + \infty = \infty + P = P$ ， $\infty = -\infty$ ；

若 $P(x, y) \in E(F)$ ，则 $P + (-P) = \infty$ ， $-P = -(x, y) = (x, -y)$ ；

若 $P(x_1, y_1), Q(x_2, y_2) \in E(F)$ ， P, Q 连线的延长线交椭圆曲线于点 R 关于 x 轴的对称点，则 $P + Q + R = \infty$ ，即 $P + Q = -R$ ，如图 5-2 所示。

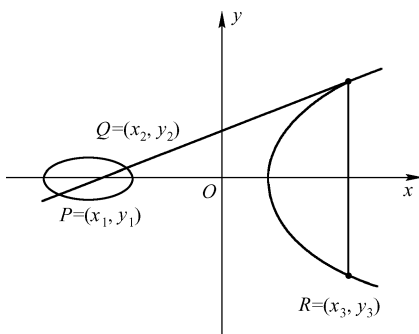


图 5-2 $P + Q = -R$

3. 椭圆曲线离散对数问题

若 $k \in \mathbf{Z}$ ， $E(F)$ 为椭圆曲线， $P(x, y) \in E(F)$ ，定义点乘 kP 如下：

当 $k > 0$ 时， $kP = P + P + \dots + P$ （共 k 个）；

当 $k < 0$ 时， $kP = (-k)(-P)$ ；

当 $k = 0$ 时， $kP = \infty$ ；

若存在最小的正整数 n 使得 $nP = \infty$ ，则称 n 为点 P 的阶。

已知 k 和 kP ，求解 k 的问题称为椭圆曲线离散对数问题（ECDLP）。当点 P 的阶为大素数时，ECDLP 的计算是不可行的（即椭圆曲线离散对数问题的难解性）。

4. 固定轮数群组密钥协商方案

在无线传感器网络中，节点簇的各节点之间的通信是一种群组通信的方式。为了保证节点簇通信的安全，在一种可认证的群组密钥协商协议基础上，结合椭圆曲线密码系统，使用基于椭圆曲线的固定轮数的群组密钥协商协议构建协商方案。该方案轮数固定，可扩展性强；轮数固定，两轮协商可得出群组密钥，简单高效。群组密钥协商方案如下：

(1) 第 1 轮协商

设有 n 个用户 U_0, U_1, \dots, U_{n-1} 参与群组密钥协商。

1) 每个用户 U_i ， $0 \leq i < n-1$ ，随机选取 $r_i \in_R \mathbf{Z}_q^*$ ，计算 $Z_i = r_i P$ ，其中， r_i 为第 i 个用户产生的随机数， P 为预先在椭圆曲线上取的一个公共点。

2) 将 Z_i 发送给用户 $U_{(i-1) \bmod n}$ 和 $U_{(i+1) \bmod n}$ 。



(2) 第2轮协商

1) 每个用户收到 $Z_{(i-1)\bmod n} = r_{(i-1)\bmod n}P$ 和 $Z_{(i+1)\bmod n} = r_{(i+1)\bmod n}P$ 后, 计算 $X_i = (Z_{(i+1)\bmod n} - Z_{(i-1)\bmod n}) \times r_i$, 并将 X_i 广播到每个用户。

2) 每个用户 U_i 计算它们的会话密钥。

3) $K_i = nr_i Z_{(i-1)\bmod n} + (n-1)X_i + (n-2)X_{(i+1)\bmod n} + L + X_{(i-2)\bmod n}$

正确性证明: 对于任意的第 i 个用户, 其计算出的密钥为

$$\begin{aligned} K_i &= nr_i Z_{(i-1)\bmod n} + (n-1)X_i + (n-2)X_{(i+1)\bmod n} + L + X_{(i-2)\bmod n} \\ &= [nr_i r_{(i-1)\bmod n} + (n-1)r_i(r_{(i+1)\bmod n} - r_{(i-1)\bmod n}) + (n-2)r_{(i+1)\bmod n}(r_{(i+2)\bmod n} - r_{(i)\bmod n}) \\ &\quad + L + r_{(i-2)\bmod n}(r_{(i-1)\bmod n} - r_{(i-3)\bmod n})]P \\ &= [r_{(i-1)\bmod n}r_i + r_i r_{(i+1)\bmod n} + L + r_{(i-3)\bmod n}r_{(i-2)\bmod n} + r_{(i-2)\bmod n}r_{(i-1)\bmod n}]P \end{aligned}$$

那么, 对于第 $i-1$ 个用户计算出来的密钥为

$$\begin{aligned} K_{(i-1)\bmod n} &= nr_{(i-1)\bmod n} Z_{(i-2)\bmod n} + (n-1)X_{(i-1)\bmod n} + (n-2)X_{(i)\bmod n} + L + X_{(i-3)\bmod n} \\ &= [nr_{(i-1)\bmod n} r_{(i-2)\bmod n} + (n-1)r_{(i-1)\bmod n}(r_{(i)\bmod n} - r_{(i-2)\bmod n}) + (n-2)r_{(i)\bmod n}(r_{(i+1)\bmod n} - r_{(i-2)\bmod n}) \\ &\quad + L + r_{(i-3)\bmod n}(r_{(i-2)\bmod n} - r_{(i-4)\bmod n})]P \\ &= [r_{(i-2)\bmod n}r_{(i-1)\bmod n} + r_{(i-1)\bmod n}r_{(i)\bmod n} + L + r_{(i-3)\bmod n}r_{(i-4)\bmod n} + r_{(i-3)\bmod n}r_{(i-2)\bmod n}]P \end{aligned}$$

显然, $K_i = K_{(i-1)\bmod n}$ 。

5.3.2 实验十八 基于椭圆曲线的群组密钥协商协议通信实验

1. 实验目的

本实验介绍了使用基于椭圆曲线的群组密钥协商协议方案实现节点群组安全通信的程序。通过实验, 使读者初步掌握无线传感网安全通信的基础知识, 并理解在传感器节点之间建立安全通信方案的原理和方法。

2. 实验设备

(1) 硬件

- 2.4GHz 增强节点 (IOT-NODE24T) 5 个。
- JTAG 编程器 1 个。
- PC (含串口) 1~5 台。

(2) 软件

- IAR Embedded Workbench Evaluation Version for MCS-51 V7.20H。
- Z-Stack for CC2430 Version 1.4.2。
- Packet Sniffer for CC2430 Version 2.7.1。
- SerialPortTerminal for Windows (based on .Net 2.0)。

3. 实验原理

本实验实现时采用 ZigBee 协议, 硬件平台是 2.4GHz 增强节点硬件 (IOT-NODE24T), 主要是由于 ZigBee 协议低复杂度、低功耗、低速率、低成本的特点, 特别适用于群组安全通信的应用环境。ZigBee 相关介绍详见第 4 章 ZigBee 网络实验相关章节。



协调器节点通过选择一个网络 ID 启动整个网络，它是整个无线传感器网络的第一个设备，在群组密钥协商中充当协商进程的调度者。路由器节点允许其他设备加入网络，负责多跳路由和协助它自己的由电池供电的子终端设备的通信，在群组密钥协商中是透明的。终端设备负责数据的采集与发送，在群组密钥协商中是真正参与协商的用户。

本实验的网络中使用了 5 个节点：1 个协调器节点，负责网络的组建以及地址表的组建；4 个终端设备节点，为参与协商的用户。实验流程如图 5-3 所示。

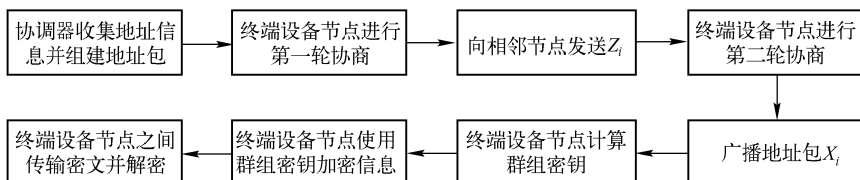


图 5-3 实验流程示意图

下面对上述主要流程分别进行介绍。

(1) 协调器收集地址信息并组建地址包

在群组密钥协商的初始化阶段，所有参与群组密钥协商的终端设备节点发送自己的网络地址给协调器。当协调器在一定的时间阈值内不再收到新的网络地址时，就将收集到的网络地址组建为地址表并打包，发送给所有参与群组密钥协商的终端设备节点，使每个终端设备节点都能够感知到群组中的个体，参与协商。

协调器每接收到一个包含地址信息的数据包，就执行以下代码：

```

//统计当前地址表中与收到地址不同的地址条目数量
//tempAddr16 为收到的 16 位网络地址，z_r_counter 初值为 0
for(z_r_i=0;z_r_i<first_r_addrTable.Total;z_r_i++)
{
    if(first_r_addrTable.Addresses[z_r_i]!=tempAddr16)
    {
        z_r_counter++;
    }
}
//若接收到新的网络地址，就将新地址插入地址表中，并将阈值计数清零
if(z_r_counter==first_r_addrTable.Total)
{
    first_r_addrTable.Addresses[first_r_addrTable.Total]=tempAddr16;
    first_r_addrTable.Total++;
    same_Total_times=0;
}
//若未接收到新的网络地址，就将阈值加 1
if(z_r_counter==first_r_addrTable.Total-1)
{
    same_Total_times++;
}
//根据阈值来控制收集地址信息的时间，阈值的大小可根据群组的大小进行调整
  
```



```
//当阈值到达时就对地址表进行规范化并广播给所有的群组密钥协商参与者
if(same_Total_times==0x05)
{
    //对地址表进行排序
    //并将地址表进行广播
}
```

(2) 终端设备节点进行第一轮协商

当终端设备节点收到协调器发送的地址包后，进入群组密钥协商第一轮，主要包括两部分：椭圆曲线的初始化以及计算点 Z_i 。

椭圆曲线的初始化主要完成椭圆曲线以及曲线上公共点的生成，根据参与协商的节点数分配点存储空间给变量*ecdh_node，通过调用函数 ecdh2_init 实现。

```
#define CURVE_M 163 //椭圆曲线位数
#define CURVE_A 7 //椭圆曲线参数 a
#define CURVE_B 6 //椭圆曲线参数 b
#define CURVE_C 3 //椭圆曲线参数 c
ECDH2 *ecdh_node ; //椭圆曲线点存储空间指针
//定义结构 ECDH2，表示椭圆曲线上的一个点
typedef struct ecdh2par
{
    int lasty[MAXNODENUM] , no , promptr;
    int nodenum ;
    big key , randnum , para;
    big Node[MAXNODENUM] ;
    epoint *tempPoint[2] ;
    miracl *mip ;
    ebrick2 binst ;
    char mem_big[MR_BIG_RESERVE(ECDN2_BIGNUM)];
    char mem_ecp[MR_ECP_RESERVE(ECDN2_ECPNUM)];
}ECDH2;

//椭圆曲线初始化
extern BOOL ecdh2_init( ECDH2 *p,mr_unsign32 seed,int no,int nodenum);
```

ecdh2_init 中的 4 个参数分别为：椭圆曲线生成的公共点、随机数种子、节点编号、参与协商节点的总数。由于无线传感器节点存储资源十分有限，公共点的存储空间需要严格控制。其申请、释放分别通过调用 ecdh2_alloc、ecdh2_free 实现。

```
//申请一个点存储空间
BOOL ecdh2_alloc( ECDH2 *p );
//释放一个点存储空间
BOOL ecdh2_free( ECDH2 *p );
```

为了充分利用每个无线传感器节点所处环境的唯一性、特殊性等物理特征，这里使用温度传感器采集的原始值作为随机数种子。

```
//获取温度传感器原始值
```



```
INT16 getTemperature(void);
```

椭圆曲线初始化成功后，每个终端设备节点分别计算点 Z_i ：

```
extern BOOL offlineCalculations( ECDH2 *p ) ;
```

`offlineCalculations` 函数产生一个随机大数 r ，然后计算 $Z_i = rp$ ，并将 Z_i 发送给它的前后节点。

(3) 终端设备节点第二轮协商

当每个终端设备节点收到其前后节点发送的点 Z_i 后，调用 `firstCalculations(ECDH2 *p)` 计算点 X_i ， $X_i = r_i(Z_{i+1} - Z_{i-1})$ ：

```
ECDH2 *ecdh_node ; //椭圆曲线点存储空间指针
extern BOOL firstCalculations( ECDH2 *p ) ;
```

节点计算出点 X_i 后，将点 X_i 广播给所有节点。

无线传感器节点携带的能源十分有限，其中数据发送消耗的能量最大。为了减少能量消耗，在第一轮和第二轮椭圆曲线上点的数据发送使用了点压缩算法。使用该算法，可以将一个点 (x,y) 压缩为 x 和一个比特，减少约 50% 的数据流量。

```
ECDH2 *ecdh_node ; //椭圆曲线点存储空间指针
//点压缩 Zi
extern BOOL setEchd2Z( ECDH2 *p , int id , char *POINT , int y ) ;
//点压缩 Xi
extern BOOL setEchd2X( ECDH2 *p , int id , char *POINT , int y ) ;
```

函数 `setEchd2Z` 和 `setEchd2X` 的参数相同。其中， p 是椭圆曲线上的点； id 是节点编号；`POINT` 存放点压缩后的 x ， y 存放点压缩的一位比特。

(4) 终端设备节点计算群组密钥

当终端设备节点收到所有其他节点发送的 X_i 后，根据 `ecdh_node` 计算群组密钥。群组密钥的计算通过调用 `secondCalculations(ECDH2 *p)` 实现。

```
ECDH2 *ecdh_node //椭圆曲线点存储空间指针
char ecdh2_key[21];
//计算群组密钥
extern BOOL secondCalculations( ECDH2 *p ) ;
```

由于将计算出来的群组密钥是椭圆曲线上的一个点，不方便直接用来加密消息。因此，将该点进行点压缩，将 x ， y 拼接为一个大数作为群组密钥。最后群组密钥保存在全局变量 `ecdh2_key[21]` 中。

(5) 使用群组密钥加密信息

得到的群组密钥 `ecdh2_key[21]` 可以用来作为各种加密算法的主密钥，这里使用 AES 加密算法，对群组密钥 `ecdh2_key[21]` 求 SHA 值，作为加密密钥。

```
shs256_init(&sh);
for ( f_r_i=0x0;f_r_i<0x15;f_r_i++)
    shs256_process(&sh,ecdh2_key[f_r_i]);
```



```
//获得最终加密密钥  
shs256_hash(&sh,sha256_key);
```

IOT- NODE24T 节点采用的主芯片为的 TI 公司的 CC2430F128 芯片。该芯片提供了支持 AES 加密算法的专用协处理器模块，以最小的 CPU 开销来实现对数据的加解密。故本实验直接调用 AES 协处理器对数据进行加解密。

调用接口 `void aesEncrypt(char * plainString, BYTE * key, int strLength)`对数据进行加密:

```
void aesEncrypt(char * plainString, BYTE * key, int strLength)  
{  
    int i;  
  
    for(i=0; i<LENGTH_KEY+LENGTH_IV; i++)  
    {  
        IV[i] = key[i];  
        key16[i]=key[++i];  
    }  
    //设置 AES 模式为 CBC, 并加载密钥  
    AES_SETMODE(CBC);  
    halAesLoadKeyOrInitVector(key16, TRUE);  
    //对明文进行加密并删除明文信息  
    halAesEncrDecr((BYTE *)plainString, strLength,  
                   (BYTE *)plainString, IV, ENCRYPT);  
}
```

(6) 解密

调用接口 `void aesDecrypt(char * cipherString, BYTE * key, int strLength)`对数据进行解密:

```
void aesDecrypt(char * cipherString, BYTE * key, int strLength)  
{  
    int i;  
    for(i=0; i<LENGTH_KEY+LENGTH_IV; i++)  
    {  
        IV[i] = key[i];  
        key16[i]=key[++i];  
    }  
    // 设置 AES 模式为 CBC, 并加载密钥  
    AES_SETMODE(CBC);  
    halAesLoadKeyOrInitVector(key16, TRUE);  
    //对密文进行解密并删除密文信息  
    halAesEncrDecr((BYTE *)cipherString, strLength,  
                   (BYTE *)cipherString, IV, DECRYPT);  
}
```

4. 实验内容

初步掌握椭圆曲线基础知识，理解基于椭圆曲线离散对数难解问题。



理解群组密钥协商协议基本原理，理解群组密钥协商协议解决群组安全通信的原理。

了解无线传感器网络环境下实现加密与解密的方法，特别是算法针对无线传感器网络资源受限所作的优化。

5. 实验步骤

1) 启动 IAR Workbench 环境，编写实验代码，对协调器节点和终端设备节点的实验代码分别进行编译。编译成功后，将相应代码分别下载到协调器节点和终端设备节点。

2) 将上位机软件 SerialPortTerminal.exe (需要.NET 2.0 支持) 复制到 PC 端，将两个或者所有终端设备节点通过串口连接到 PC (可连接至同一台 PC，也可每个节点各自连接一台 PC)。

3) 启动 SerialPortTerminal.exe，选择对应的 COM 口，波特率设置为 9600bit/s，校验位设置为 None，数据位设置为 8 位，停止位设置为 1 位，数据模式设置为 Text，根据实际情况输入节点的个数。

4) 将协调器节点和终端设备节点全部复位。启动协调器节点，开始组建 ZigBee 网络，等待红灯常亮，表示组网完成。

5) 向背向主芯片一侧扳动协调器节点的摇杆，当观察到协调器节点黄灯亮时，在 10s 内迅速向背向主芯片一侧扳动所有终端设备节点的摇杆，将终端设备节点加入群组当中，启动群组密钥协商。观察终端设备节点红灯闪烁情况，并在 SerialPortTerminal 程序的信息区观察群组密钥的协商结果，协商结果的含义见表 5-2。

表 5-2 协商结果的含义

协商结果项目	含 义
KEY	群组密钥协商得到的群组密钥
Total time	当前节点群组密钥协商时间开销 (格式为时: 分: 秒)
Scaler multiplication time (ms)	当前节点单次标量乘时间开销 (单位为毫秒)

6) 在协商结果出现后，上位机软件会弹出提示框，询问是否选择当前终端设备节点发送消息。任选一个终端设备节点单击 “Yes” 按钮作为消息发送者，其他终端设备节点单击 “No” 按钮作为消息接收者，并在消息发送者节点对应的 SerialPortTerminal 程序中输入明文 (暂不支持中文)，并单击 “发送” 按钮。观察传感器节点之间的安全通信情况，包括消息发送者对明文进行加密后回发给上位机软件的密文，以及消息接受者接收密文后解密得到的明文。

6. 思考题

- 1) 认真观察网络中协调器的数据包，说明群组密钥协商的驱动模式。
- 2) 本实验采用的群组密钥协商协议存在什么缺点？
- 3) 如何增加参与协商的节点的个数，统计群组密钥协商的时间？说明群组密钥协商的时间与参与协商的节点数的关系。

7. 补充阅读

[1] CC2430 A True System-on-Chip solution for 2.4 GHz IEEE 802.15.4.

[2] 祝烈煌，曹元大，廖乐健.一种可认证的群组密钥协商协议[J].计算机工程与应用，2006,S1:108-109.



[3] Burmester M, Desmedt Y. A Secure and Efficient Conference Key Distribution System. *Advances in Cryptology-EUROCRYPT'94*. Berlin; Springer, 1995:275-286.

[4] Z-Stack OS Abstraction Layer (OSAL) Application Programming Interface F8W-2003-0002. Texas Instruments, 2005.

[5] Z-Stack Network (NWK) Application Programming Interface F8W-2003-0008. Texas Instruments, 2005.

[6] Z-Stack Application Framework (AF) Application Programming Interface F8W-2003-0010. Texas Instruments, 2005.

[7] Z-Stack Application Support Sub-Layer (APS) Application Programming Interface F8W-2003-0025. Texas Instruments, 2005.

[8] Z-Stack Device Object (ZDO) Application Programming Interface F8W-2004-0009. Texas Instruments, 2005.

5.4 室内实时定位实验

物联网定位技术在交通运输、现代物流、旅游娱乐、工农业生产、公共安全、军事国防、智能家居和智慧医院等领域有着广泛的应用前景。目前，卫星定位技术已在交通运输、大地测绘、军事国防等部门得到了普遍使用，并且正向个人移动终端上快速渗透，有望成为手机等移动终端的标准配置。在制造业和物流行业，采用定位技术对生产人员、生产设备、制成品和运输工具进行实时追踪，可有效提高生产调度效率，减少浪费。在安全问题极其重要的煤矿、监狱、医院、学校、危险作业场所部署实时定位系统，可以随时掌握有关人员及关键设备的位置信息，紧急情况下能大大提高搜救效率。

近年来，随着 WiFi (Wireless Fidelity) 网络技术的迅速发展和移动终端的日益普及，利用 WiFi 网络对人员和物品进行定位跟踪正成为热点。WiFi 定位技术具有成本低、部署方便等优点，非常适合于办公室、车间、学校和医院等室内环境的应用。在这些环境下，采用卫星定位技术的全球定位系统 (Global Positioning System, GPS) 信号受到阻隔难以有效定位，WiFi 定位技术正好可以弥补卫星定位技术的不足。从 20 世纪末开始，WiFi 定位技术就一直受到广泛关注。微软公司早在 1999 年就研发了基于 WiFi 网络的定位系统 RADAR。它采用射频指纹 (Radio Map 或 Fingerprint) 匹配技术对办公室内的移动接入设备进行定位。目前已有少量外国公司，如 Ekahau、AeroScout 等，推出了基于 WiFi 网络的低功耗定位系统，定位精度一般在 3m 左右。国内不少科研机构，如中国科学院计算技术研究所、武汉大学等，也已经开展了 WiFi 定位技术相关的研究工作。

5.4.1 WiFi 网络协议简介

WiFi 网络协议主要包括 802.11 系列协议簇，与 IEEE 802.15.4 系列协议相比，它的最大优点是传输速率较高，通信距离相对较远，同时兼容已有的各种 802.11 DSSS 设备。802.11 系列协议规范了无线局域网 (Wireless Local Area Network, WLAN) 的媒体访问控制 (Medium Access Control, MAC) 层及物理 (PHYSical, PHY) 层。由于无线传输的方式不同，IEEE 802.11 在统一的 MAC 层下面规范了各种不同的物理层协议。下面简要介绍几个



与 WiFi 定位相关的协议。

IEEE 802.11b 是所有无线局域网标准中最著名、普及最早的标准，它支持 1~11Mbit/s 的数据传输速率，其载波频率为 2.4GHz，共有 14 个频宽为 22MHz 的频道可供使用。802.11b 的 MAC 层使用载波监听多址访问/冲突避免（Carrier Sense Multiple Access/Collision Avoidance, CSMA/CA）协议，物理层则采用了红外线、跳频（Frequency-Hopping Spread Spectrum, FHSS）和直扩（Direct-Sequence Spread Spectrum, DSSS）3 种不同的物理介质。802.11g 采用了与 802.11b 协议相同的载波频率，最大传输速率为 54Mbit/s（与 802.11a 相同）。802.11g 使用 补码键控（Complementary Code Keying, CCK）技术与 802.11b 协议兼容，同时又采用 正交频分复用（Orthogonal Frequency Division Multiplexing, OFDM）技术支持高达 54Mbit/s 的数据流。802.11b 是所有 WLAN 标准演进的基石，未来系统大都需要兼容 802.11b。

802.11a（又称 WiFi5）工作在 5.8GHz 频段，采用正交频分复用调制方式，最高物理层的数据传输速率可达 54Mbit/s。802.11a 是一个非全球性的标准，并不兼容 802.11b 协议，不过同频段系统之间的干扰较小，比较适合具有高速率传输需求的数据、多媒体通信业务。为了弥补 802.11a 与 WiFi 兼容性上的不足，人们采用了双频通信技术（802.11a+g 双模），工作在 2.4GHz 和 5GHz 两个频段，遵循 802.11a/b/g 标准，实现了与 802.11b 的后向兼容。

802.11n 协议使用 多入多出（Multi-In, Multi-Out, MIMO）和双频带（2.4 GHz 和 5 GHz）技术，最大数据传输速率高达 600Mbit/s，同时兼容 IEEE 802.11b/g 设备。802.11n 对 802.11 标准的单一 MAC 层协议进行了优化，增加了净负载所占比重，减少管理检错所占字节数，大大提升了网络吞吐量。802.11n 通过构建多组独立天线阵列，动态调整波束方向，保证用户能接收到稳定的信号，并减少其他噪声信号干扰，使无线网络的传输距离增加到几公里，移动性大大增强。

5.4.2 实验十九 基于射频指纹匹配的室内实时定位实验

1. 实验目的

了解基于 WiFi 网络的物联网定位系统的工作原理，熟悉基于射频指纹匹配定位技术的工作过程以及实现方法。

2. 实验设备

（1）硬件

- 低功耗 WiFi 实验硬件（IOT-WIFI）1~2 个。
- 无线路由器 6 个左右。
- 推车（最好高架、非金属）1 辆。
- 米尺 1 卷。

（2）软件

- WiMap 定位平台软件 1 套。

3. 实验原理

（1）定位算法及实现

目前，WiFi 定位技术大都通过测量接入点（Access Point, AP）或定位标签（Tag）发射的射频信号强度（Received Signal Strength Indication, RSSI）方法进行定位。由于射频信



号受环境变化、人员移动等因素影响较大，具有较强的时变特性和随机特性，因此，使用接收到的射频信号强度估计待定位目标与信标间的距离误差较大，因此现有的 WiFi 定位系统一般不采用基于 RSSI 测距的三边或多边定位方法，而是采用射频指纹匹配技术实现目标定位。基于射频指纹定位技术的基本原理是，由多个信标（即 AP）发射的射频信号强度 RSSI 矢量构成一个射频指纹，目标区域中每个位置与在该位置上采集的射频指纹对应，在建立起射频指纹与位置的一一对应关系之后，便可以使用射频指纹来确定目标位置。基于射频指纹的节点定位过程分为离线训练和在线定位两个阶段。

离线训练阶段，系统需要在待定位区域的不同位置大量采集相应位置的射频指纹，并保存在指纹数据库中。射频指纹库反映了位置空间与信号空间的直接映射关系，可有效避免复杂室内环境下使用接收信号强度与距离的不精确映射关系引入的定位误差。在构建射频指纹库时，一般根据定位区域的地理分布特点，将定位空间分割为若干离散网格，每个网格与该位置的 RSSI 特征向量相关联（该特征向量由多个接收到的信号强度值构成）。

上述离线训练阶段采样射频指纹的过程代码如下：

```
/*@usage:采样过程主循环*/
while(!quit)
{
    if(udpSocket.waitForReadyRead())
    {
        QDateTime now = QDateTime::currentDateTime();//采样时间

        //接收数据
        datagram.resize(udpSocket.pendingDatagramSize());
        udpSocket.readDatagram(datagram.data(), datagram.size());

        //解析
        RSSMeasurement measure;
        QString mac;//节点 MAC 地址
        Location location;//采样位置
        DataConverter::resolveData(datagram,measure,mac,location);

        //保存新观测时间项并获得 measure_no 主键
        measure_no=measureTimeDB.insertMeasureTime(now,mac.toStdString());

        //保存观测数据到数据库
        measureDBThread->save("train",measure_no,measure);

        //保存采样位置到数据库
        locationDBThread->save("train",measure_no,location);
    }
}
```

在线定位阶段，定位标签实时测量由多个信标发射的射频信号（或者多个信标测量由标签发射的射频信号强度），构建射频指纹，并与训练阶段采集的射频指纹库进行匹配计算，



选择相似度最大的位置作为待定位标签的最终位置估计。

根据 RSSI 特征的不同, 基于射频指纹识别定位方法又进一步细分为确定型定位和基于概率定位两类。前者在构建射频指纹库时, 每个网格位置仅记录接收到的对应信标 (AP) 信号强度均值, 在线定位时一般采用最近邻等方法进行定位; 后者在构建射频指纹库时, 一般基于每个网格的信号分布先验假设 (如高斯分布), 计算该分布的统计特征 (如均值和方差), 在线定位时采用贝叶斯分类器等统计方法进行目标位置估计。本实验采用概率型定位算法, 以获得较高的定位精度。

在线定位过程的主要代码如下:

```
Location positioning(RSSMeasurement & current_measurement,
                    Location* priori_location)
{
    double max_poster = -1;
    double prior=-1;
    double likelihood;
    double poster;

    Location result;

    Location location;
    vector<RSSMeasurement> measurement;

    TrainingDataSet training_dataset = mMAPLocatorModel->getTrainingDataSet();

    set<LabeledData> dataset = trainingDataSet.getDataSet();
    set<LabeledData>::const_iterator dataset_iter;
    for (dataset_iter = dataset.begin(); dataset_iter != dataset.end();
        dataset_iter++)
    {
        location= dataset_iter->getLocation();
        //计算先验概率
        prior=priori_calculate(priori_location, location);

        //计算观测似然概率
        measurement = dataSetIter->getRssVector();
        likelihood=likelihood_calculate(current_measurement,measurement);

        //计算后验概率
        poster= posteriori_calculate(prior,likelihood);

        //定位结果为后验概率最大的位置
        if(poster>max_poster)
        {
            max_poster=poster;
            result= location;
        }
    }
}
```



```
    }  
  }  
  return result;  
}
```

其中，参数 `current_measurement` 为当前观测数据；`priori_location` 为上一时刻定位结果。

在定位过程中，对于每一个位置，首先计算上一个位置与指纹库中的每一个指纹的先验概率，然后计算当前观测值与该预测位置对应指纹的似然概率，最后计算上一个位置与似然值的后验概率。计算完所有指纹库中的指纹的后验概率之后，将后验概率最大值对应的位置作为当前位置的估计值。基于概率定位方法把待定位目标所处位置看成一个分布，每一次定位时都使用上一时刻目标位置的概率发布（即先验概率），可有效避免由于较大观测噪声（影响似然函数）带来的定位风险，从而获得较高的定位精度。

定位过程使用了3个比较核心的函数，下面分别进行介绍。

`priori_calculate` 函数根据历史位置计算当前位置的先验概率，两个参数 `currentLocation` 和 `prioriLocation` 分别为当前时刻的预测目标所在位置和前一时刻目标所在位置，其实现代码如下：

```
double priori_calculate(Location * prioriLocation, Location *currentLocation)  
{  
    if (prioriLocation == NULL)          //前一位置为空，返回均匀分布概率  
    {  
        double locationSize;  
        locationSize = (double)this->trainingDataSet.getLocationSize();  
        return 1/locationSize;  
    }  
    else    //根据马尔科夫转移概率计算先验概率  
    {  
        //neighbors 前一位置 prioriLocation 的邻居位置集合  
        vector<Location> neighbors = neighborSearch(*prioriLocation);  
        vector<Location>::const_iterator iterNeighbor =  
            find(neighbors.begin(),neighbors.end(),*currentLocation);  
  
        //当前估计位置在上一位置的邻域内，则当前估计位置的先验概率为上一位置的邻域的均匀分布  
        if (iterNeighbor != neighbors.end())  
        {  
            return 1/((double)neighbors.size());  
        }  
        else    //当前估计位置不在上一位置的邻域内，则先验概率为 0  
        {  
            return 0;  
        }  
    }  
}
```

`likelihood_calculate` 函数计算当前时刻观测值与指定射频指纹数据的似然概率，两个参数



measure 和 fingerprint 分别为当前时刻的观测值和待判定的射频指纹数据。相应的代码如下：

```
double likelihood_calculate (RSSMeasurement & measure,
                             vector<RSSMeasurement> & fingerPrint)
{
    map<string,float> measureMap = measure.getRssMap();//当前观测向量
    map<string,float>::iterator measureIter;

    map<string,double> likelihoodMap;//保存临时计算结果，元素为<MAC,似然概率>
    map<string,double>::iterator likelihoodIter;

    vector<RSSMeasurement>::const_iterator fingerPrintIter;

    RSSMeasurement aFingerPrint;
    pair<string,float> aFingerPrintMeasure;

    string currentMac;
    float currentRSSI, fingerPrintRSSI;
    double likelihood = 1.0;

    //初始化 likelihoodMap
    for (measureIter = measureMap.begin();
         measureIter != measureMap.end();measureIter++)
    {
        likelihoodMap.insert(
            map<string,float>::value_type(measureIter->first,0.0)) ;
    }

    //计算观测 measure 与对应某个位置上的所有指纹 fingerPrint 之间的似然
    for (fingerPrintIter = fingerPrint.begin();
         fingerPrintIter != fingerPrint.end();fingerPrintIter++)
    {
        //取出一个指纹
        aFingerPrint = *fingerPrintIter;
        for (measureIter = measureMap.begin();
             measureIter != measureMap.end();measureIter++)
        {
            currentMac = measureIter->first;
            currentRSSI = measureIter->second;
            aFingerPrintMeasure = aFingerPrint.getRSS(currentMac);

            fingerPrintRSSI = aFingerPrintMeasure.second;
            //计算观测与指纹中对应 Mac 的观测间的高斯核函数值
            likelihoodMap[currentMac] +=
                this->calculateKernel(currentRSSI,fingerPrintRSSI);
        }
    }
}
```



```

for (likelihoodIter = likelihoodMap.begin();
    likelihoodIter != likelihoodMap.end();likelihoodIter++)
{
    //假设不同无线路由器的观测相互独立
    likelihood *= (likelihoodIter->second / fingerprint.size());
}
return likelihood;
}

```

posteriori_calculate 通过先验概率 prior 和似然概率 likelihood 计算后验概率。相关代码如下:

```

double posteriori_calculate(double prior, double likelihood)
{
    return prior*likelihood;
}

```

总的来讲,采用射频指纹定位方法比采用基于射频信号强度测距的定位方法可获得更高的定位精度,但由于射频信号动态变化较大,采用射频指纹定位方法能获得的定位精度仅在 3m 左右,难以满足高精度的定位应用需求。不过,通过增加接入点数目,以及对射频信号衰落进行补偿,一定程度上可提高 WiFi 定位技术精度。

(2) 定位实验系统框架简介

本实验定位软件利用 WLAN 基础设施实现物联网环境的目标定位。

该实验系统主要提供以下功能。

- 定位系统创建,允许同时建立多个定位系统。
- 室内地图绘制,支持多楼宇、多楼层的三维定位。
- 训练数据采集及训练。
- 传感器目标位置监测,实时显示当前系统中所有传感器节点的位置,并提供轨迹监测和目标锁定功能。
- 环境中无线路由器信息查询和位置设定。
- 支持多目标实时定位跟踪。

系统物理架构如图 5-4 定位实验系统架构示意图所示,整个框架包括定位系统、定位服务器和客户端 3 个部分。其中定位系统包括待定位的传感器节点,若干无线路由器,传感网节点接收无线路由器信号,并通过网络传送到定位服务器;定位服务器负责定位节点,同时与数据库连接,完成相关数据的记录;客户端则提供室内地图绘制,节点监测等功能。下面对各组件进行详细介绍。



图 5-4 定位实验系统架构示意图

1) 定位系统。实验所采用的 WiFi 传感器节点是基于 GS1011 芯片设计的,其主要功能是定时对环境中的无线路由器进行扫描,并测量无线路由器返回信号的信号强度,最后将各信号强度打包发送给服务器端已进行后续工



作。传感器节点采用周期性工作方式，即节点定期启动，完成工作后立即进入休眠状态。此时，CPU 和传感器皆停止工作，只有实时时钟（Real-Time Clock, RTC）处于工作状态。节点大部分时间处于休眠状态以达到满足低功耗的目的。

无线路由器在实验中的主要作用是当收到 WiFi 传感器节点的扫描信号后，为指纹识别定位提供所需的信号强度。其中一个指定的无线路由器作为网关，具有为节点动态分配 IP 地址以及将节点发送的数据转发给定位服务器的功能。

2) 定位服务器。定位服务器包含了定位引擎、采样服务器、地图服务器以及数据库，这些组件可分别部署于不同的主机。在本实验中为了减少设备数目，将它们部署于同一主机。各组件功能如下：地图服务器提供地图数据管理，以支持客户端的地图绘制和管理功能；采样服务器负责从客户端接收训练数据并持久化到数据库；作为定位服务器最重要的构成部分，定位引擎并行处理多个客户端的定位请求，并向已订阅位置服务的客户端发布传感器节点的位置信息。

3) 客户端。客户端直接与定位服务器联系，通过配置客户端可与不同的定位服务器相连。为了配置方便，采用图形界面，不仅可以直观地配置楼层、接入点（无线路由等），而且可实现地图编辑、显示，以及定位结果的可视化显示。楼宇配置可以添加、移除楼层，并设置楼层高度等参数，接入点配置以表格形式列出环境中的接入点信息，并支持对接入点位置等信息的编辑更新。地图编辑模块支持添加地图图片、设置比例尺、绘制网格等操作，并可通过显示选项卡配置显示模式，如是否以轨迹形式显示目标，或是否显示环境中的接入点等。

4. 实验内容

了解基于射频指纹定位方法的预备知识，包括贝叶斯估计等概率统计理论、定位原理、传感器节点无线收发过程以及无线通信相关基础知识。掌握定位系统软件安装及硬件部署以及定位过程，包括采样、训练及定位等。

5. 实验步骤

(1) 启动 WiMap

按照光盘提供的文档安装和配置 MySQL 和 WiMap.rar，进入 WiMap 安装目录，双击 WiMap.exe 启动 WiMap 程序。

(2) 项目创建或打开已有项目

如果需要创建项目，选择文件→新建→项目。

如果已经建有项目，可以通过“文件→打开”打开已有项目。在“选择楼宇”对话框中选择对应的项目即可。

(3) 采样

建议采用分时多次的方式进行采样工作。

1) 设置采样数。单击“定位”选项卡中的“采样”，在“采样参数配置”中设置每个网格的采样个数，如 60。

2) 放置传感器节点。启动传感器节点，将其置于待采样的网格处，在地图中单击该采样位置，此时该网格会改变颜色，当采样结束时颜色恢复。移动节点位置，单击对应地图网格继续采样，如此反复。注意：在节点处于采样位置后才可以单击待采样网格，开始该位置的训练数据收集。



(4) 训练

单击“定位”选项卡中的“训练”。

(5) 定位

1) 单击“定位”选项卡中的“定位”，开启传感器节点。此时地图显示出定位结果。

2) 显示轨迹。在“显示”选项卡中勾选“轨迹”复选框，此时会显示节点的轨迹。

6. 思考题

1) 为了提高定位精度，无线路由器部署应遵循什么策略？

2) 请指出3个影响定位精度的系统或环境因素。

3) 请给出您的采样方式，并说明这样做的理由。

7. 补充阅读

[1] 陈菁菁. Tomcat 原理与 Java Web 系统开发[M]. 北京: 清华大学出版社, 2009.

[2] GS1011 Ultra Low-Power Wireless System-On-Chip Datasheet. GainSpan Corporation. 2008.

[3] Youssef M, Agrawala A. The Horus Location Determination System[J]. Wireless Networks, 2008,14(3):357-374.

[4] Bahl P, Padmanabhan V N. RADAR: An In-building RF-based User Location and Tracking System[A]. Proc of the IEEE INFOCOM 2000. Tel Aviv, Israel, 2000:775-784.

[5] 林以明, 罗海勇, 李锦涛, 赵方. 基于动态 Radio Map 的粒子滤波室内无线定位算法[J]. 计算机研究与发展. 2011, 48(1):139-146.

[6] 赵方, 罗海勇, 林权, 马严. 基于核函数法及马尔可夫链的节点定位技术[J]. 通信学报, 2010,31(11):195-204.

[7] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band. IEEE Std 802.11b-1999.

[8] GS1011 Peripheral User Manual. GainSpan Corporation. 2008.

[9] GainSpan Embedded Platform Software Application Programming Guide V2.0[R]. GainSpan Corporation. 2008.

5.5 自组网时间同步实验

时间同步是所有分布式系统都需要解决的一个重要问题。在集中式系统中，由于任何进程或模块都可以从系统唯一的全局时钟中获取时间，因此系统内任何两个事件都有着明确的先后关系。而在分布式系统中，由于物理上的分散性，系统无法为彼此间相互独立的模块提供一个统一的全局时钟，而是由各个进程或模块各自维护它们的本地时钟。由于这些本地时钟的计时速率、运行环境存在不一致性，因此即使所有的本地时钟在某一时刻都被校准，一段时间后，这些本地时钟间也会出现失步。为了让这些本地时钟再次达到相同的时间值，必须进行时间同步。时间同步就是通过对本地时钟的某些操作，达到为分布式系统提供一个统一时间标度的过程。本节简单介绍时间同步的基础知识并结合实验进行说明。



5.5.1 无线传感器网络时间同步简介

时间同步的研究由来已久，在物联网和传感器网络中都是重要的支撑技术，有许多成熟的方法被成功地应用于解决这一问题，有代表性的解决方法有 NTP 和 GPS。NTP 协议是目前因特网上时间同步协议的标准，用于把因特网上计算机的时间同步于 UTC 时间。NTP 采用层状结构的同步拓扑，每一层均有若干时间服务器，如顶层时间服务器，第二层时间服务器等，其他均为客户机。顶层时间服务器通过广播、卫星等方式与 UTC 同步；其他层的时间服务器可选择若干个上一层时间服务器及本层时间服务器作为同步源来实现与 UTC 时间的间接同步；客户机则可通过指定一个或多个上一层时间服务器来实现与 UTC 的同步。可以看出：NTP 协议的可靠性依赖于时间服务器的冗余性和时间获取路径的多样性。

NTP 协议并不能直接适应于无线传感器网络。首先，这两种网络的节点具有很大差异：同以网络服务器为代表的因特网节点相比，无线传感器网络节点具有体积、电能供应、计算能力和存储空间的严格约束，导致复杂的、高精度的 NTP 协议不可能在无线传感器网络节点上运行；其次，无线传感器网络采用无线传输方式，而因特网却主要采用可靠得多的有线传输方式，这两种传输方式在带宽、抗干扰能力和抗衰落能力等方面具有巨大的差别，使得 NTP 针对有线传输方式特点而对参数所做的一些工程优化并不适合于无线传输方式；再次，无线传感器网络的应用要求算法具有强的局部性，即达到局部最优性，而因特网则强调整体最优性。所以，普遍认为 NTP 并不适合于无线传感器网络。

GPS 是由美国国防部为满足军事部门对海陆空设施进行高精度导航和定位的需要而建立的。GPS 系统由 3 部分构成，分别为空间星座部分、地面监控部分和用户设备部分。空间部分是由若干颗 GPS 工作卫星所组成的，每颗卫星上装置有精密的铷、铯原子钟，并由监控站经常进行校准，达到和 UTC 时间的同步。每颗卫星不断发射包含其位置和精确到十亿分之一秒的时间的数字无线电信号用于接收设备的时间校准。GPS 接收装置接收到来自于 4 颗卫星的信号，根据伪距测量定位方法不仅可以计算出其在地球上的位置，而且也可计算出 GPS 接收机时间与 UTC 时间之偏差，并进行时间校准，达到与 UTC 时间的同步。这种方法的同步精度可达 100ns。

由于 GPS 信号的穿透性差，GPS 天线必须安装在空旷的室外，并且要求附近没有高大的遮挡物，这是 GPS 应用的一个主要限制。此外，GPS 接收机的成本、体积和功耗也较大，这也阻碍了 GPS 在无线传感器网络上的应用。

综上所述，现有的成熟的时间同步方法无法直接应用解决无线传感器网络的时间同步问题，这是由于无线传感器网络具有以下自身的特点。

(1) 传输延迟的不确定性

报文传输延迟的不确定性是对无线传感器网络时间同步的主要挑战之一。一方面，传输延迟比要求的时间同步的精度大得多；另一方面，它极易受到处理器负载、网络负载等因素的影响。通常，报文的传输延迟可分为发送时间、访问时间、传送时间、传播时间、接收时间、接受时间。传输延迟的不确定性严重影响了同步精度，因此需要对传输延迟仔细地测量、分析和补偿才能设计出高精度的时间同步协议。

(2) 低功耗、低成本与小体积

低功耗、低成本与小体积的要求对无线传感器网络软硬件设计的各个方面均提出挑战。



无线传感器网络强调低功耗原则，在设计时间同步软硬件时必须遵循该原则。例如：对用于时间同步的硬件来说，类似于 GPS 接收机这样的高耗能、高成本设备是不合适的；对时间同步软件来说，虽然增大同步操作的频率能够提高同步精度，但必然伴随着同步功耗的增加，因此完全依靠增大同步操作频率来提高同步精度的方法是不合适的。低成本和小体积更是加剧了电能供应的紧张趋势。

(3) 可扩展性

无线传感器网络时间同步协议会随着网络规模的扩大而出现同步精度的劣化现象，即同步误差随着网络规模的扩大而增长，并最终导致同步误差的越界。网络规模的扩大还会引起时间同步协议其他方面性能的下降，甚至出现不能正常工作的情况。如何设计时间同步协议，使得其在大规模网络中不仅能正常工作，而且能保持较好的性能，即具有强的可扩展性。这是大规模无线传感器网络时间同步需要考虑的主要问题之一。

(4) 健壮性

尽管 NTP 会遇到短暂的链路失败，但 NTP 仍然能正常工作。这是因为 NTP 被手动配置了多个时间服务器，因此具有较强的健壮性。在无线传感器网络中，节点的移动、故障及外界环境的变化等多种因素都会导致无线传感器网络的高度动态性。静态配置方案并不能应对网络的高度动态性。时间同步协议必须能够对这些情况进行处理，以保证系统的健壮性。

5.5.2 实验二十 TPSN时间同步实验

1. 实验目的

本实验介绍了使用 TinyOS 实现传感器网络时间同步协议，(Timing-sync Protocol for Sensor Networks, TPSN)，并对两个无线传感器网络节点进行同步。通过实验，使读者初步掌握无线传感器网络时间同步问题的特点和解决方法，并锻炼其使用 TinyOS 开发实用的物联网应用程序的能力。

2. 实验设备

(1) 硬件

- 2.4GHz 节点 (IOT-NODE24) 2 个。
- JTAG 编程器 1 个。
- PC (含串口) 1 台。

(2) 软件

- TinyOS 2.x。
- WinAVR20050214。
- AVRStudio4.18。

3. 实验原理

TPSN 是一种充分借鉴了互联网同步协议 NTP 的无线传感器网络时间同步协议。该协议通过一对包含了节点时间信息的请求和应答报文对网络中的两个节点进行同步。其主要过程如图 5-5 所示。

图中的 T_1 、 T_4 用节点 A 的本地时间记录， T_2 、 T_3 用节点 B 的本地时间记录。节点 A 记录下当前时刻 T_1 ，然后向

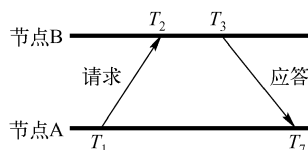


图 5-5 TPSN 协议的主要过程



节点B发送一个同步请求报文，节点B在接收到该报文后，记录下接收到时刻 T_2 ，并立即向节点A返回一个同步应答报文，把 T_2 和该报文的发送时刻 T_3 嵌入在报文中，当节点A接收到该报文时，记录下接收到时刻 T_4 。令 Δ 为当节点A的本地时刻为 T_1 时，节点A和B之间的时偏。由于 $T_1 \sim T_4$ 时间比较短，可认为当节点A的本地时刻为 T_4 时，其与节点B之间的时偏没有变化。假设报文的传输延迟相同，均为 d 。由

$$T_2 = T_1 + \Delta + d, \quad T_4 = T_3 - \Delta + d$$

可得：

$$\Delta = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}, \quad d = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

因此在 T_4 时刻，若在节点A的本地时间上增加修正量 Δ ，就达到和节点B之间的瞬时的时间同步。

由于涉及对计数器的精确操作，本实验未使用 TinyOS 提供的通用计数器 `TimerMilliC`，而用了 ATmega128 的 3 号计数器 `HplAtm128Timer3C`。本实验的 IOT-NODE24 节点采用频率为 7.3728MHz 的晶振。由于使用的是一个 16 位的计数器，为防止计数器过快溢出，实验中对计数器进行了 1024 分频，将其频率降低到 7200Hz。这样，一个计数周期最大可以持续 9.10s。当然，9.10s 对于实验来说还是不够的，因此实验中又引入了一个 16 位整型变量 `high16Bit`，表示计数值的高 16 位，与计数器一起构成一个 32 位的计数器。

另外，通过设置比较器 `HplAtm128Timer3C.Compare` 来调整同步操作的间隔，也即设置每隔 `SYNC_INTV` 个计数周期比较器溢出一次，进行一次同步，并闪烁红色 LED 灯。通过观察红灯的闪烁情况便可以观察节点是否达到同步。本实验中 `SYNC_INTV` 设置为 36000，即 5s 同步一次。

与计数器的初始化相关的代码如下：

```
task void InitAndStartTimer()
{
    post setWGM(); //设置 WGM 为 0100
    call Compare.set(SYNC_INTV); //设置比较器的值，即计数器溢出值
    call Compare.start(); //打开比较器的中断
    call Timer.set(0); //计数器清零
    call Timer.setScale(ATM128_CLK16_DIVIDE_1024); //设置分频并启动计数器
}
```

其中 `setWGM()` 为设置 Waveform Generation Mode 为 0100，具体参考 ATmega128 数据手册的表 61。相关的代码如下：

```
task void setWGM()
{
    Atm128TimerCtrlCompare_t wgm10;
    Atm128TimerCtrlCapture_t wgm32;

    wgm10 = (call TimerCtrl.getCtrlCompare());
```



```
wgm10.bits.wgm10 = 0;
call TimerCtrl.setCtrlCompare(wgm10);

wgm32 = (call TimerCtrl.getCtrlCapture());
wgm32.bits.wgm32 = 1;
call TimerCtrl.setCtrlCapture(wgm32);
}
```

同步过程是通过数据包交换完成的，交互过程包含了一个同步请求数据包和一个同步应答数据包。其定义如下：

```
//同步请求数据包
typedef nx_struct syncReqMsg
{
    nx_uint32_t T1;
} syncReqMsg;

//同步应答数据包
typedef nx_struct syncAckMsg
{
    nx_uint32_t T1;
    nx_uint32_t T2;
    nx_uint32_t T3;
} syncAckMsg;
```

在实验中，应答数据包中也包含了时间 T_1 ，并由节点B返回，这样使得节点A不需要保存与之对应的 T_1 ，并预防由于 T_1 与其他时间不对应而产生的问题。

程序运行过程如下：首先，设置两个节点，主节点A和从节点B。每当主节点比较器溢出时，记录下当前时间 T_1 ，并发送同步请求数据包。相关代码如下：

```
if (!radioBusy)
{
    syncReqMsg* btrpkt = (syncReqMsg*)
        (call RadioAMSend.getPayload(&pkt, sizeof(syncReqMsg)));

    //获得当前时间并保存到T1中
    atomic btrpkt->T1 = high16Bit * SYNC_INTV + ((call Timer.get()));

    //发送数据包
    if (call RadioAMSend.send(AM_BROADCAST_ADDR, &pkt,
        sizeof(syncReqMsg)) == SUCCESS)
    {
        radioBusy = TRUE;
    }
}
```



从节点收到同步请求数据包之后，立即记录接收到数据包的时间 T_2 ，并发送应答数据包。

```
//获得当前时间并保存在 T2 中
atomic T2 = high16Bit * SYNC_INTV + ((call Timer.get()));
//从数据包中获得 T1，并保存到全局变量中，以便将其发送回去
if (len == sizeof(syncReqMsg))
{
    syncReqMsg* btrpkt = (syncReqMsg*)payload;
    T1 = btrpkt->T1;
}
//发送数据包
post RadioSend();
```

在发送过程中，获得当前时间 T_3 ，并连同 T_1 和 T_2 一并发回。

```
syncAckMsg* btrpkt = (syncAckMsg*)
    (call RadioAMSend.getPayload(&pkt, sizeof(syncAckMsg)));

//将保存的 T1 和 T2 写入数据包
btrpkt->T1 = T1;
btrpkt->T2 = T2;

//获得当前时间 T3，并写入数据包
atomic btrpkt->T3 = high16Bit * SYNC_INTV + ((call Timer.get()));

//发回应答数据包
if (call RadioAMSend.send(AM_BROADCAST_ADDR, &pkt,
    sizeof(syncAckMsg)) == SUCCESS)
{
    radioBusy = TRUE;
}
```

主节点在收到该数据包之后，立即记录下接收时间 T_4 ，并计算时间偏移。

```
atomic T4 = high16Bit * SYNC_INTV + ((call Timer.get())); //获得接收时间T4
if (len == sizeof(syncAckMsg))
{
    syncAckMsg* btrpkt = (syncAckMsg*)payload;

    //获得其他 3 个时间点
    T1 = btrpkt->T1;
    T2 = btrpkt->T2;
    T3 = btrpkt->T3;

    //为了计算方便，将原计算时间偏移的公式转换为((T2+T3)-(T1+T4))/2
    //这里分别计算两组的和，然后根据时间偏移的正负分别进行补偿
    T23 = T2 + T3;
```



```

T14 = T1 + T4;

if (T23 > T14)                                     //时间偏移为正
{
    delta = (T23 - T14) / 2;                       //计算偏移大小
    high = delta / SYNC_INTV;                     //计算补偿值的高 16 位
    low = delta % SYNC_INTV;                      //计算补偿值的低 16 位
    low16Bit = (call Timer.get());                //获得当前时间

    atomic high16Bit += high;                     //对高 16 位进行补偿
    call Timer.stop();                            //停止计数器
    if (low16Bit + low > SYNC_INTV)               //如果补偿后的低 16 位超过溢出值
        atomic high16Bit++;                      //向高位进位
    low16Bit = (low16Bit + low) % SYNC_INTV;      //计算低 16 位的校准值
    call Timer.set(low16Bit);                    //设置计数器的计数值
    call Timer.start();                          //重新启动计数器
}
else                                               //如果偏移为负
{
    delta = (T14 - T23) / 2;                       //计算偏移大小
    high = delta / SYNC_INTV;                     //计算补偿值的高 16 位
    low = delta % SYNC_INTV;                      //计算补偿值的低 16 位
    low16Bit = (call Timer.get());                //获得当前时间

    atomic high16Bit -= high;                     //对高 16 位进行补偿
    call Timer.stop();                            //停止计数器
    if (low16Bit < low)                          //如果补偿后的低 16 位小于 0
        atomic high16Bit--;                      //向高位借位
    low16Bit = (low16Bit + SYNC_INTV - low)%SYNC_INTV;//计算低 16 位的校准值
    call Timer.set(low16Bit);                    //设置计数器的计数值
    call Timer.start();                          //重新启动计数器
}
}
}

```

4. 实验内容

阅读 TPSN 的相关论文，学习协议的原理以及运行机制。

阅读 ATmega128 的数据手册，了解其 16 位计数器的工作原理以及使用方法。

掌握 TinyOS 程序设计方法，了解 NesC 的编程过程以及程序结构，能够编写 TinyOS 程序。

对 IOT-NODE24 节点编程、运行过程进行实际的操作，了解物联网前端感知节点的工作机理。

5. 实验步骤

1) 编写实验代码，用 WinAVR 的 Programmers Notepad 打开 TPSN.h，检查 TPSN.h 中是否有 MASTER 的宏定义（即代码#define MASTER）。启动 Cygwin 环境，在实验代码所在文件夹下使用“make micaz”进行编译。编译通过之后，在实验代码文件夹的 build\micaz 子文件夹下会生成 main.ihex 文件，将其文件名改为 A.ihex。



2) 再次用 WinAVR 的 Programmers Notepad 打开 TPSN.h, 将 TPSN.h 中的 MASTER 宏定义注释掉 (即修改为 `//#define MASTER`), 并保存文件。然后使用 “make micaz” 进行编译, 编译通过之后, 在实验代码文件夹的 build\micaz 子文件夹下会生成 main.hex 文件, 将其文件名改为 B.hex。

3) 将节点与 PC 相连, 并使用 AvrStudio 将前两步编译得到的 .hex 文件分别烧写到两个节点中去。

4) 启动主节点 A, 可以看到主节点上的红灯, 每隔 5s 闪烁一次。

5) 启动从节点 B, 可以看到主节点的红灯经过调整与从节点的红灯同步闪烁。

6. 思考题

1) 为什么要对节点的时钟频率进行分频, 如果不分频或者进行 8 分频会有什么后果?

2) 为什么要使用比较器, 直接使用计数器可以吗?

3) 如何实现 3 个节点乃至多个节点的同步?

7. 补充阅读

[1] Saurabh G, Ram K, Mani B S. Timing-sync Protocol for Sensor Networks. Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, Los Angeles, California, USA. ACM, 2003: 138-149.

[2] 李晓维, 徐勇军. 无线传感器网络技术[M]. 北京: 北京理工大学出版社, 2007.

[3] Philip Levis, David Gay. TinyOS Programming. Cambridge: Cambridge University Press, 2009.

5.6 数据索引及搜索实验

海量感知数据的分类、索引和搜索是物联网的云计算平台中的核心技术, 也是物联网几乎所有规模化应用及服务的基础。本节以全文信息检索为例, 简单介绍海量数据搜索相关基础知识, 并以实验来进一步说明, 该技术可以全面应用于互联网和未来的物联网中。

5.6.1 基于倒排索引的全文搜索技术简介

文本是互联网和物联网中的重要组成部分, 海量的文本信息一般是存储在专用的数据库系统中进行统一管理。数据库管理系统存储的一般都是结构化数据, 以方便数值的计算、查询、统计与挖掘。但随着计算机存储与计算能力的迅猛发展, 越来越多的非结构化文本数据存储在数据库中, 数据库中的文本搜索日益广泛。

当前, 数据库中的文本搜索, 一般采用 SQL 语句中的 Like 操作符或者采用数据库系统自带的全文索引功能。Like 操作往往特别耗时, 数据规模超过 10 万条, 查询往往会导致网络连接超时, 无法满足在线搜索的需要; 同时, Like 查询仅仅是简单的字符串匹配, 没有考虑语言语义。例如, 检索“和服”, 同样会命中“产品和服务”。当前, 数据库内置的全文索引功能、速度和效果同样存在类似缺陷。因此, 在数据库系统基础上, 需要对非结构化文本字段构建智能的搜索引擎。

一个好的搜索引擎结构应该满足以下几个目标。

1) 良好的检索效果。系统以各种方式接受用户的查询请求, 给出令人满意的检索结



果，给用户最想要的内容。这一点是最重要的，需要优先考虑。

2) 快速的响应效率。对于搜索引擎系统来说，在保证检索效果的同时，效率是非常关键的，应该让客户感觉到无延时地返回结果。

3) 系统规模的可扩展性。众所周知，现在信息量的膨胀速度十分惊人，要想应对它的发展，必须在系统结构上作出努力，使得系统可以在几年内容纳合理数量的记录及其各种索引。

4) 合理的系统性能。信息数量飞速增长，因此系统容纳的吞吐量可能越来越多，但是，系统必须保证在有限的运行资源下取得理想的效果。

5) 系统结构最好便于修改。这是因为技术不断进步，今天看来很先进的技术，也许两年后便不再先进了。搜索引擎系统也需要不断地更新，加入对新技术的支持。要想做到系统结构的灵活性，最好的方法是保持系统各部分之间的相对独立。

6) 可以保存日常的使用情况。便于进行日志的分析和挖掘，了解用户的需求和数据对用户价值。

基于倒排索引的全文搜索技术较好地实现了上述目标，是目前应用广泛的索引和搜索方法。这种方法的基本原理如图 5-6 所示。

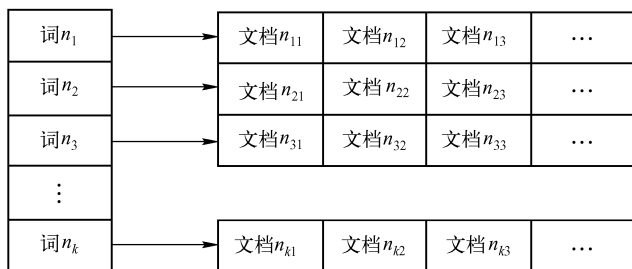


图 5-6 基于倒排索引的全文搜索技术的基本原理

从根本上说，倒排文件的基本结构是一个词表，每个表项是系统可以识别的一个词汇。从每个表项引出一个指针，指向一个数据结构，此结构记录了该词汇在所有文本中的出现位置和频率等信息。

倒排文件的生成过程是这样的：取得一个文本记录之后，将其顺序扫描，用一个分词词典对它进行分词，记录下每一个词的出现位置和频率，存入倒排表。合并所有文本产生的倒排表（当然，要记录下来词汇出现的记录），就得到倒排文件。

倒排文件方法有以下突出的优点。

1) 实现简单。因为原理很简单，所以几乎所有的工作都集中在数据结构的设计和实际编码上。一个好的数据结构是非常重要的，它会影响到系统的诸多方面，最重要的是：

- 查找速度。好的数据结构会使查询时需要检索的数据最小化，这样可以大大减少 IO 时间，提高程序运行效率。
- 存储空间。好的数据结构需要考虑极大数据量情况下的磁盘存储问题。页面的索引与页面需要的存储空间是同一量级的。因此，对于一个在大规模应用领域使用的搜索引擎来说，数据结构是非常重要的。
- 可维护性。倒排文件的结构是很难维护的，但设计良好的数据结构可以在一定程度上缓解这个难题。



2) 查找速度快。倒排表的结构是直接适应查询请求的。对于只含关键字的简单查询,当用户输入查找关键字时,系统要做的工作就是分析关键字,从倒排表中查找相应的关键词,然后把它们对应的文本记录信息提取出来进行合并输出。即使文本库中包含的文本数量达到百万级,这个过程仍然是相当快的,一般在几毫秒钟之内就会得到结果。如果系统的查询界面支持逻辑操作符和其他操作符,这个过程会复杂一些,主要是增加了中间结果的合并操作,但相对简单查询来说,增加的复杂度仍然是线性的。

5.6.2 实验二十一 数据库全文精准搜索实验

1. 实验目的

了解全文搜索引擎的工作原理,并初步掌握全文搜索引擎的实现方法,培养开发实用的搜索应用程序的能力。

2. 实验设备

(1) 硬件

- PC 1 台。

(2) 软件

- Tomcat 服务软件 1 套。
- MySQL 数据库 1 套。
- 全文搜索引擎 DBTSearch 1 套。
- 组织机构数据库 OrgInfo 1 套。

3. 实验原理

全文搜索引擎 DBTSearch 的系统框架主要包括系统核心层和应用层,系统构架如图 5-7 所示。

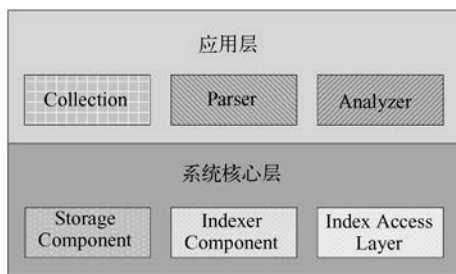


图 5-7 DBTSearch 系统架构

系统核心层包括存储组件 (Storage Component)、索引组件 (Indexer Component) 和索引访问组件 (Index Access Layer) 3 个部分。存储组件实现索引的压缩存储,存储的位置可以是本地磁盘,内存或机群等,具体的存储位置和方案对于上面的索引层是透明的。索引组件实现文档倒排,这一层又可以扩展多种不同数据类型字段的索引器。例如,对于中文文本,可以采用 Word Indexer;而对于和中文文本大不相同的 URL,则可以采用 URL Indexer,这个索引器利用了 URL 的特征能实现更高效的索引,用户也可以扩展自己的索引器。索引访问组件架构在索引组件之上,实现对索引的读写、检索功能等。这一层包含 3 部分:索引检索 (Index Searcher)、索引生成 (Index Writer) 和索引读取 (Index Reader)。索



引检索组件检索已经创建好的索引；索引生成组件输入原始文档，生成倒排索引；索引读取组件读取索引的相关信息，包括索引文档遍历、索引词检索等。

在系统核心层之上是应用层，包含文档集抽象组件（Collection）、文档格式解析器组件（Parser）和文档内容分析器组件（Analyzer）组件。文档集抽象组件从待索引的文档集中抽取出单个文档，输入到解析器组件进行文档格式解析；分析器组件则对解析器解析后的文档内容进行分析（分词、去停用词和词根还原等），将文本的字符串序列转换成数字编号的词（Term ID）序列，输入到系统核心部分进行索引。分析器组件同时也应用在检索部分，将检索词转换成词编号。

核心层的存储组件、索引组件以及应用层的文档集抽象组件、文档格式解析器组件、文档内容分析器组件都可以单独扩展、升级或替换。其中，文档格式解析器组件和文档内容分析器组件可以通过 COM 插件扩展。

（1）索引算法及实现

索引采用目前广泛使用的向量空间模型（Vector Space Model）和 TF_IDF 标引策略。主要步骤是对文本记录进行分词、去除停用词、词根还原（Stemming）等简单的预处理之后，对文本建立倒排表。主文档中的每一个词（预处理之后保留的）都在倒排表中有对应的入口词项。每个入口词都指向一个属性项链表，链表中每一项记录该入口词在主文档中的出现情况，如偏移量、（英文）大小写等信息。建立索引后，对索引文件进行压缩优化，然后进行保存。上述索引过程的实现代码如下：

```
//初始化分析对象
CParser* pParser = new CPlainParser();

//初始化中文解析对象
CAnalyzer* pAnalyzer = new CChineseAnalyzer(pParser);

//初始化索引对象
CIndexWriter* pWriter = new CIndexWriter("indexname",pAnalyzer);

//初始化文档收集对象
CCollection* pDir = new CDirCollection("docPath",pWriter);

//扫描文档集，建立倒排索引
pDir->scan();
delete pDir;
```

（2）检索算法及实现

检索过程是根据用户的查询请求，到页面库/知识库中查找相关记录，再把查询结果返回给用户。在返回的结果中，要根据记录的相关度和重要性对结果进行排序，把质量最好的结果排在检索界面的前面。系统首先对查询请求进行分析，包括分词、去除停用词、词根还原等预处理过程，然后在索引文件中找到相关的结果记录，然后进行综合性的排序和整合。上述检索过程的实现代码如下：

```
//初始化分析对象
```



```
CQuery* pQuery = CQueryParser::Parse("北京市海淀区",
                                     "content", new CChineseAnalyzer);

//初始化查询对象
CSearch* pSearch = new CIndexSearch("indexname");

//查询
CQueryHits *pHits = pSearch->search(pQuery);

//取得查询结果
pHits->startIterator();
while(pHits->hasNext())
    CDocument* pDoc = pHits->doc();

//使用 pDoc
.....

//资源释放
delete pQuery;
delete pSearch;
delete pHits;
delete pDoc;
```

4. 实验内容

了解搜索引擎的基本知识，包括索引的建立、搜索的过程。掌握 MySQL 数据库的安装与运行、搜索引擎的搭建、搜索结果检查。

5. 实验步骤

- 1) 启动 MySQL 数据库，并使组织机构数据库 OrgInfo 顺利运行。
- 2) 词表建立（该步骤可忽略，只针对定制需求，可以直接采用原有的 PDAT 文件。）
命令格式：

```
LJIndexer.exe 1 【词表文本文件名】【PDAT 词典存储文件名】
```

例如：

```
LJIndexer.exe 1 WordList.txt dictionary.pdat
```

词表每行即一个词项。

注：斜体字为输出文件名，下同。

- 3) 词表数据导入，便于调试输出（该步骤可忽略，只针对定制需求，可以直接采用原有的 WordList 文件）

命令格式：

```
LJIndexer.exe 2 【词表文本文件名】【WordList 存储文件名】【PDAT 词典文件名】
```

- 4) 数据库字段信息列表导入。



命令格式:

```
LJIndexer.exe 3 【数据库字段信息文本文件】【数据库字段数据存储文件名】
                【数据库字段数据导出文件名】
LJIndexer.exe 3 FieldInfo.txt FieldInfo.dat FieldInfo.export_txt
```

FieldInfo.txt 内容示例见表 5-3。

表 5-3 FieldInfo.txt 内容示例

序号	字段名称	索引名称	数据类型	是否需要索引	是否需要过滤使用	是否需要存储	是否为通配符检索域	解释
1	Id	id	int	0	0	1	0	ID
2	Name	name	text	1	0	0	1	机构名称
3	Address	address	text	1	0	0	1	地址
4	Zipcode	zipcode	int	0	0	1	0	邮编

相关内容说明如下:

- 字段名称为数据库中的实际字段名称, 必须一致, 否则无法访问数据库。
- 索引名称为后续检索的字段名称, 必须和提交搜索的名称一致, 否则解析错误, 无法正确搜索。
- 数据类型支持数据库的常见格式, 其中要求非字符型字段不用有索引, 只用于过滤或者存储。
- 是否需要索引, 只针对文本字段。
- 是否需要过滤使用, 可针对所有类型字段, 一般文本索引即可, 无需过滤使用。
- 是否需要存储, 存储在文件中, 方便搜索结果显示该字段。
- 是否为通配符索引域是指在不指定字段名称搜索时, 是否覆盖该字段。

5) 建立索引。

命令格式:

```
LJIndexer.exe 4 【数据库字段信息数据文件】【索引文件系列名称】【PDAT 词典文件名】
                【WordList 存储文件名】
```

例如:

```
LJIndexer.exe 4 dbConf.xml orginfo field.dat dictionary.pdat dictionary.wordlist
```

其中 dbConf.xml 配置信息如下:

```
<DatabaseConf>
<DBConnectionString>Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=E:\LJSearch\db1.mdb</DBConnectionString>#数据库连接字符串
#Provider=SQLOLEDB; Server=127.0.0.1;Database=LOS; uid=sa; pwd=sa;
<Table>[View_I3SStd]</Table>#数据库表名[LOS].[dbo].
<SortField>id</SortField>#排序字段
</DatabaseConf>
```



其中:

DBConnectString 为连接数据库的字符串, 各个不同的数据库系统的字符串都不太一致, 如

```
ACCESS: Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\LJSearch\db1.mdb
SQL Server: Provider=SQLOLEDB; Server=127.0.0.1;Database=LOS; uid=sa; pwd=sa;
```

Table 为需要建立全文搜索的表名;

SortField 为搜索结果排序字段名称 (在本例在, 是按照 id 来排序的)。

6) 构建全文搜索服务。

执行 LJSearchServer.exe 即可。

其中主要依据 SearchServer.xml 进行配置, 配置文件如下:

```
<?xml version="1.0" encoding="GB2312"?>
<SearchServerConf>
<ReleaseVersion version="1.0" lastModify="2009-12-30" />
<Config>
    <ServerIP>127.0.0.1</ServerIP>#检索服务器 IP 地址
    <Port>8002</Port>#检索服务进程的端口号
    <Standard>E:\LJServer\dat\orginfo</Standard>#原始标准数据及索引结果文件
    <FieldInfo>E:\LJServer\dat\field.dat</FieldInfo>#字段信息文件
    <PDAT>E:\LJServer\dat\dictionary.pdat</PDAT>#PDAT 文件名
    <WordList>E:\LJServer\dat\dictionary.wordlist
        </WordList>#WordList 文件名
</Config>
</SearchServerConf>
```

其中:

ServerIP 为检索服务 IP 地址;

Standard 为索引生成的数据路径名称。

7) 检索对比测试。

启动检索测试程序 SearchAgent.exe, 输入检索语句, 语句格式为:

```
[FIELD] fieldname/* [AND] value1 value2 value 3
```

其中:

Fieldname 为索引字段名称; 也可以给出通配符 *, 表示在认定的字段范围内搜索; 可以多个字段并联, 实现关联搜索。例如:

```
[FIELD] name [AND] 北京 银行
```

SQL 语句查询为:

```
select * from yq_web_docinfo where companyname like '%北京%银行%';
```

查看返回结果的数量和内容, DBTSearch 的结果如下:



```
<?xml version="1.0" encoding="gb2312" standalone="yes" ?>
<LJSearch-Result>
  <Result-Number>
    <Total-Number>29873</Total-Number>
    <Return-Number>10</Return-Number>
    <Start-Position>0</Start-Position>
  </Result-Number>
  <Result>
    <Document>
      <id>370942</id>
      <name>北京银行中关村支行</name>
      <address>北京市中关村科学院南路 7 号</address>
    </Document>
  </Result>
</LJSearch-Result>
```

6. 思考题

- 1) 为什么全文搜索引擎 DBTSearch 能够显著提高数据库全文搜索的效率?
- 2) 如何应用 DBTSearch 开发复杂的物联网服务应用程序?
- 3) 如何实现多个服务器集群的分布式检索?

7. 补充阅读

- [1] 灵玖 DBTSearch 数据库全文搜索引擎白皮书, 灵玖软件, 2010.
- [2] D.Hawking,N.Craswell,and P.Thistlewaite.Overview of the TREC-7 very large collection track.In Proceedings of TREC-7, 1998:40-52
- [3] William Frakes, Ricardo Baeza-Yates. Information Retrieval: Data Structures & Algorithms [M]. New Jersey: Prentice Hall Press, 1992.
- [4] Sergey Melnik, Sriram Raghavan, Beverly Yang. Hector Garcia-Molina Building a Distributed Full-Text Index for the Web[C]. In 10th International World Wide Web Conference, May 2-5, 2001: 396-406.
- [5] Eric W Brown, James P Callan, W Bruce Croft. Fast Incremental Indexing for Full-Text Information Retrieval[J]. In Proc. of 20th Intl. Conf. on Very Large Databases, September, 1994: 192-202.

第6章 物联网演示实验

在实验室里进行物联网技术实验，约束因素较多，实验项目往往受到所占场地、所需费用的制约，一般只能对单项关键技术进行实验，对系统级的物联网应用进行实验则相对困难。本章给出 5 个典型物联网应用的演示系统，内容涉及环境监测、出入管理、智能家居、智能医疗和智能抄表。这些系统级演示实验，更贴近于实际应用，同时也能够反映出前面各章节所包含的基础实验的关键技术，最重要的是，这些演示实验规模适中，剪表性强，适合在实验室内操作，有利于建立全局的系统的整体的概念。

本章共分 5 部分，分别对应 5 个演示实验。每个演示实验都给出了系统的概述、结构、组成、流程、特点、实际操作过程等一般性介绍，配以光盘上的应用程序，供实验室的师生或相关技术人员搭建演示系统，进行学习、研究与开发。进一步的具体行业领域应用系统，可参见本书第 7 章的应用示范系统和本系列丛书的《物联网技术应用》一书。

6.1 演示一 环境监测演示实验

1. 实验目的

在学习和理解传感器基本原理和无线传感器网络基本框架的基础上，本演示实验增加了气象、大气参数监测等多种传感器模块，构成实时环境状况监测系统。通过实验演示，有助于学生更好地理解各种气体监测模块的工作原理，以及环境监测系统的基本工作方式。

2. 实验设备

(1) 硬件（环境监测实训套件）

- 天气基本信息监测模块 1 块。
- 一氧化碳监测模块 1 块。
- 二氧化碳监测模块 1 块。
- 一氧化氮监测模块 1 块。
- 臭氧监测模块 1 块。
- VOC(有机刺激性气体)监测模块 1 块。
- 无线传感器网络开发套件（含 9 个基本节点和 1 个 USB-ISP 下载模块）1 套。
- PC 1 台。

(2) 软件

- AVR Studio 4.18。
- TinyOS 2.x。
- 串口调试助手。

3. 实验原理

基于物联网的环境监测系统包括天气基本信息监测模块、一氧化碳监测模块、二氧化碳监



测模块、二氧化氮监测模块、臭氧监测模块、有机刺激性气体监测模块等，可以监测影响人体健康的各种有害气体含量，并及时给予用户相应提示。这些监测模块将采集到的数据传输到无线传感器网络节点，网络节点通过处理模块进行相关处理，最后通过射频模块进行无线传输，将数据汇聚到用户的互联网手机上，或通过 USB 接口直接传输到 PC 上，形成一个完整的网络。其结构框图如图 6-1 所示。

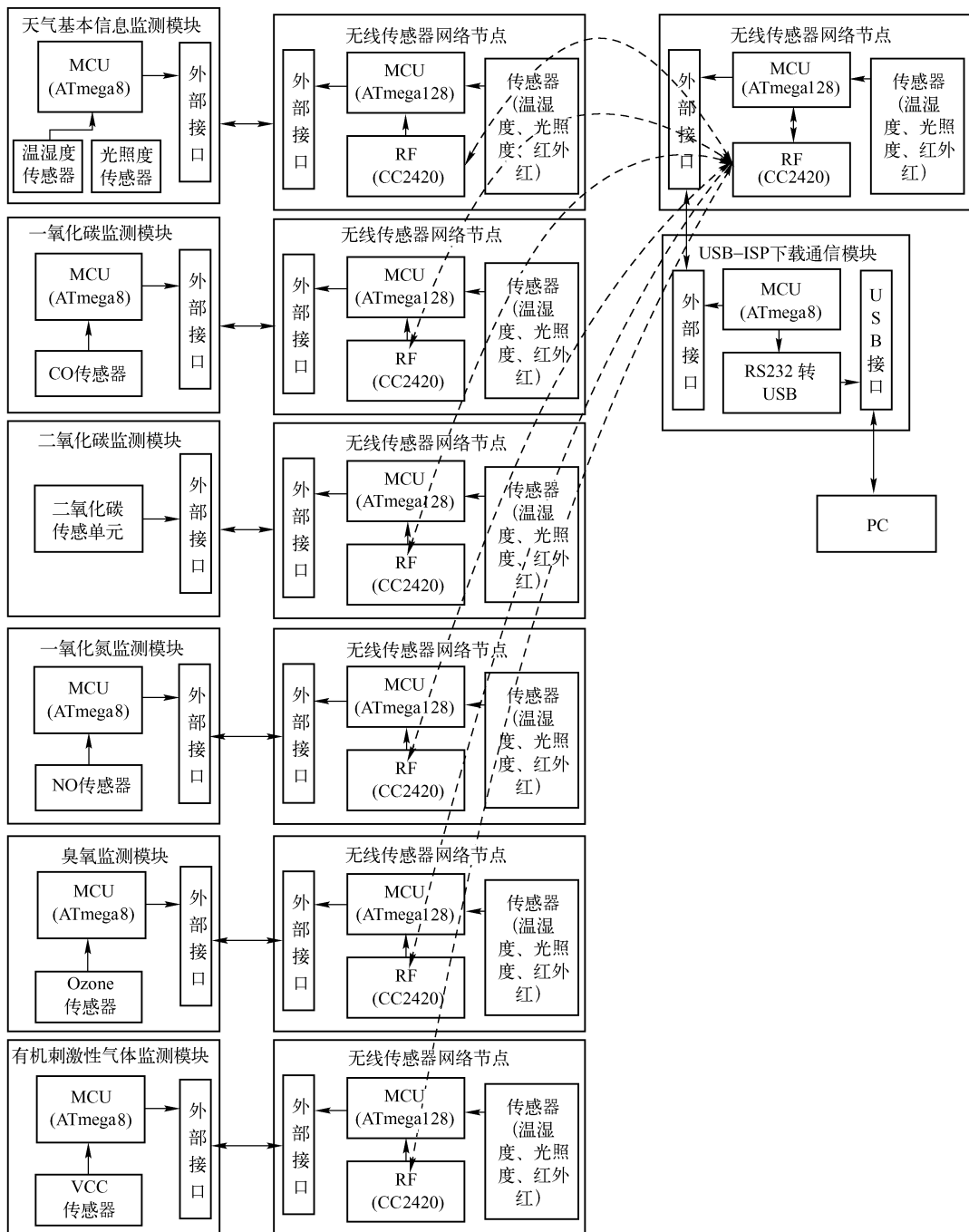


图 6-1 基于物联网的环境监测系统结构框图



(1) 原理及程序

本系统共涉及 6 个气体监测模块。下面以有机刺激性气体监测模块的工作过程为例说明原理及程序。无线传感器节点每 512ms 向传感器模块请求传感器数据，得到数据后通过无线的形式将传感器数据发送到与 PC 连接的 0 号节点上。0 号节点接收到数据后，通过串行通信将数据传送到 PC 上，整个过程如图 6-2 所示。其中，USN Mote1 代表负责采集数据的无线传感器节点；USN Mote0 代表与 PC 相连的 0 号节点。

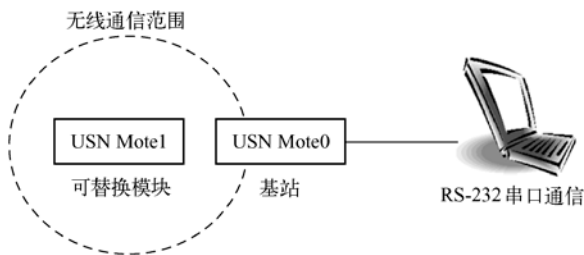


图 6-2 环境监测实验的整个过程

(2) 基站

连接 PC 的无线传感器网络节点运行基站 (BaseStation) 的程序。基站一方面将通过无线方式接收到的数据传送到 PC 中，另一方面将从 PC 中收到的数据通过无线的方式进行发送。基站的程序可在 OptModuleEX/BaseStation 文件夹中找到，在这里不详细分析。需要注意的是，基站的 GroupID 须与与其连接的传感器节点的 GroupID 相同。

(3) 实验代码

无线传感器网络节点的编程基于 TinyOS 2.x，LIB_Header 和 LIB_Interaction 两个文件夹配置了需用到的组件，OptCO 文件夹中的文件 描述了一氧化碳监测模块的具体功能实现，OptCO2 文件夹中的文件 描述了二氧化碳监测模块的具体功能实现，OptNO 文件夹中的文件 描述了一氧化氮监测模块的具体功能实现，OptOZONE 文件夹中的文件 描述了臭氧监测模块的具体功能实现，Opt VOC 文件夹中的文件 描述了有机刺激性气体监测模块的具体功能实现，BaseStation 文件夹中的文件 描述了 BaseStation 模块的具体功能实现。

4. 实验内容

(1) 构建开发环境

安装 TinyOS 系统、Cygwin、AVR Studio，构建无线传感器网络的开发环境。学习如何进行基于 TinyOS 系统的编程、编译、下载的过程。

(2) 理解、运行例题程序

理解运行在 TinyOS 下的 nesC 程序的特性，理解整个演示系统的构建过程及例题程序。掌握程序的下载、运行、调试方法。

(3) 修改程序并搭建系统

能够基于例题程序，设计出新程序，构建环境监测演示系统。

5. 实验步骤

(1) 安装 TinyOS2 及 AVR Studio

开始实际的演示之前，确保已经在 PC 上安装了基本的应用程序，主要包括 Cygwin 开发包和 AVR Studio，这一过程将有助于学生的开发过程。通常，TinyOS 作为针对传感器网



络开发的操作系统，需要运行在 Linux 环境下。然而，为了开发方便，本演示实验将通过 Cygwin 开发包在 Windows 下模拟 TinyOS 在 Linux 下的操作。除此之外，还需要安装交叉编译环境所需的其他工具。

(2) 安装例题程序

安装所有例题程序，进行程序编译。

已经安装的 Cygwin 开发包允许用户在 Windows 操作系统中编译 TinyOS 应用程序。运行 Cygwin 程序，进入 TinyOS 文件夹，可以查看各种无线传感器网络程序，其操作界面如图 6-3 所示。

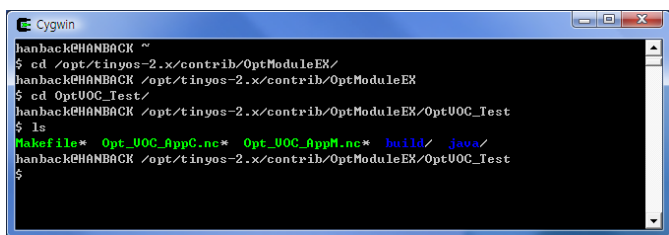


图 6-3 Cygwin 操作界面

为了把用户已生成的程序下载到传感器节点硬件中，需要用 ISP 程序。在 Cygwin 环境中，对传感器节点程序进行交叉编译后，通过 ISP 程序把编译好的结果下载到 Zigbex 节点中。运行光盘的 Program/ISP 文件夹中的 Setup.exe 程序，则 VCP 驱动将被安装。这个驱动使用 AVR 的 ISP 功能，可以将程序安装到实际节点中。

(3) 程序下载

为了下载 Zigbex 程序，需要 AVR Studio 和 USB-ISP。在使用 USB-ISP 之前，进行如下检查。

将 USB-ISP 连接到 PC 端口，通过以下步骤验证 USB-ISP 的 USB 串行端口(COM 号)：

Control panel→System→Hardware's Device Manager→Port (COM and LPT)

Port 后的数字代表当前 USB 的串行端口号。如果这个数字大于 8，AVR Studio 将不能识别该端口，需要重置。

在设置 USB 串行端口的界面中，可以通过选择“Port Settings”改变 USB 串行端口号，进入该界面后单击“Advanced”按钮，如图 6-4 所示。

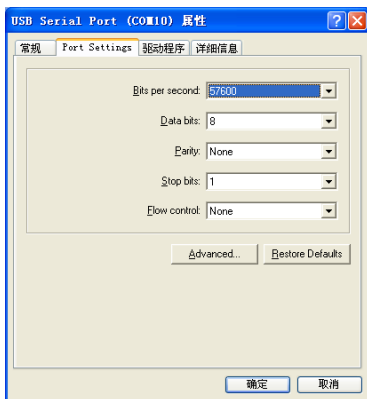


图 6-4 串行端口设置界面



如图 6-5 所示，可以重新设置 COM 端口数值使其小于“8”。

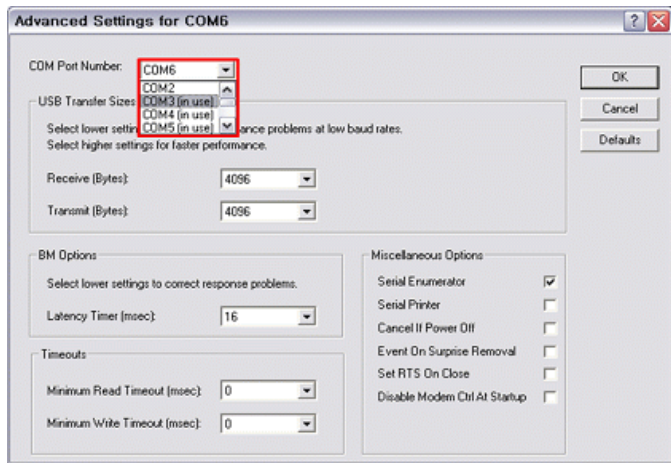


图 6-5 串行端口数值设置界面

一旦 USB-ISP 配置完毕，可以通过如下过程将程序下载到 Zigbex。

连接 USB-ISP 到 Zigbex，然后使用 USB 延长线将 USB-ISP 连接到 PC。USB-ISP 的 S1 和 S6 开关必须置于“SPI”，如图 6-6 所示（如果使用 ISP-USB 下载程序将开关置于“SPI”；如果使用串行通信功能将开关置于“UART”）。

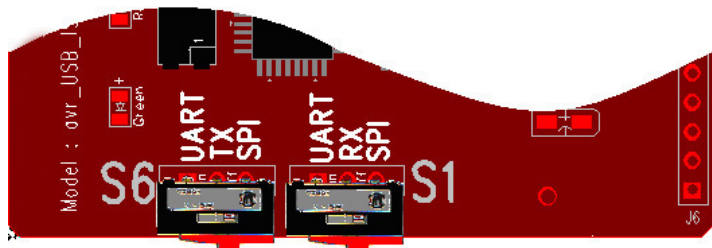


图 6-6 ISP 设置

运行 AVR Studio，然后单击“Auto Connect”连接到 USB，就会出现如图 6-7 所示的信息框。

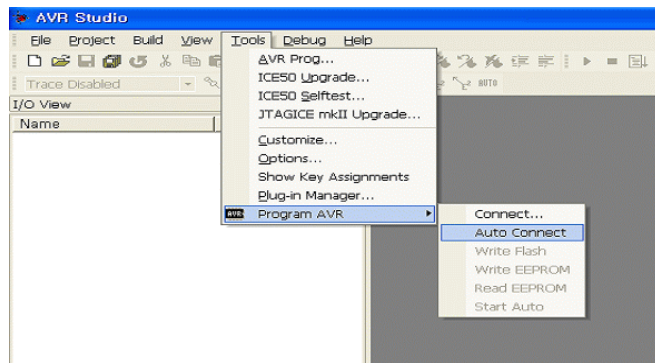


图 6-7 连接设置界面信息框



在图 6-8 所示界面中单击“Cancel”，进入图 6-9 所示的下载界面，选择下载所需文件，将其下载到 Zigbex 节点中。

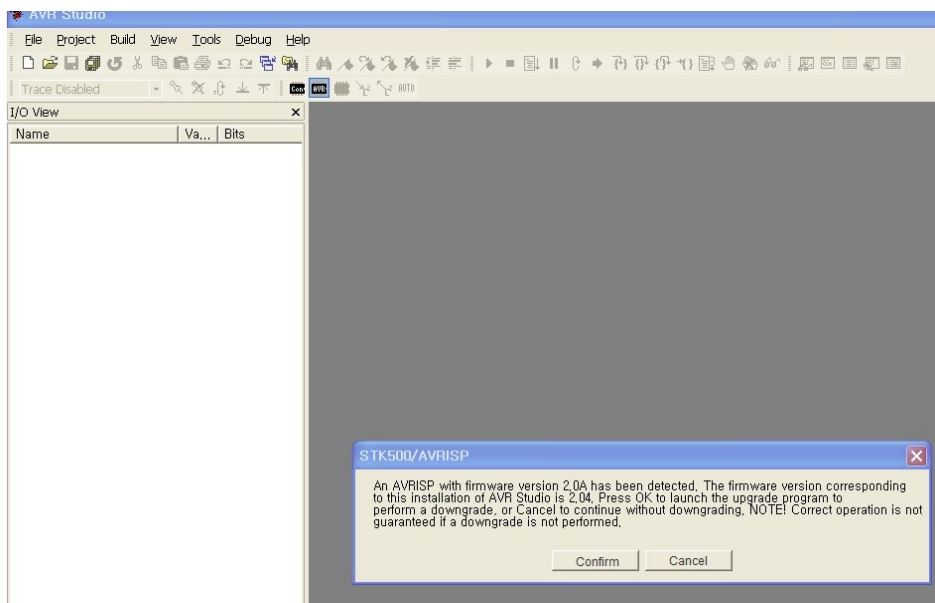


图 6-8 信息框界面

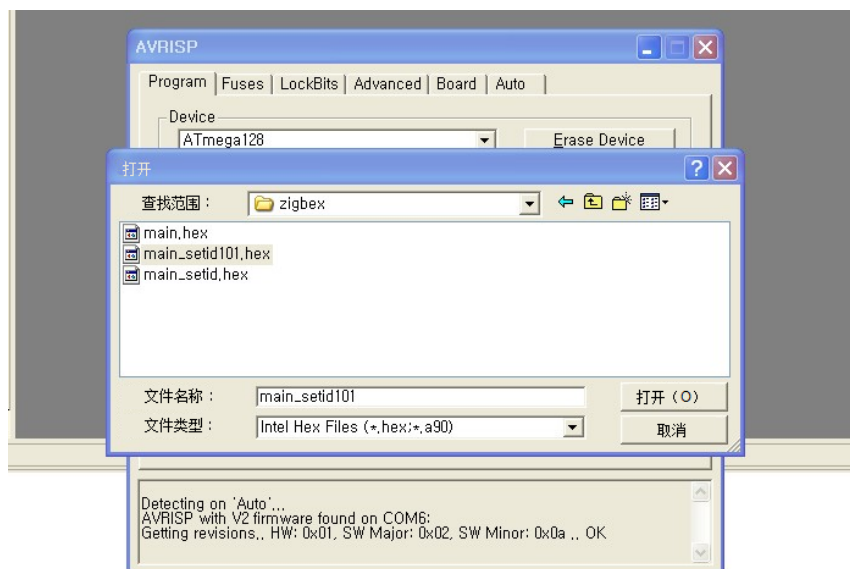


图 6-9 下载文件界面

在下载过程中，各传感器模块下载各自对应文件夹中的 .hex 文件，与 PC 相连接的节点下载 BaseStation 文件夹中的 .hex 文件，如图 6-10 所示。

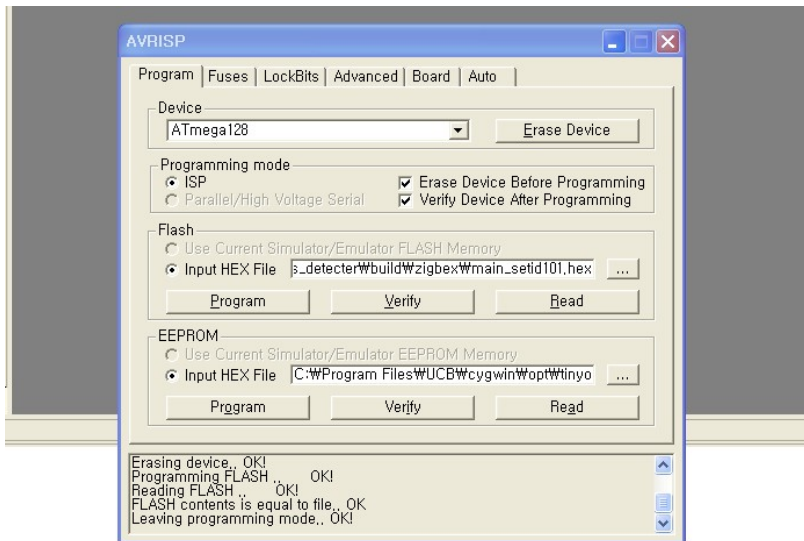


图 6-10 程序下载界面

(4) 程序运行及结果

连接 USB ISP 模块和标注 ID0 号的节点，连接 PC 与 USB ISP 模块。

经过对可视界面的一系列操作，会看到如图 6-11 所示的传感器实时数据界面。



图 6-11 传感器实时数据界面

6. 思考题

1) 除本实验给出的示例外，TinyOS 系统是否可以添加其他的传感器模块组件？如果可以，如何进行编辑，连接，编译；如果不可以，请说明理由。

2) 在软件层面，怎样给本实验中的气体监测模块节点添加固有的传感器组件（如温度、湿度、光照度、红外传感器），并定时发送实时数据？



6.2 演示二 出入管理演示实验

1. 实验目的

本演示实验在学习和理解 RFID 标签和读写器基本原理（可参阅前文实验十和实验十一）的基础上，设计了一个基于 RFID 的非接触式双头读卡方案，构建了人员出入管理系统。通过实验演示，能够使学生更好地理解 and 掌握 RFID 模块的实际应用方式，并对出入管理系统的运行方式进行全面了解。

2. 实验设备

(1) 硬件

- RFID 900MHz 读写器 2 套。
- 5V 直流电源及连接线 1 套。
- 900M 标签（EPC Class1 G2）10 个。
- 无线传感器网络开发套件（含 10 个基本节点和 1 个 USB-ISP 下载模块）1 套。

(2) 软件

- AVR Studio4.18。
- TinyOS。
- 串口调试助手。
- 读写器上位机软件。

3. 实验原理

人员出入管理系统主要包括 RFID 标签和 RFID 读写器，以及用于综合处理和显示的后端设备（计算机），其中 RFID 标签上记录着用户的个人信息。当用户进入 RFID 读写器的感知范围时，其携带的 RFID 标签会将用户的个人信息传输到 RFID 读写器，读写器再将信息传输到后端显示和处理设备，用于后续处理。目前，人员出入管理系统已经被广泛使用在学校、公司、政府机关等机构，用于人员信息管理和考勤统计。

在实验室走廊通道的前后两端各放置一套 RFID 读写器，并且两套 RFID 读写器的感知范围不互相重叠。这样，根据两套读写器接收到数据的时间先后顺序，就可以判断出用户的行进方向。当用户的行进方向是面向房间，并且靠近房间一侧的读写器接收到了用户的个人信息时，判断用户已进入房间；当用户的行进方向是背离房间，并且远离房间一侧的读写器接收到了用户的个人信息时，判断用户已离开房间。依靠这样独特的双头读卡方案，可以有效避免用户长时间滞留在读写器的感知范围内，从而导致读写器判断失误的情况。利用人员出入管理系统，管理人员就可以方便地对出入房间的人员信息进行统计和管理，避免因疏忽造成遗漏。

本演示实验中，RFID 读写器可采用两种方式向计算机进行数据传送：一种是有线方式，即将读写器上的 USB 口直接与计算机连接，将用户信息以有线方式传输到计算机上；另一种是无线方式，即将读写器后端与无线传感器网络节点相连，将用户信息首先传送到无线传感器网络节点，之后通过多个无线传感器网络节点进行多跳传输，传输到与计算机相连的 0 号节点，最后通过 0 号节点与计算机的接口传输到计算机上。

本实验系统使用了如下实验设备，结构框架及实物图如图 6-12 和图 6-13 所示。

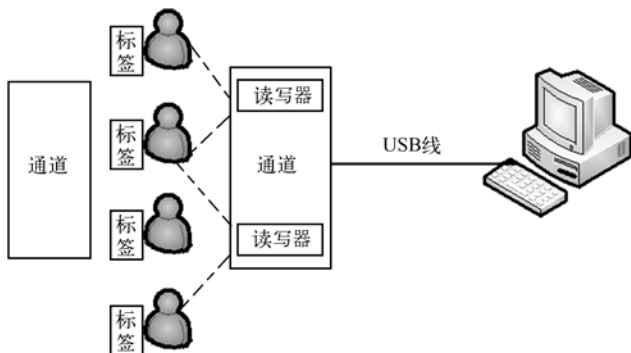


图 6-12 出入管理演示实验结构框架

(1) RFID 900MHz 读写器

RFID 900MHz 读写器用于收集用户的个人信息，并将信息传输到后端显示和处理设备。RFID 读写器可固定在门槛或是门口合适的位置上，使用时需给 RFID 读写器供以 5V 电源。

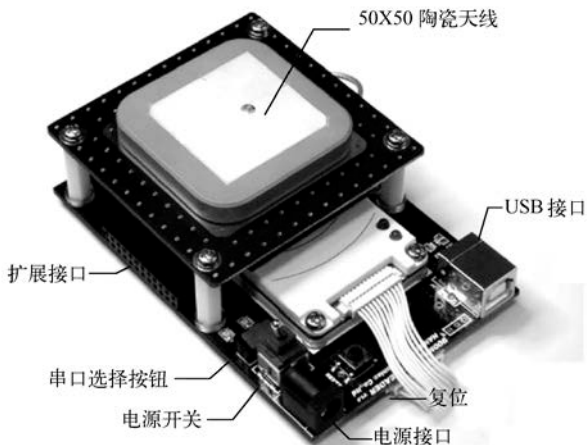


图 6-13 RFID 900MHz 读写器

(2) 标签

RFID 标签（如图 6-14 所示）记录着用户的个人信息，当用户通过 RFID 读写器的感知范围时，标签会将用户信息传输给 RFID 读写器。RFID 标签可粘贴在学生的实验指导书上，当学生携实验指导书进出实验室时，管理员可以通过 RFID 读写器对学生的出入情况进行自动记录。



图 6-14 RFID 900M 标签

3) 实验代码

LIB_Header□ LIB_Interaction两个文件夹中的文件 配置了需用到的组件，OptRFID文件 夹中的文件 描述了RFID读写器具体的功能实现。

4. 实验内容

(1) 搭建演示平台



进行各模块间的连接，安装读写器上位机软件，为演示做好软硬件准备。

(2) 进行演示实验

进行实际系统演示，使学生对 RFID 系统在日常生活中的应用模式有更深入的了解。

5. 实验步骤

(1) 模块连接

将读写器通过 USB 连接线与计算机相连接（如图 6-15 所示），为其提供 DC 5V 电源，确保数据通路的顺畅。

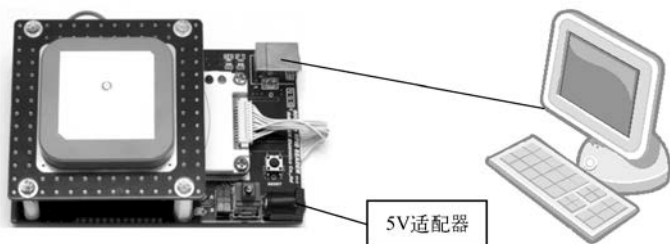


图 6-15 实际连接图

(2) 驱动及程序安装

初次将读写器连接至计算机，并开启电源后，系统会提示安装驱动。请将驱动所在目录移动到光盘的 Driver 文件夹，选择自动安装选项。驱动安装完毕，将事先准备好的程序下载到 RFID 读写器中。

(3) 开启上位机软件进行标签识别

在如图 6-16 所示的上位机操作界面中选择正确的串口连接，单击 Start 按钮，就可以进入 RFID 标签识别阶段。当有用户进入或离开时，软件界面会自动更新其个人信息。

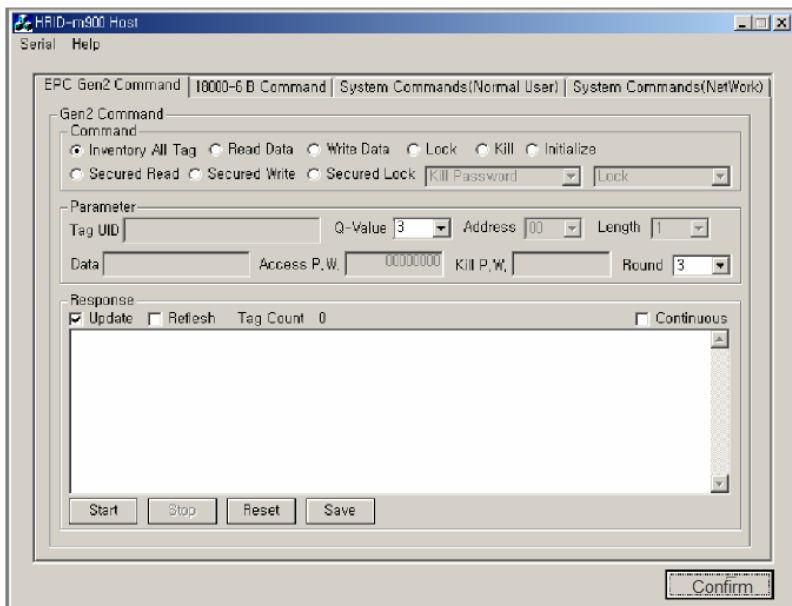




图 6-16 上位机操作界面

6. 思考题

- 1) RFID 在通信过程中会受哪些因素的影响?
- 2) RFID 读写器的读写工作原理是什么?
- 3) 沃尔玛超市中物品上所使用的标签是哪个频段的? 当大量货物进出时, 其使用的读写器在技术上会面临哪些问题?

6.3 演示三 智能家居演示实验

1. 实验目的

本演示实验在学习和理解传感器基本原理和 IEEE 802.15.4 通信标准(可参阅前文实验十五和实验十六)的基础上, 增加了室内电气监测控制传感器模块, 构建了一个实时家电监测控制系统。通过本演示实验, 能够使学生更好地掌握各种电气监测控制传感器模块的工作原理, 并形成物联网智能家居系统应用的整体概念。

2. 实验设备

(1) 硬件(智能家居实训套件)

- 继电器控制模块 4 套。
- 电灯亮度控制模块 1 套。
- 马达驱动控制模块 1 套。
- 密码锁 1 套。
- 燃气阀门控制器 1 套。
- 燃气泄漏感应报警器 1 套。
- 灯 1 套。
- 人体热释电感应模块 1 块。
- 数字量输入、输出模块 1 块。
- Zigbex 节点 7 块。
- USB ISP 下载板 1 块。
- 接线板 1 块。
- 智能家居模型 1 套。
- PC 1 台。

(2) 软件

- AVR Studio4.18。
- TinyOS 1.x。
- 串口调试助手。
- HBE-HOMENET 上位机软件。

3. 实验原理

基于物联网的智能家居系统内部组网需要控制的设备多, 网络容量大; 系统内包含多种高速、中速及低速的数据设备, 对信息传输的延时性、实时性要求严格; 兼容广泛的连接技



术；需要保证高级别的安全性。智能家居系统组网结构如图 6-17 所示。

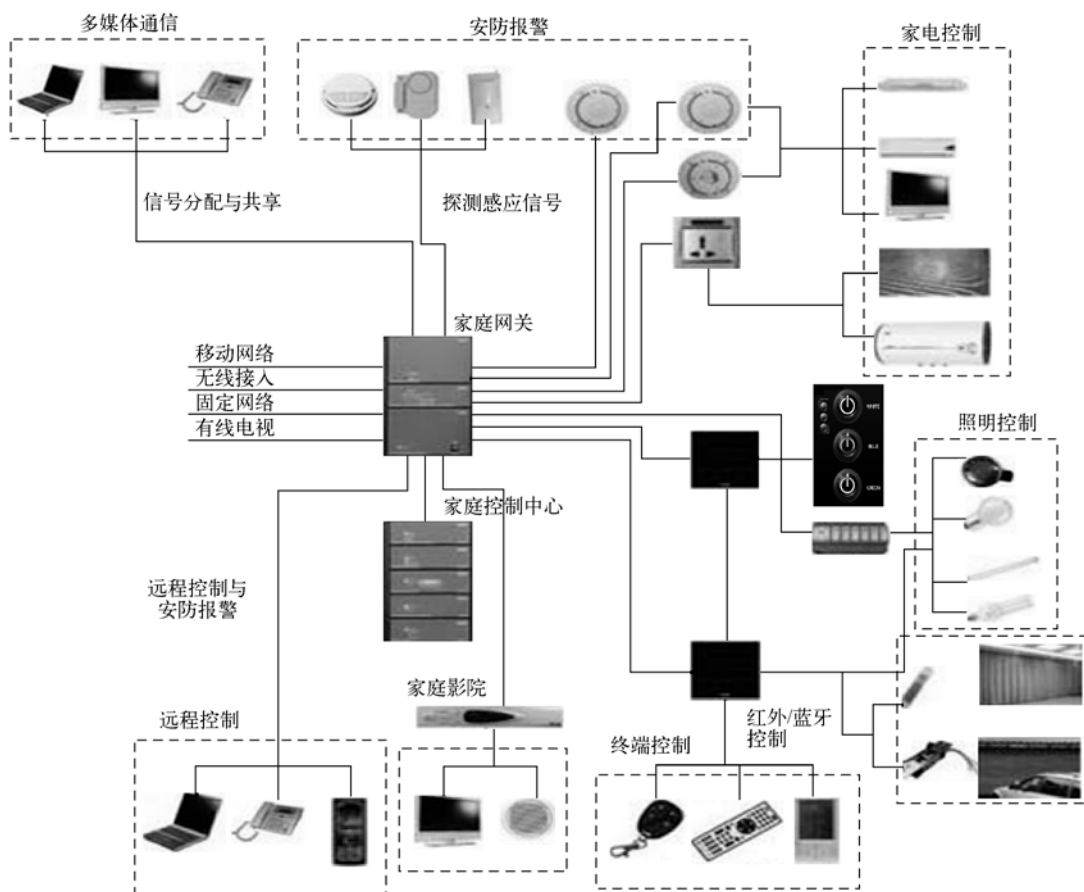


图 6-17 智能家居系统组网结构

图 6-17 所示的智能家居系统组网结构中，家居设备、移动终端以及其他域设备通过网络实现信息互通、协议转换、信息共享和交互；借助高速有线网络与无线互联，在 QoS、UPnP、DHCP 等关键技术的支撑下，各种智能家居设备可以自动、无需任何配置地接入系统，极大提高了系统的灵活性、易用性及可扩展性。

基于物联网的智能家居组网采用分层体系架构，同样可以划分为感知层、网络传输层、融合服务层 3 个层面。

智能家居感知层：包括了各种与家电、家居相关的传感器、控制器、执行器及识别装置等，以及有线网络结合无线泛在网络的物理连接。这一层还包括不同接入方式的 MAC 子层和链路控制子层，作用包括对上层网络层提供统一的接口，屏蔽异构网络之间的差异；进行不同形式家庭通信网络间的 MAC 协议数据单元（PDU）映射，以支持不同网络间进行方便互通；支持动态智能的有线及多种无线网络的接入及选择。

智能家居网络传输层：主要包括家庭内部网络和骨干网络接入两部分。家庭内部组网支持的有线方式包括电子载波 X-10 和 CEBUS、电话线 HomePNA、以太网 IEEE 802.3 和



802.3u、串行总线 USB1.1、USB2.0 和 IEEE 1394 等；无线方式包括无线局域网、家庭射频技术、蓝牙、红外、ZigBee 等。网络接入层通过家庭网关与业务网关，实现不同应用协议规范的互联互通互操作，并与骨干网络实现无缝连接。

智能家居融合服务层：以用户为中心的融合业务层，提供用户接口和不同制式的智能家居服务。通过智能家居组网多层协作的自适应 QoS，自适应匹配异构网络及终端设备，充分保证端到端的媒体、语音、安防等多种业务的服务。智能家居演示系统包括门锁控制单元、煤气泄漏监测单元、煤气阀门控制单元、灯光亮度控制单元和继电器控制单元。这些控制单元对家庭生活中常用的电器设备进行监测和控制，并通过无线传感器网络节点与室内主控单元互连互通（主控单元可以是固定的 PC，也可以是用户手持的 PDA 或手机），形成一个绿色智能的网络，使得用户可以自由操纵室内的各种电器，真正成为生活的主人。

本实验系统使用了如下实验设备。

(1) 门锁控制单元

门锁控制单元用于控制用户室门的开启和关闭，在提供出入便捷性的同时，保护家中的财产安全。门锁控制单元包括数字门锁、继电器控制模块、无线传感器网络节点 3 部分，其中无线传感器网络节点嵌入继电器控制模块，系统连接如图 6-18 所示。

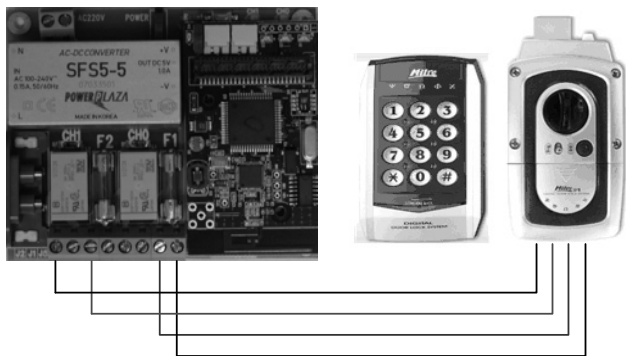


图 6-18 门锁控制单元系统连接图

(2) 燃气泄漏监测单元

煤气泄漏监测单元用于监测室内煤气含量，以便在煤气超标时及时提示主人，避免情况进一步恶化。煤气泄漏监测单元包括继电器控制模块、煤气泄漏感应器、数字量输入/输出 (DIO) 选配模块、无线传感器网络节点 4 部分。在连接时，将连接有 DIO 模块的 Zigbex 节点嵌入继电器控制模块，并连接燃气泄漏检测传感器及 DIO 模块。其系统连接如图 6-19 所示。



图 6-19 燃气泄漏监测单元连接图

(3) 燃气阀门控制单元

燃气阀门控制单元用于控制家中煤气的开启和关闭，并根据实际情况控制煤气流量，在节省能源的前提下避免出现煤气泄漏的情况。燃气阀门控制单元包括直流电机控制模块、燃气阀门、无线传感器网络节点 3 部分。在连接时，将无线传感器网络节点嵌入电机控制模块，并将燃气阀门连接至电机 1 控制输出端。其系统连接如图 6-20 所示。

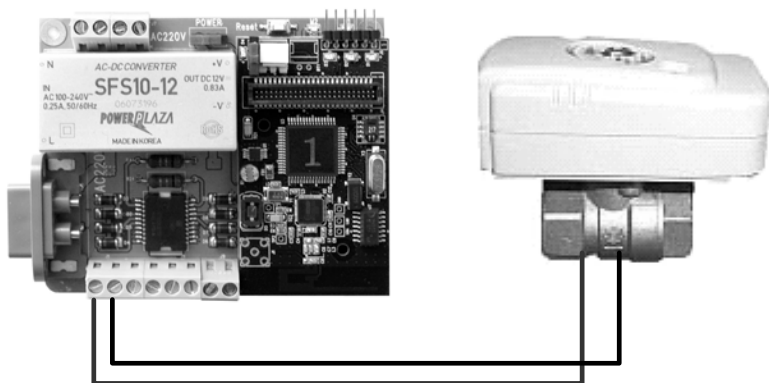


图 6-20 燃气阀门控制单元连接图

(4) 灯光亮度控制单元

灯光亮度控制单元可以根据室内现有光照情况自动调节室内灯光强度，在保证室内明亮的前提下节省能源。灯光亮度控制单元包括灯光亮度控制模块和无线传感器网络节点两部分。在连接时，将电灯电源线与亮度调节模块输出接口相连，并将无线传感器网络节点嵌入电灯亮度控制模块。其系统连接如图 6-21 所示。

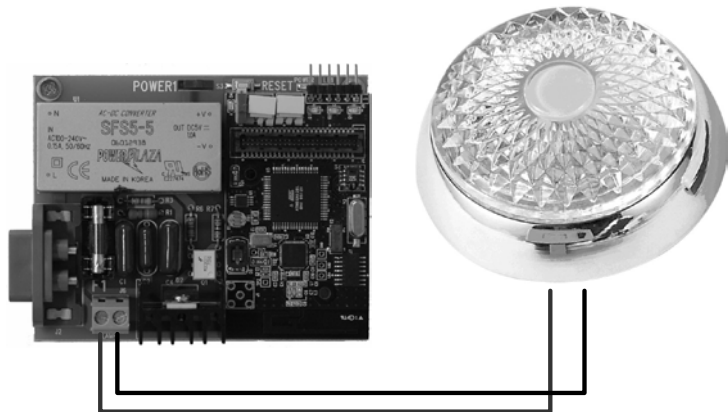


图 6-21 灯光亮度控制单元连接图

(5) 继电器控制单元

继电器控制单元用于连接家用电器与继电器，以便对家用电器的电流、电压进行自动调节和安全保护。继电器控制单元包括家用电器模块和继电器模块两部分，其系统连接如图 6-22 所示。

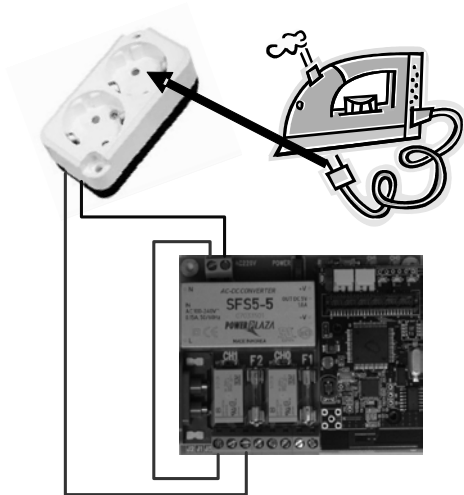


图 6-22 继电器控制单元连接图

(6) 实验代码

无线传感器网络节点的编程基于TinyOS 2.x, LIB_Header和LIB_Interaction两个文件夹中的文件 配置了需用到的组件, hanback_homenet_TinyOS2x文件夹中的文件 描述了各个电气控制检测模块的具体实现。

4. 实验内容

(1) 构建开发环境

安装 TinyOS 系统、Cygwin、AVR Studio, 构建无线传感器网络的开发环境。学习如何



进行基于 TinyOS 系统的编程、编译、下载。

(2) 理解、运行例题程序

理解运行在 TinyOS 下的 nesC 程序的特性，理解整个演示系统的构建过程及例题程序。掌握程序的下载、运行、调试方法。

(3) 修改程序并搭建系统

能够基于例题程序，设计出新程序，构建智能家居演示系统。

5. 实验步骤

(1) 安装 TinyOS 及 AVR Studio

本步骤是实验开始前的软件准备工作。通常，TinyOS 作为针对传感网络开发的操作系统，需要运行在 Linux 环境下。然而，为了开发方便，本演示实验将通过 Cygwin 开发包在 Windows 下模拟 TinyOS 在 Linux 下的操作。具体步骤详见演示一。

(2) 安装例题程序

安装所有例题程序，进行程序编译。在理解程序思路之后可以进行灵活修改。具体步骤详见演示一。

(3) 程序下载

将编译完毕的程序下载到传感器节点端。为了下载 Zigbex 程序，需要 AVR Studio 和 USB-ISP。具体步骤详见演示一。

(4) 程序运行及结果

连接 USB ISP 模块及标注有 house_server 的节点，连接 PC 与 USB ISP 模块。在 PC 端启动智能家居控制的上位机软件。选择正确的 com 端口后，可通过软件界面左侧的按钮对各家电进行控制。

6. 思考题

1) 本实验中用到继电器控制灯在操作，也学习了人体热释感应模块的工作原理和使用方法，修改代码视图实现当人体热释感应模块感应到物体移动时，同时控制灯亮。

2) 怎样在软件层面为电灯亮度控制单元添加固有的传感器组件（例如温度、湿度、光照度、红外传感器），并定时发送其实时数据？

3) 除了本实验给出的各种组合，是否还能组合成其他模块进行联动？

6.4 演示四 智能医疗演示实验

1. 实验目的

本演示实验在学习和理解传感器基本原理和无线传感器网络基本框架的基础上，增加了人体生理信息传感模块，构建了一个健康状况实时检测管理系统。通过演示实验，使学生能够了解相关医疗检测模块的工作原理，并形成基于物联网的智能医疗应用的整体概念。

2. 实验设备

(1) 硬件（智能医疗实训套件）

- 腕式血压计及接口板 1 套。
- 血氧含量检测模块及接口板 1 套。
- 心电检测模块及接口板 1 套。



- 非接触温度检测模块及接口板 1 套。
- 无线传感器网络开发套件（含 10 个基本节点和 1 个 USB-ISP 下载模块）1 套。
- PC 1 台。

(2) 软件

- AVR Studio4.18。
- TinyOS 2.x。
- 串口调试助手。

3. 实验原理

本实验系统主要包括血压计模块、血氧含量检测模块、心电检测模块、非接触温度检测模块、无线传感器网络开发套件和后端显示与处理设备。这些检测模块可以实时检测人体的各项生理指标，并将采集到的数据传输到无线传感器网络节点上面的处理模块，经过特定处理后再通过射频模块进行无线传输，将数据发送给物联网手机或 PC，最终汇聚到网络服务器上，供用户查询。本系统的结构框图如图 6-23 所示。

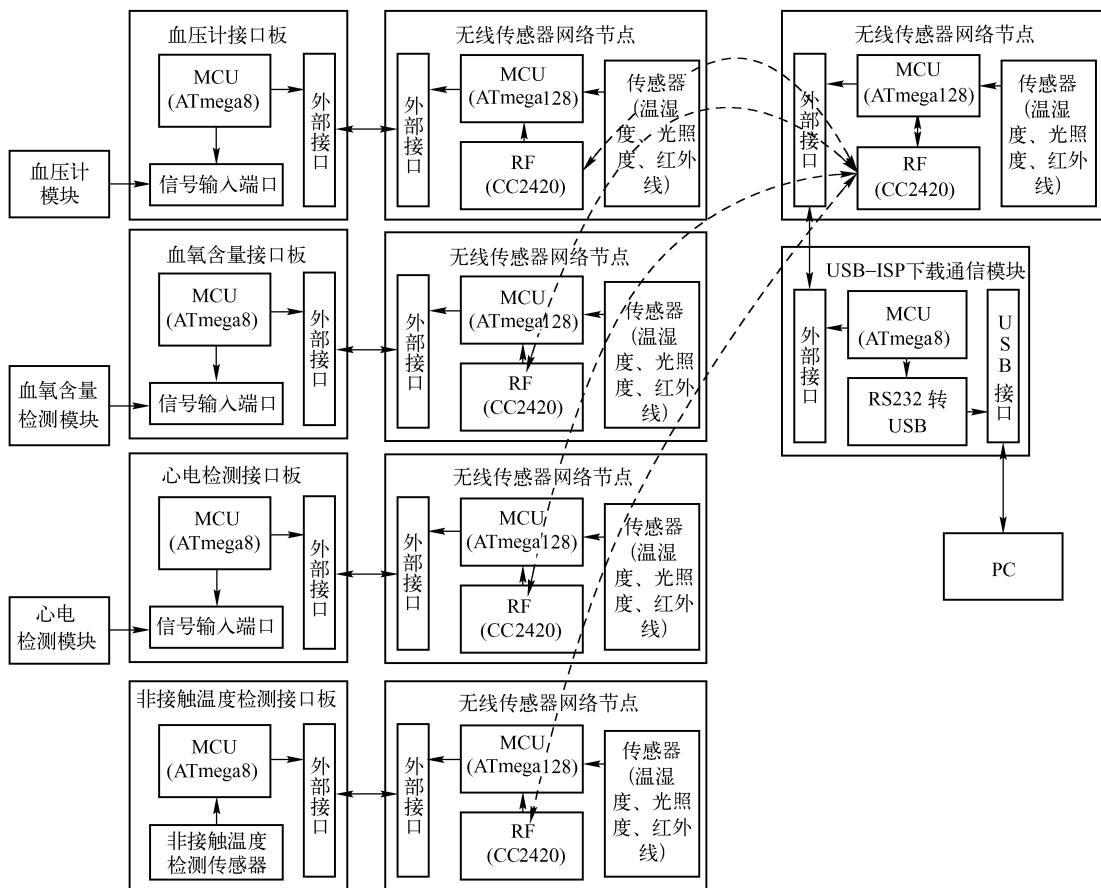


图 6-23 智能医疗演示实验系统结构框图



本演示系统使用了如下实验设备：

(1) 血压计模块说明

血压计模块包括腕式血压计、血压计接口板和无线传感器网络节点 3 部分，用于实时检测用户的血压情况。其中血压计接口板用于连接腕式血压计和无线传感器网络节点（如图 6-24 所示），将采集到的血压信息及时通过无线传感器网络发送给用户。

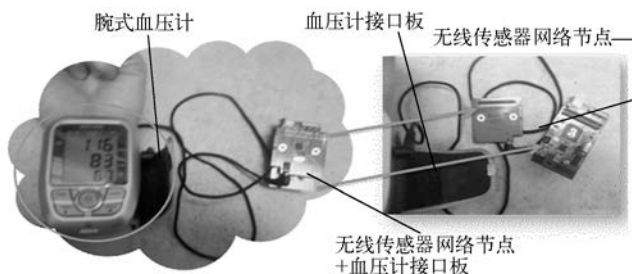


图 6-24 血压计模块连接图

(2) 血氧含量检测模块说明

血氧含量检测模块包括血氧含量检测设备、血氧含量接口板和无线传感器网络节点 3 部分，用于实时检测用户的血氧含量情况。其中血氧含量检测模块接口板用于连接血氧含量检测设备和无线传感器网络节点（如图 6-25 所示），将采集到的血氧含量信息及时通过无线传感器网络发送给用户。

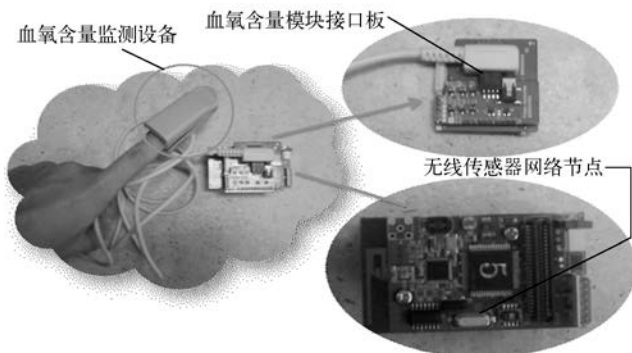


图 6-25 血氧含量检测模块连接图

(3) 心电检测模块连接说明

心电检测模块包括心电检测设备、心电检测模块接口板和无线传感器网络节点 3 部分，用于实时检测用户的心跳状况。其中心电检测模块接口板用于连接心电检测设备和无线传感器网络节点（如图 6-26 所示），以便将采集到的心跳信息及时通过无线传感器网络发送给用户。

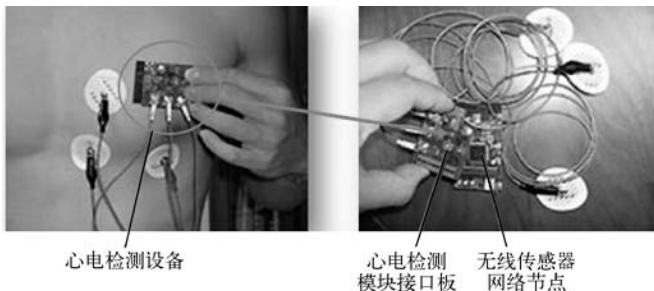


图 6-26 心电检测模块连接图

(4) 非接触温度检测模块说明

非接触温度检测模块包括非接触温度检测器和无线传感器网络节点两部分。当非接触温度检测器感知到用户的体温后，就可以通过与无线传感器网络节点相连的接口，及时将采集到的体温信息通过无线传感器网络发送给用户。非接触温度检测器如图 6-27 所示。

(5) 实验代码

无线传感器网络节点的编程基于TinyOS 2.x，LIB_Header□ LIB_Interaction两个文件夹中的文件 配置了需用到的组件，OptSPO2 文件夹中的文件 描述了血氧含量检测模块具体的功能实现，OptBio文件夹中的文件 描述了血氧含量检测模块具体的功能实现，OptTN9 文件夹中的文件 描述了非接触式温度检测模块具体的功能实现，OptBloodPress文件夹中的文件 描述了血压检测模块具体的功能实现。



图 6-27 非接触温度检测器

4. 实验内容

(1) 构建开发环境

安装 TinyOS 系统、Cygwin 虚拟机、AVR Studio，构建无线传感器网络的开发环境。学习如何进行基于 TinyOS 系统的编程、编译、下载。

(2) 理解、运行例题程序

理解运行在 TinyOS 下的 NesC 程序的特性，理解整个演示系统的构建过程及例题程序。掌握程序的下载、运行、调试方法。



(3) 修改程序并搭建系统

能够基于例题程序，设计出新程序，构建智能医疗系统。

5. 实验步骤

(1) 安装 TinyOS 2 及 AVR Studio

本步骤是实验开始前的软件准备工作。通常，TinyOS 作为针对传感网络开发的操作系统，需要运行在 Linux 环境下。然而，为了开发方便，本演示实验将通过 Cygwin 开发包在 Windows 下模拟 TinyOS 在 Linux 下的操作。具体步骤详见演示一。

(2) 安装例题程序

安装所有例题程序，进行程序编译。在理解程序思路的基础上可以进行灵活修改。具体步骤详见演示一。

(3) 程序下载

将编译完毕的程序下载到传感器节点端。为了下载 Zigbex 程序，需要 AVR Studio 和 USB-ISP。具体步骤详见演示一。

(4) 程序运行及结果

连接 USB ISP 模块及标注有 ID0 号的节点，连接 PC 与 USB ISP 模块，打开串口调试助手，设定正确的 com 号及波特率（57600bit/s）。通过串口调试助手，可以查看到运行界面。

6. 思考题

- 1) 无线传感器网络用于医疗给人们的生活带来了哪些方便之处？
- 2) 怎样在软件层面给本演示实验中的人体健康状况监测模块添加固有的传感器组件（如温度、湿度、光照度、红外传感器），并定时发送实时数据？

6.5 演示五 智能抄表演示实验

1. 实验目的

本演示实验在学习和理解传感器基本原理和无线传感器网络基本框架的基础上，增加了电表、水表传感器采集与远程数据输出的应用，构建了一个远程智能抄表系统。通过演示实验，可以使熟悉和理解电表、水表等数据采集模块的原理与应用，在实际操作中体会 3 种远程数据传输方式的优劣，并建立基于物联网的远程抄表系统应用的整体概念。

2. 实验设备

(1) 硬件（智能抄表实训套件）

- 电表数据采集器 1 套。
- 水表数据采集器 1 套。
- 无线传感器网络开发套件（含 9 个基本节点和 1 个 USB-ISP 下载模块）1 套。
- 电力线通信调制解调器 1 套。
- 电力传输线 1 条。
- 后端处理和显示终端（PC）1 台。

(2) 软件

- AVR Studio4.18。
- TinyOS。



- 串口调试助手。

3. 实验原理

如图 6-28 所示，智能抄表系统主要包括水表数据采集器、电表数据采集器、无线传输节点、电力线通信调制解调器（俗称“电力线猫”）和电力线。水表数据采集器和电表数据采集器在采集到相关数据后，可以通过 3 种方式将数据传输到后端处理设备。第一种是利用无线传感器网络节点，通过无线多跳方式进行传输；第二种是利用监控室及附近已部署的有线或无线网络（如 WiFi 覆盖）进行传输；第三种是利用电力线载波通信的方式进行传输。

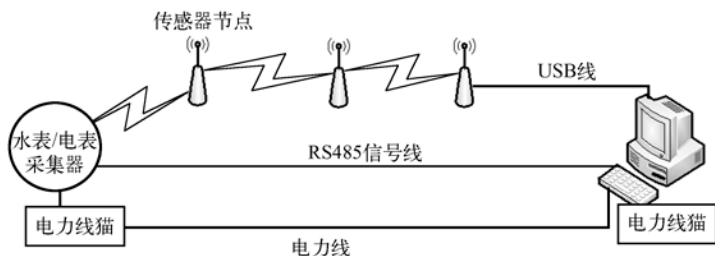


图 6-28 智能抄表演示系统框架

第一种方式适用于采集终端离监控室较近的情况。在这种情况下，数据的传输可以只通过无线传感器网络来完成。第一种方式的优点是节点间的路由可自由进行设置，因此灵活性较强，缺点是传输距离有限。

第二种方式适用于采集终端离监控室较远，但监控室及附近有有线网络或无线网络覆盖的情况。在这种情况下，可通过在采集终端植入嵌入式系统搭建网络服务器，监控室通过已部署的网络访问网络服务器，从网络服务器主动提取数据，或网络服务器定时将数据传送到监控服务器上；监控终端也可通过 RS485 信号线直接与远传水表/电表相连，进行续数。第二种方式的优点是数据传输距离远，稳定性强；缺点是有线网络或无线网络的部署较复杂，对硬件环境要求较高。

第三种方式适用于采集终端较远，并且监控室附近缺乏网络覆盖的情况。在这种情况下，为了免去布置数据采集专用线路的麻烦，可以利用已有的电力线进行传输。采用这种方式，需在终端设置电力线猫，电力线猫与远传水表/电表的 RS485 接口直接相连。在传输过程中，水表/电表数据采集器首先将数据传输给电力线猫，电力线猫将数据转换为电信号，再通过电力线传输到终端处，最后终端通过电力线猫将信号还原。第三种方式的优点是对环境硬件要求较低，适用性强；缺点是数据传输的稳定性较差，容易受到干扰。

对应三种不同的传输方式，处理和显示终端需要进行不同的配置。对于第一种方式，终端通过 USB 口与发送节点进行连接，接收从其他无线传感器网络节点多跳传输来的数据；对于第二种方式，终端通过 RS485 信号线直接从水表/电表中采集数据；对于第三种方式，终端通过 RS485 接口连接至电力线猫，通过电力线接收数据。

当水表和电表数据传输到处理和显示终端后，终端对数据进行简单处理，就可以通过软件界面展现在用户面前了，避免了用户专程去查看水表和电表数据的麻烦，为供水和供电部



节省了大量人力和财力。

本演示系统使用了如下实验设备：

(1) 电表数据采集器

电表数据采集器也称远传电表，用于采集电表的 RS485 数据（或脉冲信号），再将电表数据转换为可传输的数字信号进行远程传输。其实物如图 6-29 所示。



图 6-29 电表数据采集器

(2) 水表数据采集器

水表数据采集器也称远传水表，用于采集水表上记录的用户用水量，再将水表数据转换为可传输的数字信号进行远程传输。其实物如图 6-30 所示。



图 6-30 水表数据采集器

(3) 实验代码

无线传感器网络节点的编程基于 TinyOS 2.x，Common Component 文件夹配置了 TinyOS 系统的相应组件，ETS_Solar_base 文件夹配置了与电脑相连接的 Base 节点端程序实现，ETS_Solar_node 文件夹配置了与水表/电表相连接的数据采集节点端程序。

4. 实验内容

(1) 搭建演示平台



连接无线传感器网络节点、接口模块、水表数据采集器和电表数据采集器，为演示系统搭建网络环境。

(2) 进行远程抄表演示实验

进行远程抄表系统的演示过程，具体演示水表/电表数据的3种传输方式，使学生感受3种数据传输方式的优劣，从而更深入地理解物联网应用系统在具体环境中因限制不同而产生的不同变化模式。

5. 实验步骤

(1) 通过无线多跳方式进行数据传输

1) 安装 TinyOS2 及 AVR Studio。实验开始前的软件准备工作。通常，TinyOS 作为针对传感网络开发的操作系统运行在 Linux 环境下。然而，为了开发方便，本演示实验将通过 Cygwin 开发包在 Windows 下模拟 TinyOS 在 Linux 下的操作。具体步骤详见演示一。

2) 安装TinyOS组件。从 Common Component文件夹中安装必备的组件。

3) 程序下载。将编译完毕的程序下载到传感器节点端。为了下载Zigbex程序，需要 AVR Studio和USB-ISP。具体步骤详见演示一。具体将 ETS_Solar_base文件夹中的程序下载到Base节点端，将ETS_Solar_node文件夹中的程序下载到数据采集节点端。需要提及的是，由于水表/电表的型号众多，学生可根据使用水表/电表的实际型号，按照厂商给定的数据采集格式，对ETS_Solar_node文件夹中的程序进行相应修改。

4) 程序运行及结果。连接USB ISP 模块及 Base节点，连接PC与USB ISP模块，打开串口调试助手，设定正确的com口号及波特率（57600bit/s）。通过串口调试助手，查看水表及电表数据。

(2) 通过有线网络进行数据传输

1) 模块连接。将水表/电表与 RS485 信号线连接、监控终端通过 RS485/RS232 转换模块与 RS485 信号线的另一端相连。

2) 开启上位机软件进行读表操作。开启监控终端处的上位机软件，对获取的数据进行解析处理并呈现给用户获取水表和电表数据。

(3) 通过电力线载波通信方式进行数据传输

1) 模块连接。将水表/电表数据采集器与电力线猫进行连接，后端处理和显示终端与电力线猫进行连接，建立一个完整的网络通路。

2) 驱动安装。初次将电力线猫连接至计算机，并开启电源后，系统会提示安装驱动。请将驱动所在目录移动到光盘的 Driver 文件夹，选择自动安装选项。

3) 开启上位机软件进行读表操作。开启计算机上安装的上位机软件，通过电力线远程获取水表和电表数据。

6. 思考题

无线传感器网络用于智能抄表系统，不仅可以实时感知水表、电表的数据，还可以通过无线传感器节点上的哪些传感器获取当前水表、电表的环境状况？

第7章 物联网应用示范系统

物联网实验的设计约束因素较多，实验规模也受到一定限制。本章主要介绍规模较大的应用示范系统，进一步拉近物联网与行业应用的距离，主要包括智能医疗、城市应急、智能搜索、智能家居、智能物流、地质监测等应用系统的设计，每部分包括系统概述、系统架构、工作流程、系统特点及组成几方面内容。这些都是的工作在相关领域的专家完成的。其中智能医疗和智能家居已在第6章以演示实验的方式提供，本章的系统设计可以从另外层面提供参考。关于其他更多的应用示范系统，将通过出版社或相关网站为读者提供。

7.1 系统一 智能远程医疗急救系统

智能远程医疗急救系统是一种以呼叫中心和网络平台为中心，与急救中心、医疗机构、健康咨询机构等相结合，集监测、定位、呼救于一体的健康救助系统，是十分典型的物联网应用系统。该系统包括无线心电监护仪、无线血压计、无线血氧仪等医疗设备以及健康手机和网络平台。借助物联网相关技术的应用，这些医疗设备可以让用户无创伤地测量心电图、血压、血氧等数值，并将这些代表生命体征的数据发送和同步显示在手机上，以使用户实时监测自身健康状况；同时手机自动将这些测量数据转发到网络平台的个人电子健康档案库，供医院和专家参考，以实现准确快速的治疗。当用户出现意外时，按下手机上的紧急救助键，手机会自动和急救中心、服务中心接通，实现紧急救助。

1. 系统概述

国家统计局的调查数据显示，全国60岁以上人口为1.3亿，占总人口的10%，是世界上唯一一个老年人口超过1亿的国家。也就是说，中国已正式进入“老龄时代”。随着新中国成立后50年代出生高峰的人口即将进入老年，预计到2015年，我国老年人总数将突破2亿，2027年超过3亿，2044年将超过4亿。

中国在养老、医疗、长期照料服务等社会保障制度方面还不是很完善，在城乡二元结构尚未根本改变的情况下，解决老有所养、老有所管问题，已成为一个涉及千千万万个家庭、牵动着社会神经的敏感话题。胡锦涛主席曾指出：“尊重老年人、关爱老年人、照顾老年人，是中华民族的优良传统，也是一个国家文明进步的标志。我们要弘扬中华民族尊老敬老的传统美德，大力发展老龄事业，给予老年人更多生活上的帮助和精神上的安慰，让所有老年人都能尽享幸福的晚年”。这也充分体现了国家对人口老龄化问题的高度重视。

目前社会出现了许多的“2+4家庭”，即夫妻双方要照顾4位老人。而由于生存压力以及人们生活习惯的改变，老人和子女们不在一起生活的空巢家庭也越来越多。资料显示，我国目前身边无子女、又无他人照料的“空巢老人”有2340万之多，约占老年人家庭的20%。专家认为，随着我国社会老龄化的加剧，家庭空巢化成为社会发展的必然趋势。预计



到 2030 年，我国空巢老年人家家庭可能达到老年人家家庭的 90%。届时，我国老年人家庭将“空巢化”。老年人的养老与健康问题已经成为社会和谐发展的重要议题。

当空巢老人遇到疾病突发时，往往会因为身边无人照料而错失救治良机。尤其是有心脏病的老人，在病情发作的情况下多伴有神志不清、无行为能力等症状，很多情况下病人已经无法自己完成呼救等一系列救护措施，最后因错过最佳抢救时机而造成不良后果。在 2002 年，中国有 230 万人的死亡与心血管疾病相关。世界卫生组织公布的《2008 年世界卫生统计》报告中指出，全球每年有 1700 万人患心脏病，患心脏病等慢性病的死亡人数还将不断增加，这种趋势将延续至 2030 年。据统计，全世界心血管病每年夺走 1200 万人的生命，接近世界人口总死亡人数的 1/4。“中国约有 1 亿高血压患者，每年新发生脑卒中病例约 300 万，心肌梗死 60 万人，心脑血管病死亡人数约 260 万，平均每小时死亡 300 人”，这是出席第二届五洲国际心血管病研讨会的北京安贞医院院长张兆光教授提供的数字。他说，近年的统计数据显示，中国人群的心脑血管病正在逐年上升，总数已经大大高于美国、加拿大、法国、瑞士、日本等国家。

远程健康管理及救助系统正是基于“尊敬老年人、关爱老年人、帮助老年人”的优良传统，为空巢老人尤其是心脏病患者、高血压患者等提供了一种能够实现自动呼救功能的远程保健救助系统，在一定程度上解决关系国计民生的老年人健康问题。

2. 系统架构

基于物联网的智能家庭远程医疗急救系统包括无线血压计、无线心电监护仪、无线血氧仪以及基于物联网的手机和网络服务平台等，多种健康监护及测量设备佩戴在身体的不同部位，尽量做到对正常生活无影响。这些设备最终通过物联网手机汇聚到网络服务器上，并通过多种手段与亲属、急救中心及医院专家形成一个远程救助的网络，如图 7-1 所示。

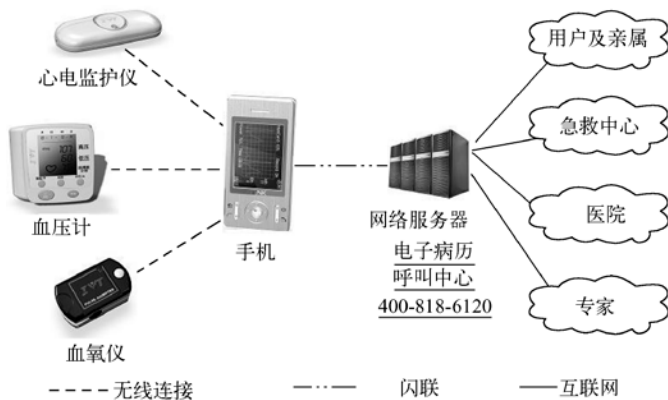


图 7-1 智能家庭远程医疗急救系统

本系统紧密结合物联网技术，具有以下技术特点。

(1) 终端设备之间的无线通信技术

目前，基于物联网的手机与电子医疗设备（无线血压计、心电监测仪、血氧仪）之间的通信采用手机上的常用蓝牙接口，使得用户携带和使用非常便捷。当然，随着物联网应用的发展，也可以采用 ZigBee、低功耗蓝牙、WiFi 等短距离无线通信技术。



在健康监测感知设备方面,用户可根据需要自行选择使用无线血压计、心电监护仪、血氧仪等便捷式微型医疗设备,利用这些医疗设备测得的血压、心电图、血氧等数值,通过蓝牙通信的方式实时传送给手机,以便于用户通过手机进行自行监测,并传输到远程。

(2) 动态心电、血压、血氧含量实时监测

智能物联网手机从无线血压计、心电监护仪、血氧仪等医疗设备接收测量数据后,能够相应地实时显示血压数据、心电图和血氧数值,也可以将测量结果以短信形式自动发到子女的手机上,便于子女对老人身体状况的随时了解和把握。同时,手机还设置有自动告警功能。当这些测量结果处于设置的异常值范围内,手机会发出不同级别的告警提示,提醒用户注意休息或及时就医。

医学研究表明,心脏的电活动与机械活动或血液动力学,是相互联系的过程。心脏电活动的改变会引起血液动力学的改变。动态心电图和血压的同步监测,可提供心血管疾病患者的实时动态心电图与血压之间的时间对照。为揭示心电与血压这两种信息的内在联系提供了重要资料。血氧是呼吸循环系统的重要生理参数,很多心脑血管疾病是由于血氧含量不足引起的。心肌对缺氧十分敏感。因此,动态心电、血压、血氧含量三合一检测,为缺血性心脏病、缺氧性心脏病、高血压的诊断提供了新的有效检测手段,为制定动态给药治疗提供强有力的依据。

(3) 物联网智能手机的“一键呼救”功能

当有意外突发时,用户可以按下手机上的紧急救助键、一键接通急救中心,手机也会自动和急救中心接通,而且手机听筒和话筒同时打开,用户在呼救时不用将手机举到嘴边。同时,手机还给服务中心、监护人发送求救短信,求救信息中包含用户所在位置,以便于急救中心根据准确定位及时前来救助,使救护更加迅速、及时,给挽救生命赢得更多的时间。

手机还将支持蓝牙无绳电话功能。如果用户在家里同时安装蓝牙网关,则手机平时当无绳电话使用,避免老年人用户接听电话时行动不便导致滑倒摔伤、扭损肌肉韧带等事故,实现了固网与移动的融合;而当用户出现意外而室内手机信号弱时,手机“一键呼救”自动通过网关接入固定电话拨打急救电话,安全可靠。

(4) 网络平台的电子健康档案库

手机通过物联网自动将血压、心电、血氧等测量结果转发到网络服务平台,以建立个人电子健康档案库。网络服务平台可以提供24小时的自动录入服务。用户及亲属、急救中心、医院皆可以远程登录,随时查询、关注老人的健康状况。该平台还能提供系列健康信息及其他关怀服务,如健康指导、饮食指导、健身指导、各种纪念日提示等。另外,网络平台还会提供绿色食品等与日常生活息息相关的信息等内容。

针对后台数据库,在文件存储方面采取安全周密的备份方案,采用安全可靠的磁盘阵列,具有灾难恢复等技术,从而保证系统的稳定、正常运营;会员档案管理平台,采用安全可靠的加密通道机制访问,避免数据被其他人在中间截取,页面登录采用加密验证码,防止重复登录、暴力登录;完善的权限管理和信息安全设置,对于登录用户,所有能登录的都必须是通过了400电话人工实名认证的会员,并且根据用户类型不同还具有不同的认证和使用方式,如用户需要强制的实名制认证和审核方可使用,并且用户只能访问自己的信息或者家人信息,而不能去访问其他用户的信息,从多方面保证用户隐私信息的安全性。



网络服务平台的服务人员需要 24 小时接听电话，随时提供监测服务，及时处理客户终端发出的救援信息，为急救中心提供必要的健康信息及位置信息，以随时协调救治。

(5) 医疗增值服务功能

本系统可以和医院合作，请有经验的医生、专家根据病人传送到网络平台的电子健康档案提供健康咨询服务以及远程治疗建议。医生可以通过网络平台查询病人的过往病历，了解病史以及心电、血压、血氧等检测记录，从而更好地判断病人的病情以及对病人治疗效果的跟踪。如病人在家治疗，可以直接在网上跟踪病人的血压、心跳情况。

3. 工作流程

(1) 手机和无线设备的工作流程

如图 7-2a 所示，手机先要和无线设备进行连接，连接成功后，用户开始测量，测量的数据能够显示在手机屏幕，测量结束后，手机通过 GPRS 将数据上传到网络服务器。用户也可以通过网页查看测量结果。

(2) 网络服务器的数据处理与服务流程

后台软件按照设定的标准，对接收到的数据进行分析，分析结果保存到服务器，同时可以将结果发送到用户以及监护人的手机上，如图 7-2b 所示。同样，用户也可以通过网页或其他方式定制查看分析结果。

(3) 医疗检测与实时健康报警工作流程

测量过程中，通过测量得到数据判断，如果数据不在正常值范围内，手机会发送报警信息到后台，服务器通过网络将信息发送给客服。客服看到告警信息后，会立即电话联系使用者或者其监护人，如果问题属实且严重，客服会帮助联系急救中心，如图 7-2c 所示。

(4) 医生对病人进行实时监测

用户在测量心电过程中，可以上传数据到服务器。医生通过客户端程序或者网站能够实时浏览用户上传的心电图，对用户出现的问题进行及时、准确的分析。分析结果会保留在数据库中，用户也可以通过网页进行查看，如图 7-2d 所示。

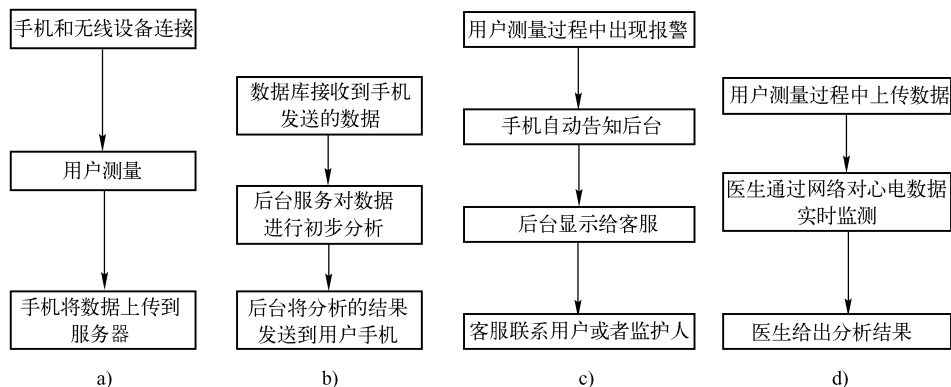


图 7-2 智能家庭远程医疗急救系统部分工作流程图

a) 数据接收 b) 数据分析 c) 医疗服务 d) 监测查询

4. 系统特点

1) 智能家庭远程医疗急救系统是一套融合多项物联网技术的综合应用系统，以呼叫中



心和网络服务平台为中心，与急救中心、医疗机构、健康咨询机构等相结合，集监测、定位、呼救于一体的远程保健救助服务系统。

2) 手机支持与电子医疗设备之间的蓝牙自动群连接技术，能对组成群的电子医疗设备进行自动建立蓝牙连接和断开蓝牙连接，免去用户的烦琐操作，同时蓝牙通信解决了线缆的困扰问题，使得用户携带与使用非常便捷。

3) 实现了动态心电、血压、血氧含量三合一检测，为缺血性心脏病、缺氧性心脏病、高血压的诊断提供了新的有效检测手段，为制定动态给药治疗提供强有力的依据。

4) 将蓝牙网关与手机结合，可以克服 GPS 无法进行室内定位的技术难题，成功解决了事发地的室内环境实时定位问题。手机支持一键呼救功能。意外突发时，只要用户长按此键，手机就给监护人、服务中心发送求救短信并拨打急救电话。

5) 手机通过网关绑定固定电话，平时当无绳电话使用，避免老人接听电话时行动不便导致出现滑倒摔伤、扭损肌肉韧带等事故，实现了固网与移动的融合。当室内手机信号弱时，手机自动通过网关接入固定电话拨打急救电话，安全可靠。

6) 手机从血压计、心电监护仪、血氧仪获取测量数据后实时显示，如血压值、心电图、SpO₂ 值，方便用户 24 小时随时监测并短信通知亲友。同时手机可通过 GPRS 将测量数据传送给网络平台，建立用户个人电子病例。该平台还能提供系列健康信息及其他关怀服务，如健康指导以及饮食指导以及健身指导以及各种纪念日提示等。另外，网络平台还会提供绿色食品等与日常生活息息相关的信息。

7) 网络平台建立个人电子病历供医院和用户随时查询，以实现准确、快速的治疗。采用先进的网络平台后台技术与数据加密技术，确保用户信息安全。

5. 系统组成

基于物联网的智能远程家庭医疗急救系统包括物联网手机、无线血压计、无线血氧仪、无线心电监护仪以及基于物联网的网络服务软件平台等。这些设备及系统形成一个完整的物联网应用示范系统。另外，为了方便读者进一步了解智能医疗，第 6 章设立了相关演示实验，可以通过动手操作甚至编程对本系统有深入了解。系统终端设备的主要参数如表 7-1~7-4 所示。

表 7-1 智能远程家庭医疗物联网手机主要参数

技术参数	说明
网络制式	GSM/GPRS
频率/MHz	900/1800
重量/g	100
体积/mm ³	113mm×55mm×14.8 mm
屏幕/in	2.4 英寸
通话时间/h	3
待机时间/h	300
电池/mAh	980



表 7-2 智能远程家庭医疗无线血压计主要参数

技术参数		说明
显示方式		液晶数字显示屏
测量方法		脉搏扫描法/示波测定法
测量范围	血压/mmHg	20~280
	脉搏/(跳/min)	40~190
存储容量/策略数值		80组
加压方式		全自动加压方式
减压方式		全自动减压方式
排气方式		全自动急速排气方式
测量间隔/mmHg		1
使用温度/°C		+10~+40
使用湿度/%		40~85R.H.
保存温度/°C		-20~+50
使用湿度/%		15~85 R.H.
主机重量/g		约 105
外形尺寸/mm ³		70mm×72mm×23 mm

表 7-3 智能远程家庭医疗无线血氧仪主要参数

技术参数	说明
血氧饱和度显示/(%)	35~99
脉率显示/b/min	30~250
脉搏波显示	脉搏波形
功耗/mA	小于 20
测量精度	当 $80 < SpO_2 \leq 99$ 时, 精度为 $\pm 2\%$ 当 $70 < SpO_2 \leq 80$ 时, 精度为 $\pm 3\%$ 当 $SpO_2 < 70\%$ 时, 精度无定义

表 7-4 智能远程家庭医疗无线心电监护仪主要参数

技术参数	说明
电压/V	3 (2节7号电池)
连续工作时间/h	不小于 30
使用温度/°C	0~40
保存温度/°C	-10~50
体积/mm ³	80mm×20mm×5mm
重量/g	50

7.2 系统二 城市防洪应急指挥系统

城市是一个地域的经济、文化、政治中心和工商业集中地, 人口密度大。纵观历史, 许



多城市靠近滨海、湖或依山傍水修建，都受到不同类型的洪水威胁，而很多城市由于发展较快，在排水防水方面也缺乏系统投入。大多数防洪系统中，部分电站数字自动化程度低，数据依靠人工采集，各自系统独立，难以统一管理和信息共享。系统没有建立统一的空间数据库，没有整合防汛抗旱指挥系统和会商系统。

随着物联网技术的发展，利用传感器技术、无线自组网等物联网技术，全面感知防洪相关数据，实现自动化信息采集，提高辅助应急决策的水平，从而建立基于物联网技术的现代化城市洪水监测与应急指挥系统。本应用系统中所讨论的基于物联网的城市防洪应急指挥系统集成城市洪涝的预警、监测及灾后应急指挥于一体，是城市保障系统的重要组成部分。

1. 系统概述

近年来，我国城市化的步伐不断加快，但城市防洪指挥系统多年来却一直没有获得足够的重视，技术手段也显得相对落后，主要包括以下问题。

1) 部分电排站自动化程度低，数据大都靠人工采集，可靠性低。管理系统各自独立，数据的一致性管理难度大，共享程度低，严重影响科学系统的应急使用，而物联网的感知技术，通过各种传感和标识技术的应用，正好可以解决这一难题（注：电排站是通过水泵来排除低洼地带的积水，防止内涝发生的水利设施）。

2) 空间感知数据没有整合进防汛抗旱指挥系统，即没有建立统一的空间数据库，没有全面发挥 3S 技术（遥感技术（Remote Sensing, RS）、地理信息系统（Geography Information System, GIS）和全球定位系统（Global Positioning Systems, GPS））在防汛指挥中的积极作用。通过全面使用物联网及云计算技术，可以有效完成海量感知数据的管理与检索，不但可以积累大量有效信息，更是大大提升了它们的用武之地。

3) 各个应用系统所涉及的范围窄，有待拓宽，且相互独立，不协同工作。没有基于 3S 技术架构的防汛会商系统，目前的系统可视化程度不高，供所需的信息还不够全面，信息显示的方式没有利用 GIS 技术来增强展示效果。物联网的应用层可以提供有效的用户界面及数据共享方式，大大提高了整个系统的可用性。

本节介绍物联网城市防洪应急指挥系统的一种实现方案，对电排站进行自动化升级改造并加装无线传感器设备（流量计、水位计、温度传感器、电流电压传感器等）、PLC 编程，并开发部署物联网城市防洪监测预警和应急指挥平台，城市洪水淹没演进分析系统，防汛会商指挥系统。科学地确定洪水淹没范围和水深分布，为洪水风险图制作、防洪指挥调度和洪涝灾害的损失评估提供准确的评判依据。在利用各种水利数据的基础上，提供灾情预报、防洪防涝措施、调度预案、评估、灾害评估、会商决策支持等。

2. 系统架构

基于物联网的城市防洪应急指挥系统平台基于传感器、视频、遥感等物联网感知技术、城市宽带无线通信技术、大型综合数据库和 Web 技术、GIS 技术等，采用多层架构体系，开发应用支持平台实现数据高度共享及预警分析（如图 7-3 所示）。

系统集成实时雨水情检测预报、防洪排涝措施、调度预案、评估、灾害评估、防汛会商决策支持、数据汇集、内外河水位检测、预警、响应等功能于一体。集成系统平台通过实时雨水情预报分析内外河水位是否已超过警戒水位，同时通过防汛会商决策支持平台去做相应响应流程的处理解决，比如内河水位超警戒，将水位数据及流程交与会商解



决平台分析，会商平台则结合多方水位数据（内河水位、外河水位、最近水库水位），并通过定制解决数据模型，找出最好的排涝方式，是开闸放水，还是泵站抽水等操作。数据由各子系统提供，决策处理由会商平台来处理，同时可以利用多方数据做出洪水淹没演变过程。

3. 系统组成

基于物联网的城市防洪应急指挥系统可以为城市防洪提供重要保障。城市防洪支持系统涵盖了信息采集、信息传输、信息集成、信息加工、方案生成、方案评估、方案发布、决策实施及信息反馈等内容，是防洪非工程措施中比较重要的部分。本系统可以分为 5 个紧密联系的子系统。

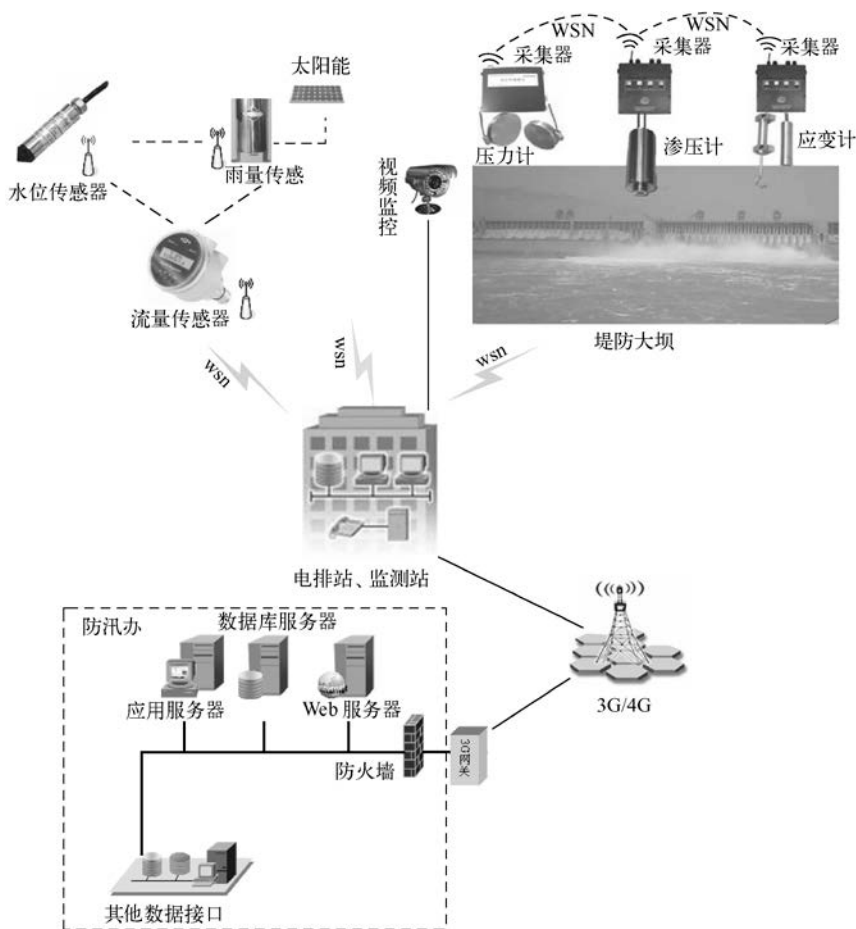


图 7-3 物联网城市防洪应急指挥系统网络架构

(1) 城市防洪应急指挥系统感知前端

系统前端主要通过多种传感器，采集与防洪相关的物理世界的信息，然后根据前端信息来控制电排站。前端主要感知的信息包括以下方面。

- 电排站设备信息：设备基础信息、模拟量、开关量、脉冲量、数字量、信号量值及状态设定等。



- 雨水情信息：雨量、流量、内河蓄水量、进出流量等。
- 工情信息：大坝压力、应变参数、渗压参数等。前端系统中使用的传感器有水位传感器、流量传感器、渗压计、应变计、压力计。

水位的测量通常采用浮子式水位计、超声波水位计、压力式水位计、气泡式水位计；而浮子式水位计因其仪器结构简单，性能稳定，工作可靠，得到广泛的应用，如图 7-4 所示。



图 7-4 浮子式水位计

流量传感器分为 4 类：超声波管道流量计，测量技术的核心在于超声波传播时间的测量；旋涡流量计，通过检测流量管中的旋涡个数来对过程流体进行流量测量；靶式流量计，通过流体队靶的力感应技术来对过程流体进行流量测量；电磁流量计，依据电磁感应定律工作，当导电液体在磁场中切割磁力线时，将产生与流速成正比的感应电势信号。上述 4 种流量计，以电磁流量计结构简单、安装方便、性能稳定、工作可靠，得到广泛的应用。

无线硅晶片渗压计是设计用于精确测量完全饱和或不完全饱和岩土中的孔隙水压力，用于基水坝中的孔隙水压力监测，渗压计顶端由透水石和与之相接的硅晶片振动膜传感器组成。应变计可直接埋设在大坝的混凝土内，用以监测混凝土内部的应变，内置的温度传感器可同时监测安装位置的温度。压力计用来测量填土和堤坝等软基的土体压力，主要部件均选用特殊钢材制造，适合在各种恶劣环境中使用。

(2) 城市防汛会商信息系统

城市防汛会商系统由防汛会商核心系统、天气预报子系统、卫星云图子系统等子系统组成，它是整个系统的智能决策中心，可以提供多种定制服务。

1) 防汛会商核心系统。防汛会商核心系统包括了在线查询、工程图编辑和系统管理 3 大功能。

在线查询根据查询对象不同分为 4 类：河道水情查询（图 7-5）、雨情查询（图 7-6）、水库水情查询（图 7-7）和工情查询（图 7-8）。在线查询主要是由业务层的查询组件、等值线组件、地图组件等完成，往下调用通用模块，由业务模型提供查询的依据和标准。在线查询要求快速响应、查询结果准确；系统还需要提供各种错误处理，比如查询条件输入不合理或者查询结果异常等，保证在错误操作的情况下系统仍能稳定运行。在服务器和网络状况良好的情况下，可以实现查询的快速响应，各种业务组件和通用组件也易于移植、维护和扩展。

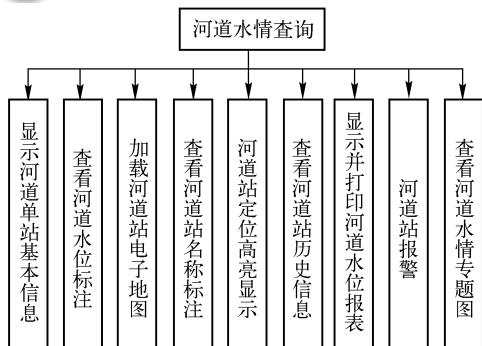


图 7-5 河道水情查询

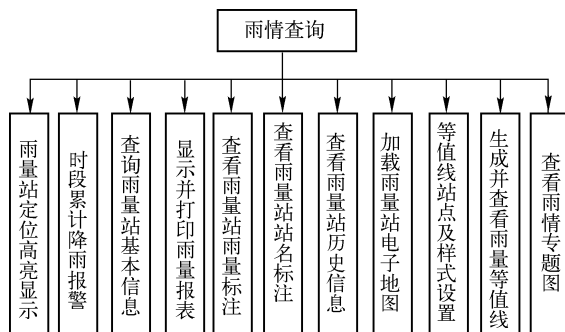


图 7-6 雨情查询

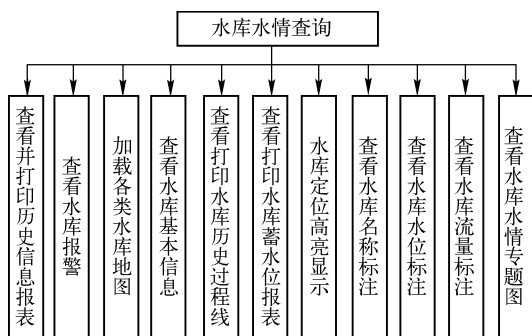


图 7-7 水库水情查询

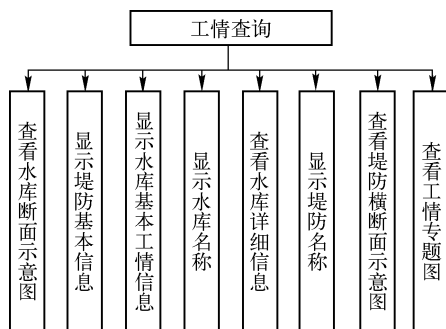


图 7-8 工情查询

工程图编辑和系统管理功能给用户提供了数据更新、维护、管理以及系统权限管理等接口（如图 7-9 所示），以方便用户进行日常维护。

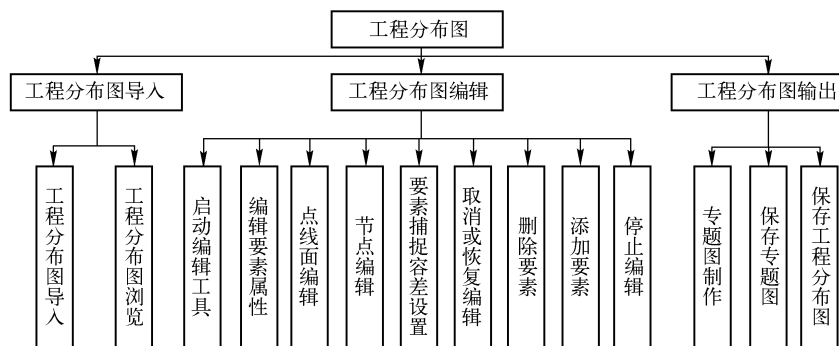


图 7-9 工程分布图管理功能分解图

2) 天气预测子系统。天气预测子系统包括天气预测、台风路径采集和降水分布预报图等功能（图 7-10）。台风路径图、降雨强度预报图、降水预报图等天气预测文件由天气预测服务器采集。该系统主要根据用户输入条件查询服务器上的天气预测文件，对其进行演示或显示。

3) 卫星云图子系统。卫星云图子系统作为一个独立的子系统集成到“基于 3S 的防汛会商系统”中（图 7-11）。由卫星云图接收系统提供云图结果文件。该开放给本系统读



取。该系统用于采集、管理和显示卫星云图以及动态演示卫星云图变化过程。

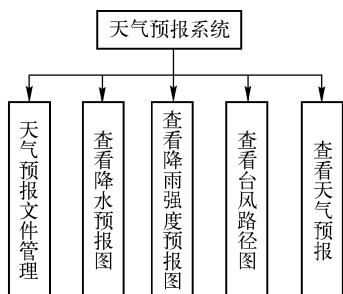


图 7-10 天气预报系统功能分解图

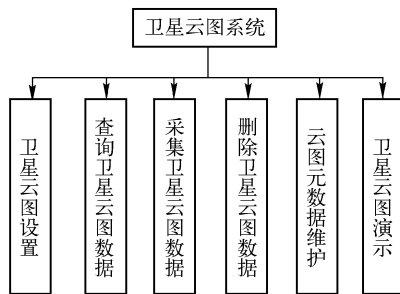


图 7-11 卫星云图系统功能分解图

(3) 城市防汛会商分析决策支持子系统

该子系统建立了基于地理信息技术的实时性强、可视化程度高的洪水淹没分析地理信息系统，对防洪工程图、流域地形地貌、淹没范围信息进行分析与管理；以大比例尺电子地图和数字高程模型（DEM）为基础，将地理信息的三维可视化、空间分析以及连通性分析等技术和淹没分析模型相结合，应用到洪水淹没范围的分析以及淹没过程动态模拟中，能够更准确、科学地确定洪水淹没范围和水深分布，为洪水风险图制作、防洪指挥调度和洪涝灾害的损失评估提供准确的评判依据。

(4) 城市电排站管理子系统

该子系统包括以下 7 大管理模块。

1) 排涝泵站及涵闸管理模块。通过内河水位监控设备，获取内河水位数据，通过 GPRS 等其他通信传输方式，将数据发送到系统平台上，如果发现内河水位数据较高，则系统管理控制泵站及涵闸的相关设备启动，进行排涝工作。

2) 工程控制管理模块。在系统后台添加设置排涝站及涵闸控制条件，保证积水尽快排出。比如，在系统上设置自排条件，当降雨发生时，如果外河水位监控设备将监控到外河水位数据发送到系统平台上，系统平台分析发现外河水位较低，则通过自排将涝水及时排出且不影响农作物生长的，采取自排方式，监控管理密切提取外河水位变化情况的数据，系统管理人员可以通过平台上的数据展示，看是否外河水位已经超出，当外河水位上涨，不符合自排条件时，要立即由系统发送信息关闭排水涵闸，以防倒灌。当汛期水位超过警戒水位，开启进水涵闸，必须以公文的形式发送报批手续。启动排涝泵站排涝时，泵站工作人员要严格检查涵闸是否关闭，注意机械运行状况，做好机械行车纪录，确保机械正常运行。如果气象部门预报有特大暴雨发生，应适时腾空底水，尽可能减少内涝损失。

3) 堤防管理模块。当地方管理人员发现堤防需要维修时，通过乡镇、街道管理人员登录系统平台然后填写申请；水利部门在平台上看到该信息，则安排相关人员进行勘察设计，经审批后，由堤防所在地政府按照项目管理要求组织实施，维修完成后，通过验收，按照工程决算结算维护补助费。

4) 内外河水位监控管理模块。主要是排水量、内外河水位实时监控，包括：对泵机、闸门等设备的运行参数进行实时监测，有异常情况发出告警，告警以邮件或短信的方式自动



通知相关人员进行检修；对各泵站的前池水位、排水量、内外河水位等进行监测。通过各种通信方式将数据发送到系统平台上，对没有按规定（如时间、格式）上传（人工或自动）数据的电排站发出告警，告警以邮件或短信的方式自动通知相关人员，然后系统平台结合数据进行各种相关操作设置，比如系统管理人员可以通过数据启动设备。系统可以按区域、电排站名称、时间等条件进行查询检索，并可以打印、导出查询结果。（监测以电子地图、表格等方式进行显示。）

5) 排涝泵站及涵闸管护人员的管理模块。实行分级管理体制，成立排涝管理办公室，有关乡镇、街道也要相应成立排涝管理专门机构，分级分段负责；乡镇、街道排涝泵站、涵闸，按照属地管理原则，由工程所在地的乡镇、街道及村（居）委员会管理；各堤防由所属乡镇、街道管理。

6) 系统后台设置管理模块。日志管理：对管理人员的操作步骤记录数据库，作为系统及数据维护的依据。短信管理：当发生安全问题时，通过短信发送到各级别的管理机构及人员。

7) 数据维护管理模块。校对不正确的数据（增、删、改），但要在数据库中进行标注。

（5）城市内涝预报预警子系统

该子系统自动收集自动站降水资料，对城市水利工程、城市排水能力、水雨情信息等信息进行有效管理

根据城市排水能力、降水量值、降水强度、降水持续时间等因素进行分析，若短时间内出现强降水而且达到一定量时，系统就会对城区内涝做出短时预报，并结合水文局提供的洪水预报信息，对洪水进行警示，在电子地图上标示相关的行政区域和所涉及的社会经济信息。

4. 系统特点

本系统引入传感器、无线自组织网络、智能决策、云计算等物联网相关技术，实现在无移动通信信号地点的数据感知、传输、管理与智能决策的问题。提高水情管理水平，将信息技术更广泛地应用于水利管理日常业务工作中，实现水利数据采集的全面覆盖及自动化、管理工作的规范化和决策科学化。

7.3 系统三 智能家居示范系统

智能家居系统在很多物联网的书籍中都有所提及，被认为是与人们生活最近的物联网应用。本节简单介绍一种物联网智能家居系统的示范方案，主要功能包括面向家居应用的电机控制、设备状态监测、家庭网络构建、智能家居服务等。智能家居的需求各异，但其物联网的关键技术却大同小异，实际应用方案可以根据具体情况进行定制。

1. 系统概述

智能家居是以住宅为平台，兼备建筑、网络通信、信息家电、设备自动化和传感器网络，集设备、系统、结构、服务、管理为一体的高效、舒适、安全、便利、环保的居住环境。基于物联网的智能家居通过无线/有线多种接入方式接入外部网络，通过家庭内部网络的有线/无线方式构成家庭内部网络，外部网络和内部网络构成智能家居服务泛在化的网络



基础。智能家居系统是物联网应用的重要舞台。

智能家居目标是通过融合的网络环境跟各种物联网应用服务建立连接，通过统一的业务支持平台提供的标准化业务接口获取各种服务。提供通过多种终端如触摸屏、多功能遥控器、手机等的简单可靠操作，有效完成内部家居的管理和对外交流。

智能家居具有丰富的外部网络接入方式，有线方式可支持 PSTN、FTTx、ADSL、VDSL 等；无线方式可支持 GSM、CDMA 1x、GPRS、WCDMA、TD-SCDMA、CDMA 2000、LTE、WLAN、WiMAX 等。智能家居内部网络连接有线方式可支持 LAN、X-10 等；无线方式可支持 RFID、WLAN、Bluetooth、ZigBee 等。

2. 系统架构

基于物联网的智能家居系统内部组网需要控制的设备多，网络容量大；传输信息延时性、实时性要求复杂，包含高速、中速及低速的数据设备；兼容广泛的连接技术；保证高级别的安全性。智能家居内部网络组网结构如图 7-12 所示。

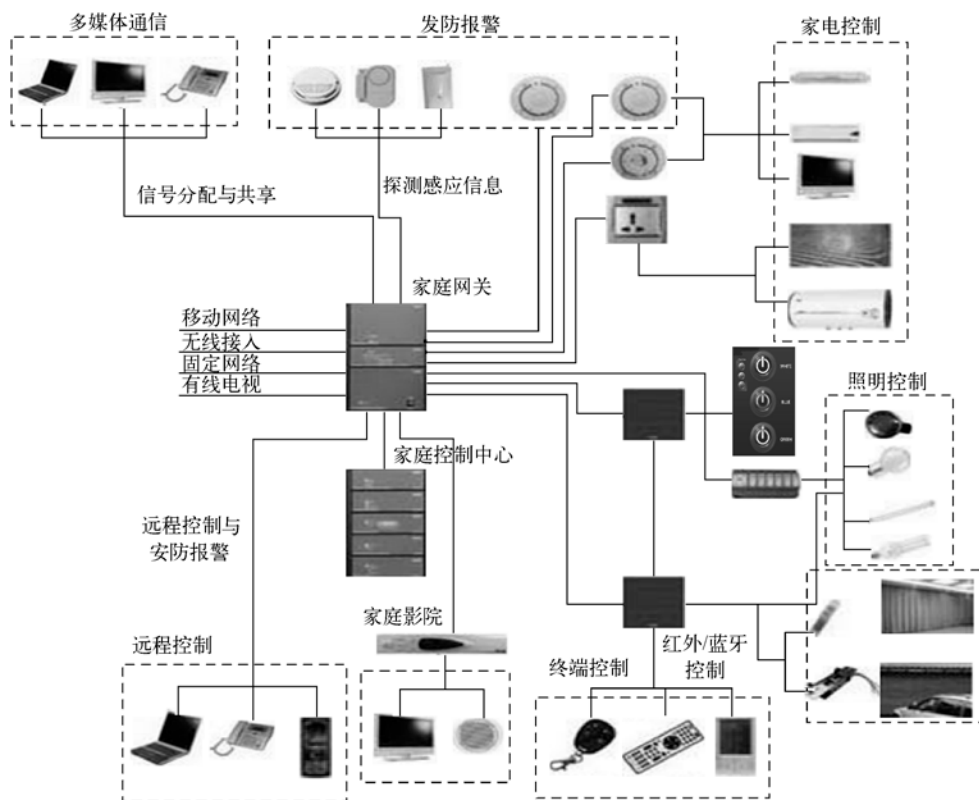


图 7-12 智能家居内部网络组网结构

在图 7-12 所示的家庭网络组网结构中，家居设备、移动终端以及其他域设备通过网关实现信息互通及协议转换、信息共享和交互；高速有线网络结合无线互联，支持 QoS、UPnP、DHCP 等关键技术，使各种智能家居设备可实现自动、无配置的接入系统，极大提高系统的灵活性、易用性及可扩展性。



3. 工作流程

基于物联网的智能家居业务流程支持几种方式，具体可以从图中较清楚地看出其具体操作流程，由于篇幅的关系，这里就不一一介绍了。

1) 本地终端设备访问本地终端设备（业务平台不参与），如图 7-13 所示。

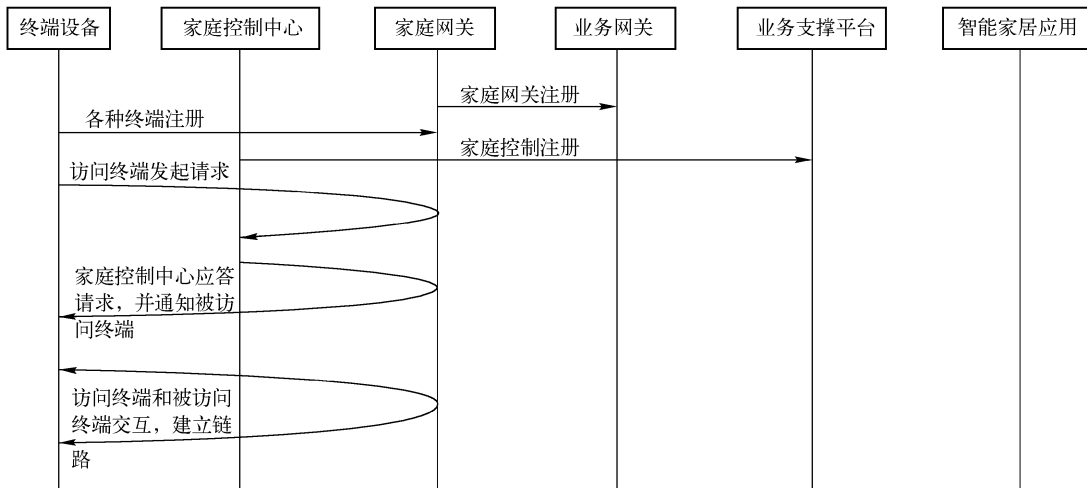


图 7-13 智能家居系统本地非业务平台流程

2) 本地终端设备访问本地终端设备（业务平台参与），如图 7-14 所示。

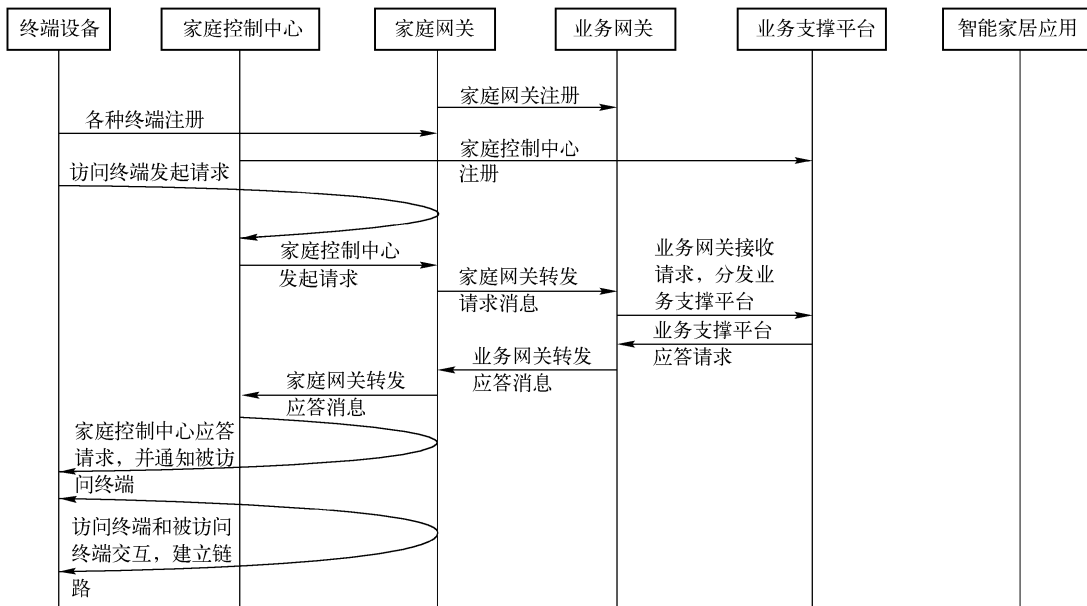


图 7-14 智能家居系统本地业务平台流程

3) 本地终端访问远程终端，如图 7-15 所示。

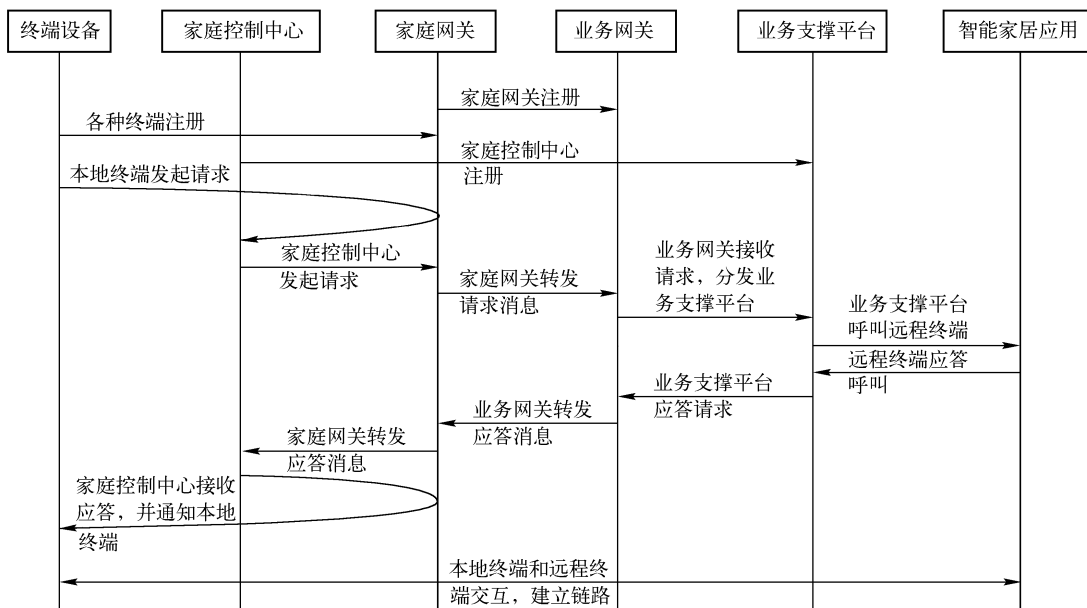


图 7-15 智能家居系统本地访问远程流程

4) 远程终端访问本地终端, 如图 7-16。

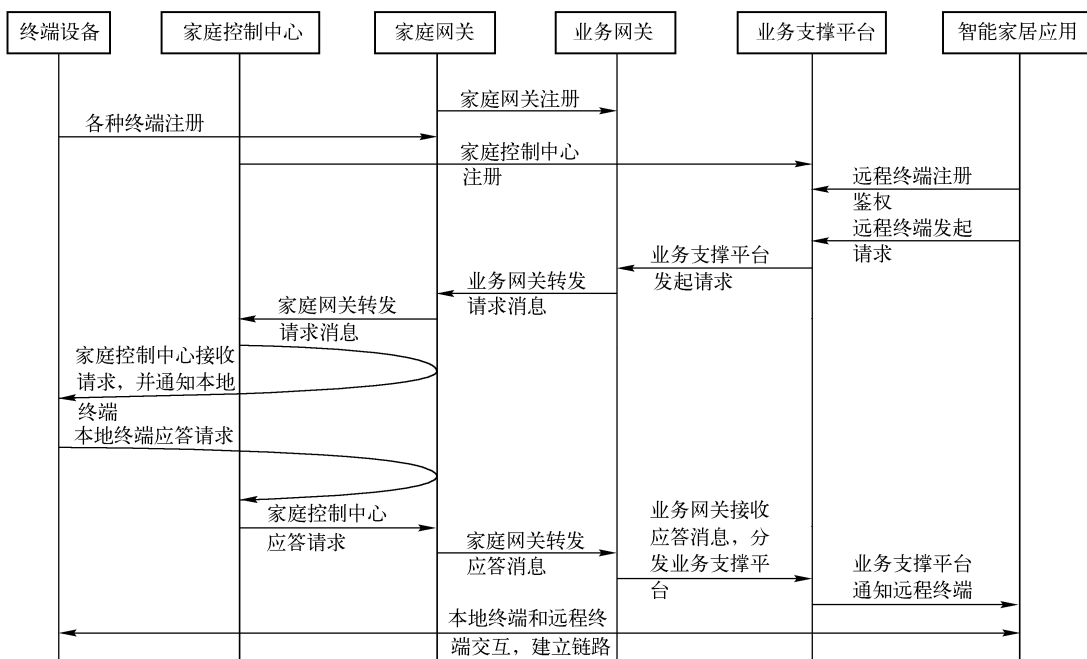


图 7-16 智能家居系统远程访问本地流程

4. 系统特点

基于物联网的智能家居组网采用分层体系架构, 同样可以划分为感知层、网络通信层、业务层等几个层面。



(1) 感知层

智能家居感知层包括了各种与家电、家居相关的传感器、控制器、执行器及识别装置等，有线网络结合无线泛在网络的物理连接。这一层还包括不同接入方式的 MAC 子层和链路控制子层，对上网络层提供统一的接口，屏蔽异构网络的差异；进行不同形式家庭通信网络间的 MAC 协议数据单元(PDU)映射，以支持不同网络间方便互通；支持动态智能的有线及多种无线网络的接入及选择。

(2) 网络通信层

智能家居网络通信层主要包括家庭内部的网络与与骨干网络的接入两个部分。家庭内部组网支持的有线方式包括电子载波 X-10 和 CEBUS、电话线 HomePNA、以太网 IEEE 802.3 和 802.3u、串行总线 USB 1.1、USB 2.0 和 IEEE 1394 等；无线方式包括无线局域网、家庭射频技术、蓝牙、红外、ZigBee 等。网络接入层通过家庭网关与业务网关，实现不同应用协议规范的互联互通互操作，并与骨干网络实现无缝连接。

(3) 业务层

智能家居业务层是以用户为中心的融合业务层，提供用户接口、不同制式的智能家居服务等。通过智能家居组网多层协作的自适应 QoS，自适应匹配异构网络及终端设备，充分保证端到端的媒体、语音、安防等多种业务的服务。

7.4 系统四 智能闸口示范系统

随着科学技术的高速发展、贸易壁垒的消除、全球化进程的加快，物流领域越来越成为现代信息技术应用比较普遍和成熟的领域，物流企业正在转变为信息密集型企业，物流信息化成为现代物流业的灵魂，是现代物流业发展的必然要求和基石。港口物流作为物流领域的中心枢纽，在物流的各个环节中扮演着核心的角色。我国港口经济建设发展正进入一个崭新的阶段，中国已成为世界上港口吞吐量和集装箱吞吐量最多、增长速度最快的国家。港口的发展（特别是大型集装箱港口）是我国经济发展的基石。

物联网技术是物流信息化最重要的技术手段，借助物联网智能采集、传输与处理等技术手段，物流企业的信息化水平可以显著提升，实现目达耳通的智慧物流管理，降低物流成本，提升物流效率。闸口是港口的核心和中心，也是影响其效率与安全的最重要的环节，本示范系统以全自动化智能闸口为应用背景，介绍物联网在港口物流中的应用模式，本节只是对应用系统进行介绍，关于详细的智慧物流，读者可参阅《物联网技术应用》。

1. 系统概述

闸口一般特指港口的通关道口，这里也泛指所有物流转运枢纽。闸口的管理是整个物流管理中的核心要素，直接关系到整个物流链的效率与安全。我国目前各大港口、物流园区、转运站及海关，都拥有规模很大的通关闸口，但其信息化方案及水平很不一致，各闸口在功能设计上也是差异很大，系统中都存在着相当多的人为干预，急需物联网技术的深入应用。传统的基于纸质单据的通关系统存在以下致命问题。

(1) 系统效率低下

由于进关过程需要多次的纸质信息交互，并需要打印很多纸质凭证，使闸口的处理能力



严重下降，大量等待入港的车辆在闸口处滞留，货物在码头内停留时间过长，严重降低了集装箱调度周期，降低了服务质量，使闸口成为物流效率中最大的瓶颈。需要引入高效率的识别与感知技术来提升系统效率。

(2) 系统安全性不足

由于没有先进的车辆识别技术，导致一些套牌车辆出现，在闸口处蒙混过关，非法进入码头偷运货物。人工干预过多（如通过肉眼识别车牌和箱号）导致一定的出错概率，而纸质单据的交互过程还存在伪造、抵赖、丢失、遗忘、遗漏等问题。基于物联网的信息化不但可以解决安全性问题，更可以大大提升系统的可靠性。

(3) 系统可扩展性差

随着闸口 OCR 系统、称重系统、智能拍照系统等逐渐建立，电子关锁、智能堆场等系统的实施，目前传统的作业方式已不能适应现代化港口的要求，需要从应用层提升系统的可扩展性，建立统一的应用服务平台，提供统一的服务接口。

基于物联网的智能闸口方案可以彻底解决以上问题，可以使集卡车入港速度显著加快，明显提高码头的货物吞吐率，提高码头效率和效益；系统独特的车道识别能力能准确读取对应车道上的车辆信息，保证起落杆的正确开启；全程采用先进的加密算法和确认逻辑，保证系统的安全性、可靠性，杜绝了非法车辆的偷运；自动的工作方式使人为出错的概率降到了最低，防抵赖机制使各环节的工作更加透明；智能闸口系统不仅可应用在闸口处，还可使用先进的无线通信技术，在作业堆场内部实现作业引导、集装箱定位、电子关锁等功能，在码头的现代化建设中发挥重要的作用。

2. 系统架构

基于物联网技术的智能化闸口系统通过对车辆、调度、作业、控制等系统全面联网，并使用安全高效的无线车载终端设备，免去一切纸质信息的交互。车队通过互联网从电子数据交换中心获得预约号后，将预约号以无线方式下发到车载终端，安装有车载终端的集卡车辆在进入到闸口时，将预约号、车牌、码头号等所有相关信息传送到闸口读写器，读写器通过和码头系统互连，将码头产生的行车指南通过无线发送到车载终端，司机只要通过车载终端上的智能终端设备就可知道入港后的作业位置。该方案可以从根本上消除人工干预过程，全程实现自动化。本系统已经在宁波港下属的各大码头及中铁联合国际集装箱有限公司（简称中铁联集）相关中心站成功运行，实际效果印证了设计初衷。本系统可分为 5 部分，每部分的具体功能如下：

1) 智能闸口中心数据库：整个智能闸口系统的中心统筹管理智能车载终端，通过 RFID、OCR 等技术，记录各个地点作业的时间信息和重量信息等，与 EDI 系统交互数据，中心数据库具备查询、检索、添加、删除、统计等功能，方便各方对各类数据的要求。

2) 智能车载识别终端系统：包括 RFID 车牌和无线通信、智能交互等功能，负责智能车载终端的注册、分发。可以在频繁进入港口的车上安装智能识别终端，在临时车上安装带座子的终端并内置锂电池。回收临时租用的终端，与长期设备的年检等配合。

3) 预约号、码头号等预录入信息分发系统：能够通过互联网或内部专网从 EDI 获得预约号、码头号等预录入信息，通过移动通信网络以无线的方式下发给对应的车载终端，并将车辆 ID 和预录入信息的绑定信息回传到中心数据库。



4) 闸口入口系统：司机能通过车载终端上的按键和显示屏操作，从码头系统获得本次作业的行车指南，闸口上的地磅和 OCR 等系统联合工作。而车道检测装置，可以准确区分车道，将行车指南下发到对应的车载终端内。集卡车在收到行车指南后根据此信息进入港口作业，并将时间、车牌等信息上传到智能闸口中心数据库。

5) 闸口出口系统：集卡车在完成港口内部作业后，到达闸口出口，此时闸口设备会检测到车辆，并通过后台比对车辆是否允许出港，如果是，则自动删除车载终端内部的行车信息和预约信息，并将时间、车牌等信息上传到智能闸口中心数据库。

3. 工作流程

传统闸口系统的业务流程在不同的国家、地区或港区都有很大的差异，特别是很多人工参与环节较多的场合，更是形形色色。本节所讨论的基于物联网的智能闸口系统的工作流程可按智能车载终端的数据流向来描述，如图 7-17 所示。

智能车载识别终端系统：该系统不但集成了 GPS/GPRS 系统，也集成了车载无线 RFID 识别和有源数据收发系统，如图 7-18 所示，初次使用必须将智能车载终端注册到系统中。长期使用通过支架固定在车辆驾驶台，并接入车载电源。如果临时使用，则在终端上携带吸盘装置，可方便吸附在挡风玻璃上或其他方便的临时位置。另外，整个硬件系统的安装还包括天线、读写器、控制器等，这一切都要充分考虑到系统的性能与方便性。

预约号、码头号等信息分发：如图 7-18 所示。本作业的主要目的将预约信息下载到智能车载终端内，为车辆入港时做准备。当然，如果智能车载终端已经嵌入 GPRS 模块，则可在任何地点、任何时间操作。

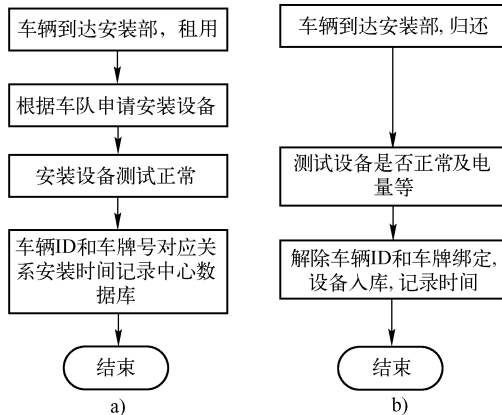


图 7-17 智能闸口系统工作流程

a) 租用流程 b) 归还流程

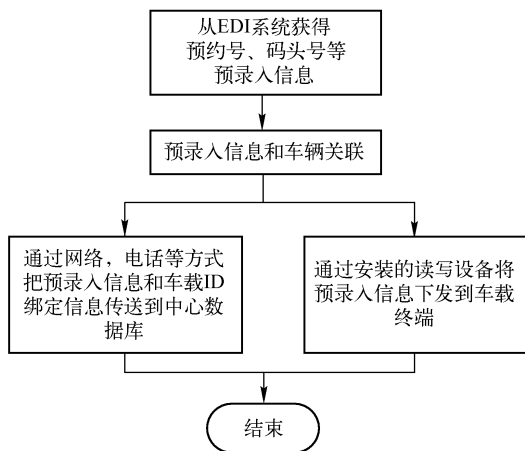


图 7-18 智能车载识别终端系统工作流程

闸口入口识别：如图 7-19 所示，当安装有智能车载终端的车辆到达入口时，车辆检测可以激活读写器工作，将车载终端内的数据读取到码头系统并产生行车指南，行车指南通过无线方式发送到车载终端，司机也可根据车载终端上的按键来进行查询和回复操作，每项操作的时间都会记录到中心数据库以备各种统计查询需要。系统具备很高的实时性，确保车辆在入港前能获得行车指南。

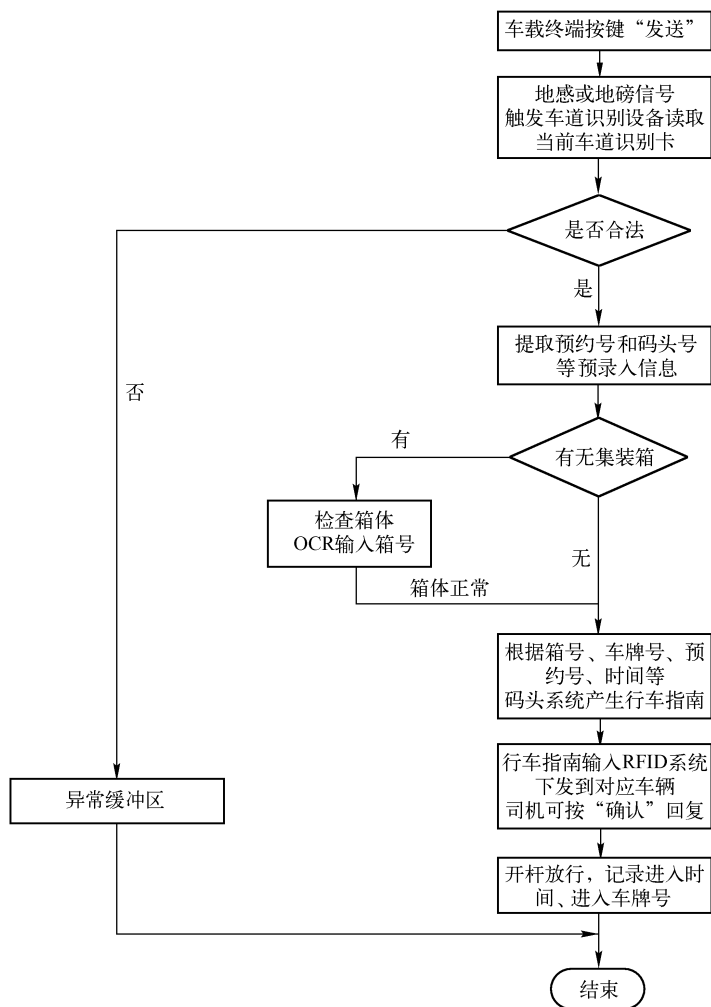


图 7-19 智能车载终端入口工作流程

闸口出口识别：车辆在港口内部完成作业后准备离港，离港时读写器需要读取车辆信息（如图 7-20 所示），判断是否允许离港，并删除终端内的作业信息，而系统还会采集车辆的重量信息并记录到中心数据库。

4. 系统特点

本智能闸口系统通过超高频无源 RFID 完成了车辆及人员识别，并将倒道闸系统、地磅系统、摄像系统、箱号及车号识别系统等进行了互联，并和智能物流的中心调度系统进行接入，形成了一个功能全面的物流网系统。本系统是物联网的典型应用，具有以下特点。

1) 远距离识别。系统通过使用较高性能的无源超高频 RFID 装置，典型识别距离可达到 12~15m 以上，可以方便多种类型的大型闸口系统。智能车载识别设备可集成大容量的片内存储器，存储容量从 4096 字节可一直扩展到几十千字节甚至上兆字节，可进行多种应用扩展，并可以为进一步安全加密留有余地。

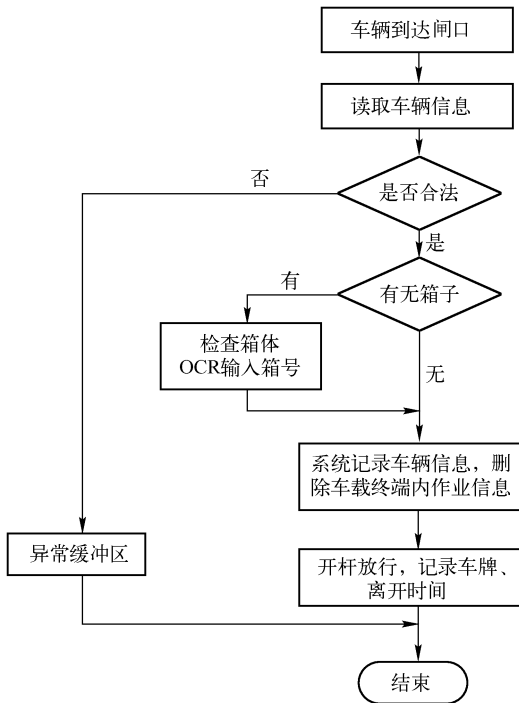


图 7-20 智能车载终端出口工作流程

2) 防邻道干扰。本系统使用的方向性天线，能够很准确地控制读取角度，保证在道口上的几个车道间不产生邻道干扰，保证每个安装在车道上的读写器读取到的都是本车道上的 RFID 识别终端。

3) 设备高速读取。智能车载终端的读取速度只要几十毫秒，这样就保证了当车辆在行驶过程中可以有效读取，而不需要车辆停在读取区域内。快速读取带来的最大好处是，车辆不用在道口排队等候，彻底解决道口阻塞问题，提高码头的吞吐量，增加了经济效益。

4) 系统安全。智能车载终端和读写器之间采用加密技术，可以保证数据在传输过程中的安全，不能随意被破解。而智能车载终端的 ID 号都为全球唯一，也就避免了被随意地复制和修改。使用专有无线信道，不受其他无线信号干扰。

5) 系统可靠。智能车载终端由于采用无线识别，读取时不存在终端和读写器的磨损，避免了由于接触而产生的各种故障。智能车载终端安装在车内后，不受天气、温度、湿度、振动等外界条件的影响。

6) 可扩展。智能车载终端具备主动的无线通信能力，能结合车载 GPS、电子关锁等各类通信设备，实现更广范围内的车辆、货物管理。

7) 操作简便。RFID 由于使用远距离读取，并且配合定向天线覆盖整个车道，只要车辆经过就可以完成识别，可极大方便司机作业及港口的管理。以往需要纸质打印的行车指南，通过电子方式显示到 RFID 的车载终端，更加直观、方便。车载终端上包含各种提示符，能帮助司机解决各种行车需要。

5. 系统组成

核心设备：智能车载识别终端、发卡器、读写器、控制器及天线。其主要参数见



表 7-5~7-8。

辅助设备：车辆检测器、道闸、计算机。

软件系统：智能物流综合调度系统、智能闸口中心管控系统。

表 7-5 智能车载识别终端主要参数

技术参数	说明
电源	8~35V 车载直流电源或大容量充电电池 1500MA/H
天线	内置
容量/KB	4 (可扩展)
反复擦写次数	>100000
人机系统	带背光 LCD(一屏 48 个汉字), 3 个按键
工作温度/°C	-10~+55
储存温度/°C	-20~+85
使用寿命/年	大于 3

表 7-6 智能闸口专用发卡器主要参数

技术参数	说明
通信接口	RS-232
电源	DC 9V 电源适配器
功耗/W	<2
工作温度/°C	-10~+55
工作状态提示	蜂鸣器、通信指示灯、电源指示灯
使用寿命/年	>3

表 7-7 智能闸口专用读写器主要参数

技术参数	说明
天线	可外接天线 5 个、SMA 接头
读卡方式	定时自动读卡、外触发控制读卡或软件发命令读卡, 读卡方式可设置
通信接口	RS-232、RS-485、Wiegand26
电源	DC 9V 电源适配器
工作温度/°C	-10~+55
工作状态提示	蜂鸣器、天线指示灯、通信指示灯、电源指示灯、读卡指示灯、RF 发射指示灯
使用寿命/年	大于 3

表 7-8 智能闸口专用控制器主要参数

技术参数	说明
CPU	ARM7
通信接口	RS-232、RS-485、10M 以太网
电源	DC8V~24V
功耗/W	<1
工作温度/°C	-10~+55
储存温度/°C	-20~+85
系统接口	2 路输入、2 路输出
储存温度/°C	-20~+85



技术参数	说明
工作状态提示	蜂鸣器、通信指示灯
使用寿命/年	大于3

7.5 系统五 海量地址匹配与搜索系统

在现今社会信息量急剧膨胀的情况下，对信息进行智能分析、搜索、加工和匹配越来越受到人们的重视，尤其是物联网的规模应用，会导致感知数据的爆炸式增长。在物联网应用中，信息的智能处理尤其是地址信息的分析处理，能够自动分析物体与物体之间的距离和位置，进行物体的智能定位与跟踪，成为物联网技术领域必不可少的组成部分。

地址匹配与搜索系统可直接应用于邮政、银行、保险以及广泛的物联网相关服务，也可以在此基础上，提供进一步的地址标准化、地址信用以及地址一致性判别等服务。本应用示范系统以实际案例，介绍地址匹配与搜索技术在物联网中的应用。本系统主要基于海量感知数据做出，充分体现了在物联网应用中的智能搜索技术。

1. 系统概述

地址匹配与搜索系统可以快速便捷地匹配输入的任何地址片段，给出匹配出的最佳结果，并有针对性地给出计算依据。地址匹配与搜索系统充分融合了自然语言理解、信息检索等方面的核心技术，具有智能、高效、自学习 3 大特点。图 7-21 所示为中文地址匹配系统架构。

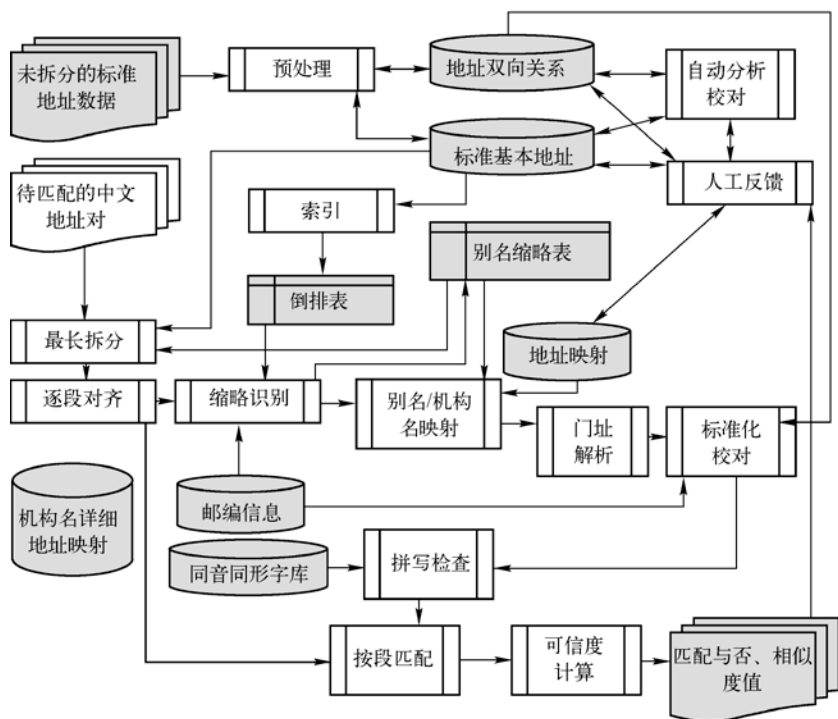


图 7-21 中文地址匹配系统架构



① 智能：主要体现在系统可以智能识别用户输入的真实意图，智能模糊匹配后台知识库，进行逻辑推理，并给出逻辑推理的知识依据。

② 高效：体现在本系统可以单机每秒处理 5000 条记录。

③ 自学习：是指整个系统无需人工干预，直接导入正确标注的邮编数据，即可完成整个系统的学习，自适应地调整处理结果。

2. 系统架构

图 7-22 中基本概念的定义如下。

● 地址双向关系：指的是地址区划之间的上下位关系。比如“海淀区”，其上位地址为“北京市”、下位地址有“中关村”、“黄庄”等。地址双向关系实质上表示的是地址树形结构的双向关系，如图 7-22 所示。

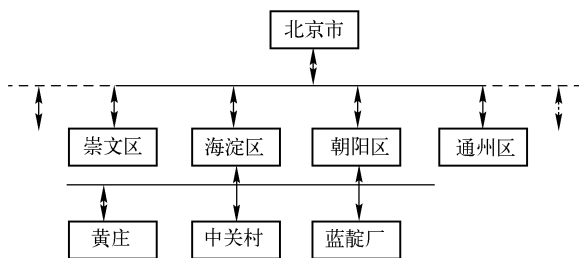


图 7-22 地址树形结构的双向关系

- 预处理：包括标准地址的自动分析校对、各级地址的索引、标准地址的生成以及双向关系的建立等。
- 索引：按字倒排索引。
- 最长拆分：按照“长地址优先”的原则对待拆分的地址进行拆分处理，输出按照级别解析拆分。
- 地址对齐：按照标准地址库及其级别定义，将两个地址按照级别进行对应比较。
- 缩略识别：缩略的地址名、机构名识别匹配，最终匹配出对应的标准名称。
- 别名解析：各种别名的对应转换。
- 门址解析：各种门牌号及缩略形式的标准化解析，如“2-3-12”匹配“2 门 3 单元 12 号”。
- 标准化校对：主要是针对标准地址深层次的关系校对，如“北京市浦东区”内在的错误关系。
- 拼写检查：主要处理同音、同形、顺序颠倒的汉字拼写错误。
- 可信度计算：结合各个阶段的处理结果，进行综合打分。
- 人工反馈：结合人工的检查反馈，进行错误驱动的自学习。

3. 主要功能

- 邮编智能匹配：输入地址片段，自动模糊匹配，计算出该地址所对应真实准确的邮政编码。
- 结果判别：根据实际情况，依次给出输入有误、匹配上唯一一条邮编，匹配上多条邮编（如有多条邮编数据，则给出唯一的推荐结果）。



- 结论依据：并在结论后自动给出所作判断的依据。
- 复杂门牌号码的处理：门牌号码中有阿拉伯数字，全角数字，汉字数字等多种形式，AddrSmart 系统可以做统一的标准化处理。
- 丰富的后台邮编数据加工功能：可以对邮编数据的门牌号码自动扩展分析，建立索引，方便查询。

4. 系统特点

地址匹配与搜索系统具备以下技术特点。

- 智能模糊匹配：可以模糊匹配用户的各种形式的输入，包括简略输入、智能匹配相关的邮编数据。
- 快速高效：单线程每秒可以处理 5000 条数据。
- 准确率高：根据 80 万条地址的实际测试，准确率高达 99.97%。
- 支持多线程：可以应用于支撑物联网服务。
- 智能学习功能：所有知识库自动学习已经正确标注过邮编的数据即可获取，无需复杂的人工干预。
- 技术先进：综合运用了自然语言理解技术、信息检索技术、模糊匹配与机器学习技术。

7.6 系统六 室内位置服务示范系统

低成本无线定位技术作为物联网应用基础支撑技术的重要组成部分，在安全生产、现代物流、公共安全、旅游娱乐、军事国防等领域有着广泛的应用前景。在煤矿生产中使用人员实时定位系统，可以提高紧急情况下的搜救能力；在制造及物流行业中使用装置实时定位系统，可显著提高生产与物流效率；在公共安全问题比较敏感的大型场馆内，使用定位系统对流动人员及关键设备进行实时定位跟踪，可提高应急管理水平。随着物联网应用的进一步拓展、移动计算设备的广泛使用以及无线通信技术的迅速发展，基于用户的位置信息可以为人们提供各种个性化的智能服务。

本示范系统针对大型建筑目前亟待解决的定位导航问题（如室内外复杂环境定位导航服务，统一平台下监控预警与应急安防的互联、互通、互动，需要提供个性化的位置服务定制，基于位置的展馆个人社交等），通过对室内外无缝精确定位技术和基于无线传感器网络的监控预警技术的突破，提供面向大型场馆、会场等建筑应急管理的高精度室内外导航定位服务，满足现代大型场馆快速应急管理的需求，同时提供方便、灵活的室内个性化的位置服务定制，基于位置的个人社交服务，并通过无线传感器网络、RFID 等技术实现建筑场馆的设备或人员无线监控管理，提高运维效率，并达到节能减排的目的。

1. 系统概述

随着经济的迅速发展，大型建筑开始在我国大量设计并建造，基于建筑室内环境的定位及其位置服务应用需求日益广泛。然而，像奥运场馆、世博会场馆、国家大剧院及各种大型娱乐场所等大型场馆和学校、厂房、监狱、会议中心等建筑群体，由于结构复杂，人员密集，火灾疏散困难，火势蔓延迅速，扑救难度大，大型建筑已成为国内外防灾减灾研究的重要课题领域。建立基于位置的应急及灾难出现的导航服务势在必行。与此同时，室内环境下基于位置的个性服务（如基于位置的欢迎信息、最近餐馆查询、车务通等）的改善将为人们



的生活带来更多的便利。此外，潜在的位置服务应用场景还有位置广告（精准化小区广告、位置竞价排名、动态优惠券）、位置社区（动态社区、位置交友、社区服务现场展开）、位置游戏（实景大富翁、寻宝探索游戏、种菜游戏的深入）、位置分析（基于用户位置等上下文的信息挖掘、商业网点选址、交通流量预测分析、城市规划优化分析、个人商务活动分析等）将极大地丰富人们的日常生活。

在日益增长的定位服务需求的大力推动下，伴随无线通信技术的迅速发展，定位技术研究取得了系列成果。到目前为止，出现了基于 GPS、移动网络、WiFi、RFID、蓝牙、ZigBee、UWB、超声波、红外、计算机视觉等定位系统，它们为各种基于位置服务提供了良好的基础设施。目前，室外场景下基于 GPS 的位置服务应用发展较快，而人们活动比较频繁的室内环境下位置服务才刚刚起步，有待进一步拓展。

现有室内环境基于位置服务系统存在如下不足。

（1）系统定位精度不足

由于卫星信号受到阻挡，现有的基于 GPS 的位置服务系统在室内环境难以应用。各种基于移动网络的定位技术（GPRS/CDMA）尽管在室内可以实现定位，但精度有限（定位误差为 50~100m），难以满足室内场景下的高精度位置服务应用需求。而各种适用于室内环境的短距离无线定位技术，如 WiFi、RFID、ZigBee、UWB，目前还处于发展完善之中，有待于与移动终端的进一步融合。

（2）系统成熟度低

随着人们工作、生活环境的日益智能化，室内位置服务应用潜力巨大。另一方面，由于室内环境复杂，缺乏像室外 GPS 那样统一的高精度定位基础设施，目前室内基于位置服务还缺乏成熟的应用方案。室内定位技术的多样性和差异性为提供统一的位置服务提出了诸多挑战，不仅需要解决多种异构定位技术的融合问题，而且还需要建立支持广域范围的室内位置服务平台，并实现与室外位置服务的无缝对接。

（3）系统可扩展性差

现有的室内定位系统都是一个个相互孤立的自治系统，不同系统之间不仅缺乏统一的标准，兼容性和互通性较差，而且缺乏统一、广域覆盖的位置服务平台，限制了室内位置服务的大规模应用推广，无法满足各种室内定位需求。

为了满足室内环境日益增长的位置服务应用需求，利用部署日益普及的 WiFi 网络，提供大范围、低成本、高精度的定位服务，正成为室内位置服务的一个重要发展方向。值得一提的是，目前 WiFi 芯片、模块等正在打破封锁，成为各种移动终端的标准配置，为 WiFi 定位技术的大规模应用提供了基础条件。在室内环境使用 WiFi 定位技术，不用增加任何硬件开销，直接通过测量射频接收信号强度，使用高精度定位引擎软件，可提供满足大多数室内位置服务需求的基础位置数据。

基于 WiFi 定位技术能较好地适应室内复杂场景，不像其他室内定位技术那样需要构建专门的硬件基础设施；WiFi 网络在为人们提供高速数据通信服务的同时，还能为人们提供良好附加值的实时位置信息，网络部署简单，具有低成本、普遍适用等特点；目前 WiFi 产品非常成熟，市场仅仅需要添加部分软件就可以实现定位功能。

2. 系统架构

基于物联网的位置服务应用系统架构如图 7-23 所示，主要包括低功耗、小型化定位标



签、阅读器和定位服务器等，其中定位服务器驻留了高精度定位引擎、楼宇内基于位置服务应用的中间件等模块。

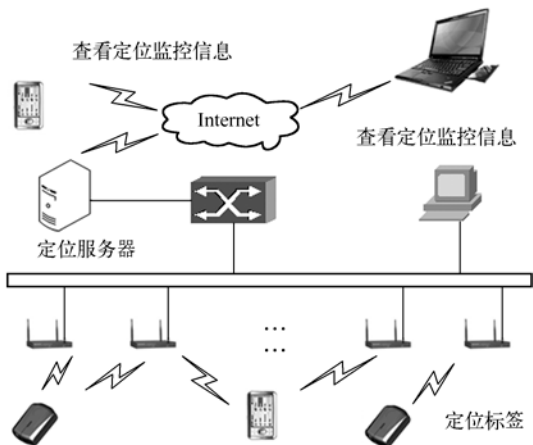


图 7-23 基于物联网的位置服务的应用系统架构

基于位置服务的应用系统可为用户提供室内外定位、路径导航、综合位置服务查询、应急事件监测、信息发布等物联网定位应用，如图 7-24 所示，主要包括“环境实时监测”、“个人定位及实时跟踪”、“应急导航”、“导航服务”、“位置查询/信息推送”、“社交服务”。

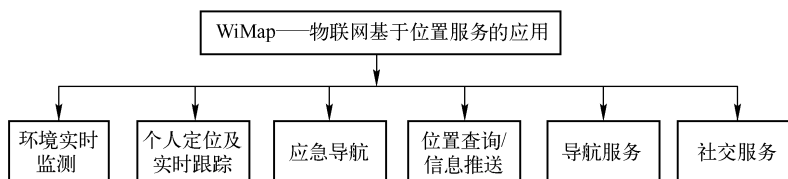


图 7-24 物联网基于位置服务的应用功能

1) 环境实时检测。支持远程 Web 访问的实时环境监测服务，使用部署在室内环境的温湿度、光照度等传感器，辅助传感器节点的空间位置信息，实现对室内环境的温湿度、光照度的实时监测，并提供基于 GIS 的友好人机界面。

2) 个人及定位实时跟踪。基于低功耗 SoC、WiFi 通信技术等开发的便携式定位标签，使用基于射频指纹的高精度定位引擎，不仅可以满足室内场景的高精度定位跟踪服务应用需求，而且顺应了低能耗这一发展趋势。

3) 应急导航。基于环境智能感知技术，通过部署在环境中的温湿度等传感器，获得环境的预警信息；在紧急情况下，系统中基于位置的动态应急处理引擎可为室内处于危险区域的人员进行广播、短信等预警提示，并提供最优逃生路线。

4) 位置查询/信息推送。把用户终端获取的高精度位置信息与在特定空间上开展的讲座、报告、商业活动等广告信息相关联，使用无线网络传输和 Web 信息推送技术，在用户屏幕上展示“基于位置的信息查询和推送”服务。



5) 导航服务。在带有 WiFi 模块的移动终端上安装相关插件, 可获取从当前位置到目的地的最佳路径, 与 GIS 结合, 可提供直观、便捷的出行服务。

6) 社交服务。利用高精度室内定位、Web 技术、数据库挖掘技术, 基于用户海量信息挖掘所获得的社交关系, 结合展区内 (展馆或大型会议环境) 用户位置信息, 及时通告用户感兴趣的社交好友所处位置, 为用户提供良好的社交体验。

3. 工作流程

基于位置服务应用系统的工作流程可按定位节点标签和基于位置服务应用终端 (可以是平板电脑或智能手机) 的数据流向来描述。图 7-25 所示为室内基于位置服务的应用工作流程。

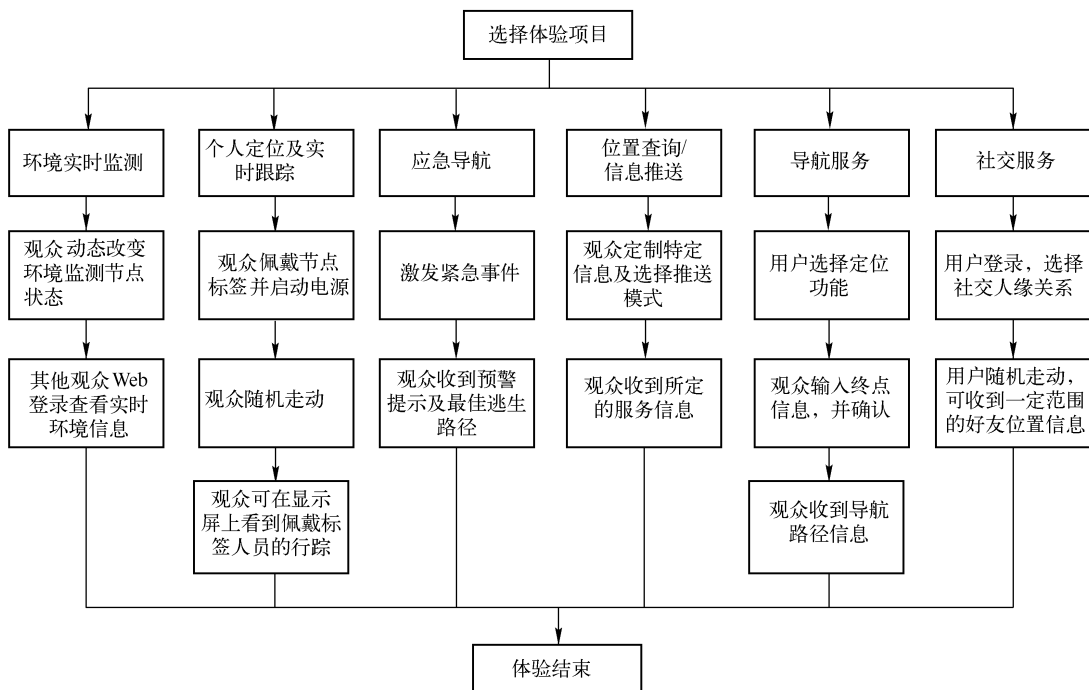


图 7-25 室内基于位置服务的应用工作流程

1) 环境实时监测。用户可以通过动态改变环境监测节点状态 (如用手盖住节点并释放、用电吹风向节点吹热风等), 其他用户可以通过 Web 登录方式查看远程环境实时信息, 可以看到某室内位置明显环境监测变化。其流程图见图 7-26。

2) 个人定位及实时跟踪。用户进入展区时可以佩戴具有实时定位、跟踪功能的嵌入式节点标签, 并启动标签电源, 可以在液晶屏幕上看到佩戴定位标签人员的位置。该人员随机走动, 可以在屏幕中看到该人员的实时行踪。其流程图见图 7-27。

3) 应急导航。该服务为默认服务, 在用户进入展区后, 在特定时间间隔内激发紧急事件, 启动基于位置的应急导航服务。将根据产生紧急事件的级别、用户的位置范围, 分别进行广播、短信、彩信、屏幕预警提示, 同时为定制应急逃生路线导航的用户发送最佳逃生路径。其流程图见图 7-28。



4) 查询/信息推送。用户走进展厅, 定制“基于位置查询/信息推送”中特定服务(如学术报告信息、招聘宣讲会、商业广告推送等), 可以选择信息推送模式, 系统将根据用户选择的服务推送模式进行短信、彩信或邮件信息推送。在特定服务相应服务范围内(如半径为20m), 为相应订阅服务的用户实时推送特定服务信息, 方便开发各种增值服务。其流程图见图7-29。

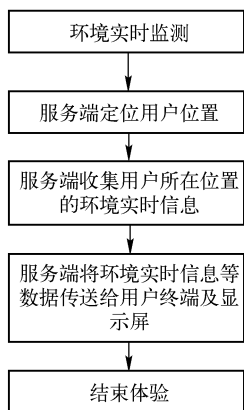


图 7-26 基于位置服务的实时环境监测流程图

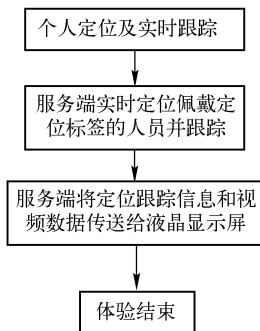


图 7-27 个人定位及实时跟踪流程图

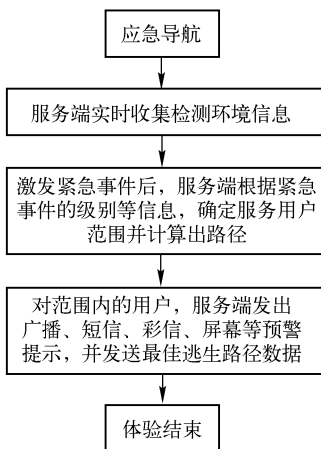


图 7-28 基于位置服务的应急导航流程图

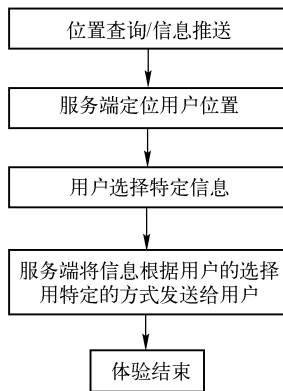


图 7-29 基于位置服务的查询/信息推送流程图

5) 导航服务。用户进入展区内可以手持具有 WiFi 功能的定位终端, 匿名启动系统, 选择“定位”功能。在按下“导航”按钮后, 输入“终点”(或选择终点), 系统根据当前位置到目的地之间完成最佳路径导航。如果用户希望从特定位置作为起点进行导航, 需要在导航“起点”中输入相应位置。其流程图见图7-30

6) 社交服务。用户进入服务区, 定制基于位置社交服务, 输入个人信息(同学、朋友、同事、同校)。多名用户启动基于位置的社交服务, 可以在展区内自由行走。当半径在10m(可定制)范围内出现该用户的社交关系人员时, 将在两位用户的定位终端上出现相关提示, 同时给出其“好友”的精确位置。其流程图见图7-31。

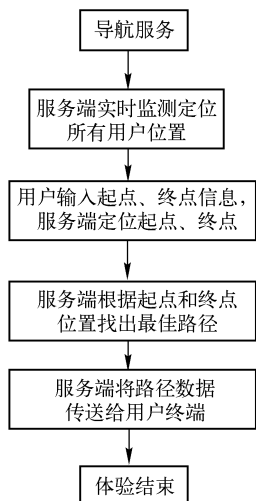


图 7-30 基于位置服务的导航服务流程图

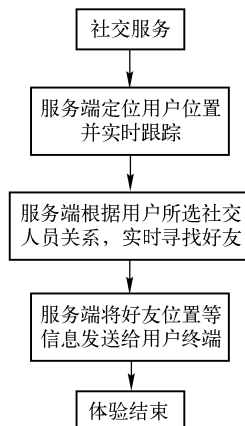


图 7-31 基于位置服务的社交服务流程图

4. 系统特点

本室内位置服务系统基于目前已经广泛部署的 WiFi 网络 (802.11b/g/n)，使用高精度定位引擎获取人员及物品的实时位置，并基于获取的位置信息，提供环境信息实时监测、智能导航、信息查询及推送、应急逃生导航、社交等多种服务。

本系统是一种典型的物联网应用，具有如下特点。

1) 全空间定位导航。本系统使用了目前已广泛部署的 WiFi 网络。由于 WiFi 信号不受室内外环境限制，因此只要部署有 WiFi 网络的区域，就可以使用 WiFi 网络实现定位。

2) 定位精度高。与可以应用于室内环境移动定位技术相比，基于 WiFi 定位可以获得更高的定位精度，可满足大部分室内位置服务应用需求。

3) 系统适应性广。本系统使用已在办公等室内环境得到大量部署的 WiFi 网络，无需增加任何硬件，仅仅增加软件就可以实现定位服务功能，系统部署和推广方便。

5. 系统组成

本系统组成主要包括软硬件两部分。硬件设备主要包括低功耗定位标签、带 WiFi 模块的移动终端 (智能手机、平板电脑)、无线路由器、定位服务器等。其中，低功耗便携式定位标签、阅读器、定位服务器等核心设备的技术参数见表 7-9~7-11。

表 7-9 低功耗便携式定位标签主要参数

技术参数	说明
通信接口	802.11b/g
电源	800mA 3.6VDC 锂电池
功耗/mW	<300 (无线发射功耗)
工作温度/℃	-10~+55
工作状态提示	蜂鸣器、通信指示灯、电源指示灯
使用寿命/年	大于 3



表 7-10 阅读器主要参数

技术参数	说 明
天线	可外接天线 2 个、SMA 接头
读卡方式	支持定时自动读卡、外触发控制读卡或软件发命令读卡，读卡方式可设置
通信接口	802.11b/g、RJ-45、TTL232
电源	DC 9V 电源适配器
工作温度/°C	-10~+55
工作状态提示	蜂鸣器、天线指示灯、通信指示灯、电源指示灯、RF 发射指示灯
使用寿命/年	大于 3

表 7-11 定位服务器主要参数

技 术 参 数	说 明
CPU	多核 CPU
通信接口	10/100M 自适应以太网
电源	220 AC
功耗/ W	<300
工作温度/°C	-10~+55
储存温度/°C	-20~+85
系统接口	一路以太网接口
储存温度/°C	-20~+85
工作状态提示	运行指示灯
使用寿命/年	大于 3

软件部分主要包括在服务器上运行的定位引擎，其系统架构如图 7-32 所示。该系统架构定义了服务器端定位软件各层的主要功能。图 7-33 给出定位引擎层为不同的定位算法提供标准的钩子函数接口，可以针对不同应用场景配置不同的定位算法，也可同时采用多种定位算法；定位中间件层主要对不同定位实体创建相应的定位进程，并对定位实体进程进行管理；功能服务层是本系统软件的核心层，为用户提供相应的功能服务，主要分为 GIS、管理、预警、日志和可视化重现与交互 5 个子系统。其中，GIS 子系统负责完成地图管理、位置管理等功能；安全子系统负责用户管理和权限管理；预警子系统包括区域管理、报警策略管理与应急系统联动接口等功能模块；日志子系统则负责记录用户、定位实体以及系统发生的事件和事务；可视化重现与交互子系统是与用户直接交互的软件模块，嵌入在服务器端，负责接受用户指令，并完成定位结果的显示。各系统独立运行，通过客户服务器（CS）方式进行信息和控制的交互。使用 CS 接口方式进行信息和控制方式便于实现服务的分布式控制，即不同的服务可以运行在不同的主机上，以提高系统的处理能力。

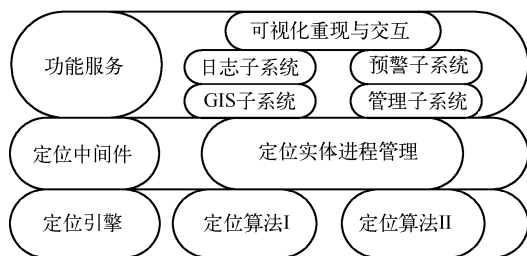


图 7-32 服务器端定位软件架构

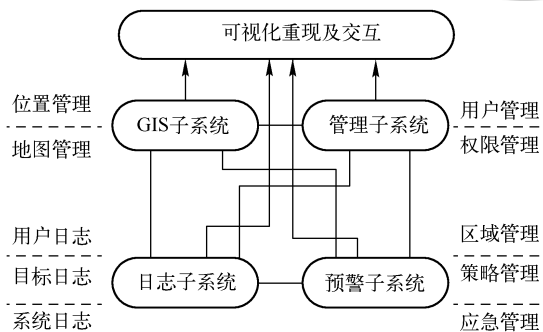


图 7-33 定位引擎层子系统关系

7.7 系统七 地质灾害监测预警系统

突发性滑坡、崩塌、泥石流是我国最严重的地质灾害类型，因其发生时间短、隐蔽性强、破坏性大，极易造成重大人员伤亡和巨大经济损失。基于物联网的地质灾害监测预警系统通过多种地质传感器及可靠的野外无线通信方式，可以有效地解决范围广、无人值守的灾害监测及预警问题，也是物联网在民生工程中的重要应用。

1. 系统概述

据不完全统计，我国近 10 年来，因滑坡、崩塌、泥石流造成死亡和失踪的人员每年平均约 1000 人，造成直接经济损失年均 38.6 亿元。5.12 汶川特大地震引起的滑坡、崩塌、泥石流造成大量人员伤亡和失踪。据现场调查统计，地震区地质灾害类型以滑坡为主，其次为崩塌和泥石流。四川、甘肃和陕西等省地震区 84 个县（市、区）发现重大地质灾害点 8439 处，其中，滑坡 4372 处，崩塌 2309 处，泥石流 515 处，威胁 109 万人的生命安全。

2008 年 6 月 23 日，胡锦涛总书记在两院院士大会上发表讲话时指示：“要加强对自然灾害分布规律和致灾机理的研究，加强自然灾害监测预警能力建设，建立灾害监测-研究-预警预报网络体系，加强防灾减灾关键技术研发，建立国家综合减灾和风险管理信息共享平台，完善国家和地方灾情监测、预警、评估、应急救助指挥体系。”《国家中长期科学和技术发展规划纲要（2006—2020 年）》也把构建重大地质灾害保障体系作为重点领域及其优先主题。

长期以来，我国建立的群测群防网络基本上覆盖了全国山区丘陵县市，每年有效避灾上千起。然而，现有监测手段大多采用人工收集方式，技术方案存在数据收集不及时，信息覆盖面不足的缺点。即使现有的野外自动监测，也仅限于少数几个独立的示范区，且区内监测设备没有成网，没有形成一个实时动态监测体系，难以满足滑坡泥石流快速预警和突发应急处置的需求，导致真正灾害到来预警不够、抢救不及时而损失惨重。

物联网技术在地质灾害监测方面可以大显身手。通过终端无线传感器网络可以延伸传感器的感知触角，实现对地质状态信息的非接触传递、实时监测、协作处理、本地化决策，以及与互联网和移动通信网的泛在接入，大大提高了信息采集的实时性、可靠性和灵活性。通过宽带移动通信技术，选择不同地质环境背景、不同气候条件的地质灾害监测示范区建设，解决滑坡泥石流监测预警的关键技术，建立监测预警信息系统，获取实时动态监测数据，建



立滑坡泥石流的预测模型，及时捕捉重大滑坡泥石流的前兆信息，为灾害防治提供技术支持，保障国民经济稳定健康发展。

系统针对我国地质灾害防治工作的现实需求，将无线传感器网络、宽带移动通信技术与滑坡泥石流监测预警技术相结合，开展应用示范和产业化建设，使我国地质灾害野外监测技术和预警方法取得突破性进展，达到国际领先水平，为我国地质灾害监测预警网络体系建设和应用推广提供技术支持。

2. 系统架构

面向监测地质灾害监测的物联网系统是通过各感知节点构成的庞大的监测网络。感知数据具有一定的差异，雨量传感器、含水率传感器、测斜传感器、位移传感器产生的数据量小，且数据不是时刻变化的；而地声传感器、GPS 设备与视频传感器产生的数据量大。为了在满足数据要求的条件下更好地利用带宽资源，采用低速率与高速率传感器节点共存的异构网络体系结构。雨量传感器、测斜传感器、位移传感器采用低速率网络传输协议和设备；地声传感器、GPS 设备与视频传感器采用高速率宽带网络传输协议和设备。

在实际应用中，单个传感器网络的网络规模直接影响网络容量的提升，故将传感器网络设计为 3 层的 MESH 网络，如图 7-34 所示。

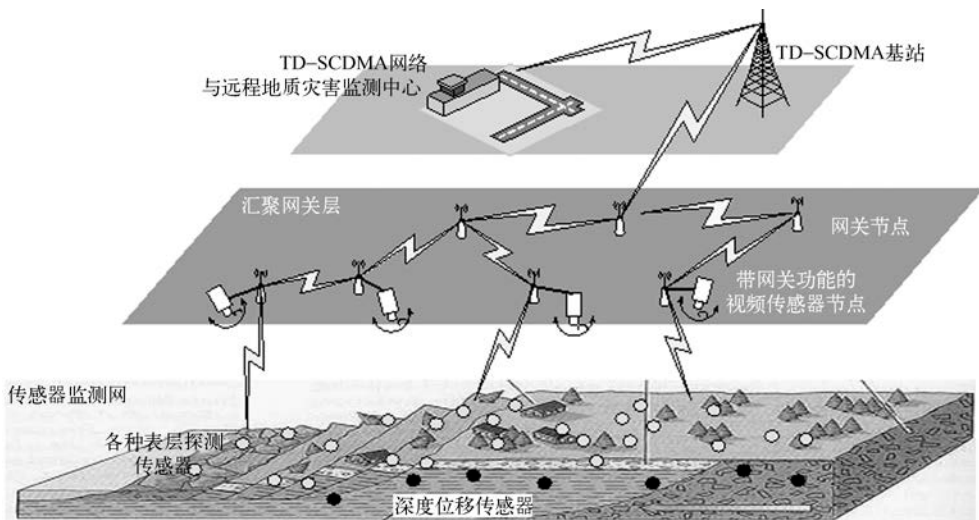


图 7-34 地质灾害监测预警系统体系结构示意图

第一层为使用 ZigBee 协议的低速率探测感知层。网络中的各种探测传感器节点（也包括大数据量的音视频传感器节点、GPS 传感器节点和地声传感器节点）与中继节点通过自组织成网的方式形成最底层的传感器监测网。协议的自组织性主要体现在，传感器节点故障时的网络快速愈合能力和新的节点增加部署时的快速组网能力。采用带有快速邻居发现与更新的自组织网络协议，实现预期的目标。

第二层为网络传输层，使用 WiFi 进行数据的传输，该层包括 ZigBee-WiFi 网关、WiFi 中继节点、TD-SCDMA 网关等。探测传感网络中的数据可以通过任一个 ZigBee-WiFi 网关接入汇聚网关层，通过在汇聚网络中中继到达某 TD-SCDMA 网关，TD-SCDMA 网关具有



WiFi 到 TD-SCDMA 的网关功能, 能够将使用 WiFi 汇聚到的数据通过 TD-SCDMA 网络发送到监测中心。为了满足大数据量的传感器设备(如音视频传感器等)的数据速率要求, 也将它们放入到汇聚网关层, 它们可以以单跳或多跳的方式将数据传送至移动通信网络中。

第三层为地质灾害监测应用服务中心站。为了更好地利用移动通信基础设施, 系统使用了 TD-SCDMA 网络与远程地质灾害监测中心站进行数据交换。使用 TD-SCDMA 网络可以满足当前的监测传感器与多媒体数据的传输需求, 并可以大大降低研发和部署成本。对于地质灾害监测的汇集传输是一个很好的选择。

3. 工作流程

本系统针对长期监测和灾后应急通信设计了不同的工作流程。如图 7-34 所示, 长期监测采用三层结构。

1) 网络中的各种探测传感器节点与中继节点通过自组织成网的方式形成最底层的传感器监测网。该监测网负责长期的从野外采集数据, 并将其传送至第二层汇聚网关层。

2) 汇聚网关层包括 ZigBee-WiFi 网关、WiFi 中继节点、TD-SCDMA 网关以及其他高速率传感器节点。ZigBee-WiFi 网关在收到监测网的数据后, 通过在汇聚网络中中继到达某一 TD-SCDMA 网关; TD-SCDMA 网关具有 WiFi 到 TD-SCDMA 的网关功能, 能够将使用 WiFi 汇聚到的数据通过 TD-SCDMA 网络发送到监测中心站。

3) 地质灾害监测中心站和 TD-SCDMA 网络构成第三层。该层主要对收集到的数据进行处理, 通过系统或者专家分析之后, 再根据得出的结论进行反馈控制。另外, 中心站通过与互联网相连可供远程的监控与会商。

除了长期实时监测的功能外, 系统还提供了支持多媒体的灾后快速部署与应急通信保障体系, 系统结构如图 7-35 所示。灾后通信保障采用了支持宽带无线通信技术 TD-LTE 技术等远距离基站。灾后 TD-SCDMA 网络可能受损, 应急通信设备采用远距离的宽带无线通

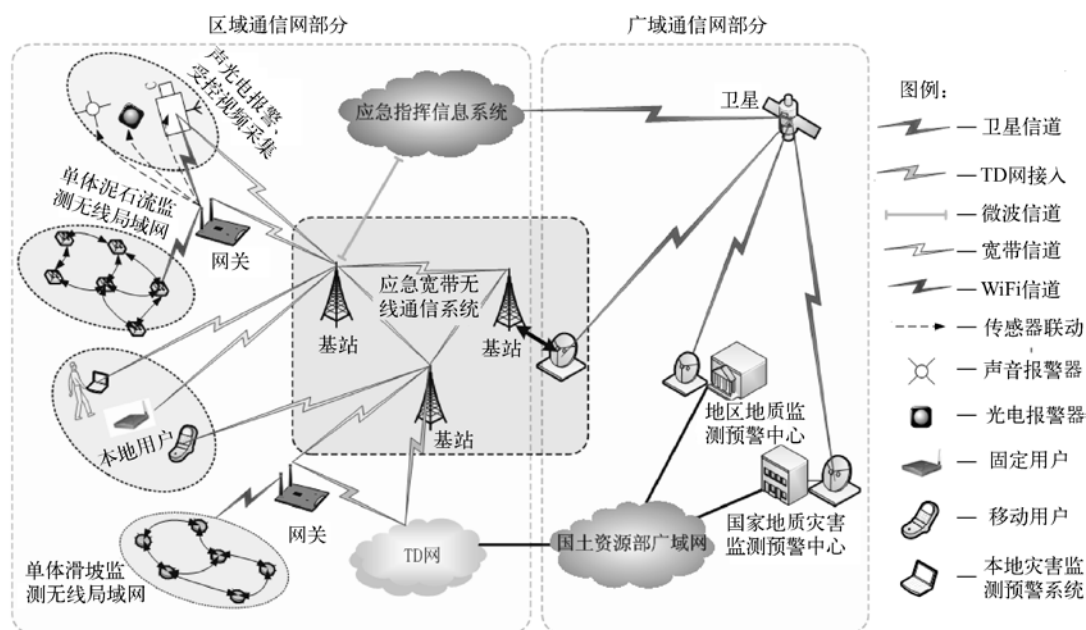


图 7-35 灾后快速部署与应急通信保障体系结构示意图



信基站。该基站具有 WiFi-LTE 网关的功能，将受灾区域内部署的传感器节点监测到的数据通过 WiFi 汇聚到该基站进行远距离的接力传输。该基站具有传输范围远、传输速率高的特点。感知数据通过基站传输到远程指挥监控中心，可通过多媒体信息与多感知数据实时监控灾区或易发灾区的破坏情况，并可根据数据分析结果对滑坡泥石流等进行预警。

4. 系统特点

(1) 能够解决地质灾害连续动态监测难题

滑坡泥石流监测系统需要布设在野外山地环境，现有的监测数据多采用群测群防人工采集，即便少量的采用自动采集，但因数据的采样和收集周期较长，不可能达到实时监控和预测。由于缺乏连续的动态观测，对滑坡泥石流的致灾临界点的判断造成困难，严重影响了地质灾害的防治效果。通过滑坡泥石流传感网络系统研制和应用，有效地解决了地质灾害连续动态数据采集和汇集的难题，提高监测实时性，从而提高灾害防治水平。

(2) 提高自动地质灾害监测覆盖范围

据不完全统计，全国滑坡、崩塌、泥石流灾害隐患点超过 20 万处。目前，这些灾害点的监测主要依靠群测群防来进行信息通报，靠专业设备自动监控的不足 1%。未能采用专业仪器自动监控的主要原因是自动观测成本高。通常，监测一个滑坡泥石流点需要投入超过数十万元的专业设备。本系统研制的低成本滑坡泥石流传感网监测系统，提高了自动地质灾害监测覆盖范围以及地质灾害监测可靠性，可有效推动滑坡泥石流自动监测系统的应用。

(3) 加速物联网规模实用化进程

从 20 世纪末提出传感器网络的概念以来，传感器网络技术已经得到了长足的发展，但是传感器网络除了通信优势外，如何从功能、部署、运行等方面替代专业的传感器系统，从而满足行业领域的业务需求还需要进行不断的实践、检验和磨合。通过组网协议开发、芯片组集成、网管软件开发、通信网关的研制，系统将对国产传感器芯片组、传感网技术进行综合实践检验，以加速物联网在行业应用领域的实用化、规模化进程。

(4) 拓展宽带无线移动通信应用领域

地质灾害监测、应急处置对无线宽带多媒体网通信提出了很高的要求。本系统使用了我国自有知识产权的无线宽带多媒体网络通信技术，以拓展宽带无线移动通信设施在国家专业数据汇集传输应用领域，优化新一代移动通信技术从传输带宽、通信距离等方面满足地质减灾领域的深度应用的需求，促进国家和行业应急通信的技术融合。

第 8 章 物联网实验室建设

本书以前章节，主要从物联网的感知与标识、通信与网络、接入与处理 3 个方面切入物联网 3 个层次，精选了 23 个实验项目、5 个演示实验和 7 个应用示范系统。通过课前预习、系统搭建、编程测试、实验报告编写（一般含有调试过程、结果分析和实验体会）等环节，能够深入理解物联网的核心技术原理与关键技术内容，了解当前国内外主流芯片和实现方案，并能够对单项技术达到举一反三的效果；通过实验、演示实验和应用示范系统，可以让读者对物联网在行业领域中的各种应用，进一步建立起全局的系统概念。

随着物联网工程及传感网工程本科专业建设的兴起，迫切需要配套物联网实验室建设方案，以进一步提升学生的动手能力和实践水平。本章根据作者多年来在电子与信息技术领域的教学和实验室建设经验，结合不同的学科、专业背景和行业属性，提出物联网实验室建设的原则与具体方案，希望能够在一定程度上起到指导与参考作用。

8.1 物联网实验室环境建设

本书前面的章节围绕物联网基本问题设计了 23 个实验，基本覆盖了物联网感知互动层、网络传输层及应用服务层的比较重要的关键技术，主要涉及了单片机、嵌入式及智能终端 3 个方面的实验平台环境。当然，云计算环境也应是物联网的核心内容，但由于在实验室环境下难以操作，只能进行大量的简化和适应性改造工作，因此这部分内容将在本书的升级版本中加以考虑。不同行业与专业背景的学校可根据各自的需要，配置不同的实验环境。表 8-1 列举了本书所涉及的物联网主要实验环境的建设目标。

表 8-1 物联网主要实验环境的建设目标

实验环境	建设目标
单片机环境	<ul style="list-style-type: none">● 建立起对单片机的基本认识，熟悉基于单片机的应用程序的开发流程，掌握基于汇编语言、C 语言和轻量单片机操作系统的开发，并对它们之间的区别有较深的理解● 掌握典型单片机片内资源的使用方法。学习用单片机开发片外设备的基本流程和方法● 初步掌握软硬件协同调试的基本方法和思路。学习调试一些基本的硬件信号问题● 学会开发一些较为复杂的基于单片机的应用项目
嵌入式环境	<ul style="list-style-type: none">● 建立起对嵌入式微处理器的基本认识，掌握基于嵌入式微处理器的应用程序的开发流程，熟悉基于嵌入式操作系统开发的特点● 学习在嵌入式 Linux 上开发外设驱动的基本流程和方法● 掌握软硬件联合调试的基本方法和思路。学习调试一些基本的硬件信号问题● 学会开发一些较为复杂的基于嵌入式的应用项目
智能终端环境	<ul style="list-style-type: none">● 建立起对目前主流智能终端环境的基本认识，熟悉基于移动终端的应用程序的开发流程，掌握基于智能终端环境的应用开发的特点● 学习在基于 N 的 QT、苹果的 iOS 上开发应用程序的基本流程和方法● 掌握智能终端模拟器环境的基本开发方法和思路。进一步理解网络应用的开发方法● 学会开发一些较为复杂的智能手机应用项目

物联网实验室环境建设所需实验室设备，可根据具体的专业要求、师资力量、场地规模



等实验室建设需求和教学背景进行选择，并结合上述物联网实验室的 3 种实验平台，建设符合物联网实验室要求的软硬件环境。例如，假设实验室可容纳学生人数一般为 35~40 人，实验设备装置可配成 1 人一组或 2~3 人一组，每组实验设备的基本配置如表 8-2 所示。

表 8-2 物联网实验室设备基本配置

设备类型	数量	物联网实验设备描述
感知节点	3~5 个	可选包括感知模块、射频模块、RFID 模块、GPS 模块等（其中：感知模块包括高精度温湿度传感器模块、光照传感器模块、磁场传感器模块、加速度传感器模块、图像传感器模块等；射频模块包括 ZigBee 模块、3G 通信模块、蓝牙模块和 WIFI 模块等）
网关节点	2 个	可选模块除了具备感知节点中的感知模块、射频模块、RFID 模块、GPS 模块等，还可以配备电源模块（供电）、ARM 处理模块、DSP 模块、FPGA 处理模块
终端处理显示设备	1 套	PC（1 台，含串口、USB 接口）、智能手机/PDA（1 部）、iSD 卡（2 张）、USB 接口网关（1 台）
开发环境	1 套	与感知节点、网关节点相配套的编程、调试软硬件环境

在表 8-2 中，物联网终端节点分为两类，一类是普通的感知节点，采用单片机或嵌入式芯片和射频芯片，用于各种感知数据的采集和传输；另一类是网关节点，带有额外的处理模块，用于数据的上报、转发和一定规模的数据处理。本书所涉及的所有实验均涉及感知节点和网关节点。为了节约实验成本，部分实验（如实验二十一）会涉及一定规模的数据处理，均在台式计算机上完成。另外，与智能终端相关的实验，既可以在智能手机终端上完成，也可以通过模拟器在台式计算机上完成。

利用表 8-2 中给出的基本实验设备，并结合表 1-1 的具体设备内容，可以完成本书中所涉及的所有课程实验及演示实验，并可以从概念上很好地展现出物联网的感知与标识、通信与组网及接入与处理等方面的核心概念。另外，通过结合本书的第 6、7 章关于 5 个演示实验和 7 个应用系统的具体介绍，能够让学生较清晰地了解物联网应用中可能面临的一些问题，具体可以进一步参考《物联网技术应用》。图 8-1 给出了通过基本设备的实物，在实验室中物联网基本示意图，计算机通过网关可以获取感知节点的信息，智能终端（手机或 PDA）充当着移动网关和移动终端的角色，这里服务器也可以用台式计算机来代替。

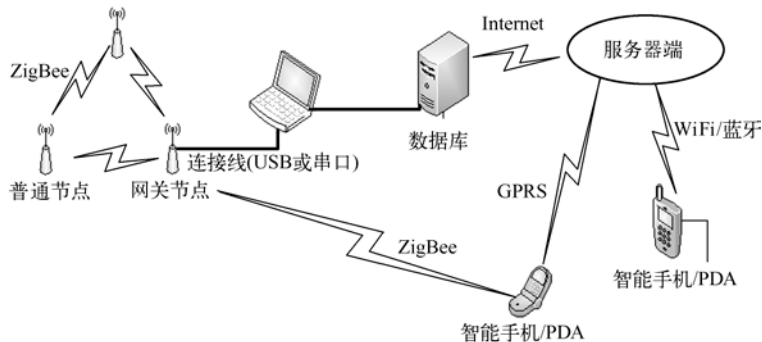


图 8-1 物联网实验方案基本示意图

8.2 物联网实验室实验内容

虽然物联网工程及传感网工程均已经作为独立的专业获得批准，但由于师资、教材、实



验设备、社会人才需求以及对物联网认识等因素,当前很多高校或者企业仍然从各自的专业偏向出发来设置物联网实验室,或者说根据本身的行业背景进行物联网实验室的设置。从专业方向上来讲,主要包括计算机、自动化、网络工程、微电子、微机电、通信等专业。由于本书所涉及的实验内容较多,对于初学者很难也没有必要全部深入学习,“伤其十指不如断其一指”,深入掌握若干实验并触类旁通,往往能够达到更好的效果。

在物联网实验室中,为了更好地安排实验,根据对课程内容的掌握程度要求不同,一个学期的实验教学过程(一个学期16周,除去开学1~2周的大班绪论课,期末考试周,实验课程一般动手实际操作为13周)可以分为以下3种模式(详见表8-3)。

- 模式一:每周1次,每次3学时(连续135min),共13次课内实验。
- 模式二:每周1次,逢单周或双周加1次,每次3学时,共20次课内实验。
- 模式三:每周2次,每次3学时,共26次课内实验,课时充足,建议加深了解。

当然,如果为了更好、更深入地掌握或扩展实验内容,也可以安排两个学期的实验课程,这样可以有更多的实验时间,进一步加强实践教学环节。

表8-3 物联网实验室实验内容安排

模 式	实验量/次	物联网实验内容
模式一	13	实验一:单片机开发平台基础实验 实验二:嵌入式平台基础实验 实验三:智能移动终端平台基础实验 实验四A:单片机平台高精度温湿度传感器实验 实验五B:嵌入式平台运动感知实验 实验六:图像采集实验 实验九:嵌入式平台基于MiniGUI的简易编辑器实验 实验十:高频RFID读写实验 实验十三:2.4GHz无线通信实验 实验十四:433M低功耗自组网实验 实验十七:Android系统下互联网接入实验 实验十九:基于射频指纹匹配的室内实时定位实验 实验二十一:数据库全文精准搜索实验
模式二	20	实验一:单片机开发平台基础实验 实验二:嵌入式平台基础实验 实验三:智能移动终端平台基础实验 实验四A:单片机平台高精度温湿度传感器实验 实验四B:嵌入式平台高精度温湿度传感器实验 实验五A:单片机平台运动感知实验 实验五B:嵌入式平台运动感知实验 实验六:图像采集实验 实验七:步进电动机实验 实验八:键盘与显示屏实验 实验九:嵌入式平台基于MiniGUI的简易编辑器实验 实验十:高频RFID读写实验 实验十二:433MHz无线通信实验 实验十三:2.4GHz无线通信实验 实验十四:433M低功耗自组网实验 实验十五:IEEE 802.15.4协议通信实验 实验十六:ZigBee协议组网通信实验 实验十七:Android系统下互联网接入实验 实验十九:基于射频指纹匹配的室内实时定位实验 实验二十一:数据库全文精准搜索实验



模 式	实 验 量	物联网实验内容
模式三	26	实验一：单片机开发平台基础实验 实验二：嵌入式平台基础实验 实验三：智能移动终端平台基础实验 实验四 A：单片机平台高精度温湿度传感器实验 实验四 B：嵌入式平台高精度温湿度传感器实验 实验五 A：单片机平台运动感知实验 实验五 B：嵌入式平台运动感知实验 实验六：图像采集实验 实验七：步进电动机实验 实验八：键盘与显示屏实验 实验九：嵌入式平台基于 MiniGUI 的简易编辑器实验 实验十：高频 RFID 读写实验 实验十一：超高频 RFID 读写实验 实验十二：433MHz 无线通信实验 实验十三：2.4GHz 无线通信实验 实验十四：433MHz 低功耗自组网实验（2 次） 实验十五：IEEE 802.15.4 协议通信实验 实验十六：ZigBee 协议组网通信实验 实验十七：Android 系统下互联网接入实验 实验十八：基于椭圆曲线的群组密钥协商协议通信实验 实验十九：基于射频指纹匹配的室内实时定位实验（2 次） 实验二十：TPSN 时间同步实验 实验二十一：数据库全文精准搜索实验

考虑到国内主要高校的师资队伍均来自于与物联网相关的计算机、自动化、网络工程、电子等专业，均具有一定的专业基础和技术擅长，完全可以结合各自的专业背景迅速建立起物联网专业实验室，这样可以大大缩短实验室建设周期，降低初次建设成本。下面结合相关专业方向的特点，给出重点实验设置，具体参考如表 8-4 所示。

表 8-4 物联网主要实验重点选修参考

专业偏向	实验设计目标	重点实验
计算机	重点围绕单片机或嵌入式平台深入学习实验，不但了解实验原理并尽可能引导学生自行设计实验，基础实验和物联网实验同等重要。建议做完本书全部实验，并针对本单位的教学重点进行深入学习	实验一、三、四 A、五 A、六、七、八、十二、十三、十四、十八、二十一
自动化	重点围绕单片机或嵌入式某一平台深入学习，并结合控制部分，对网络相关内容要求适当放低，建议进行包括 ARM 平台、RFID 实验及 ZigBee 的组网实验	实验二、四 B、五 B、九、十、十一、十六
网络工程	重点围绕单片机平台及智能终端相关实验，并把握网络相关实验，特别是自组网、IEEE 802.15.4 以及 ZigBee 等实验，以及网络接入实验，建议配备支持 QT 和 Android 平台的智能终端	实验一、三、四 A、五 A、十二、十三、十四、十五、十六、十七、二十
电子微电子	重点围绕单片机和嵌入式两大实验平台进行深入学习，做到精通硬件设计，实验中可考虑为这两个平台独立设计一些传感器、射频模块等外部设备，也可以适当了解学习无线通信协议方面的实验	实验一、二、四 A、四 B、五 A、五 B、六、七、八、九、十二、十四
无线通信	重点关注无线通信相关实验，可以以单片机平台为主，做到深入理解各类无线通信协议（可以 ZigBee 协议为例）、智能终端平台基础实验以及公网接入实验，建议配备支持 QT 和 Android 平台的智能终端	实验一、三、四 A、五 A、六、七、八、十二、十三、十四、十七、十九
信息处理	重点关注嵌入式和智能终端平台及相关实验，很多实验都可以在 PC 上完成，建议辅助支持 QT 和 Android 平台的智能终端	实验二、三、四 B、五 B、九、十八、十九、二十、二十一
单片机	以单片机平台为主的相关实验，包括单片机开发平台基础实验、感知实验及射频通信实验，由于本书大部分实验都可以在单片机平台上完成，建议深入学习	实验一、四 A、五 A、六、七、八、十二、十三、十四、十五、二十
嵌入式	以嵌入式和智能终端平台的相关实验为主，包括嵌入式开发平台基础实验以及部分感知实验，并适当辅助一些与单片机相关的网络实验，建议配备支持 QT 和 Android 平台的智能终端	实验二、三、四 B、五 B、九、十七、十九
机电一体化	重点关注本书中的单片机和嵌入式相关基础实验，并完成和传感器相关的所有实验，力求做到能够自主设计，学有余力者可适当选修其他相关实验	实验一、二、四 A、四 B、五 A、五 B、六、七、八、九、十九



另外,从具体应用领域上来看,物联网会涉及众多产业背景,在很多时候需要将以上实验环境及内容进行适应性开发,建设专门开发团队或者与其他专业团队合作,开发设计出具有不同行业特色的实验和应用演示系统。当然,这方面的工作必须结合实际应用的需求和特点,并且是建立在读者对本书基本实验内容完全掌握的基础上的。

8.3 相关课程和预备知识

物联网实验要用到多门相关的课程和预备知识,其中有的课程是各个专业都会设置的,如C语言程序设计,而多数课程是难以在实验之前都能修完的。

按照所需知识的先后顺序,分为实验先修课程、大班绪论课和实验专项知识3部分进行详细介绍。实验先修课程是进行物联网实验前修的课程,学生必须对绝大部分课程都已经修过;大班绪论课是物联网实验开学的第1~2周进行讲解的课程,重点还在于学生自学并在后续的实验中进行实践;实验专项课程是在进行具体实验前的15~30min进行专门讲解的课程,目的是进行相关基础知识的介绍和梳理,以便于学生对理论知识的掌握和具体实验的开展。

1. 实验先修课程

- C语言程序设计(必备)
- 单片机原理与应用(必备)(或微机原理与接口技术、计算机组成原理、嵌入式系统设计之类的课程)
- 计算机网络(必备)
- 电路原理
- 半导体技术基础
- 模拟电子技术基础
- 数字电子技术基础(数字电路/数字逻辑设计)
- 数据结构
- 通信原理

2. 大班绪论课

1) 物联网产业与技术导论:讲解物联网概述,并介绍物联网产业情况,用相关行业的典型案例分析,把“世界万物相连”的概念讲授给学生,激发学生兴趣。

2) 无线传感器网络:讲解有关无线通信网络的基本概念,各种无线网络的基本结构以及无线通信系统的关键技术和实现方法(包括蜂窝网络、IEEE 802.11、802.15.4标准)。使学生能够了解无线传感器网络和移动计算系统的基本原理、体系结构等。

3) 通信系统与SOC集成:利用单片机及嵌入式平台简单介绍仿真知识并演示,通过对多种集成芯片的解剖与分析,重在培养学生自己的动手操作。

4) 传感器原理与应用:介绍各类传感器的基本理论,各种传感器的工作原理、主要性能及其特点,使学生能合理地选择和使用传感器,掌握常用传感器的工程设计方法和实验研究方法,并通过应用层的实验,进一步了解感知数据的处理方法。

3. 实验专项知识

在进行一个或几个相关的实验之前,根据实验内容,对所涉及的知识模块或专项技能进



行有针对性的讲解，时间大约 15~30min 左右，一般在实验室内进行。

实验专项知识包括以下内容。

- 传感器及多传感器融合方法及其应用（高精度温湿度传感器、光照传感器、磁场传感器、加速度传感器、图像传感器等）。
- 智能移动终端系统开发（Symbian、Android、iOS、Linux、Palm、BlackBerry、Windows Mobile、Mac OS）。
- 图像采集与处理技术、电机工作原理及应用、键盘与显示屏简单介绍。
- MiniGUI 的简易编辑器介绍。
- RFID 系统工作原理、室内实时定位基本原理。
- 常见的无线传输通信协议（蓝牙、WiFi、3G、ZigBee、GPRS）。
- 应用层软件开发（Qt、Android、Python）。
- 自然语言理解的基本原理、数据库、信息检索的原理。

物联网实验室的建设需要学校、学院和专业结合自身条件，充分整合各方面资源，构建优越的实验环境。学生真正通过物联网实验的学习和实践，建立物联网概念，掌握物联网理论和相关软硬件开发知识，并对物联网整个产业有一个全面、系统和深入的理解，为从事相关的科研工作打下坚实基础。学校通过物联网实验室建设，为国家培养更加全面、符合当代产业方向和需求的高水平、高素质的现代化人材。

ISBN 978-7-111-34147-5

策划编辑：郝建伟

封面设计：zsc | 子时文化 | ZiShi Culture



· 内容简介 ·

本书从实验的角度介绍物联网相关技术，针对物联网的感知与标识、通信与网络、接入与处理 3 个不同层面的技术，精心设计了 21 项实验，并提供了内容丰富的配套光盘（包括参考文档、实验源程序、实验环境及配置文档、实验手册等其他实验相关资料）。

本书结合智能医疗、智能物流、环境监测等领域的应用场景，介绍了 5 个演示实验和 7 个应用系统，为读者进一步开发物联网应用解决方案提供思路。

最后，针对物联网教学及研究实验室建设的实际需求，特别是不同专业或行业背景，提供了一些具体的参考建议，希望能够对广大读者从事相关工作起到借鉴作用。

上架建议 计算机/网络

ISBN 978-7-111-34147-5



9 787111 341475 >

地址：北京市百万庄大街22号
 电话服务
 社服中心：(010)88361066
 销售一部：(010)68326294
 销售二部：(010)88379649
 读者购书热线：(010)88379203

邮政编码：100037
 网络服务
 门户网：<http://www.cmpbook.com>
 教材网：<http://www.cmpedu.com>
 封面无防伪标均为盗版

定价：43.00 元(含1DVD)