摘　要

众所周知，传统的投票表决方式存在着投票耗时长、计票任务重，而且容易出错、容易受人控制等缺点。针对这些缺点，本文研究并设计了一套无线下载电子投票表决系统。

本系统采用电子表决器代替传统的选票显示选票信息、采集选民意愿，并通过无线的方式将信息传送给计算机，由计算机代替人工进行选票统计。与现有的其它投票表决系统相比，本系统最大的特点是能将需要进行投票表决的内容在计算机上编辑后通过无线方式下载到电子表决器中，在该表决器中就能显示全部投票内容。本系统具有性能先进、使用方便灵活、计票快速准确、安全可靠等特点。

本文从硬件电路设计、实时操作系统设计、上位机软件设计、通信协议设计和数据加密算法等五个方面对无线下载电子投票表决系统进行详细阐述。首先，本文阐述了以超低功耗 16 位单片机 MSP430F149 为核心的硬件电路设计；详细论述了各模块元器件的选择和应用。其次，在详细阐述了μC/OS-II 嵌入式实时操作系统内核结构的基础上讲述了其在无线下载电子投票表决系统中的移植和优化。再次，给出了上位机软件的设计方案，包括数据的定义、基本操作和界面的设计。然后，讲述了本系统使用的通信协议，包括数据帧的格式、命令格式，以及无线网络的建立和维护过程。然后，给出了 AES 数据加密算法的加、解密过程和安全性、效率分析。最后，在实际测试的基础上给出了设计结论。

关键词：电子表决器；MSP430 单片机；nRF905；μC/OS-II；AES 数据加密算法

Wireless Download Electronic Voting System

Abstract

As everyone knows, the traditional voting has several shortcomings, such as needs long time to throw the ballots into the ballet box, the task of count the tickets be heavy, apt to make mistakes, and be controlled easily, etc.. To these shortcomings, this artical designes a wireless download electronic voting system.

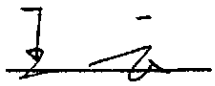
In this system, electronic voting machines are used to replace the traditional ballots for showing the information of ballots and gather the voter's will. And by radio, they send the information to the computer which counts the ballots instead of human. Compared with other voting systems existing, the most characteristic of this system is it can download the content of voting to the electronic voting machines by radio after edited on computer, and can show all content by the electronic voting machines. Besides it is easy to use, fast and accurately to count the ticket, secure and reliable etc..

This artical explains the system in detail from hardware design, on chip Real-Time operating system, software on computer, protocol of communication, and data encrypt algorithms. First of all, this text explains the hardware design. The circuit uses the ultra low power singlechip computer MSP430F149 as the core. This portion describes the choice and application of the components of all modules in detail. Secondly, after detailed explained the μC/OS-II real-time on chip operating system, explains the transplantation and optimization of μC/OS-II in wireless voting system. Moreover provides the plan of software design on computer, besides data structure basic operations and interface. Then, tells the communication protocol that this system uses, besides frame structure, instruction structure and the process of wireless network setting up and maintenance. Then tells the process of encrypting and decipherring of AES data encrypt algorithm. The analysis of AES's security and efficiency is also provided. Finally, provides the conclusion on the basis of testing actually.

Key Words: Electronic voting machine; MSP430 singlechip computer; nRF905; μC/OS-II; AES data encrypt algorithm

独创性说明

作者郑重声明：本硕士学位论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得大连理工大学或其他单位的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

作者签名：  日期：2005.3.20

1 绪论

1.1 投票形式的演变与发展

投票是指选民在规定的时间和地点，按照法律规定的原则和方式，就决定代表及国家公职人员的合适人选，或者就决定国家重大事务做出个人选择的一种表达方式[1]。投票是选民选举权的具体行使和实施，是整个选举过程的一个十分重要的环节，其对于民主政治的发展有着极为重要的意义。

1.1.1 国外投票形式的演变与发展

投票从出现到现在已经经历了几千年的历史变迁，经历了一个由粗放到精细、由随意到规范的历史发展过程[1]。早期的民主选举全部采用公开投票的形式。古罗马帝国采取的是口头投票的方式，即选举人高呼被选举人的姓名，被呼叫最多者即为当选者。这种口头选举的方式，选举人的意志表达得十分坦率也十分粗糙，而且仅仅适用于选民人数较少、被选举人的威望较高的情形。如果被选举人中有两人的威望相当，很可能会造成一些失误。这种口头投票的方式曾在西方一些国家流行相当长的时期。公开投票还有一些其他形式，如鼓掌欢呼、举手表决等。但是随着经济的发展和民主运动的高涨，导致选举权的扩大和利益群体的分化，公开投票方式日益暴露其难以掩饰的弊端[1]。19世纪中叶开始，秘密投票方式逐步取代了公开投票方式而成为各国通行的一种选举方式，选票也应运而生。

1.1.2 我国投票形式的演变与发展

在我国，投票选举起步得比较晚，直到国民党南京政府倒台，旧中国并没有真正实行过民主选举[1]。但在新民主主义革命时期，在中国共产党领导广大民众建立的红色政权和抗日民主政府中也实行过真正的民主选举，并制定了一些适应我国当时经济、文化背景，适合广大民众的投票方式[1]。例如：圈名法、投豆法等为当时的民主选举做出了巨大贡献。所谓圈名法，就是由选举的组织人员将候选人的名字写在选票上，让识字不多的选民在自己同意的候选人名字上画圈[1]。所谓投豆法，就是让候选人面对选民坐成一排，每人背后放一个碗，开始选举后选民依次排队走到候选人背后将事先发给他们的豆粒投给他们赞成的候选人背后的碗中，以身后碗中豆多者当选[1]。随着社会主义经济的发展和人民群众文化素质、政治素养的提高，1979 年的选举法中规定全国和地方各级人民代表大会代表的选举一律采用无记名投票方法。

1.1.3 现行投票方式的缺点

现在比较盛行的投票方式为无记名投票方式，采用选票的形式。这种投票方式的缺点显而易见：首先，需要用人工来统计选票，对于大型的选举来说这是一件繁重的工作，容易出错而且容易受人控制导致不公平的结果。其次，这种投票方式需要选民将选票投入投票箱中，这会耗费相当长的时间；统计选票的过程也需要相当长的时间。第三，不够灵活，如果第一轮选举并没有产生理想的结果则需要进行第二轮甚至是第三、第四轮选举，这时需要重新制作和发放选票。基于这些原因，在西方一些采用直接选举的国家采用网上投票的方法来达到方便快捷的目的。而我国采用的是直接选举和间接选举相结合的方式，即乡、县两级人大代表由选民直接选举产生，然后由县级人大代表选举产生省级人大代表，最后由省级人大代表选举产生全国人大代表[1]。在这种选举方式下网上投票则并不适合，在我国采用电脑计票的方式来克服普通投票方式的缺点。

1.2 投票表决系统研究背景

自从开始出现使用电脑进行选票统计以来，电子投票表决系统形式可以分为以下三种形式：检票箱方式、电子表决器方式和网上投票方式。

1.2.1 检票箱方式

这种方式是在投票箱中安装能自动识别选票的电子、机械装置，这些装置将选票信息传送给计算机，由计算机对选票进行分析和统计。这种方式已经发展了两代：第一代计算机控制管理的投票表决系统和第二代计算机控制管理的投票表决系统。

第一代计算机控制管理的投票表决系统在投票箱中安装的是一套基于光电识别的系统[2]。它利用光电传感器阵列读取选票上固定位置的信息点是否被涂黑，来统计选票上的信息，其原理类似于阅卷机系统[3]。系统对选票的纸张和印刷的质量要求较高；对选票的设计格式有严格的限制；对于有另选人的选票还要有专门的机械结构进行分拣，以便人工统计。以上各点都限制了第一代投票系统的广泛应用[3]。

第二代计算机控制管理的投票表决系统与第一代投票系统有着本质区别，这个系统在投票箱内安装 CCD 摄像头，对投入的选票进行照相，计算机对所得的照片数据进行分析从而得到选票信息。

检票箱方式将计算机代替人工进行选票统计，大大缩短了选票统计的时间，提高了选票统计的正确性，但仍然需要选民将选票投入到选票箱中，不能最大的缩短会议的时间。

1.2.2 电子表决器方式

这种方式在每个选民手中都持有一个电子表决器，选民通过电子表决器上的按钮对候选人进行投票。电子表决器通过有线或无线的方式将选民意愿传送给统计计算机。

这种方式无需选民将选票投入投票箱，投票表决和选票统计的速度很快，准确性也很高。但目前的电子表决器相当简单，往往只有几个按钮，不能在表决器上显示候选人的信息，候选人的信息需要通过大屏幕集中显示。选举的过程比较麻烦，需要对候选人逐一投票，选票的有效性很难得到保证。

1.2.3 网上投票方式

网上投票方式是建立在完善的网络建设之上的一种投票方式，这种方式在西方一些采用直接选举的国家被采用。其实质是一个采集数据的网站，选民通过访问网站将自己的意愿传送到网站数据库进行统计[4]。这种方式方便、灵活、快捷，只要有网络、有计算机便可以进行投票表决，适合于大型分散的投票表决。

1.3 系统设计目标和特点

针对传统的投票方式的缺点和目前投票系统的不足，本课题研究并开发一套全新的电子投票表决系统——无线下载电子投票表决系统。该系统采用电子表决器代替传统的选票显示选票的内容、采集选民的意愿；采用计算机计票代替人工计票；电子表决器与统计计算机之间通过无线通信的方式进行数据交换，选票信息可以通过电脑编辑后下载到电子表决器中。

系统性能指标：

- 有效作用距离：室内视距 100m 范围内；
- 最高支持投票表决人数：3000 人；
- 最长计票时间：最后一个选民按下发送按钮 1 分钟内完成计票任务；
- 投票结果能自动存储和打印；
- 和大屏幕相连，集中显示投票内容和结果；
- 在规则较简单的投票表决活动中，能检查并提醒选票的有效性。

与现有的其它投票表决系统相比，本文中的无线下载电子投票表决系统有以下几个特点：

- 能将需要投票表决的内容在计算机中编辑后通过无线方式下载到电子表决器中，投票人在电子表决器中就能看到全部要进行投票的内容；

• 添加了对选票的有效性检测和提醒功能，能有效地减少废票数，提高投票表决质量；

• 采用无线通信的方式，使无线投票系统使用方便灵活；

• 采用先进的数据加密算法对信息进行加密，提高了系统的安全性；

• 采用超低功耗、高性能的嵌入式处理器作为电子表决器的核心，采用低功耗的无线通信芯片担任通信任务，大大降低了电子表决器的功耗，延长了电池的使用寿命，减小了电子表决器的体积，减轻了其重量，降低了系统运行成本；

• 采用性能优异的嵌入式实时操作系统作为电子表决器的开发平台，与其它嵌入式操作系统相比具有实时性高、所需的资源少、开发成本低的优点。

1.4 本课题完成的工作

本课题在对无线下载电子投票表决系统的各个方面作了大量的研究、设计和试验工作的基础上，完成了样机的制作、调试和测试。本课题首先完成了系统结构设计和各模块元器件的选型，完成了电路原理图和 PCB 图的设计，以及实物外形的设计。其次，研究了低功耗无线通信网络的设计，其中包括数据帧结构的设计、命令设计和工作过程设计，并对所设计的无线通信网络进行了实际验证。再次，完成了μC/OS-II 在无线下载电子投票表决系统中的移植和优化。然后，编写了上位机控制软件，并编写了 AES 数据加密算法的 C 语言代码。最后，对整个系统进行了调试，对设计目标进行了检验。

1.5 本文的文章结构

本文分六章对无线下载电子投票系统的硬件、软件、通信协议以及信息加密等几方面进行了详细的阐述。

第一章是绪论，首先介绍了投票表决的演变和发展，其次介绍了当今中外投票表决系统的各种形式，接着就目前投票表决系统存在的缺点提出了本课题的设计目标，最后说明了本课题完成的工作和本文的文章结构。

第二章是硬件电路设计，在论述了系统结构、主机结构和电子表决器结构的基础上详细说明了各模块中元器件的选择和使用，并对个别重要的模块给出了原理图。

第三章介绍了操作系统设计，从操作系统的内核结构、任务管理、任务间通信以及操作系统在无线下载电子投票表决系统中的移植和优化等几方面对其进行了详细的说明。

第四章阐述了上位机控制软件的设计，从内部的数据结构到控制界面都作了详细的说明，还说明了一些关键控件的使用方法。

第五章阐述了无线下载电子投票表决系统的通信协议设计，包括主机与电子表决器间的无线通信协议和主机与统计计算机间的有线通信协议，从数据帧格式、命令定义和系统工作过程等方面进行了详细说明。

第六章阐述数据加密设计，详细说明了数据加密和解密的算法过程，并且给出了数据加密算法的安全性和效率分析。

第七章是本文的总结与展望，对本课题设计开发的无线下载电子投票表决系统进行了总结，并对有待改进的地方提出了一些看法。

2 系统的硬件电路设计

2.1 系统的硬件结构

2.1.1 系统结构

投票表决系统由：计算机、主机、电子表决器和大屏幕显示器等四部分组成。系统结构如图 2.1 所示，各部分功能如下：

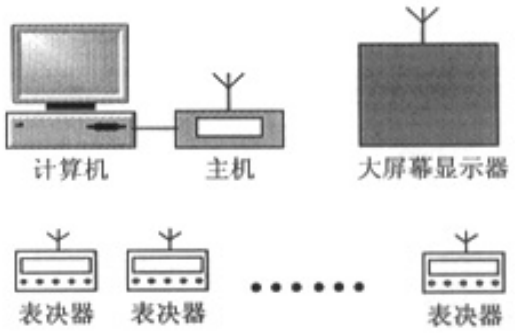


图 2.1 无线投票系统结构

Fig. 2.1 Structure of wireless voting system

计算机 计算机是整套系统的核心，它有三个功能：输入候选人名单或表决的内容，统计投票结果，控制投票进程；

主机 主机是计算机与电子表决器和大屏幕显示器的通信的桥梁和纽带，它负责无线通信网络的建立和维护；

电子表决器 电子表决器是系统与选举人信息交换的窗口，它显示候选人名单和议题信息，采集选举人意愿并将其传送给计算机，在投票结束后显示投票结果；

大屏幕显示器 大屏幕显示器对候选人信息、表决议题、投票结果进行集中和详细显示。

2.1.2 主机结构

主机是连接计算机和电子表决器、大屏幕显示器的桥梁，它负责无线通信网络的建立和维护。主机由核心处理器 MSP430F149 及其外围电路组成，其结构如图 2.2 所示，各部分功能如下：

无线通信模块将数据报通过无线电波发送给电子表决器和大屏幕显示器，接收由电子表决器和大屏幕显示器发送过来的数据。在发送时，无线通信模块自动在数据报前加

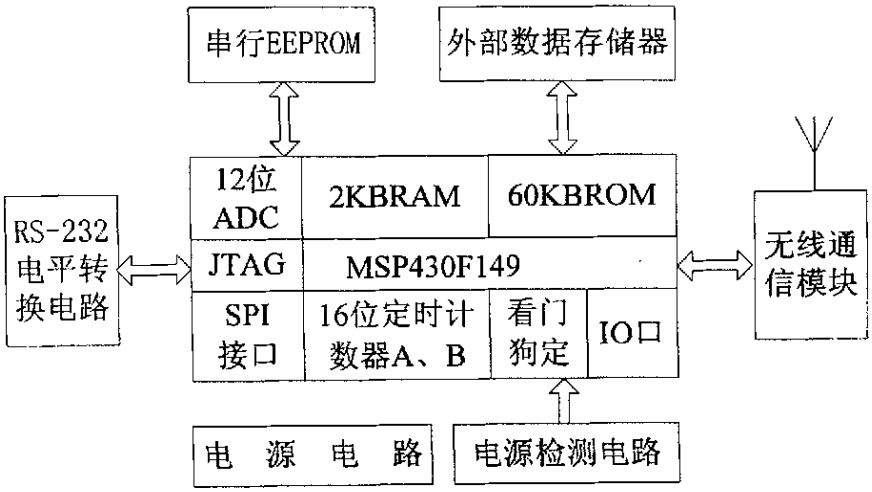


图 2.2 主机硬件结构

Fig.2.2 Hardware structure of server

上同步头和识别码，在数据报尾加上 CRC 校验；在接收时识别数据报是否属于本系统，并检验数据的正确性。

主机与计算机通信采用 RS-232 通信，RS-232 电平转换电路将单片机的 TTL 电平转换成 RS-232 电平，将计算机的 RS-232 电平转换成 TTL 电平。RS-232 电平转换芯片采用 MAXIM 公司的 MAX3232。

外部数据存储器用来存储两部分数据和信息：与计算机、电子表决器、大屏幕显示器通信过程中的数据（作为数据的缓存）、建立的无线通信网络信息和各终端的状态标志。

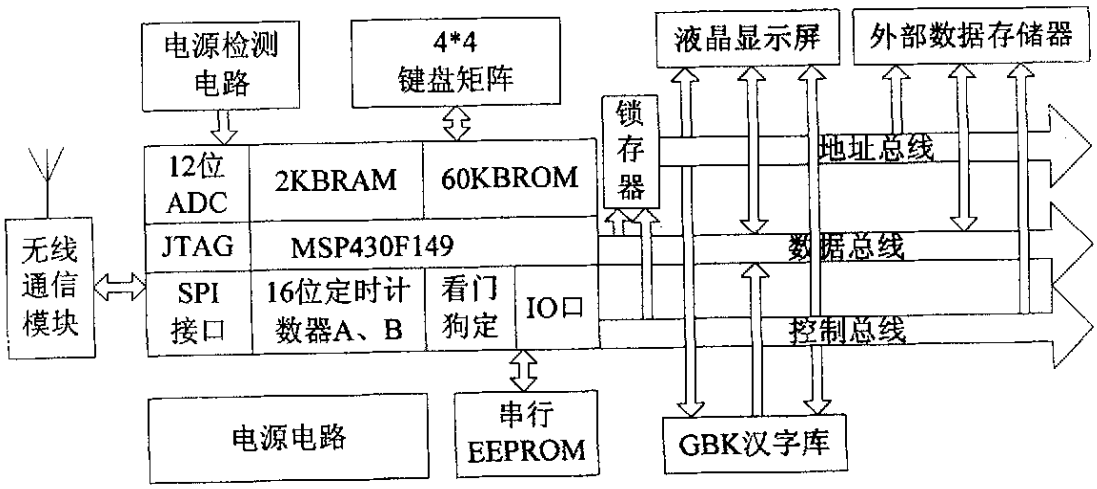


图 2.3 电子表决器硬件结构

Fig.2.3. Hardware structure of electronic voting machine

串行 EEPROM 用以存储系统识别码、本机地址、无线通信的工作信道和投票结果等信息，采用 ATMEL 公司的 AT24C32。

2.1.3 电子表决器结构

电子表决器由核心处理器 MSP430F149、无线通信模块、液晶显示屏、汉字库、键盘和数据存储器等组成。电子表决器的硬件结构如图 2.3 所示。

2.2 无线通信模块设计

无线投票系统对无线通信模块的要求：

- 较远的通信距离，以适应较大的会议场所；
- 较强的抗干扰能力，使系统在较强的外界干扰中也能正常工作；
- 较多可供选择的工作信道，尽可能的避开存在干扰的信道；
- 较高的传输速率，以满足较大数据量的传输要求；
- 较低的功耗，以降低系统的功耗。
- 工作在民用频段，方便使用；

2.2.1 无线收发芯片 nRF905

nRF905 是挪威 Nordic 公司推出的一款单片无线收发一体的芯片。

1. Nordic 公司的 RF 芯片系列

Nordic 公司是 1983 年在挪威成立的专门从事 ASIC 设计的电子公司，其主要产品是 RF 收发芯片。自推出第一款 RF 收发芯片 nRF401 以来，Nordic 公司已经开发了十几款 RF 芯片。

表 2.1 Nordic 公司 RF 系列芯片

型号	功能	工作频段	调制方式	通信速率	最大发射功率	灵敏度
nRF401	收、发	433MHz	FSK	20kbps	+10dBm	-105dBm
nRF402	发射	433MHz	FSK	20kbps	+10dBm	-105dBm
nRF403	收、发	315/433MHz	FSK	20kbps	+10dBm	-105dBm
nRF902	收、发	868MHz	FSK	50kbps	+10dBm	-105dBm
nRF903	收、发	433/868/915MHz	FSK/GMSK	76.8kbps	+10dBm	-100dBm
nRF905	收、发	433/868/915MHz	GFSK	100kbps	+10dBm	-100dBm
nRF2401	收、发	2.4GHz	GFSK	1Mbps	0dBm	-90dBm
nRF2402	收、发	2.4GHz	GFSK	1Mbps	0dBm	-90dBm

2. nRF905 的特点

nRF905 是 Nordic 公司成功的 RF 收发芯片之一，其功能特点如下[5]：

• 工作在 433/868/915MHz 三个 ISM(工业、科学和医学)频段，使用时无须申请许可证；

• 最大的发射功率 10dBm，接收灵敏度-100dBm，在开阔地的最大传输距离可达到 1000m；

• nRF905 采用抗干扰能力强的 GFSK 调制，具有很强的抗干扰能力；

• 有多个工作信道可供选择；

• 传输速率 100kbps；

• 极低的功耗，并且有多种工作模式可供选择，发射功率和接受灵敏度可以调节；

• 工作电压范围宽 1.9～3.6V，适合于低电压工作场合；

• 工作模式之间和信道之间的切换时间短，见表 2.2。

表 2.2 不同工作模式间的切换时间

切换模式	最大时间
关机→待机	3ms
待机→发送	650μs
待机→接收	650 μs
接收→发送	550 μs
发送→接收	550 μs

• 将低噪声放大器 LNA 、功率放大器 PA 、压控振荡器 VCO 、GMSK 调制解调等大部分功能集成在芯片内，外围电路简单易于开发；

• 集成了曼切斯特编解码、CRC 校验、组帧等功能，易于程序设计。

3. nRF905 结构

nRF905 是一款高度集成的 RF 芯片，它内部集成了低噪声放大器 LNA、功率放大器 PA、压控振荡器 VCO、GFSK 调制解调、频率基准源、PLL 频率合成、数字静噪电路以及单片机接口电路，它的内部结构图如图 2.4 所示。

4. nRF905 典型应用

由于很多功能都在片内集成，因此使用时只需使用少量的外围器件，无需使用价格昂贵的变容管、声表滤波器等，大大降低了成本，增加了开发的灵活性，而且易于调试

开发。nRF905 的典型工作电路如图 2.5 所示。

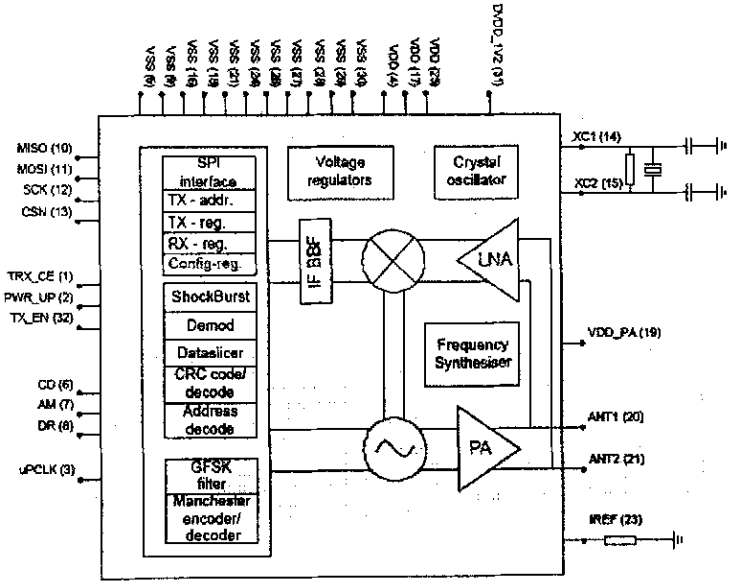


图 2.4 nRF905 内部结构和引脚分布

Fig.2.4 Internal structure of nRF905 and pinout

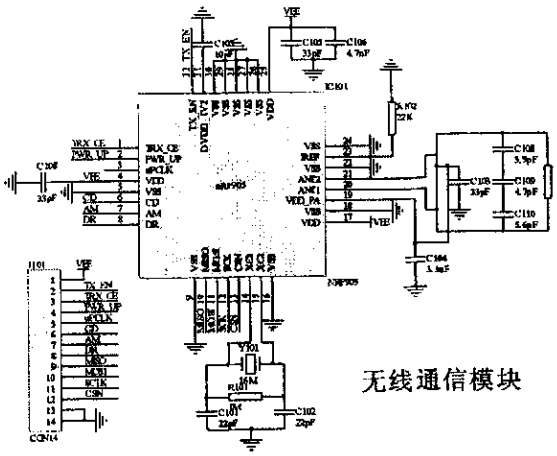


图 2.5 nRF905 典型工作电路

Fig.2.5 Typical application of nRF905

5. nRF905 与单片机接口

nRF905 采用串行外围设备接口 SPI（Serial Peripheral Interface）与单片机连接，单片机工作在 SPI 的主机模式，nRF905 工作在从机模式，接口图如图 2.6 所示。

串行外围设备接口 SPI 总线技术是 Motorola 公司推出的一种同步串行接口。它由三根信号线组成：串行时钟（SCLK）、主入从出（MISO）、主出从入（MOSI）。连接在

SPI 总线上的设备工作在主机、或从机模式，主机控制串行时钟（SCLK），产生传输数据所需要的时钟信号。在时钟的上升沿锁存数据，在下降沿改变总线上的数据。

对 nRF905 进行读、写操作时，必须先通过 CSN 的由高到低的跳变来使能 nRF905，读、写操作的时序如图 2.7、2.8 所示。

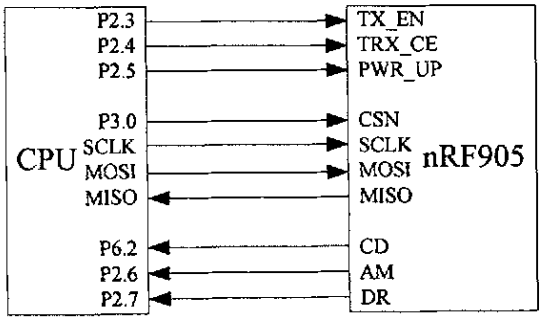


图 2.6 nRF905 与单片机接口

Fig.2.6 Connection between nRF905 and CPU

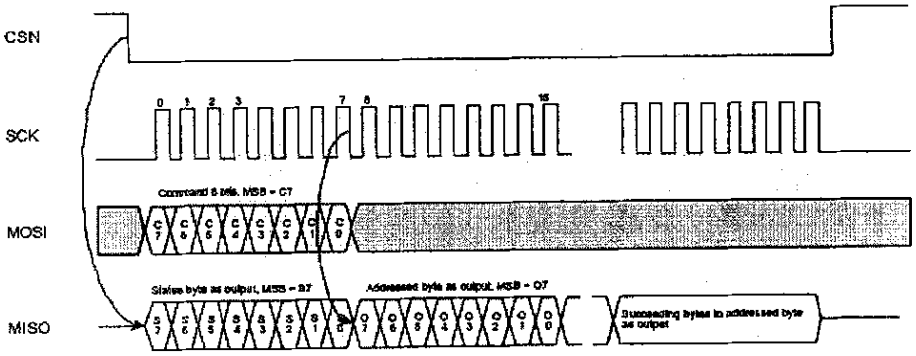


图 2.7 SPI 读操作

Fig.2.7 SPI readoperation

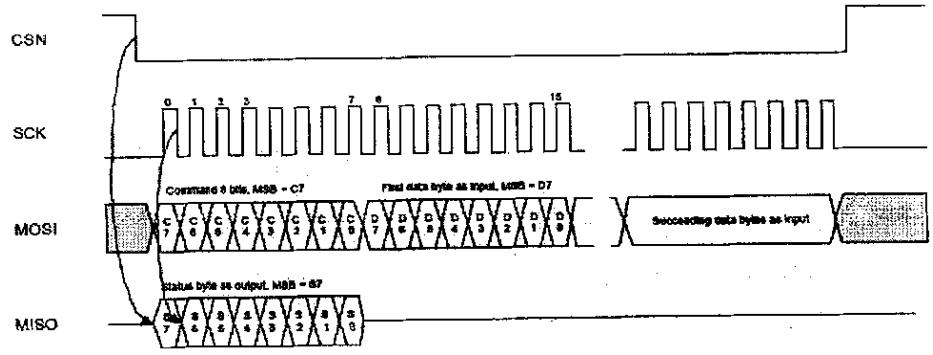


图 2.8 SPI 写操作

Fig.2.8 SPI write operation

6. nRF905 的工作模式

nRF905 有两种工作模式和两种节能模式可供选择。两种工作模式分别是：ShockBurstTM 接收模式和 ShockBurstTM 发送模式；两种节能模式分别是关机模式和待机模式。通过控制 TX_EN、TRX_CE 和 PWR_UP 的电平可控制 nRF905 的工作模式，见表 2.3。

表 2.3 nRF905 的工作模式

PWR_UP	TRX_CE	TX_EN	工作模式
0	X	X	关机模式
1	0	X	待机模式
1	1	0	ShockBurstTM 接收模式
1	1	1	ShockBurstTM 发送模式

ShockBurstTM 工作模式 Nordic 公司的 ShockBurstTM 工作模式使低速的处理器也能使用高速的 nRF905 数据报发射率。nRF905 将与 RF 协议有关的高速数据处理集成在片内而仅仅通过 SPI 接口与处理器相连，SPI 接口的速度由处理器决定，这种工作方式有利于节能。在 ShockBurstTM 接收模式下，当一个包含正确地址和数据的数据报被接收到后，地址匹配(AM)和数据准备就绪(DR)两引脚通知微控制器。在 ShockBurstTM 发送模式，nRF905 自动产生字头和 CRC 校验码，当发送过程完成后，数据准备就绪引脚通知微处理器数据发射完毕。由以上分析可知，nRF905 的 ShockBurstTM 收发模式有利于节约存储器和微控制器资源，同时也减小了编写程序的时间。下面具体分析 nRF905 的发送流程和接收流程。

发送流程 nRF905 典型的发送流程分以下几步[5]：

- 当微控制器有数据要发送时，通过 SPI 接口按时序把接收机的地址和要发送的数据传送给 nRF905；
- 微控制器置高 TRX_CE 和 TX_EN，激发 nRF905 的 ShockBurstTM 发送模式；
- nRF905 进行 ShockBurstTM 发送；
- 射频寄存器自动开启；
- 数据打包(加字头和 CRC 校验码)；
- 发送数据报；
- 当数据发送完成，数据准备就绪引脚被置高；

• 如果 AUTO_RETRAN 被置高，nRF905 不断重发，直到 TRX_CE 被置低；

• 当 TRX_CE 被置低，nRF905 发送过程完成，自动进入空闲模式。

ShockBurstTM 发送模式保证一旦发送数据的过程开始，无论 TRX_EN 和 TX_EN 引脚是高或低，发送过程都会被处理完。只有在前一个数据报被发送完毕，nRF905 才能接收下一个发送数据报。发送时序图如图 2.9 所示。

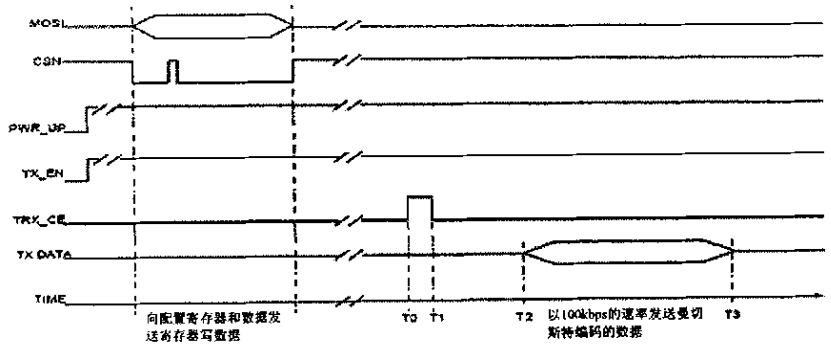


图 2.9 发送过程时序图

Fig.2.9 Timing diagram for transmit operation

接收流程　nRF905 典型的接收流程分以下几步[5]：

• 当 TRX_CE 为高、TX_EN 为低时，nRF905 进入 ShockBurstTM 接收模式；

• 650us 后，nRF905 不断监测，等待接收数据；

• 当 nRF905 检测到同一频段的载波时，载波检测引脚被置高；

• 当接收到一个相匹配的地址，地址匹配引脚被置高；

• 当一个正确的数据报接收完毕，nRF905 自动移去字头、地址和 CRC 校验位，然后把数据准备就绪引脚(DR)置高；

• 微控制器把 TRX_CE 置低，nRF905 进入空闲模式；

• 微控制器通过 SPI 口，以一定的速率把数据移到微控制器内；

• 当所有的数据传送完毕，nRF905 把数据准备就绪引脚和地址匹配引脚置低；

• nRF905 此时可以进入 ShockBurstTM 接收模式、ShockBurstTM 发送模式或关机模式。

当正在接收一个数据报时，TRX_CE 或 TX_EN 引脚的状态发生改变，nRF905 立即把其工作模式改变，数据报则丢失。当微处理器接到地址匹配引脚的信号之后，就知道 nRF905 正在接收数据报，其可以决定是让 nRF905 继续接收该数据报还是进入另一个工作模式。接收时序图如图 2.10 所示。

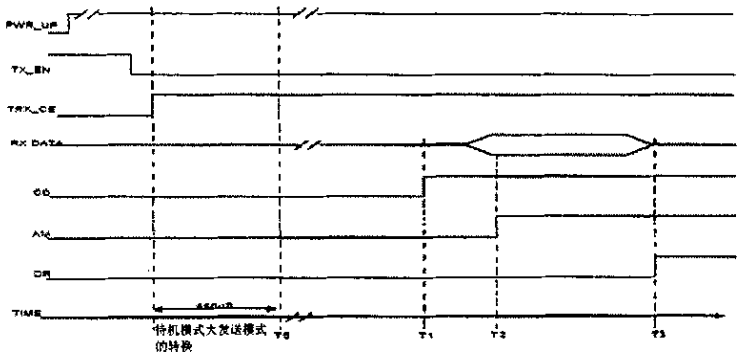


图 2.10 接收过程时序图

Fig.2.10 Timing diagram for receiving operation

节能模式　nRF905 的节能模式包括关机模式和待机模式。工作在关机模式时，nRF905 的工作电流最小，一般为 2.5uA。进入关机模式后，nRF905 保持配置字中的内容，但不会接收或发送任何数据。待机模式有利于减小工作电流，其从待机模式到发送模式或接收模式的启动时间也比较短。在待机模式下，nRF905 内部的晶体振荡器部分处于工作状态。nRF905 在待机模式下的工作电流跟外部晶体振荡器的频率有关。

7. 配置 nRF905

所有对 nRF905 的配置都通过 SPI 接口来完成，SPI 接口包括五类内部寄存器：Status-Register 、 RF-Configuration Register 、 TX-Address 、 TX-Payload 、 RX-Payload，见图 2.11。用表 2.4 所示的 8 条指令可对这些寄存器进行读写操作。

表 2.4 nRF905 配置命令

命令名	命令结构	操作
W_CONFIG(WC)	0000AAAA	写配置寄存器，AAAA 指定开始写操作的寄存器
R_CONFIG(RC)	0001AAAA	读配置寄存器，AAAA 指定开始读操作的寄存器
W_TX_PAYLOAD(WTP)	00100000	写发送寄存器，1-32 字节，始终从 0 字节开始
R_TX_PAYLOAD(RTP)	00100001	读发送寄存器，1-32 字节，始终从 0 字节开始
W_TX_ADDRESS(WTA)	00100010	写发送地址，1-4 字节，始终从 0 字节开始
R_TX_ADDRESS(RTA)	00100011	读发送地址，1-4 字节，始终从 0 字节开始
R_RX_PAYLOAD(RRP)	00100100	读接收寄存器，1-32 字节，始终从 0 字节开始
CHANNEL_CONFIG(CC)	1000pphc	快速设置 nRF905 的工作频率，在配置寄存器中
	ccccccccc	CH_NO=ccccccccc，HFREQ_PLL=h，PA_PWR=pp

表 2.5 RF 配置寄存器

字节号	内容 bit[7:0] 高位在前	初始值
0	CH_NO[7:0]	01101100
1	bit[7:6]未用, AUTO_RETRAN, RX_RED_PWR, PA_PWR[1:0], HFREQ_PLL, CH_NO[8]	00000000
2	bit[7]未用, TX_AFW[2:0], bit[3]未用, RX_AFW[2:0]	01000100
3	bit[7:6]未用, RX_PW[5:0]	00100000
4	bit[7:6]未用, RX_PW[5:0]	00100000
5	RX_ADDRESS(设备地址)字节 0	E7
6	RX_ADDRESS(设备地址)字节 1	E7
7	RX_ADDRESS(设备地址)字节 2	E7
8	RX_ADDRESS(设备地址)字节 3	E7
9	CRC_MODE, CRC_EN, XOF[2:0], UP_CLK_EN, UP_CLK_FREQ[1:0]	11100111

RF 工作配置寄存器由 10 个字节组成，这 10 个字节指定了 RF 的工作频段、频率、发射功率、接收灵敏度、数据报的长度及校验类型等。各字节的作用如表 2.5 所示。

对配置寄存器的读写时序如图 2.7、2.8 所示。

RF 工作中心频率由 HFREQ_PLL 和 CH_NO 共同决定，计算公式如式 2.1。

$$f_{op} = (422.4 + (CH\_NO/10)) \times (1 + HFREQ\_PLL)MHz \tag{2.1}$$

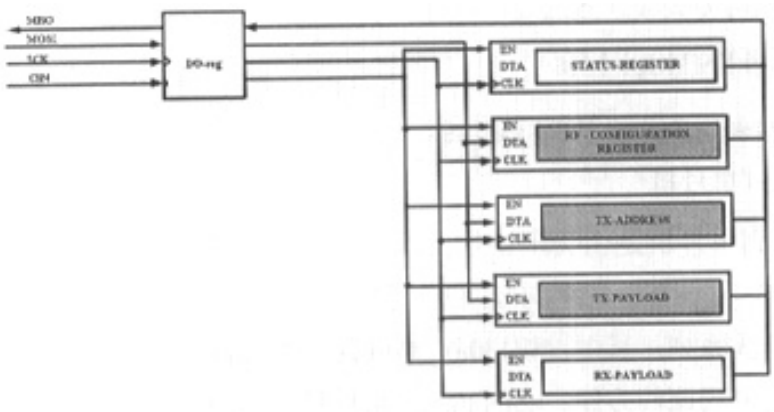


图 2.11 SPI 接口和 5 类内部寄存器

Fig.2.11 SPI-interface and the five internal registers

2.2.2 无线通信模块 PCB 设计

由于工作在RF频段，走线对无线通信模块的质量会有很大影响，即使一根很短的导线也会如电感一样，粗略估算，每毫米长度导线的电感量约为1nH[6]；而接收电路中的高增益放大器对噪声相当敏感。因此，要使无线通信模块有好的通信效果，在PCB设计中必须注意以下几点：

• 至少采用双层PCB，布置一个可靠的地线层。在表面贴装的PCB上，所有信号走线可在元件安装面的同一面，地线层则在其反面。理想的地线层应覆盖整个PCB(但要注意天线处除外)。如果不采用地线层，大多数地线将会较长，电路的设计特性将无法保证；

• 如果采用两层以上的PCB，地线层应放置在邻近信号层的层上（如元件面的下一层）；

• 将信号布线层的空余部分也用地线平面填充。这些地线平面必须通过很多过孔与主地线层面连接；

• 另外，所有对地线层的连接应尽量短。接地过孔应放置在（或非常接近）元件的焊盘处。决不要让两个地信号共用一个接地过孔，避免由于过孔连接阻抗在两个焊盘之间产生串扰；

• 在离VDD引脚尽可能近的地方用高性能的RF电容去耦；

• 电源必须经过良好的滤波，并且与数字电路供电分离。

天线采用在板环形 PCB 天线。PCB 天线具有成本低，方向性较好，对人体不敏感的特点，特别适合在手持产品中使用。环形天线通常应用于相对较窄的带宽，有助于抑制强的不需要的信号以免干扰接收器。在天线附近避免排布数字信号线路，并使天线周围保持自由空间，可使天线达到较好的效果。

2.3 核心处理器MSP430F149

核心处理器采用单片机MSP430F149。

2.3.1 MSP430 系列单片机

MSP430 系列单片机是由美国德州仪器（TI）开发研制的一种具有超低功耗特点的功能强大的 16 位 RISC 结构单片机。它有 MSP430x1x、MSP430x2x、MSP430x3x、MSP430x4x 等几大系列；其中 MSP430Fx 系列具有 FLASH 存储器，使系统设计、开发调试和实际应用都变得比较方便。MSP430F1x 系列是 MSP430FLASH 型单片机中的一种，其具有以下特点：

• 超低功耗　运行在 1MHz 时钟条件下时，工作电流视工作模式不同为 0.1～400　，工作电压为 1.8～3.6V。

• 强大的处理能力　MSP430F1x 具有丰富的寻址方式（7 种元操作数，4 种目的操作数），但只需要简洁的 27 条指令；片内寄存器数量多，存储器可实现多种运算；具有高效的查表处理方法。这些特点保证了可以编制出高效的程序。MSP430F1x 具有较多的中断源，并且可以任意嵌套，使用时方便灵活。当系统处于省电模式时，只需要 6s 就能用中断请求将它唤醒。

• 丰富的片上外围模块　MSP430F1x 中的各成员集成了较多的片上外围资源，如：12 位 ADC、精密模拟比较器、硬件乘法器、带有捕获/比较寄存器的 16 位定时器、看门狗、可实现异步/同步及多址访问的串行通信接口、数十个可实现方向设置及中断功能的并行输入输出端口等。

• 方便高效的开发方式　具有 FLASH 存储器的 MSP430F1x 使它的开发工具相当简便。利用单片机本身具有的 JTAG 接口或片内 BOOT ROM，可以在一台 PC 机和一个结构小巧的 JTAG 控制器的帮助下实现程序的下载，完成程序调试。

• MSP430F1x 的各个型号大多有性能相同而存储器不同的 ROM 型、OTP 型，以适应产品在设计、开发、生产的各个不同阶段的需要。

• 适应工业级运行环境　MSP430F1x 的运行环境温度范围为-40～+85℃，所设计的产品适合运行于工业环境下。这个系列目前有 MSP430F11x、MSP430F11x1、MSP430F13x、MSP430F14x、MSP430F15x、MSP430F16x 等型号，所集成的存储器容量及外围模块各不相同，可适应不同需要。

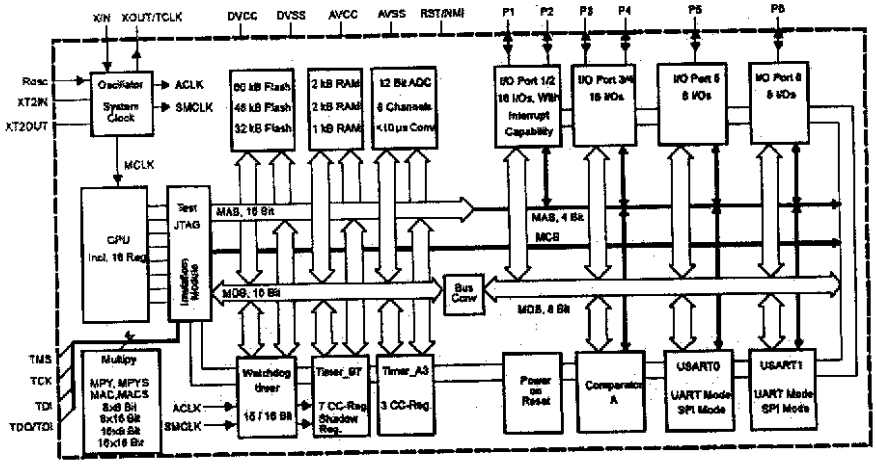


图 2.12 MSP430F14x 系列单片机内部结构图

Fig.2.12 Internal structure of MSP430F14x

1．MSP430 系列单片机的内部结构

MSP430 系列单片机采用存储器-存储器结构，即用一个公共的空间对全部功能模块寻址，同时用精简指令组对全部功能模块进行操作[8]。由于不同型号的单片机具有不同的外围模块和不同大小的存储器，因此在内部结构上有细微的差别。图 2.12 是 MSP430F14x 系列单片机内部结构图。

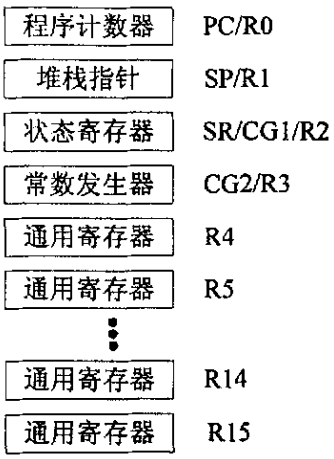


图 2.13 MSP430 的 CPU 结构
Fig.2.13 CPU structure of MSP430

2．MSP430 系列单片机的 CPU

MSP430 系列单片机的 CPU 采用正交的精简指令集，由 16 位 ALU、指令控制逻辑和 16 个寄存器组成[8]，如图 2.13 所示。

寄存器中有 4 个具有特殊用途，即：程序计数器 PC/R0、堆栈指针 SP/R1、状态寄存器和常数发生器 SR/CG1/R2、常数发生器 CG2/R3。除常数发生器 CG1 和 CG2，所有寄存器都可作为通用寄存器，用所有指令操作[8]。

3．MSP430 系列单片机的存储空间

MSP430 单片机的存储空间采用“冯·诺依曼”结构，ROM、RAM 和外围模块由同一组地址及数据总线连接在同一地址空间中，因此可以用相同的指令访问 ROM、RAM 和外围模块，也可以执行 RAM 中的程序代码[8]。无论 ROM、RAM、SFR 或外围模块，都位于同一公共地址空间中，寻址空间为 64KB。图 2.14 为 MSP430 存储空间结构图。

数据总线可以是 16 位或 8 位宽度[8]。16 位外围模块用 16 位数据宽度（字）访问，8 位外围模块用 8 位数据宽度（字节）访问。程序存储器（ROM）和数据存储器（RAM）可以用字也可以用字节进行访问。

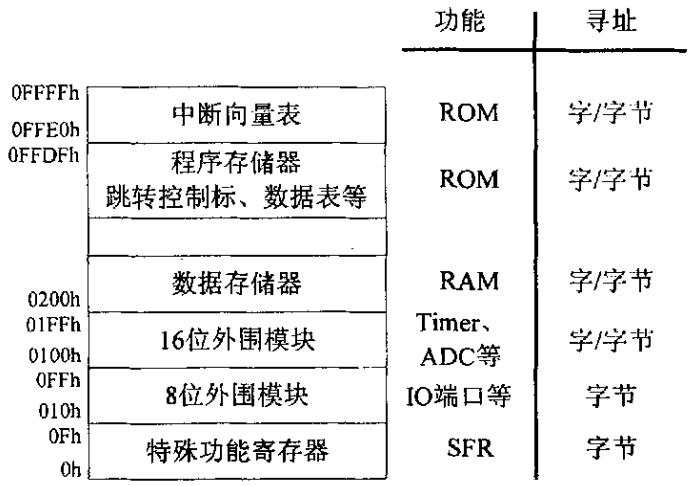


图 2.14 MSP430 存储空间基本结构
Fig.2.14 Memory structure of MSP430

字节数据可以定位在偶地址或奇地址。字数据定位在偶地址，低字节在偶地址，高字节在奇地址，因此，字指令访问时总是用偶地址。

ROM 的开始地址因 ROM 体积大小的不同而不同。中断向量位于 ROM 的最高 16 字地址区。其中最高优先级的向量位于最高的 ROM 字地址 0FFFEh。

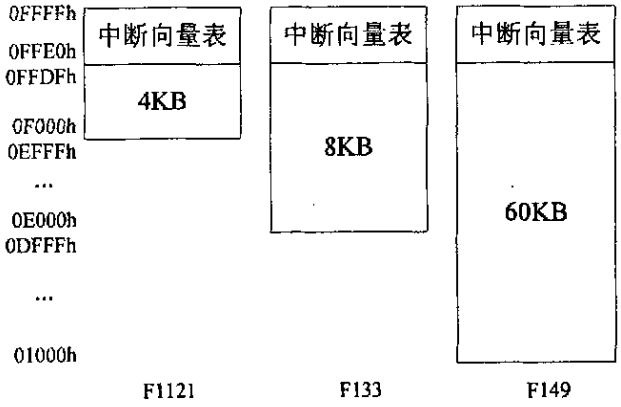


图 2.15 不同型号 ROM 地址分布
Fig.2.15 ROM distributing of different chip

4. MSP430 系列单片机的中断系统

MSP430 系列单片机有丰富的中断资源，这些中断可以分为四类：

- 系统复位
- 可屏蔽中断（maskable）

表 2.6 F13x 及 F14x 中断向量表

中断源	中断标志	系统中断	地址	优先级
上电、外部复位				
看门狗复位	WDTIFG	复位	0FFFE	15, 最高
FLASH	KEYV			
NMI	NMIIFG	（非）屏蔽		
振荡器故障	OFIFG	（非）屏蔽	0FFFC	14
FLASH 存储器非法访问	ACCVIFG	（非）屏蔽		
Timer_B7	BCCIFG0	可屏蔽	0FFFA	13
Timer_B7	BCCIFG1 TBIFG	可屏蔽	0FFF8	12
比较器 A	CMPAIFG	可屏蔽	0FFF6	11
WDT	WDTIFG	可屏蔽	0FFF4	10
USART0 接收	URXIFG0	可屏蔽	0FFF2	9
USART0 发送	UTXIFG0	可屏蔽	0FFF0	8
ADC	ADCIFG	可屏蔽	0FFEE	7
Timer_A3	CCIFG0	可屏蔽	0FFEC	6
Timer_A3	CCIFG1 CCIFG2 TAIFG	可屏蔽	0FFEA	5
P1	P1IFG.0-7	可屏蔽	0FFE8	4
USART1 接收	URXIFG1	可屏蔽	0FFE6	3
USART1 发送	UTXIFG1	可屏蔽	0FFE4	2
P2	P2IFG.0-7	可屏蔽	0FFE2	1
		可屏蔽	0FFE0	0，最低

- 非屏蔽中断（non-maskable）
- （非）屏蔽中断（（non）-maskable）

系统复位包括：上电复位（POR）和上电清除（PUC）。上电复位在：芯片上电和在 RST/NMI 引脚上出现低电平信号（RST/NMI 被设置成复位模式）时产生。上电清除在：

上电复位、看门狗计时溢出、看门狗定时器写入错误的安全键值和 FLASH 存储器写入错误的安全键值时产生。上电复位总是会产生上电清除，但上电清除不会产生上电复位。

非可屏蔽中断是无法屏蔽的，这些中断既无中断允许位，也不会受通用中断允许位（GIE）的影响。

（非）屏蔽中断不能用通用中断允许位（GIE）屏蔽，但是可以用各自的中断允许位控制。当一个（非）屏蔽中断请求被接受时，相应的中断允许位就自动复位，因此也禁止了新的中断请求。RETI 指令不影响相应的（非）屏蔽中断的允许位，因此必须用软件在中断服务程序中的 RETI 指令前将中断允许位置位，以保证在中断服务结束后能再次被允许。（非）屏蔽中断可以有以下条件产生：

- 当选择 NMI 模式时，RST/NMI 引脚产生跳变沿。
- 当振荡器失效中断允许时，发生时钟失效。
- 当非法访问中断允许时，发生对 FLASH 存储器的非法访问。

中断向量表位于 ROM 的最高地址的 16 个字。每一个中断源有唯一的中断向量与之对应，但一个中断向量所对应的中断源并不唯一。中断向量在中断向量表中的地址越大对应中断的优先级越高。由于开发时间的先后不同，不同型号的单片机的中断向量表设计有较大的差别，表 2.6 是 MSP430F13x 及 MSP430F14x 的中断向量表。

5. 工作模式

MSP430 是为超低功耗应用开发的，它有多种工作模式来实现这一功能。MSP430 的各种工作模式状态以强有力的方式支持超低功耗的各种要求，这是通过各模块的智能化运行管理和 CPU 的状态组合而得到的。一个中断事件可以将系统从各种工作模式中唤醒，而 RETI 指令又使 MSP430 返回到中断事件发生前的工作模式[8]。

MSP430 单片机通过控制位于状态寄存器（SR）中的 4 个功能位：SCG1（系统时钟发生器控制位 1）、SCG0（系统时钟发生器控制位 0）、OSCOff（晶振关闭位）和 CPUOff（CPU 关闭位）来控制单片机工作在不同的节能模式。通过软件可组合成 6 种工作模式：

- 活动模式（AM）：SCG1=0，SCG0=0，OSCOff=0，CPUOff=0；CPU、时钟处于活动状态。
- 低功耗模式 0（LPM0）：SCG1=0，SCG0=0，OSCOff=0，CPUOff=1；CPU 被禁止；系统主时钟（MCLK）被禁止；系统从时钟（SMCLK）、辅助时钟（ACLK）保持活动。
- 低功耗模式 1（LPM1）：SCG1=0，SCG0=1，OSCOff=0，CPUOff=1；CPU 被禁止；MCLK 被禁止；如果数控振荡源（DCO）未作用于 MCLK 或 SMCLK，又处于活动状态，则直

表 2.7 低功耗工作模式控制位

省电模式	SCG1	SCG0	OSCOff	CPUOff
LPM0	0	0	0	1
LPM1	0	1	0	1
LPM2	1	0	0	1
LPM3	1	1	0	1
LPM4	X	X	1	1

流发生器（DC）被禁止，否则仍处于活动状态；SMCLK、ACLK 保持活动。

• 低功耗模式 2（LPM2）：SCG1=1，SCG0=1，OSCOff=0，CPUOff=1；CPU 被禁止；MCLK 被禁止；SMCLK 被禁止；DCO 被禁止；ACLK 保持活动。

• 低功耗模式 3（LPM3）：SCG1=1，SCG0=1，OSCOff=0，CPUOff=1；CPU 被禁止；MCLK 被禁止；SMCLK 被禁止；DCO 被禁止；DCO 的 DC 发生器被禁止；ACLK 保持活动。

• 低功耗模式 4（LPM4）：SCG1=X，SCG0=X，OSCOff=1，CPUOff=1；CPU 被禁止；MCLK 被禁止；SMCLK 被禁止；DCO 被禁止；DCO 的 DC 发生器被禁止；ACLK 被禁止；晶振停止工作。

MSP430F13x/MSP430F14x 在不同工作模式下的电流消耗的典型值如下图。

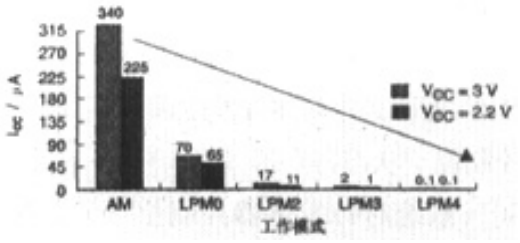


图 2.16 F13x/F14x 的工作模式和工作电流关系

Fig.2.16 F13x/F14x relation between current and working modes

6. JTAG 接口

MSP430 的 JTAG 接口是标准的 JTAG 仿真接口，仿真器通过 JTAG 接口可以调试系统的硬件和软件。JTAG 接口的连线图如图 2.17 所示。

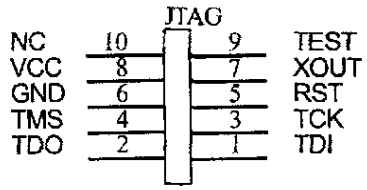


图 2.17 MSP430 的 JTAG 仿真接口
Fig.2.17 The JTAG port for MSP430

2.3.2 MSP430F149

MSP430F149 是 MSP430 FLASH 型单片机中资源较为丰富的一种，其有以下几部分组成[9]：

- 基础时钟模块，包括 1 个数控振荡器（DCO）和 2 个晶体振荡器；
- 看门狗定时器 Watchdog Timer，可用作通用定时器；
- 带有 3 个捕获/比较寄存器的 16 位定时器 Timer_A3；
- 带有 7 个捕获/比较寄存器的 16 位定时器 Timer_B7；

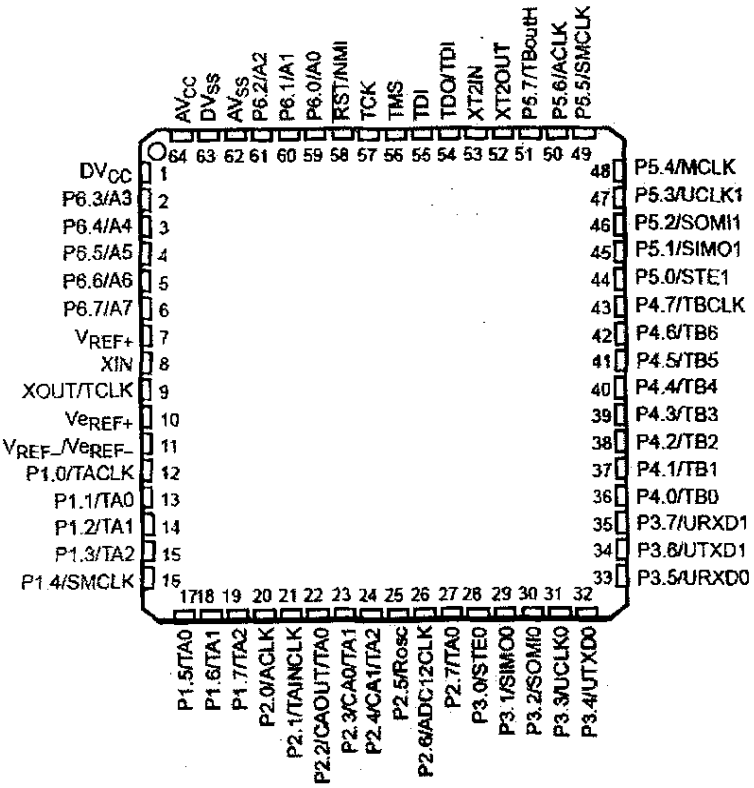


图 2.18 MSP430F149 引脚分布图
Fig.2.18 MSP430F149 pinout

- 2 个具有中断功能的 8 位并行端口：P1 与 P2；
- 4 个 8 位并行端口：P3、P4、P5 与 P6；
- 模拟比较器 Comparator_A；
- 12 位 A/D 转换器 ADC12；
- 2 个串行通信接口：USART0 与 USART1；
- 1 个硬件乘法器；
- 60KB+256 字节 FLASH；
- 2KB RAM。

MSP430F149 采用 64PM 封装形式，其引脚分布如图 2.18 所示。

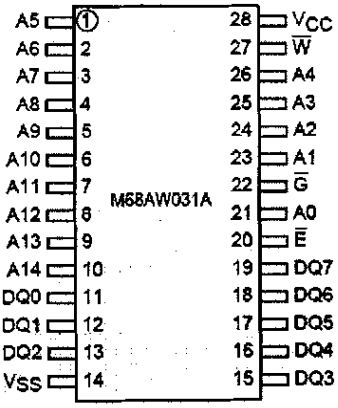


图 2.19 M68AW031A 引脚分布图

Fig.2.19 M68AW031A pinout

表 2.8 M68AW031A 的引脚功能

引脚名	引脚号	功能	描述
A0-A14	21，23-26，1-10	输入	地址输入
DQ0-DQ7	11-13，15-19	输入/输出	数据输入/输出
E	20	输入	片选信号，低电平有效
G	22	输入	输出允许，低电平有效
W	27	输入	输入允许，低电平有效
Vcc	28	电源	电源正，2.7~3.6V
Vss	14	电源	电源地，0V

2.4 外部数据存储器

虽然 MSP430F149 有较多的内存空间（2KB），但不足以存储大量的选票数据、工作状态等信息，因此需要外扩随机数据存储器。另外，也需要一定容量的非易失性存储器来存储设备信息、便于事后查询的投票结果等。

本设计中选择了 M68AW031A 作为外部随机数据存储器，选择 AT24C32 作为外部非易失性数据存储器。

2.4.1 外部数据存储器 M68AW031A

M68AW031A 是意法半导体公司生产的 256Kb（32KB×8）并行 SRAM，工作电压 2.7～3.6V，读、写周期 70ns。M68AW031A 的引脚分布图如图 2.19 所示，引脚功能如表 2.8 所示[10]。

2.4.2 AT24C32

AT24C32 是 ATMEL 公司生产的串行 EEPROM，容量 32Kb（4KB×8）。AT24C32 内部 4KB 的存储空间被分成 128 页，每页 32 字节[11]，通过 I^2C 总线（Inter IC Bus）与外部器件相连。AT24C32 的引脚分布图如图 2.20，引脚功能如表 2.9。

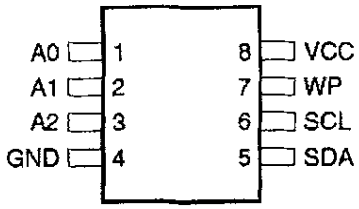


图 2.20 AT24C32 引脚分布图
Fig.2.20 AT24C32 pinout

表 2.9 AT24C32 引脚功能表

引脚名	引脚号	功能	描述
A0-A2	1-3	输入	地址输入
SDA	5	输入/输出	数据输入/输出
SCL	6	输入	时钟输入
WP	7	输入	写保护输入
VCC	8	电源	正电源，2.7~5.5V
VSS	4	电源	电源地，0V

AT24C32 与单片机的连接如图 2.21 所示。

I²C 总线是荷兰 Philips 公司推出的一种用于 IC 器件之间连接的二线制串行扩展总线[7]。与并行扩展总线相比，串行扩展总线由电路结构简单、程序编写方便、易于实现用户系统软硬件的模块化和标准化等优点[7]。目前，I²C 总线已被众多厂家应用在高档电视机、电话机、音响、摄像机等系统中。

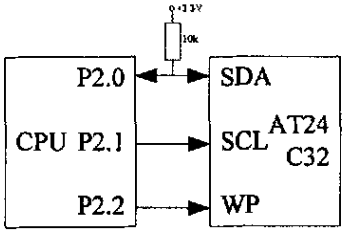


图 2.21 AT24C32 与单片机连接图
Fig.2.21 AT24C32 connect with CPU

如图 2.22 所示，I²C 总线是通过 SDA（串行数据线）和 SCL（串行时钟线）两根线在连接到总线上的器件之间传送数据，并根据地址识别每个器件。连接在总线上的所有设备工作在主机或从机模式，而且同一时刻有且仅有一台主机。I²C 总线根据器件的功能通过软件程序使其工作与发送或接收方式。当某个器件向总线发送信息时，它就是发送器（也叫主器件）；而当其从总线上接收信息时，其又成为接收器（也叫从器件）[7]。SDA 和 SCL 均为双向 I/O 线，采用漏极开路输出形式，需要在芯片外部用合适的电阻上拉到正电源。总线空闲时，两根线均为高电平。I²C 总线最大传输速率有 3 种，分别为：低速的 100kbps、中速的 400kbps 和高速的 1Mbps。

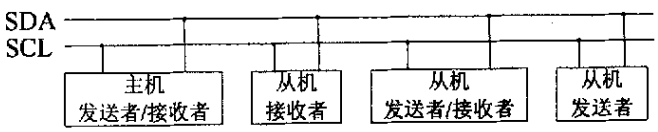


图 2.22 I²C 总线结构图
Fig.2.22 I²C bus structure

在数据传送过程中，必须确认数据传送的开始和结束。在 I²C 总线技术中，开始和结束信号（也称为启动和停止信号）的定义如图 2.23 所示。当时钟线 SCL 为高电平时，数据线 SDA 由高电平跳变为低电平定义为“开始”信号；当 SCL 线为高电平时，

SDA 线发生低电平到高电平的跳变为“结束”信号。开始和结束信号都是由主器件产生。

在 I²C 总线传输的数据的格式是以主设备发送启动信号开始，跟着发送第一字节，此字节的高 7 位为从设备地址，最低位指明数据传送方向的读/写位，该位为 0 表示主设备向从设备发送数据，为 1 表示从设备向主设备发送数据。如果地址的高 4 位不全是 0 或 1，则接着就可以发送所需的数据字节。但在每个传送的字节后面必须插入一个应答位（ACK）；如果 ACK 为 0 表示接收成功，为 1 表示接收失败。

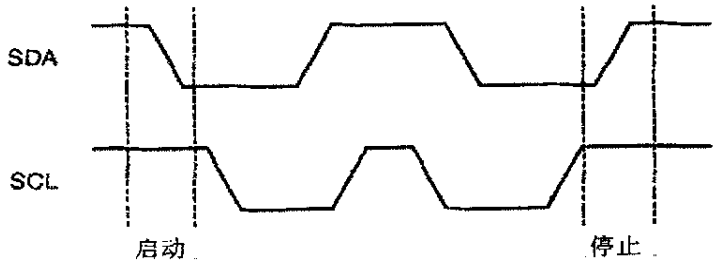


图 2.23 I²C 总线启动和停止条件

Fig.2.23 Start and stop conduction of I²C bus

AT24C32 的写操作严格按照 I²C 总线的规定。其写操作可以分为单字节写（向 AT24C32 写入一个字节）和页写（向 AT24C32 写入一个页），写过程的 SDA 线上的位流如图 2.24 和 2.25 所示。无论是单字节写还是页写都需要在器件地址字节和应答位后向 AT24C32 写入 2 个字节存储单元地址。在页写过程中第一个数据被写在写入地址的存储单元中，后续的字节被依次写入后续的存储单元；但如果地址（内部地址计数器产生）到达本页的末尾，则下一个字节将被写入本页的页首。

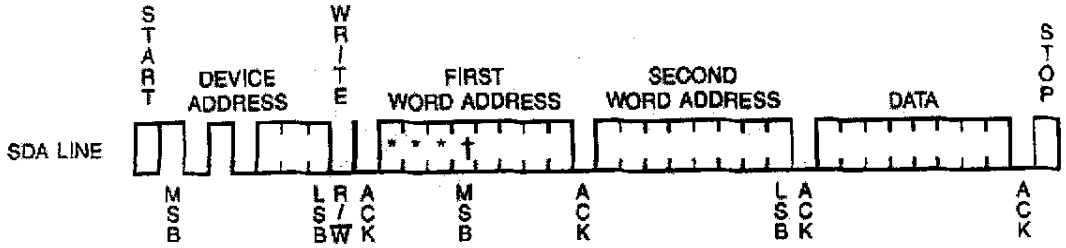


图 2.24 AT24C32 单字节写入

Fig.2.24 Byte write of AT24C32

AT24C32 的读过程可以分为当前地址读取、任意读取和连续读取三种。当前地址读取严格按照 I²C 总线的规定，任意读取与 I²C 总线的规定有些差异。在对 AT24C32 进行

任意读取之前需要先启动一个写操作，写操作的地址为要读取的存储单元的地址，如图2.26所示。

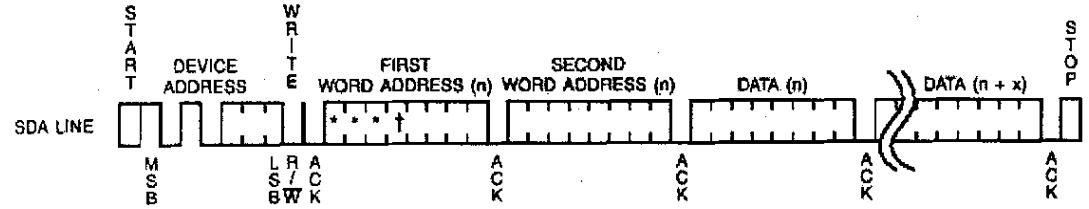


图 2.25 AT24C32 页写

Fig.2.25 Page write of AT24C32

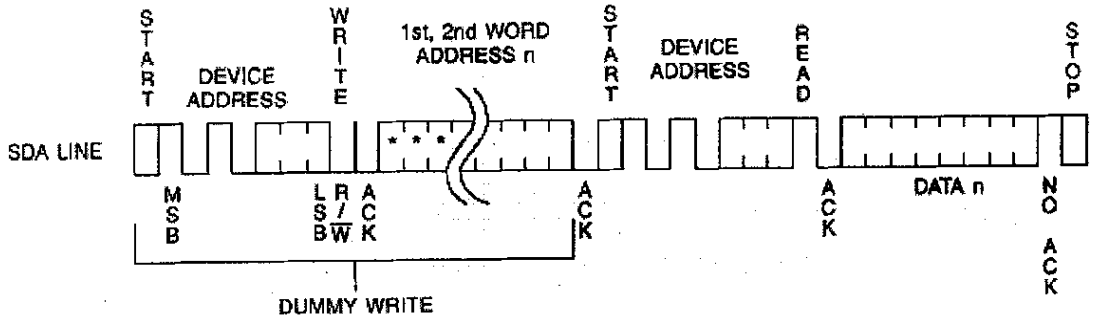


图 2.26 AT24C32 任意读取

Fig.2.26 Random Read of AT24C32

2.5 GBK 中文字库

2.5.1 GBK 汉字编码

为了规范和方便信息交换，1981年中华人民共和国国家标准总局发布了中华人民共和国国家标准汉字信息交换用编码，全称《信息交换用汉字编码字符集　基本集》，标准号为GB2312-80。GB2312-80收录简化汉字及一般符号、序号、数字、拉丁字母、日文假名、希腊字母、俄文字母、汉语拼音符号、汉语注音字母，共 7445个图形字符。其中汉字以外的图形字符 682个，汉字 6763个。在该字符集中按汉字使用的频度又将其分为一级汉字3755个（按拼音排序），二级汉字3008个（按部首排序）。

GB2312-80收入了大部分常用字和次常用字，能基本满足一般的使用需要。但在投票表决系统中需要显示大量的人名和地名，而很多人名和地名中往往有一些不常用字，甚至是一些罕见字，GB2312-80已不能满足要求，需要用更大的字库作为显示用字库。

GBK（Chinese Internal Code Specification）是又一个汉字编码标准，全称《汉字内码扩展规范》(GBK)。GBK向下与GB2312-80编码兼容，亦采用双字节表示，总体编

码范围为8140-FEFE，首字节在81-FE之间，尾字节在40-FE之间，剔除xx7F一条线。总计23940个码位，共收入21886个汉字和图形符号，其中汉字（包括部首和构件）21003个，图形符号 883个。

全部编码分为三大部分：

1. 汉字区。包括：

a. GB2312 汉字区。即 GBK/2：B0A1-F7FE。收录 GB2312 汉字 6763 个，按原顺序排列。

b. GB13000.1 扩充汉字区。包括：

(1) GBK/3：8140-A0FE。收录 GB13000.1 中的 CJK 汉字 6080 个。

(2) GBK/4：AA40-FEA0。收录 CJK 汉字和增补的汉字 8160 个。CJK 汉字在前，按 UCS 代码大小排列；增补的汉字（包括部首和构件）在后，按《康熙字典》的页码／字位排列。

2. 图形符号区。包括：

a. GB2312 非汉字符号区。即 GBK/1：A1A1-A9FE。其中除 GB2312 的符号外，还有 10 个小写罗马数字和 GB12345 增补的符号。计符号 717 个。

b. GB13000.1 扩充非汉字区。即 GBK/5：A840-A9A0。BIG-5 非汉字符号、结构符和“〇”排列在此区。计符号 166 个。

3. 用户自定义区：分为(1)(2)(3)三个小区。

(1) AAA1-AFFE，564个码位。

(2) F8A1-FEFE，658个码位。

(3) A140-A7A0，672个码位。

2.5.2 GBK 汉字库制作

GBK 共有 23940 个码位，采用 16×16 点阵的字体（即 32 个字节表示一个汉字或图形符号），则 GBK 至少需要有 23940×32 字节（748.125KB）的空间。选用意法半导体公司生产的 1MB 的 FLASH 存储器 M29W008AT[12]来存储字库。每个汉字按照区位码依次存放在 M29W008AT 中，汉字的区位码与存放位置的关系如下：

$$Add = [(Q-129)\times 192 + (W-64)]\times 32 \qquad (2.2)$$

其中：Add ——汉字在汉字库中的起始地址；

Q ——汉字的区号；

W ——汉字的位号。

2.6 RS232 电平转换电路

主机与计算机的通信采用串口通信，由于计算机的串口是 RS232 电平，而 MSP430 单片机的信号是 3.3vTTL 电平，因此需要转换电路对电平进行转换。用 MAXIM 公司的 RS232 转 TTL 专用集成电路 MAX3232[13]对电平进行转换，电路如图 2.27。

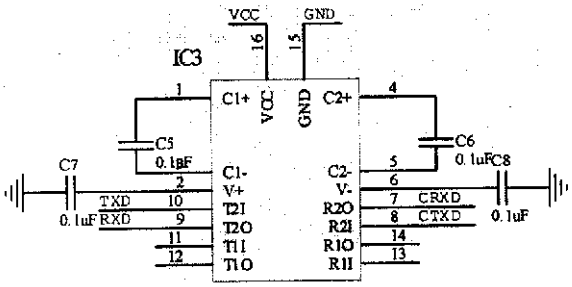


图 2.27 RS232-TTL 电平转换电路

Fig.2.27 Circuit for voltage transit between RS232 and TTL

2.7 电源电路

在电子表决器中要求尽可能少的使用电池，以减小表决器的体积、减轻表决器的重量，降低系统使用成本。因此，采用两节电池供电方案，采用升压、稳压电路得到系统需要的 3.3V 工作电压。由于 RF 模块对电源噪声非常敏感，因此 RF 电源需要有良好的电源滤波，并且与数字电路供电分离。图 2.28 是本设计中的供电电路。

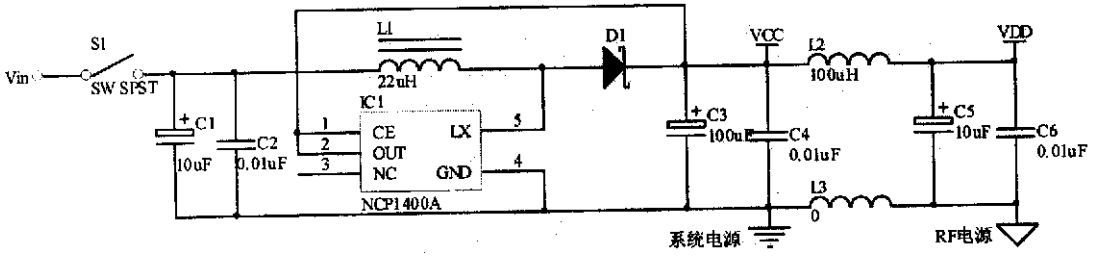


图 2.28 电源供电电路

Fig.2.28 Power supply circuit diagram

3 实时操作系统设计

电子表决器是一个较复杂的电子系统，需要完成较多的任务：与主机间的通信、显示输出、接受外部按键输入等。这些任务对实时性的要求较高，如果能采用实时操作系统（RTOS，Real Time Operating System）来调度这些任务、分配时间，则各任务的实时性将得到有效的保证，同时也会大大方便软件的开发。

3.1 实时操作系统简介

从 1981 年 Ready System 发展了世界上第一个商业嵌入式实时内核（VRTX32），至今已经有近 20 年的历史。那时候的 RTOS 还只有内核，以销售二进制代码为主，产品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，如 Ready System 的 VRTXsa、Windriver 的 Vxwork。这个时期，RTOS 在嵌入式系统设计中的主导地位已经确定，越来越多的工程师使用 RTOS。由于新的处理器越来越多，RTOS 自身结构的设计变得更易于移植，以便在短时间内支持更多种微处理器。开放源码之风已波及 RTOS 领域，许多的 RTOS 厂家在出售 RTOS 时附加了源程序代码。

目前市场上流行的，使用最多的RTOS 产品包括有：Windows CE、Lynx、 Vx-Works、Nucleus、Palm OS、pSOS等。除了商业化的操作系统外还有一些非商业化的嵌入式操作系统，如嵌入式Linux、μC/OS等。

3.2 μC/OS-II 操作系统简介

μC/OS-II 是由 Jean J.Labrosse 开发的源码公开的占先式实时嵌入式内核，其性能完全可以与商业产品相媲美。μC/OS-II 是μC/OS V1.11 的升级版本，是在μC/OS V1.11 的基础上作了很多改进而来的[14]。

μC/OS-II 具有以下特点[15]：

• 源代码公开 μC/OS-II 的作者 Jean J.Labrosse 先生在《μC/OS-II: the Real Time Kernel》一书中提供了μC/OS-II 的全部源代码，并且花了很大力气使μC/OS-II 源代码干净漂亮和谐一致，注解详尽组织有序。还开通了μC/OS 的正式网站：www. μC/OS-II.com，方便使用者互相讨论和学习。

• 可移植性强（Portable） μC/OS-II 的源代码是采用可移植性很强的 ANSI C 写的。和微处理器相关的那部分是用汇编语言写的，并且已经别压到了最低限度，使得μC/OS-II 便于移植到任何微处理器上。

• 可固化性（ROMable） μC/OS-II 是为嵌入式系统设计的，它可以嵌入到产品中成为产品的一部分。

• 可裁减性（Scalable） 对于产品中需要的μC/OS-II 中的部分功能，可以通过在用户的应用程序中用“#define constants”语句来定义，在编译的时候，编译器只会编译需要的功能而舍弃不需要的功能。这样可以减少产品中的 RAM 和 ROM 的用量。

• 占先式（Preemptive） μC/OS-II 采用占先式的实时内核，在运行条件就绪条件下，总是运行优先级最高的任务。

• 多任务 μC/OS-II 可以管理 64 个任务，目前这个版本保留 8 个给系统。应用程序最多可以有 56 个任务，而每个任务的优先级必须不同。

• 可确定性 全部μC/OS-II 的函数调用与服务的执行时间具有可确定性，μC/OS-II 系统服务的执行时间不依赖于应用程序的多少。

• 每个任务有自己单独的栈 μC/OS-II 允许每个任务有不同的栈空间，以便压低应用程序对 RAM 的需求。

• μC/OS-II 提供很多系统服务，例如邮箱、消息队列、信号量、块大小固定的内存的申请与释放、时间相关函数等。

• 中断可以使正在执行的任务暂时挂起，中断嵌套层数可达 255 层。

• μC/OS-II 具有高稳定性与可靠性 μC/OS-II 是基于μC/OS 的，μC/OS 自 1992 年以来已经有好几百个商业应用。μC/OS-II 只是在μC/OS 的基础上增加了更多的功能。

3.3 μC/OS-II 的内核结构

μC/OS-II 的内核是由一系列的变量、数据结构和对这些变量、数据结构进行访问、改写的函数组成。这些变量和数据结构主要包括：记录 CPU 工作状态的全局变量、任务就绪表、任务控制块地址表、任务控制块和任务栈等。函数主要有：系统初始化函数（OSInit()）、开始多任务调度（OSStart()）、任务建立函数（OSTaskCreate()、OSTaskCreateExt()）、任务删除函数（OSTaskDel()）、任务调度函数（OSSched()、OSIntExt()）、任务调度器上锁（OSSchedlock()）、任务调度器开锁（OSSchedUnlock()）和时钟节拍函数（OSTimeTick()）等。

3.3.1 内核中的变量和数据结构

1. 内核中的全局变量　内核中的全局变量主要用来记录 CPU 的工作状态，包括：

- 当前任务的优先级：OSPrioCur
- 准备就绪的高优先级任务的优先级：OSPrioHighRdy
- 当前任务任务控制块地址：OSTCBCur
- 准备就绪的高优先级任务块地址：OSTCBHighRdy
- 系统时间：OSTime
- 中断嵌套次数计数器：OSIntNesting
- 调度器上锁次数计数器：OSLockNesting
- 任务切换次数计数器：OSCtxCtr
- 任务数：OSTaskCtr
- 系统运行标志：OSRunning
- CPU 占用率：OSCPUUsage
- CPU 空闲时的计数器：OSIdleCtrMax、OSIdleCtrRun、OSIdleCtr
- 统计任务就绪标志：OSStatRdy

2. 任务就绪表　任务就绪表用来记录准备就绪的任务标志。任务就绪表中有两个变量 OSRdyGrp 和 OSRdyTbl[]。在 OSRdyGrp 中，任务按优先级分组，每 8 个任务为一组。OSRdyGrp 中的每一位表示 8 组任务中的一组是否有进入就绪态的任务。任务进入就绪态时，就绪表 OSRdyTbl[]中的相应元素的相应位置位[14]。

根据任务就绪表中的数据为下标查表 OSUnMapTbl[256]，可得到就绪任务中优先级最高的任务，其算法如下：

```
y= OSUnMapTbl[OSRdyGrp];
Prio = (y<<3)+ OSUnMapTbl[OSRdyTbl[y]];                    (3.1)
```

其中 Prio 就是所要求的任务的优先级。

将准备就绪的任务添加到任务就绪表的算法如下：

```
y= Prio>>3;
OSRdyGrp |= OSMapTbl[y];
OSRdyTbl[y] |= OSMapTbl[Prio & 0x07];                      (3.2)
```

其中 Prio 就是所要求的任务的优先级。

3. 任务控制块地址表（OSTCBPrioTbl） 任务控制块地址表是一个一维指针数组，其长度为任务最低优先级加一。以优先级为下标的数组元素存储的是该优先级所对应的任务的任务控制块地址。

4. 任务控制块 任务控制块是一个数据结构，当任务的 CPU 使用权被剥夺时，μC/OS-II用它来保存该任务的状态。当任务重新得到 CPU 使用权时，任务控制块能确保任务从当时被中断的那一点丝毫不差地继续执行。OS_TCBs 全部驻留在 RAM 中[14]。任务控制块包括以下内容：

- 指向当前任务栈顶的指针：OSTCBStkPtr
- 指向用户定义的任务控制块扩展指针：OSTCBExtPtr
- 指向任务栈底的指针：OSTCBStkBottom
- 栈中可容纳的指针数目：OSTCBStkSize
- 传给 OSTaskCreateExt()的“选择项”：OSTCBOpt
- 任务的识别码：OSTCBId
- 任务控制块双重链接指针：OS_TCBsOSTCBNext 和 OSTCBPrev
- 指向事件控制块的指针：OSTCBEventPtr
- 指向传给任务的消息的指针：OSTCBMsg
- 任务延时节拍计数器：OSTCBDly
- 任务的状态字：OSTCBStat
- 任务优先级：OSTCBPrio
- 加速任务进入就绪态的过程或进入等待事件发生状态的过程：OSTCBX、OSTCBY、OSTCBBitX 和 OSTCBBitY
- 该任务是否需要删除：OSTCBDelReq

众多任务的任务控制块是一个双向链表，用指针：OSTCBList 指向链表的首节点。双向链表结构有利于快速插入和删除一个任务控制块。最近建立的任务的任务控制块总是被放在链首，OSTCBList 始终指向最近建立的任务的任务控制块。

4. 任务栈 任务栈用来保存处理器寄存器的值。μC/OS-II 允许每个任务有不同的栈空间。

3.3.2 内核中的函数

1. μC/OS-II 系统初始化和多任务调度初始化是通过调用 OSInit()函数来完成的。在 OSInit()函数中对一些全局变量进行了初始化。调用 OSStart()函数就开始多任务调度，但在调用该函数前必须先建立任务，至少存在一个任务才允许调用该函数。

2．任务的建立和删除　建立一个新任务可以通过调用两个函数来完成：OSTaskCreate()或 OSTaskCreateExt()。OSTaskCreateExt()函数是 OSTaskCreate()函数的扩展函数，它扩展了一些附加功能。任务的删除可以通过调用 OSTaskDel()完成。

3．多任务调度　函数 OSSched()和 OSIntExt()都能实现任务的调度，区别在于 OSSched()用在任务级调度，中断级的调度则由 OSIntExt()来完成。如果用函数 OSSchedlock()给调度器上锁，则无法对任务进行调度，即使是比当前任务更高优先级的任务已经处于就绪状态。但中断服务程序仍然可以在中断到来时被执行。OSSchedUnlock()用来对调度器开锁。OSSchedlock()和 OSSchedUnlock()必须成对出现。

4．时钟节拍函数 OSTimeTick()将任务控制表中的非零的时间延时项 OSTCBDly 作减一操作，当某个任务的 OSTCBDly 为 0 时，如果此任务未被挂起则此任务进入就绪态。

3.3.3 临界区代码处理

和其它内核一样，μC/OS-II 为了处理临界区（critical section）代码需要关中断，处理完毕后再开中断，以避免同时有其它任务或中断服务进入临界段代码。μC/OS-II 定义两个宏来关中断和开中断，这两个宏调用分别是：OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()。

3.4 μC/OS-II 的任务管理

μC/OS-II 对任务的管理包括：在应用程序中建立新任务、删除任务、改变任务的优先级、挂起和恢复任务，以及获得有关任务的信息。

3.4.1 建立任务

想让μC/OS-II 管理用户的任务，用户必须要先建立任务。用户可以通过传递任务地址和其它参数到函数：OSTaskCreate() 或 OSTaskCreateExt()来建立任务。OSTaskCreateExt()是 OSTaskCreate()的扩展版本，提供了一些附加的功能。任务可以在多任务调度开始前建立，也可以在其它任务的执行过程中被建立，但不能由中断服务程序(ISR)来建立。在开始多任务调度(即调用 OSStart())前，用户必须建立至少一个任务。

OSTaskCreate()需要四个参数：任务代码的指针 task，当任务开始执行时传递给任务的参数指针 pdata，堆栈栈顶指针 ptos，任务的优先级 prio。OSTaskCreate()建立任务的过程如下：

• 检测分配给任务的优先级是否有效，任务的优先级必须在最高优先级 0 到最低优先级 OS_LOWSEST_PRIO 之间；

• 确保在规定的优先级上还没有建立任务，μC/OS-II 中每个任务都有特定的优先级。如果某个优先级是未使用的，μC/OS-II 通过放置一个非空指针在 OSTCBPrioTbl[] 中来保留该优先级；

• 调用 OSTaskStkInit()建立任务的堆栈；

• 调用 OSTCBInit()从空闲的 OS_TCB 池中获得并初始化一个 OS_TCB；

• 将 OS_TCB 插入到已建立任务的 OS_TCB 的双向链表中，任务处于就绪状态；

• 调用用户自己定义的函数 OSTaskCreateHook()；

• 如果系统正在运行(即 OSRunning 置为 True)，则调用任务调度函数会来判断是否新建立的任务比原来的任务有更高的优先级。如果新任务的优先级更高，内核会进行一次从旧任务到新任务的任务切换。

用 OSTaskCreateExt()函数来建立任务会更加灵活，但会增加一些额外的开销。OSTaskCreateExt()需要九个参数，前四个参数(task,pdata,ptos 和 prio)与 OSTaskCreate()的四个参数完全相同，后面五个分别为：要建立任务的一个特殊的标识符 id、指向任务的堆栈栈底的指针 pbos、堆栈成员数目的容量 stk_size、指向用户附加的数据域的指针 pext、用于设定 OSTaskCreateExt()的选项 opt。用 OSTaskCreateExt()函数建立任务的过程与用 OSTaskCreate()建立任务的过程基本相同。

3.4.2 删除任务

删除任务是指将任务返回并处于休眠状态，代码不再被μC/OS-II 调用，并不是说任务的代码被删除了。删除任务意味着：

• 它的任务控制块从 OSTCBList 链表中移到 OSTCBFreeList，这样时钟节拍函数中就不会处理它了，这样把它置入就绪表的可能性也就没有了；

• 如果它已经处于就绪表中，那么它将被移出，这样调度器函数就不会处理它，这样它被调度运行的机会就没有了；

• 如果任务处于邮箱、消息队列、信号量的等待表中，也要把它移出等待表。这些都完成了，任务就不会有机会占用 CPU 了。

通过调用 OSTaskDel()就可以完成删除任务的功能，OSTaskDel()删除任务的过程如下：

• 确保用户所要删除的任务并非是空闲任务；

• 确保用户不是在 ISR 例程中去试图删除一个任务；

• 保证被删除的任务是确实存在的；

• 如果任务处于就绪表中，它会直接被移除，如果任务处于邮箱、消息队列或信号量的等待表中，它就从自己所处的表中被移除；

• 调用用户自定义的 OSTaskDelHook()函数，删除或释放 TCB 附加数据域；

• 减少μC/OS-II 的任务计数器。简单地将指向被删除的任务的 OS_TCB 的指针指向 NULL，从而达到将 OS_TCB 从优先级表中移除；

• 将被删除的任务的 OS_TCB 从 OS_TCB 双向链表中移除，并返回到空闲 OS_TCB 表中；

• 调用任务调度程序来使准备就绪的高优先级任务运行。

3.4.3 改变任务优先级

在用户建立任务的时候会分配给任务一个优先级。在程序运行期间，用户可以通过调用 OSTaskChangePrio()来改变任务的优先级。但空闲任务的优先级不能被改变。OSTaskChangePrio()函数有两个参数：被更改任务的优先级 oldprio 和更改后的优先级 newprio。改变任务优先级的过程如下：

• 检验新优先级是否合法，如果新优先级是合法的，在 OSTCBPrioTbl[newprio]中存储非空地址来保留这个优先级；

• 检验目前的任务是否想改变它的优先级；

• 检查想要改变优先级的任务是否存在；

• 插入空指针将指向当前任务 OS_TCB 的指针从优先级表中移除；

• 检验想要改变优先级的任务是否就绪，如果该任务处于就绪状态，它必须在当前的优先级下从就绪表中移除，然后在新的优先级下插入到就绪表中；

• 将任务从事件控制块的等待队列(在旧的优先级下)中移除。并在新的优先级下将事件插入到等待队列中；

• 将指向任务 OS_TCB 的指针存到 OSTCBPrioTbl[]中。新的优先级被保存在 OS_TCB 中，重新计算的值也被保存在 OS_TCB 中。

3.4.4 挂起和恢复任务

挂起任务可通过调用 OSTaskSuspend()函数来完成。被挂起的任务只能通过调用 OSTaskResume()函数来恢复。但空闲任务既不能被挂起，也不能被恢复。OSTaskSuspend()和 OSTaskResume()的参数均为任务优先级 prio。任务挂起的过程如下：

- 确保用户的应用程序不是在挂起空闲任务；
- 确认用户指定优先级是有效的；
- 检验用户是否通过指定 OS_PRIO_SELF 来挂起调用本函数的任务本身；
- 检验要挂起的任务是否存在；
- 如果该任务存在的话，它就会从就绪表中被移除；
- 在任务的 OS_TCB 中设置 OS_STAT_SUSPEND 标志，以表明任务正在被挂起；
- 如果被挂起的任务是调用本函数的任务，调用任务调度程序。

OSTaskResume()是通过清除 OSTCBStat 域中的 OS_STAT_SUSPEND 位来取消挂起的。要使任务处于就绪状态，OS_TCBDly 域必须为 0，这是因为在 OSTCBStat 中没有任何标志表明任务正在等待延时期满。只有当以上两个条件都满足的时候，任务才处于就绪状态。最后，任务调度程序会检查被恢复的任务拥有的优先级是否比调用本函数的任务的优先级高。

3.4.5 获得有关任务的信息

用户的应用程序可以通过调用 OSTaskQuery()来获得自身或其它应用任务的信息。应用程序必须要为 OS_TCB 分配存储空间。这个 OS_TCB 与μC/OS-II 分配的 OS_TCB 是完全不同的数据空间，它是要查询的任务的 OS_TCB 的一个副本。通过这个副本可以了解被查任务的信息。

3.5 μC/OS-II 的时间管理

像其它内核一样，μC/OS-II 也要求用户提供定时中断来实现延时与超时控制等功能，这个定时中断叫做时钟节拍。时钟节拍的实际频率是由用户的应用程序决定的。

μC/OS-II 通过函数：OSTimeDly()、OSTimeDlyHMSM()、OSTimeDlyResume()、OSTimeGet()、OSTimeSet()来对时间进行管理。

OSTimeDly()、OSTimeDlyHMSM()用以任务的延时。OSTimeDly()是以时钟节拍为单位的，而 OSTimeDlyHMSM()则是以时(h)、分(in)、秒(s)、毫秒(ms)来定义时间。

调用 OSTimeDlyResume()可将处在延时期的任务重新处于就绪状态。但采用 OSTimeDlyHMSM()函数进入延时期的任务由于 OSTimeDlyHMSM()本身的问题而不一定能够结束任务的延时期。

系统用一个 32 位的计数器来记录系统时间（以时钟节拍为单位）。在每一个时钟节拍到来时加一。函数 OSTimeGet()可以获得这个系统时间，OSTimeSet()可以更改系统时间。

3.6 μC/OS-II 任务间的通信与同步

除了利用宏 OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()关闭中断和打开中断，利用函数 OSSchedLock()和 OSSchedUnlock()对任务调度函数上锁和开锁来保护任务间的共享数据外，μC/OS-II 还通过信号量、邮箱和消息队列来实现任务间的数据共享和通信。

3.6.1 事件控制块

一个任务或者中断服务子程序可以通过事件控制块 ECB（Event Control Blocks）来向另外的任务发信号。数据控制块是一个数据结构，除了包含了事件本身的定义，如用于信号量的计数器，用于指向邮箱的指针，以及指向消息队列的指针数组等，还定义了等待该事件的所有任务的列表。其结构如下：

```
typedef struct {
    void   *OSEventPtr;              /*指向消息或者消息队列的指针*/
    INT8U   OSEventTbl[OS_EVENT_TBL_SIZE]; /*等待任务列表*/
    INT16U  OSEventCnt;              /*计数器(当事件是信号量时)*/
    INT8U   OSEventType;             /*事件类型*/
    INT8U   OSEventGrp;              /*等待任务所在的组*/
} OS_EVENT;
```

OSEventTbl[]和 OSEventGrp 跟任务就绪表中的 OSRdyTbl[]和 OSRdyGrp 完全一致，只是前两者包含的是等待某事件的任务，而后两者包含的是系统中处于就绪状态的任务。

事件控制块的总数由用户所需要的信号量、邮箱和消息队列的总数决定。所有事件控制块被链接成一个单向链表——空闲事件控制块链表。每当建立一个信号量、邮箱或者消息队列时，就从该链表中取出一个空闲事件控制块，并对它进行初始化。由于信号量、邮箱和消息队列一旦建立就不能删除，所以事件控制块也不能放回到空闲事件控制块链表中。

与事件控制块有关的函数：初始化一个事件控制块 OSEventWaitListInit()使一个任务进入就绪态 OSEventTaskRdy()、使一个任务进入等待某事件发生状态 OSEventTaskWait()、由于等待超时而将任务置为就绪态 OSEventTO()。

3.6.2 信号量

μC/OS-II 中的信号量由两部分组成：一个是信号量的计数值，它是一个 16 位的无符号整数；另一个是由等待该信号量的任务组成的等待任务表。在使用一个信号量之前，首先要建立该信号量，也即调用 OSSemCreate()函数。信号量的初始计数值为 0 到 65,535 之间的一个数。如果信号量是用来表示一个或者多个事件的发生，那么该信号量的初始值应设为 0。如果信号量是用于对共享资源的访问，那么该信号量的初始值应设为 1。如果该信号量是用来表示允许任务访问 n 个相同的资源，那么该初始值为 n，并把该信号量作为一个可计数的信号量使用。

μC/OS-II 提供了 5 个对信号量进行操作的函数。它们是：OSSemCreate()，OSSemPend()，OSSemPost()，OSSemAccept()和 OSSemQuery()函数。OSSemPost()函数可以由任务或者中断服务子程序调用，而 OSSemPend()和 OSSemQuery()函数只能由任务程序调用。

3.6.3 邮箱

邮箱是μC/OS-II 中另一种通信机制，它可以使一个任务或者中断服务子程序向另一个任务发送一个指针型的变量。该指针指向一个包含特定“消息”的数据结构。使用邮箱之前也必须先建立该邮箱，可以通过调用 OSMboxCreate()函数来完成。建立邮箱是要指定消息指针的初始值。一般情况下，这个初始值是 NULL，但也可以初始化一个邮箱，使其在最开始就包含一条消息。如果使用邮箱的目的是用来通知任务某一个事件已经发生，那么就要初始化该邮箱为 NULL。如果用户用邮箱来共享某些资源，那么就要初始化该邮箱为一个非 NULL 的指针。在这种情况下，邮箱被当成一个二值信号量使用。

μC/OS-II 提供了 5 种对邮箱的基本操作：OSMboxCreate()，OSMboxPend()，OSMboxPost()，OSMboxAccept()和 OSMboxQuery()函数。任务或者中断服务子程序可以调用函数 OSMboxPost()，但只有任务可以调用函数 OSMboxPend()和 OSMboxQuery()。

3.6.4 消息队列

消息队列是μC/OS-II 中又一种通信机制，它可以使一个任务或者中断服务子程序向另一个任务发送以指针方式定义的变量。因具体的应用有所不同，每个指针指向的数据结构变量也有所不同。消息队列像多个邮箱。在使用一个消息队列之前，必须先建立该消息队列。这可以通过调用 OSQCreate()函数，并定义消息队列中的单元数（消息数）来完成。

μC/OS-II 提供了 7 个对消息队列进行操作的函数：OSQCreate()，OSQPend()，OSQPost()，OSQPostFront()，OSQAccept()，OSQFlush()和 OSQQuery()函数。任务或者中断服务子程序可以调用 OSQPost()，OSQPostFront()，OSQFlush()或者 OSQAccept()函数。但是，只有任务可以调用 OSQPend()和 OSQQuery()函数。

当建立了一个消息队列时，一个队列控制块也同时被建立，并通过事件控制块中的 OSEventPtr 域链接到对应的队列控制块。在建立一个消息队列之前，必须先定义一个含有与消息队列最大消息数相同个数的指针数组，数组的起始地址以及数组中的元素数作为参数传递给 OSQCreate()函数。

队列控制块是一个用于维护消息队列信息的数据结构，它包含了以下的一些域：

OSQPtr 在空闲队列控制块中链接所有的队列控制块的指针。一旦建立了消息队列，该域就不再有用了。

OSQStart 是指向消息队列的指针数组的起始地址的指针。用户应用程序在使用消息队列之前必须先定义该数组。

OSQEnd 是指向消息队列结束单元的下一个地址的指针。该指针使得消息队列构成一个循环的缓冲区。

OSQIn 是指向消息队列中插入下一条消息位置的指针。当 OSQIn 和 OSQEnd 相等时，OSQIn 被调整指向消息队列的起始单元。

OSQOut 是指向消息队列中下一个取出消息位置的指针。当 OSQOut 和 OSQEnd 相等时，OSQOut 被调整指向消息队列的起始单元。

OSQSize 是消息队列中总的单元数。该值是在建立消息队列时由用户应用程序决定的。在μC/OS-II 中，该值最大可以是 65535。

OSQEntries 是消息队列中当前的消息数量。当消息队列是空的时，该值为 0。当消息队列满了以后，该值和 OSQSize 值一样。在消息队列刚刚建立时，该值为 0。

3.7 μC/OS-II 的内存管理

在μC/OS-II 中，操作系统把连续的大块内存按分区来管理。每个分区中包含有整数个大小相同的内存块。利用这种机制，μC/OS-II 可以分配和释放固定大小的内存块。

为了便于内存的管理，在μC/OS-II 采用内存控制块（Memory Control Block）来管理每一个内存分区，系统中的每个内存分区都有它自己的内存控制块。一个内存控制块是一个数据结构，其结构如下：

```
typedef struct {
```

```
    void    *OSMemAddr;      /*指向内存分区起始地址的指针*/
    void    *OSMemFreeList; /* 指向下一个空闲内存控制块或者下一个空闲的内
                               存块的指针*/
    INT32U  OSMemBlkSize;    /*内存分区中内存块的大小*/
    INT32U  OSMemNBlks;      /*内存分区中总的内存块数量*/
    INT32U  OSMemNFree;      /*内存分区中当前可以使用的空闲内存块数量*/
} OS_MEM;
```

在使用一个内存分区之前，必须先建立该内存分区。这个操作可以通过调用OSMemCreate()函数来完成。OSMemCreate()函数共有 4 个参数：内存分区的起始地址、分区内的内存块总块数、每个内存块的字节数和一个指向错误信息代码的指针。如果OSMemCreate()操作失败，它将返回一个 NULL 指针。否则，它将返回一个指向内存控制块的指针。

通过调用 OSMemGet()可分配一个内存块。如果指定的内存分区中已经没有空闲的内存块，则返回一个空指针；反之，则返回内存块的指针。OSMemPut()函数可以释放内存块。传递给 OSMemPut()的内存分区必须是正确的，因为这个函数并不检验内存块是否属于指定的内存分区，如果被释放的内存块不属于指定的内存分区，就可能产生严重的错误。

应用程序也可以通过调用 OSMemQuery()函数来查询特定内存分区的使用状况。

3.8 μC/OS-II 在无线下载电子投票表决系统中的移植

μC/OS-II 是一个优秀的可移植的操作系统。它可以轻松的移植到满足以下 5 个条件的任何一个微处理器或微控制器上。

- 处理器的 C 编译器能产生可重入代码；
- 用 C 语言就可以打开和关闭中断；
- 处理器支持中断，并且能产生定时中断(通常在 10 至 100Hz 之间)；
- 处理器支持能够容纳一定量数据(可能是几千字节)的硬件堆栈；
- 处理器有将堆栈指针和其它 CPU 寄存器读出和存储到堆栈或内存中的指令。

3.8.1 μC/OS-II 在 MSP430F149 上的移植

1. μC/OS-II 系统结构

μC/OS-II 的可移植性不但要归功于它是用可移植性强的 C 语言编写，合理的系统结构也是功不可没。下图是μC/OS-II 的硬件和软件体系结构图。

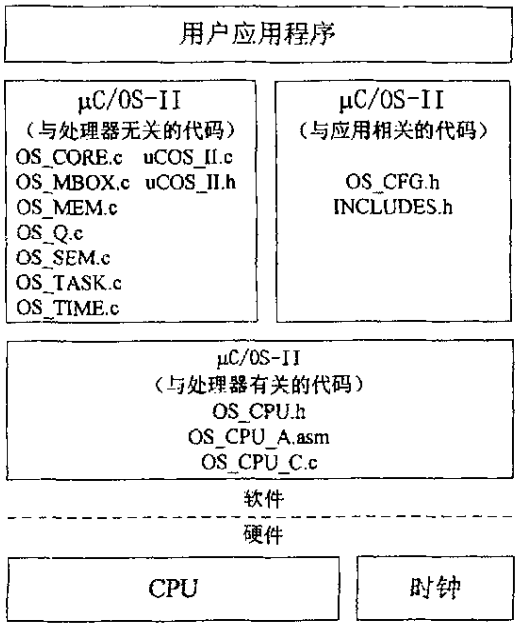


图 3.1 μC/OS-II 硬件和软件体系结构

Fig.3.1 The hardware and software system structure of μC/OS-II

从上图可以看出只需要修改与处理器相关的代码：OS_CPU.h、OS_CPU_A.asm、OS_CPU_C.c，μC/OS-II 就可以移植到相应的微处理器或微控制器中；修改与应用相关的代码：OS_CFG.h、INCLUDES.h，μC/OS-II 就可以满足相应的应用需要。

与处理器相关的代码包括：与处理器相关的常量、宏和数据类型（位于 OS_ CPU.h 文件中），对 CPU 寄存器进行操作的底层汇编语言程序（位于 OS_CPU_A.asm 文件中）以及用户在任务建立、删除等特定时刻进行的操作（位于 OS_CPU_C.c 文件中）。

在 OS_CPU.h 文件中，定义了堆栈的宽度和堆栈的增长方向，对于 MSP430 来讲堆栈的宽度是 16 位的，堆栈是由高字节向低字节增长的。关中断宏 OS_ENTER _CRITICAL() 和开中断宏 OS_EXIT_CRITICAL() 也在这里定义，按照从子程序返回后的中断使能状态，可以将这两个宏的实现方法分为两种：退出子程序后中断使能状态可能改变和退出子程序后中断使能状态不变。前一种的实现方法简单，在 MSP430 中分别被定义为：_DINT() 和_EINT()[18]。后一种方法的实现比较麻烦，需要在关中断的情况下先保存当前中断使能情况，在开中断时将中断恢复到原来的状态。文件中还定义了 OS_TASK_SW()，这是真正进入任务切换的代码。对于某些有软中断功能的处理器，它是一条软中断指令；而像 MSP430 这类没有软中断功能的处理器，它是一条对子程序的调用指令。

在 OS_CPU_A.asm 文件中定义了四个函数：OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()、OSTickISR()。OSStartHighRdy()函数的作用是允许处于就绪态的最高优先级任务，它在 MSP430 中的汇编代码如下：

```
OSStartHighRdy
        call    #OSTaskSwHook
        mov.b   #1, &OSRunning          ; 设置运行标志
        mov.w   &OSTCBHighRdy, R13      ; 调入任务堆栈
        mov.w   @R13, SP                ;
        POPALL                          ; 所有寄存器退栈
        reti                            ; 模拟一次中断返回
```

OSCtxSw()进行的是任务级的任务切换，它在 MSP430 中的汇编代码如下：

```
OSCtxSw
        push    sr                      ; 保存状态寄存器
        PUSHALL                         ; 将所有寄存器压栈
        mov.w   &OSTCBCur, R13          ; 保存堆栈指针
        mov.w   SP, 0(R13)
        call    #OSTaskSwHook
        mov.b   &OSPrioHighRdy, R13     ; 获取任务优先级
        mov.b   R13, &OSPrioCur         ;
        mov.w   &OSTCBHighRdy, R13      ; 获取任务控制块指针
        mov.w   R13, &OSTCBCur          ;
        mov.w   @R13, SP                ; 获取任务栈堆栈指针
        POPALL                          ; 寄存器退栈
        reti                            ; 模拟一次中断返回
```

OSIntCtxSw()进行的是中断级的任务切换，它在 MSP430 中的汇编代码如下：

```
OSIntCtxSw
        call    #OSTaskSwHook
        mov.b   &OSPrioHighRdy, R13     ; 获取任务优先级
        mov.b   R13, &OSPrioCur         ;
```

```
mov.w    &OSTCBHighRdy, R13    ; 获取任务控制块指针
mov.w    R13, &OSTCBCur        ;
mov.w    @R13, SP              ; 获取任务栈堆栈指针
POPALL                         ; 寄存器退栈
reti                           ; 返回中断
```

OSTickISR()是时钟节拍中断服务程序，它在MSP430中的汇编代码如下：

```
WDT_ISR                        ;
        PUSHALL                ; 将所有寄存器压栈
        inc.b    &OSIntNesting ; OSIntNesting 加 1
        call     #OSTimeTick   ; 调用 SOTimeTick()
        call     #OSIntExit    ; 调用 OSIntExit()
        POPALL                 ; 所有寄存器退栈
        reti                   ;
```

OS_CPU_C.c 文件中包含的是一些在任务建立、切换、删除、时钟节拍发生等时刻用户要进行的特殊处理的代码，用户可以根据需要来编写。

3.8.2 μC/OS-II 在无线下载电子投票表决系统中的优化

无线电子表决系统是一个特殊的系统，其在主机和电子表决器中所需要处理的任务不是很多（在 8 个以内），任务间的通信也不是非常复杂，MSP430F149 的 RAM 容量又非常有限。因此，非常有必要对μC/OS-II 进行优化，使其更适合于在无线电子投票表决系统中的应用。

1．处理任务数的优化

μC/OS-II 可以管理多达 64 个任务，但在无线电子表决系统的主机和电子表决其中的任务数却远没有这么多。因此，可以降低μC/OS-II 所能管理的任务数。这样做的好处是可以降低系统对内存的需求量，还可以降低程序的复杂程度从而减少代码量和程序执行时间。

任务划分　主机和电子表决器都是多任务的系统，按照实现的功能和各功能间的通信，将任务划分如表 3.1、表 3.2。

在对系统任务进行划分的时候需要经过慎重考虑，任务并不是划分的越多、越细越好。任务划分得太多不但占用很多的内存、使系统任务调度的负担增加，而且对系统的

表 3.1 主机任务表

任务名称	任务优先级	功能描述
SystemCheck	0	系统检测（电源电压等）
RS232DataReceiveDeal	1	RS232 数据的接收和处理
RFDataReceiveDeal	2	RF 数据的接收和处理
RS232DataSend	3	RS232 数据发送
RFDataSend	4	RF 数据发送
Net Maintain	5	无线通信网络建立与维护
	6	保留
OSTaskIdle	7	空闲任务

表 3.2 电子表决器任务表

任务名称	任务优先级	功能描述
SystemCheck	0	系统检测（电源电压等）
DataReceiveDeal	1	数据的接收和处理
DataSend	2	发送数据
KeyReadDeal	3	读键盘并处理
	4	保留
Display	5	显示
OSTaskStat	6	CPU 占用情况统计
OSTaskIdle	7	空闲任务

中断响应速度产生不利的影响。例如，在电子表决器中任务 DataReceiveDeal，在接收到数据后马上进行处理。也可以分成两个任务（数据接收和数据处理）来处理，数据接收任务接收到数据后发送消息给数据处理任务，并将 CPU 控制权交给数据处理任务。显然，第二种方法比第一种方法多一个任务，需要多花费一块内存空间来作为任务栈；两个任务在切换过程中又浪费了一段时间；而且又多了一次任务间的通信。当然，任务划分也不是越少越好，这样会使任务的复杂度增加。划分任务应该对任务复杂度、内存占用、任务间的通信量、中断相应等因素综合考虑。

操作系统修改　从表 3.1、表 3.2 中可以看出μC/OS-II 只需要能调度 8 个工作就能够满足在主机和电子表决器中的应用。与管理任务数相关的数据结构主要是任务就绪表

和事件控制块中的等待任务表。这两张表具有相同的结构，都是由一个字节的任务组标志和若干字节（由任务的最低优先级决定）的任务表组成，减少能管理的任务数后，可只用一个字节（OSRdyTbl）来作为任务表，但同时须规定任务的最低优先级为 7，即空闲任务的优先级为 7。修改后查询任务就绪表中最高优先级的任务的算法可由(3.1)简化为：

```
Prio = OSUnMapTbl[OSRdyTbl]                    (3.3)
```

其中 Prio 就是所要求的任务的优先级。

添加任务到任务就绪表的算法可简化为：

```
OSRdyTbl |= OSMapTbl[Prio];                    (3.4)
```

其中 Prio 就是所要求的任务的优先级。

其他操作与这两个操作类似，事件控制块中的等待任务表与任务就绪表的操作类似。

任务控制块也可以得到简化。如前所述，任务控制块中的用于加速任务进入就绪态的过程或进入等待事件发生状态过程的四个域：OSTCBX、OSTCBY、OSTCBBitX 和 OSTCBBitY 可以省略，因为任务减少后的操作已经很简单了。

2. 任务栈的优化

μC/OS-II 在运行的过程中需要占用相当多的内存（一般为几 K），这对于像 MSP430 系列这样内存比较小的单片机来说是无法忍受的，幸运的是μC/OS-II 还可以通过优化任务栈来减少对内存的占用。

使用μC/OS-II 存在的问题　在 MSP430 的硬件设计中没有把中断堆栈和任务堆栈分开。这样，应用μC/OS-II 的时候，在考虑每个任务的任务堆栈大小时，不单单需要计算任务中局部变量和函数嵌套层数，还需要考虑中断的最大嵌套层数。因为，对于μC/OS-II 原始的中断处理的设计、中断处理过程中的中断嵌套中所需要压栈的寄存器大小和局部变量的内存大小，都需要算在每个任务的任务堆栈中，则对于每一个任务都需要预留这一部分内存，所以大量的 RAM 被浪费[16]，如图 3.2 所示。

如果能把中断堆栈和每个任务自己的堆栈分开，这样，在计算每个任务堆栈的时候，就不需要把中断处理中（包括中断嵌套过程中）的内存的占用计算到每个任务的任

务堆栈中，只需要计算每个任务本身需要的内存大小[16]，这样就可以提高了 RAM 的利用率，缓解内存紧张的问题，如图 3.3。

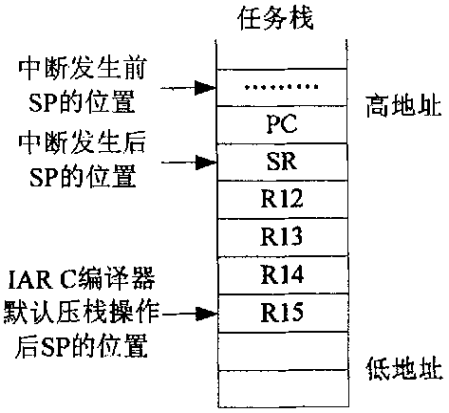


图 3.2 中断发生时的任务栈操作

Fig.3.2 Task stack operation when interrupt happen

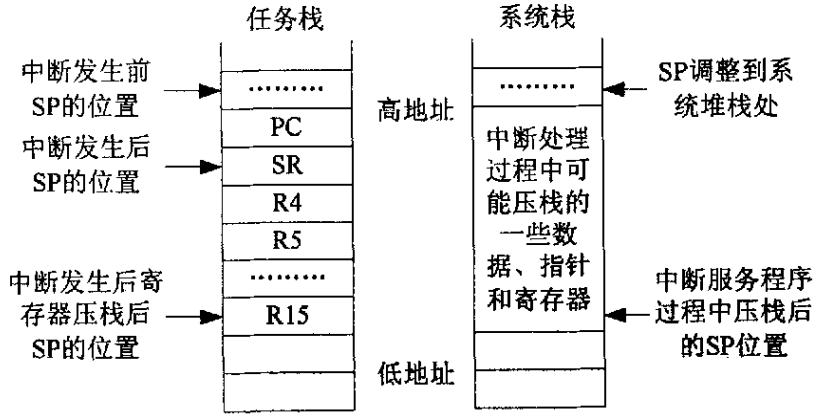


图 3.3 中断发生时的任务栈操作

Fig.3.3 Task stack operation when interrupt happen

操作系统修改　要实现上述目的需要进行如下操作：当中断发生时，中断服务程序先保存寄存器，然后根据全局变量 OSIntNesting 判断中断前执行的程序是任务还是其它中断服务程序。如果是其它中断服务程序（OSIntNesting〉=1），则不对堆栈指针作任何操作。如果是任务，则将 SP 保存到任务控制块中，然后将 SP 指向系统栈。退出中断服务程序时调用 OSIntExit()，系统会自动判断是否需要任务切换和切换任务。修改后的时钟节拍中断服务程序如下：

```
WDT_ISR                                          ; wd timer ISR
        PUSHALL                                  ; 保存所有寄存器
        bic.b   #0x01, IE1                       ; 关定时器中断
        cmp.b   #0, &OSIntNesting                ; 判断是否中断嵌套
        jne     WDT_ISR_1
        ;非任务嵌套
        mov.w   &OSTCBCur, R13                   ; 保存 SP 到当前任务栈
        mov.w   SP, 0(R13)
        mov.w   &OSISRStkPtr, SP                 ; 调整堆栈到系统栈
WDT_ISR_1
        inc.b   &OSIntNesting                    ; OSIntNesting 加 1
        bis.b   #0x01, IE1                       ; 允许时钟中断
        EINT                                     ; 允许中断
        call    #OSTimeTick                      ; 调用时钟节拍程序
        DINT                                     ; 关中断
        call    #OSIntExit                       ; 调用出中断程序
        POPALL                                   ; 恢复寄存器
        reti                                     ; 退出中断
```

另外，需要将 OSIntExit()程序中判断任务就绪表中最高优先级任务不是当前（中断前）的程序那条语句“if (OSPrioHighRdy != OSPrioCur)”去掉，因为在原来的系统中，OSPrioHighRdy = OSPrioCur 时，当前（中断中）的堆栈就是中断前执行的任务的堆栈，因此不需要切换任务。而经过修改之后的操作系统，无论 OSPrioHighRdy 是否等于 OSPrioCur，当前（中断中）的堆栈已经是系统堆栈了，而不是任务栈，所以无论如何都需要切换任务。

4 上位机软件设计

投票表决系统在投票前需要输入包括候选人名单、候选人信息、选举规则等的大量信息，选举过程中需要及时统计和显示选举结果，选举结束后又要将选举结果保存成文档以备日后查阅。

4.1 Delphi 编程工具介绍

Delphi 软件开发工具是 Borland 公司推出的一个完全导向的，可视化系统开发工具，被评为美国最优秀软件之一。Delphi 结合了可视化技术、面向对象技术、数据库技术、网络开发技术等多种先进的软件编程技术和思想，成为创建功能丰富、界面友好的 Windows 应用软件工具之一 [19]。它与 Microsoft 公司的 VC/C++相比，具有简单易懂的 IDE 开发环境；与 VB 相比，它的代码更加规范，增强了程序的可移植性，并且开发效率更高[20]。目前 Delphi 的最新版已经发展到了 8.0 版本。

Delphi 实际上是 Pascal 语言的一种版本，但它与传统的 Pascal 语言有着天壤之别[20]。Delphi 程序首先是一个应用程序框架，在这个框架下，即使不添加一行代码也可以编译成一个 Windows 应用程序，它可以接受用户对他的任何输入。Delphi 将 Windows 编程的回调、句柄处理等繁复的工程都集成在这个程序框架中，因而程序员可以轻松地对可视部件进行编程。

Borland Delphi 6 是 Borland 公司于 2001 年发布的 Delphi6.0 版本。它是 Delphi 5.0 的升级版，并且比 5.0 增加了更多的功能。与 5.0 版本一样，Delphi6.0 有三种版本：标准版（Standard）、专业版（Professional）和企业版（EnterPrise）。

- 标准版主要面向刚入门和不常使用 Delphi 的编程人员。标准版中提供了 Delphi 的基本功能配置和基本的辅助工具；
- 专业版面向专业程序员。它不仅包括标准版中提供的基本功能而且提供有扩展的数据库编程支持、部分 Internet 支持与一些外部工具；
- 企业版是功能最全的版本。它面向开发大型应用程序的程序开发人员。他在专业版的基础上扩展了对 Web 服务器、CORBA、ADO、多级数据库等的支持。同时提供了一些附加的辅助工具。

无线下载电子投票表决系统的上位机软件是在 Delphi6.0 的企业版的基础上开发的。

4.2 数据结构定义

在 C 语言中可以将类型不同的但意义相关的数据定义成结构体，以便于数据在程序中的引用。在 Pascal 中，这种数据类型被称作记录型。

记录类型的声明的一般形式如下：

```
Type
记录类型名 = Record
域名表 1 : 类型 1;
域名表 2 : 类型 2;
.....
域名表 n : 类型 n;
End;
```

4.2.1 会议记录类型定义

记录一次投票表决会议需要有两部份内容：会议的信息（包括名称、时间、地点等，在 4.2.2 中介绍）和投票表决的内容。投票表决的内容可以是从候选人中选代表（即选举形式，在 4.2.3 中介绍），也可能是对议题进行表决（即表决形式，在 4.2.4 中介绍）。因此，会议记录类型被定义成如下形式：

```
Type
    Conference = Record                                    //会议记录名
    ConferenceInf:ConferenceInformation;                   //会议基本信息
    Candidates:Array[1..96] of CandidateInformation;  //候选人信息
    Discussions:Array[1..20] of DiscussionInformation;//议题信息
End;
```

在这个记录类型中，允许有 96 个候选人，或者 20 个讨论议题。

4.2.2 会议基本信息记录类型定义

对一次投票表决会议所要记录的基本信息包括：会议名称、会议时间、地点、与会人数、负责人以及与选举、表决有关的一些信息（如：本次会议进行的是选举还是表决，选举的候选人数、被选人数，表决时是否允许弃权等）。会议基本信息记录类型定义如下：

```
Type
    ConferenceInformation = Record              //会议信息
```

```
    Name          :string[64];                  //会议名称
    Time          :array[0..4] of Byte;         //会议时间
    Address       :String[100];                 //会议地点
    Principal     :string[12];                  //负责人
    MemberNumber:integer;                       //与会人数
    Information :array[0..5] of Byte;           //会议性质、选举规则
End;
```

4.2.3 候选人信息记录类型定义

候选人信息需要记录：候选人姓名、年龄、民族、政治面貌、现任职务、获得的票数以及其它一些信息（包括简单的工作经历、政绩等）。候选人信息记录类型定义如下：

```
Type
    CandidateInformation = Record
    Name          :String[13];                  //候选人姓名
    Age           :Byte;                        //年龄
    Nationality :Enumerated;                    //民族
    PoliticalBackground:Enumerated;             //政治面貌
    Duty          :String[50];                  //现任职务
    VoteNumber    :word;                        //得票数
    Information :String[255];                   //候选人信息
End;
```

4.2.4 议题信息记录类型定义

议题信息需要记录议题内容、赞成票数、反对票数和弃权票数。议题信息类型定义如下：

```
Type
    DiscussionInformation = Record
    Content         :String[255];               //议题内容
    AgreeNumber     :word;                      //同意票数
    DisAgreeNumber:word;                        //反对票数
    GiveUpNumber    :word;                      //弃权票数
End;
```

4.2.5 其它记录类型定义

另外，串行通信的发送、接收缓冲器、数据缓冲区也需要有一定的结构，它们分别按如下定义。

发送、接收缓冲区：

```
Type
    CommBuffer = record                    //串口缓冲区
    Destination :word;                     //目的地址
    Origin      :word;                     //源地址
    Command     :byte;                     //命令
    Data        :array [0..26] of byte;    //数据
    Check       :Byte;                     //校验
    DataLength  :byte;                     //数据长度
End;
```

数据缓冲区：

```
Type
    DataBuffer = record                    //数据缓冲区
    DataLength  :word;                     //要发送的数据长度
    Offset      :word;                  //要发送的数据在缓冲区中的起始位置
    Destination :word;                     //目的地址
    Command     :byte;                     //发送数据的命令
    Data        :array [0..2047] of byte;  //数据
End;
```

4.3 Delphi 中的文件操作

Delphi 继承了 Object Pascal 的文件管理功能，并有很大的发展，其中最主要的是提供了用于文件管理的标准组件，同时也提供了更多的文件管理函数[21]。利用这些强大的功能，开发一个自己的文件管理系统就成为很容易的事。

上位机软件中对文件的基本操作通过以下函数完成：

AssignFile:　　把一个外部文件名和一个文件变量相关联

Reset:　　打开一个存在的文件

Rewrite:　　创建并打开一个新文件（或覆盖原文件）

CloseFile: 关闭一个打开的文件

Read: 从文件中读取数据，以文件变量为操作对象

Write: 向文件中写数据，以文件变量为操作对象

4.3.1 文件类型

在传统的 Pascal 中对文件的操作是通过一个变量（文件变量）来实现对磁盘上的文件（外部文件）的 I/O 操作，像其它变量一样，用于实现对文件地操作的这个变量必须有一种数据类型，这种数据类型就是文件类型[21]。传统 Pascal 中文件类型有三种：Text（或 Textfile）、file 和用户自定义类型[21]。用户自定义文件类型声明的一般形式为：

```
Type
文件类型名 = file of 基类型;
```

在无线下载电子投票表决系统的上位机软件中所操作的文件的类型采用自定义类型。由于所要存储的是整个会议的信息，因此采用会议记录类型（Conference）来作为文件的类型，定义如下：

```
Type
    ConferenceFile = file of Conference;
```

4.3.2 文件的创建和保存

要把数据保存到文件中，需要先创建一个文件。文件的创建过程如下：

```
AssignFile(文件变量,文件名);  //把一个外部文件名和一个文件变量相关联
Rewrite(文件变量);            //创建并打开一个文件
```

在创建文件时需要先申请一个文件变量，并且将文件变量与外部的文件名（即需要保存成的文件名）相关联。

文件名的输入可以通过普通的输入框来完成，也可以通过 Delphi 的控件 SaveDialog 来完成。上位机软件采用 SaveDialog 控件来接收用户的文件名输入。

SaveDialog 控件通过“Execute”方法打开 Windows 的标准保存对话框，如图 4.1 所示。通过它可以方便的选择保存文件的路径，在“文件名”中输入需要保存的文件名后点击“保存”即可将完整的文件名传给应用程序。SaveDialog 对话框打开和文件名传递的代码如下：

```
if (SaveDialog1.Execute) then          //打开 SaveDialog 对话框
    FileName:=SaveDialog1.FileName;    //获取保存文件的文件名
```

获得了文件名后，就可以通过调用 AssignFile()、和 Rewrite()函数创建文件。

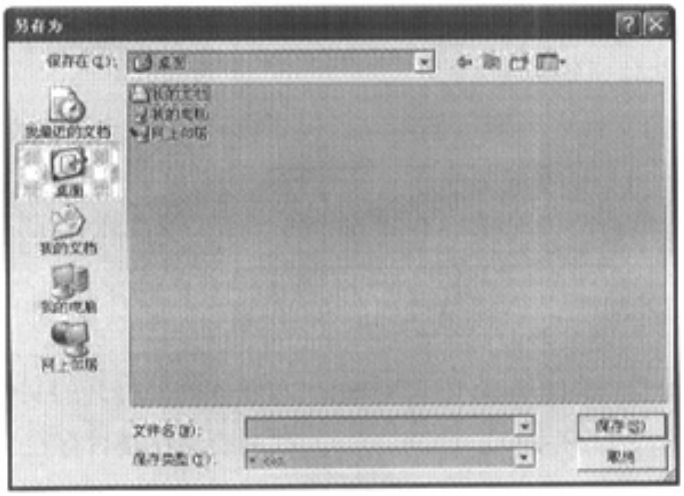


图 4.1 保存文件对话框
Fig.4.1 Savedialog box

创建完文件就可以调用 Write()函数向文件中写入数据。Write()函数的参数为需要写入的数据，由于在定义文件时已经将文件的类型定义成 Conference 型，因此只需要一条语句就可以完成保存全部会议信息。

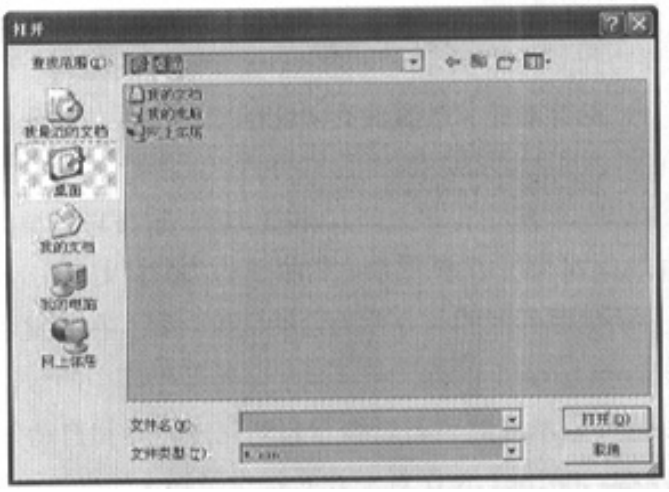


图 4.2 打开文件对话框
Fig.4.2 Open dialog box

4.3.3 文件的打开和关闭

与创建文件一样，打开文件需要获得文件的文件名。打开文件的文件名通过 Delphi 的 OpenDialog 控件来完成。OpenDialog 通过“Execute”方法打开 Windows 的

标准打开文件对话框，如图 4.2。选择需要打开的文件，点击“打开”按钮即可将需要打开的文件的文件名传递给程序。

获得需要打开的文件名后通过调用 AssignFile()函数将外部文件名和文件变量相关联，然后调用 Reset()函数打开文件。文件被打开后可调用 Read()函数来读取文件的数据。CloseFile()函数可以关闭一个被打开的文件。文件的打开、读取和关闭的代码如下：

```
if (OpenDialog1.Execute) then
    FileName:=OpenDialog1.FileName;  //获取需要打开的文件名
if (FileExists(FileName)) then       //判断文件是否存在
begin
    assignfile(ConfFile,FileName);   //将外部文件名和文件变量关联
    Reset(ConfFile);                 //打开文件
    Read(ConfFile,AConference);      //读取文件内容
    Closefile(ConfFile);             //关闭打开的文件
End;
```

4.4 数据导出 Excel 电子表格

上位机软件需要将投票结果保存成文件，以便于以后的查询。但这些文件只能通过上位机软件才能打开，这样对结果的发布有一定的限制。因此，需要将投票结果保存成常用的 Excel 格式。

OLE 自动化是 Windows 应用程序之间互相操纵的一种技巧。被操纵的一方称为自动化服务器（也称自动化对象），典型的自动化服务器有 Microsoft Word、Excel 和 Powerpoint。操纵自动化服务器的一方称为自动化控制器。在开发数据库应用程序中，经常需要借助 Microsoft Excel 的强大报表功能，把数据库中的数据输出到 Excel 表格中。Delphi 5.0 以前的版本虽然也可以编写自动化控制器和自动化服务器，但编写程序较为复杂，不易掌握。Delphi 5.0 对于 OLE 提供了强大的支持，利用 Delphi 5.0 最新提供的 Servers 栏控件可以很容易开发 OLE 自动化控制器实现对 OLE 自动化服务器的调用，发挥 Word、Excel、Powerpoint 的强大功能。

要对 Excel 进行控制，首先需要在操作系统中安装 Microsoft Excel 软件。对 Excel 的控制是通过 Delphi 的 Servers 栏控件中的 ExcelApplication、ExcelWorkbook、ExcelWorksheet 三个控件来完成。控制的过程如下：

• 用 ExcelApplication 控件与 Excel 建立连接，并创建一张工作簿。程序清单如下：

```
ExcelApplication1.Connect;                          //连接并打开 Excel
ExcelApplication1.Visible[0]:=True;                 //设定 Excel 的可见性
ExcelApplication1.Caption:='Excel Application';//设定标题名
ExcelApplication1.Workbooks.Add(Null,0);            //创建一个工作簿
```

• 用 ExcelWorkbook 控件连接已创建的工作簿。代码如下：

```
ExcelWorksheet1.ConnectTo(
        ExcelWorkbook1.Worksheets[1] as _Worksheet);
```

• 用 ExcelWorkbook 控件往工作簿中添加数据。例如：

```
ExcelWorksheet1.Cells.Item[row,column]:= '5 票' ;
```

• 保存文档。保存文档通过 ExcelApplication 控件的 SaveAs() 函数完成。

• 关闭文档。调用 ExcelApplication 控件的 Quit 方法可关闭文档。

4.5 多线程设计

无线下载电子投票表决系统的上位机软件需要处理发送数据、接收数据、数据处理等较多的任务，如果能采用多线程则可以大大方便程序的设计。

Delphi 的多线程程序设计是通过 TThread 类来实现。TThread 是一个能够在应用程序中创建互不相干的执行线程的类。每一个 TThread 对象的子类的新实例就是一个新的执行线程。从 TThread 类派生的多个实例就可以使 Delphi 应用程序完成多个线程的工作。要在应用程序中使用线程，必须从 TThread 中派生一个新类，然后重载其相关的方法[19]。

线程的创建通过 Creat 方法来实现，过程如下：

```
SendThread:=Transmit.Create(True);
```

其中，参数 True 表示线程在创建后处于被挂起的状态。Suspend() 函数用来挂起一个线程；Resume() 函数将处于挂起状态的线程处于就绪状态。

无线下载电子投票表决系统的上位机软件中，从 TThread 中派生了一个新类来进行数据发送。重载 TThread 的 Execute 方法为数据发送的过程。数据发送的过程是一个无限循环的过程，在这个过程中首先检查缓冲区是否有数据需要发送，如果有则将数据拆分成合适的大小，并加上报头和校验，然后将数据报写入到串口发送缓冲区。发送完数据后通过调用 Suspend() 来挂起线程。在需要发送数据的时候调用 resume() 来启动线程。

4.6 串行通信设计

计算机与主机间的通信采用 RS232 通信。RS232 接口是计算机最常用的通信接口之一。通过调用 Windows 通信 API，可以较方便的编写一个 Delphi 通信控件。和串行口通信相关的 API 函数有：打开通信端口 CreateFile()、获取串口状态 GetCommState()、设置串口状态 SetCommState()、获取串口事件屏蔽数据 GetCommMask()、设置串口事件屏蔽数据 SetCommMask()、读端口 ReadFile()、写端口 WriteFile()、设置发送接收缓冲区 SetupComm()以及等待串口消息 WaitCommEvent()等。

串行通信控件分以下五个部分：

1. 首先打开通信端口：

```
CommHandle := CreateFile( 'Com1' ,Generic_Read or Generic_Write,0,
Nil, Open_Existin,0,0);          //打开串口 1，返回串口句柄。
```

2. 设置串口通信有关参数：

```
GetCommState(CommHandle,MyDCB);  //获取串行口状态参数。
MyDCB.BaudRate := 115200;        //设置波特率 115200bps。
MyDCB.Parity := 0;               //没有奇偶校验位。
MyDCB.ByteSize := 8;             //8 位二进制数据格式。
MyDCB.StopBits := 1;             //停止位个数为 1。
...
SetCommState(CommHandle,MyDCB);  //设置通信状态参数。
SetupComm(CommHanle,4800,4800);  //设置发送接收缓冲区大小。
```

3. 设置串行口事件屏蔽参数：

```
GetCommMask(CommHandle,MyMask);  //获取当前事件屏蔽数据。
MyMask := MyMask or EV_RXCHAR;   //使能接收到数据事件。
MyMask := MyMask or EV_TXEMPTY;  //使能发送缓冲区空事件。
...
SetCommMask(CommHandle,MyMask);  //设置事件屏蔽数据。
```

4. 允许接收数据线程：

```
if WaitCommEvent(CommHandle,MyCommEvent,Nil) then
begin                            //产生串行口有关事件。
Case MyCommEvent of              //事件分支处理。
```

```
EV_RXCHAR:SendMessage(...);      //发送接收到数据消息。
EV_TXEMPTY:SendMessage(...);     //发送发送缓冲区空消息。
...
end;
end;
```

5. 消息处理程序：

相应的消息处理程序被串行口事件触发并执行，完成相应的操作。

Delphi 的控件设计也十分方便，完成源代码编写后，进行编译、注册即可生产被 Delphi 使用的新的控件，在程序中和系统原有的控件同样使用。

4.7 界面设计

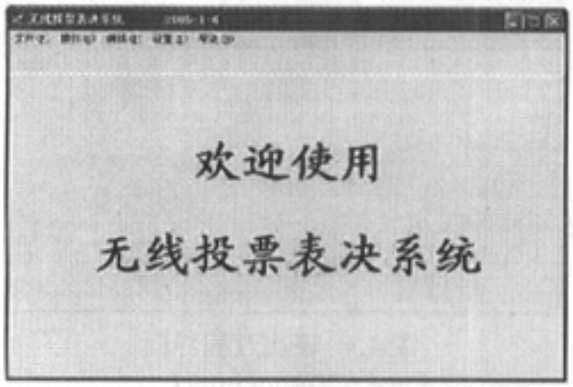


图 4.3 欢迎界面
Fig.4.3 Welcome interface

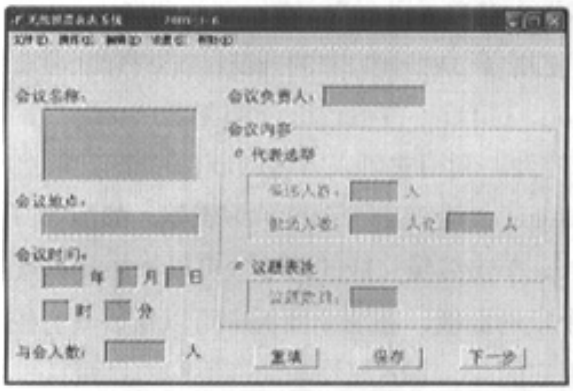


图 4.4 会议编辑界面
Fig.4.4 The conference Editing Interface

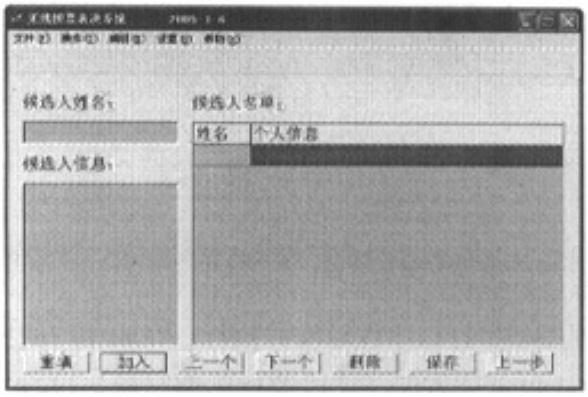


图 4.5 候选人名单录入界面
Fig.4.5 The candidates list input interface

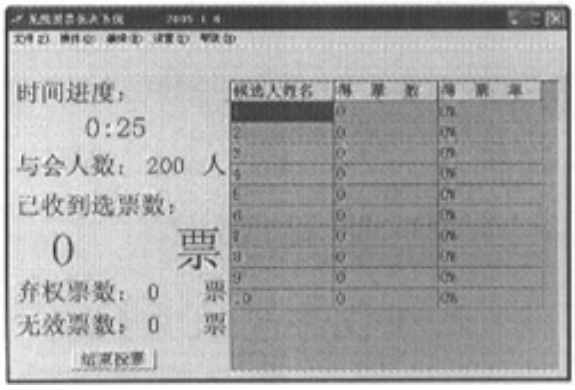


图 4.6 表决过程界面
Fig.4.6 Voting interface

无线下载电子投票表决系统上位机软件的界面包括：欢迎界面（图 4.3）、会议编辑界面（图 4.4）、候选人名单录入界面（图 4.5）和选举过程界面（图 4.6）等。

这些界面中主要使用了 Delphi 中的 mainmenu、panel、label、edit、memo、groupbox、radiobutton、button 和 stringgrid 等控件。

Mainmenu 是菜单控件，利用它可以方便的制作一个菜单。Delphi 允许菜单有无限的层次。双击控件 Mainmenu 会弹出一个菜单编辑器，如图 4.7。菜单的名称通过菜单项的 Caption 属性输入。单击编辑完成的菜单项可以添加菜单点击事件的代码。

Panel 控件是一个空的面板，在这个面板中可以摆放其它控件。本上位机软件中利用它来制作界面，利用它的 visible 属性来切换不同的界面。

Lable 是一个在窗口显示文字的控件，显示的内容通过 label 的 caption 属性输入。

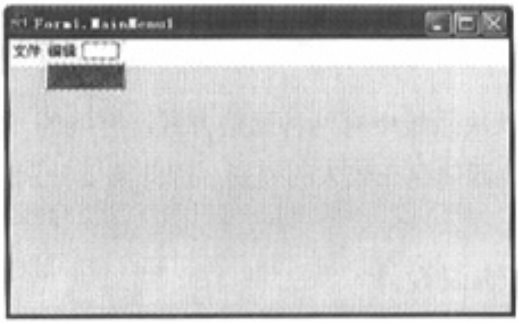


图 4.7 菜单编辑器
Fig.4.7 Menu editer

Edit 是一个接收用户字符串输入的编辑框。可以通过 edit 的 text 属性获得用户的输入信息。可以通过 maxlength 属性来限定用户输入的字符串的长度。

Memo 是一个输入多行文本的控件。Lines 属性是 memo 中一行文本，它是一个字符串数组，text 属性包含 memo 中的全部文本。Memo 控件也可以通过 maxlength 来设定输入的文本的长度。

Radiobutton 控件是一个多选一的控件，处在同一组的 radiobutton 具有排它性，即一个 radiobutton 被选中后其它的 radiobutton 就会处于未被选中的状态。它的 caption 属性可以输入选择项的内容。

Button 是一个按钮控件，主要用于接受用户鼠标的点击。Button 的 caption 属性用以显示按钮的意义。

Stringgrid 控件是一个二维表格，cells 是它的元素，向 cells 中写入需要显示的内容就可以在表格中显示。

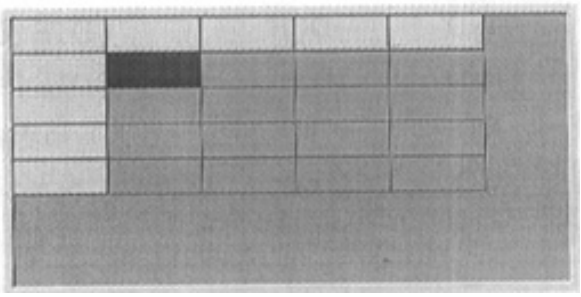


图 4.8 Stringgrid
Fig.4.8 Stringgrid

5 通信协议设计

无线下载电子投票表决系统中有两种通信方式：有线的 RS232 通信和无线射频通信。由于有线通信和无线通信存在较大的差别，因此需要分别对有线和无线通信进行协议设计。

5.1 主机与计算机通信协议设计

计算机与主机的通信采用 RS-232 通信，主要是基于以下几个原因：

- RS-232 通信速率已经完全能满足无线下载电子投票表决系统的要求；
- RS-232 通信比较简单，并且易设计开发。

5.1.1 RS-232 接口简介

目前最常用的串行通信总线接口是美国电子通信业协会（EIA）于 1969 年推荐的 RS-232C[22]。RS-232C 标准接口的全称是“使用二进制进行交换的数据终端设备和数据通信设备（DCE）之间的接口”。“RS-232C”中的 RS 是 Recommended Standard 的缩写，232 是标识符，C 表示此标准修改了三次。

由于 RS-232C 是早期为促进公用电话网络进行数据通信而制定的标准，其逻辑电平对地是对称的，与 TTL、MOS 逻辑电平完全不同。逻辑 0 电平规定为+5～+15V 之间，逻辑 1 电平为-5～-15V 之间。

表 5.1 RS-232C 接口常用引线信号定义、分类及功能

引脚号	信号名称	简称	方向	信号功能
1	保护地	——	——	接设备外壳
2	发送数据	TxD	→DCE	DTE 发送串行数据
3	接收数据	RxD	DTE←	DTE 接收串行数据
4	请求发送	RTS	→DCE	DTE 请求切换到发送方式
5	清除发送	CTS	DTE←	DCE 以切换到准备接受（清除发送）
6	数传设备就绪	DSR	DTE←	DCE 准备就绪
7	信号地	——	——	信号地
8	载波检测	DCD	DTE←	DCE 已接收到远程信号
20	数据终端就绪	DTR	→DCE	DTE 准备就绪
22	振铃指示	RI	DTE←	通知 DTE，通信线路准备就绪

通常使用 25 芯的接插件（DB25 插头和插座）来实现 RS-232C 标准接口的连接。RS-232C 定义了 20 根信号线，其中最常用的信号线的定义、分类及功能见表 5.1[23]。

RS-232C 信号连接方式，一般情况下有以下几种[22]：

• 全双工标准连接（DTE-DCE）方式：如计算机与调制解调器之间的连接，如图 5.1 所示；

• 三线连接（DTE-DTE）方式：如计算机与计算机之间连接，如图 5.2 所示；

• 反馈连接（DTE-DTE）方式：如计算机与打印机之间采用反馈连接方式，如图 5.3 所示；

• 交叉连接（DTE-DTE）方式：这种方式一般用于同一型号计算机之间连接，如图 5.4 所示。

在无线下载电子投票表决系统中主机和计算机的连接采用三线连接。

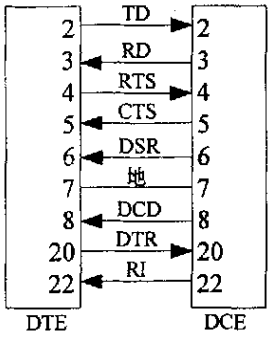


图 5.1 DTE 与 DCE 连接

Fig.5.1 Connection between DTE and DCE

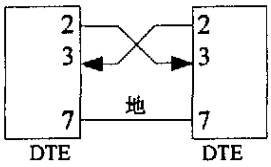


图 5.2 三线连接

Fig.5.2 Connection with three wires

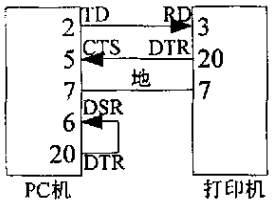


图 5.3 PC 机与打印机连接

Fig.5.3 Connection between PC and printer

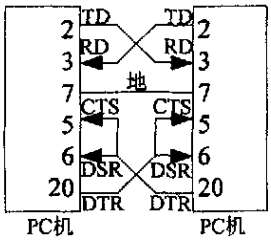


图 5.4 PC 机之间的连接

Fig.5.4 Connection between PC and PC

5.1.2 异步串行通信简介

串行通信分为同步和异步两种方式，RS-232C 采用异步方式。

在异步通信中，接收端与发送端各自运行在不同的时钟，因为发送端与接收端的时钟不同步，因此称为异步通信[24]。

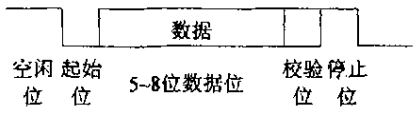


图 5.5 异步串行通信格式

Fig.5.5 Asynchronous serial communication form

如图 5.5 所示，为确保数据的同步，异步串行通信总是按固定的格式传送。在要发送数据前，首先发送一个由高电平到低电平跳变的起始位。然后，从低位开始发送数据位。接着发送奇偶校验位（可有可无）。最后发送一个高电平的停止位，停止位可以是 1 位也可以是 1.5 位。没有数据时，数据线持续发送结束位。

与异步串行通信相关的参数主要有：

- 通信速率（波特率）
- 数据位数
- 有无校验位，采用奇校验或偶校验
- 停止位位数（1 位或 1.5 位）

5.1.3 异步串行通信协议设计

参数设定：

- 波特率：115200bps
- 数据位数：8bit
- 校验：无校验
- 停止位：1bit

数据帧格式 在同步通信中往往按数据块发送数据。把需要发送的数据按如图 5.6 的顺序连接起来，组成数据块[25]。

同步字节1	同步字节2	n个数据字节	校验字节1	校验字节2

图 5.6 典型的同步通信数据格式

Fig.5.6 Typical synchronous communication data form

在无线下载电子投票表决系统的 RS-232 异步串行通信中采用类似的帧结构，并根据无线下载电子投票表决系统的特点，修改成如图 5.7。

帧同步字节	目的地址	源地址	命令字	n个数据字节	校验字节

图 5.7 帧结构

Fig.5.7 Structure of the frame

其中：同步字节 ：2 字节：0xAA55

目的地址 ：2 字节，0x0000~0xFFFF

源地址 ：2 字节，0x0000~0xFFFF

命令字 ：1 字节

数据 ：数据的长度随不同的命令而不同，但同一个命令的数据长度固定，数据的最大长度为 27 个字节。

校验字节 ：1 字节，采用 CRC-8 校验，校验从目的地址的首字节开始到数据的最后一个字节。

5.2 无线射频通信协议设计

与有线信道相比，无线信道是极其复杂的系统。对有线信道来说，其传输质量是可以控制的。通过选择合适的材料与精心加工，可以确保在有线传输系统中有一个相对稳定的电气环境。而无线信道是一个开放的环境，众多的无线信号都在同一个空间中传播，因此在无线信道中不可避免地存在着影响通信质量的噪声的干扰，需要采取措施来提高系统的抗干扰能力。

无线下载电子投票表决系统中无线通信的调制表示采用抗干扰能力强的 GFSK 调制，编码方式采用曼切斯特编码。

GFSK 调制方式是在调制前先利用高斯滤波器将基带信号滤波成为高斯形脉冲，然

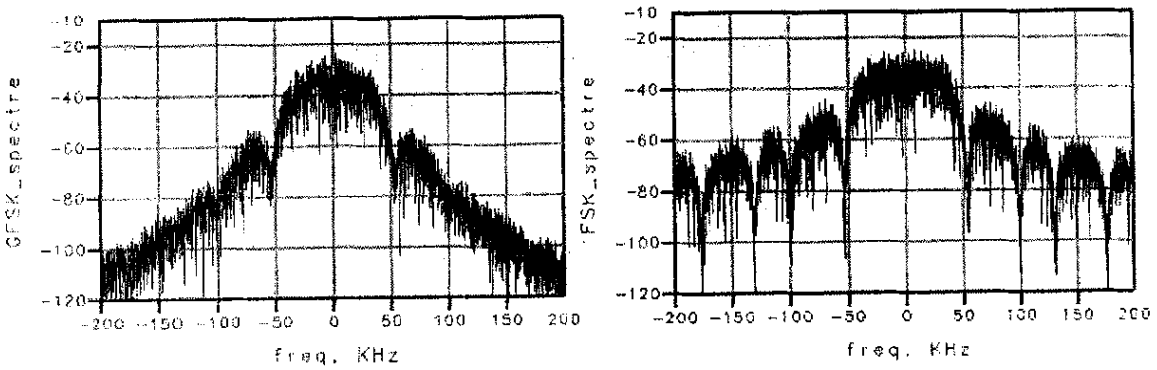


图 5.8 GFSK 和 FSK 的功率谱对比

Fig.5.8 Power density spectrum of a GFSK and FSK-signal

后进行 FSK 调制，因此与 FSK 相比在同等的带宽下 GFSK 具有更高的数据率。GFSK 和 FSK 调制的功率谱对比如图 5.8 所示。

曼切斯特编码是一种总是存在跳变的编码方式。0 时，在间隔的中间位置从高向低跳变；1 时，在间隔的中间位置从低向高跳变[26]。如图 5.9 所示。

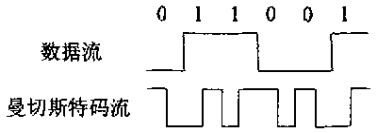


图 5.9 曼切斯特编码
Fig.5.9 Manchester coding

数据的曼切斯特编码和 GFSK 调制在无线通信模块内部完成。数据帧结构采用与串行通信类似的帧结构，数据帧的同步字节和校验字节由无线模块完成。

5.3 CRC 校验

CRC 校验即循环冗余校验码（Cyclic Redundancy Check），是一种通过模 2 运算来建立信息位和校验位之间的关系的编码形式。CRC 循环冗余校验具有比奇偶校验强得多的检错能力，可以证明它可以检测出所有的单个位错、几乎所有的双个位错、低于 $G(x)$ 对应二进制校验列位数的所有连续位错、大于或等于 $G(x)$ 对应二进制校验列位数的绝大多数连续位错。例如 CRC-16 码能百分之百地检查出单、双位错，奇数位、偶数位错以及多于 16 位的突发错误；能检查出 99.9969%的 17 位错误和 99.9984%的其他错误。

因为 CRC 的高效能，其在计算机磁盘数据存储、数字程控交换机 PCM 数据传输过程以及以太网数据传输、蓝牙技术中得到了广泛的应用。

5.3.1 CRC 编码原理

循环校验码将所传输的数据（信息）看作是一个高次（k 次）多项式，假设所传的 k 位数据（信息）为：$M=(m_{k-1},\ m_{k-2}\cdots m_1,\ m_0)$，它所对应的信息多项式为：$M(x)=m_{k-1}x^{k-1}+m_{k-2}x^{k-2}+\cdots+m_1x^1+m_0x^0$。

编码的过程是先将信息多项式乘上 x^{n-k} 得：

$$x^{n-k}\cdot M(x)=m_{k-1}x^{n-1}+m_{k-2}x^{n-2}+\cdots+m_1x^{n-k+1}+m_0x^{n-k} \tag{5.1}$$

用给定的(n,k)循环码生成多项式：

$$G(x)=g_{n-k-1}x^{n-k-1}+g_{n-k-2}x^{n-k-2}+\cdots+g_1x^1+g_0x^0 \tag{5.2}$$

除$x^{n-k}\cdot M(x)$得：

$$x^{n-k}\cdot M(x)=Q(x)\cdot G(x)+R(x) \tag{5.3}$$

其中$Q(x)$和$R(x)$分别是除得的商和余式。由于生成多项式$G(x)$为$(n-k)$次的，所以余式$R(x)$的次数一定小于等于$(n-k-1)$次，即：

$$R(x)=r_{n-k-1}x^{n-k-1}+r_{n-k-2}x^{n-k-2}+\cdots+r_1x^1+r_0x^0 \tag{5.4}$$

由于在计算中采用的模2操作，所以由(5.3)式可得：

$$x^{n-k}\cdot M(x)+R(x)=Q(x)\cdot G(x) \tag{5.5}$$

(5.5)式表明$x^{n-k}\cdot M(x)+R(x)$是生成多项式$G(x)$的倍式，即用$G(x)$去除$x^{n-k}\cdot M(x)+R(x)$所得的余式为零。$x^{n-k}\cdot M(x)+R(x)$即为循环码的码多项式，其展开式为：

$$\begin{aligned}x^{n-k}\cdot M(x)+R(x)=&m_{k-1}x^{n-1}+m_{k-2}x^{n-2}+\cdots+m_1x^{n-k+1}+m_0x^{n-k}\\&+r_{n-k-1}x^{n-k-1}+r_{n-k-2}x^{n-k-2}+\cdots+r_1x^1+r_0x^0\end{aligned} \tag{5.6}$$

它所对应的码字为：$(m_{k-1},\ m_{k-2}\cdots m_1,\ m_0,\ r_{n-k-1},\ r_{n-k-2}\cdots r_1,\ r_0)$，码字为不加改变的k位信息码元和在其后附加的$(n-k)$位校验码元组成。

在接收端对收到的码字组成的码多项式除以生成多项式$G(x)$，如果余式为零则收到的数据正确，其前k位即为所要传送的信息位；如果余式不为零则进行纠错或重传。

值得注意的是并不是任意的生成多项式$G(x)$都能满足要求，不好的生成多项式反而有适得其反的效果。从检错和纠错的要求出发生成多项式应具备下列条件：

• 当任何一位发生错误时，都能使余数不为 0；

• 不同位发生错误时余数应不相同；

• 对余数继续做模 2 除法运算时余数应是循环的。

生成多项式的选取主要依靠经验，下面几种生成多项式已经成为标准，它们具有极高的检错和纠错能力，它们是：

CRC-8 $G(x)=x^8+x^2+x+1$

CRC-12 $G(x)=x^{12}+x^{11}+x^3+x^2+x+1$

CRC-16 $G(x)=x^{16}+x^{15}+x^2+1$

CRC-CCITT $G(x)=x^{16}+x^{12}+x^5+1$

CRC-32 $G(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}$

5.3.2 查表法实现 CRC-8

CRC 校验可由多种方法实现，其中查表法是较常用的一种。多字节信息的 CRC 码可由单字节的 CRC 值经过特定的运算来实现。设 $M_n(x)$ 为 n 字节的信息，由 CRC 校验原理可知：

$$\frac{x^{16}\cdot M_n(x)}{G(x)}=Q(x)+\frac{R(x)}{G(x)}=Q(x)+\frac{x^8\cdot R_{nH}(x)+R_{nL}(x)}{G(x)} \tag{5.7}$$

当 $M(x)$ 增加一个字节时，$M_{n+1}(x)=x^8\cdot M_n(x)+M_1(x)$，则有：

$$\begin{aligned}\frac{x^{16}\cdot M_{n+1}(x)}{G(x)}&=\frac{x^{16}\cdot x^8\cdot M_n(x)}{G(x)}+\frac{x^{16}M_1(x)}{G(x)}=x^8\cdot Q(x)+x^8\cdot\frac{x^8\cdot R_{nH}(x)+R_{nL}(x)}{G(x)}\\&\quad+\frac{x^{16}M_1(x)}{G(x)}\\&=x^8\cdot Q(x)+\frac{x^{16}\cdot[R_{nH}(x)+M_1(x)]}{G(x)}+x^8\cdot\frac{R_{nL}(x)}{G(x)}\\&=x^8\cdot Q(x)+q(x)+\frac{x^8\cdot r_{nH}(x)+r_{nL}(x)}{G(x)}+x^8\cdot\frac{R_{nL}(x)}{G(x)}\\&=Q'(x)+\frac{x^8\cdot[r_{nH}(x)+R_{nL}(x)]+r_{nL}(x)}{G(x)}\end{aligned}$$

$$= Q'(x) + \frac{x^8 \cdot R_{n+1H}(x) + R_{n+1L}(x)}{G(x)} \tag{5.8}$$

由以上推理可知，当已知 n 字节信息的 CRC 校验值，求 n+1 字节信息的校验值时只要先由增加字节异或原 CRC 值的高 8 位，形成新的字节，求新字节的 CRC 值；新字节 CRC 值的高 8 位与原 CRC 的低 8 位相异或，便可求得要求 CRC 值的高 8 位，新字节的低 8 位 CRC 值为要求的 CRC 值的低 8 位。

5.4 通信命令定义

命令采用 8 位长度。其中高 3 位表示命令的功能分类信息。第 4 位表示命令的方向：0 表示从电子表决器到主机或从主机到计算机；1 表示从主机到电子表决器或从计算机到主机。低 4 位是命令字，表明命令的意义。命令格式如图 5.10 所示。

B7	B6	B5	B4	B3	B2	B1	B0
功能分类			方向	命令			

图 5.10 命令字结构

Fig.5.10 Structure of command

按照命令的功能和命令的对象将命令分为三类：网络建立类命令、网络维护和调度类命令和广播类命令。命令具体值见表 5.2。

5.5 无线通信网络的建立与维护

5.5.1 无线通信网络建立

要进行无线通信需要先建立无线网络，无线网络建立主要有以下步骤：

• 选择无线信道　主机上电初始化后依次在各个信道上监听，找出一个干扰相对较小（处于接收状态时，CD 引脚有较小的电压）的信道，作为系统工作的主信道；

• 主机发送呼叫信息　找到系统工作主信道后，在此信道上发送呼叫信息，等待从机（电子表决器）的接入。呼叫信息的格式如图 5.11，其中剩余容量为主机还允许接入的从机的数量。发送一帧呼叫信息后，主机等待若干时隙（时隙结构如图 5.12），等待从机的接入；

呼叫命令	剩余容量

图 5.11 呼叫帧格式

Fig.5.11 Structure of call frame

发送、接收转换时间	从机发送接入请求	发送、接收转换时间	主机发送接入允许	保护间隔

图 5.12 时隙结构

Fig.5.12 Structure of time slot

表 5.2 命令表

指令名	功能值	方向	命令	指令值	数据	数据长度
网络建立类命令						
呼叫	001	1	0001	0x31	剩余容量	2B
接入请求	001	0	0010	0x22	无	0B
接入允许	001	1	0011	0x33	无	0B
地址更改	001	1	0100	0x34	更改后的地址	2B
地址更改确认	001	0	0101	0x25	无	0B
网络维护和调度类命令						
查询	010	1	0001	0x51	工作状态	1B
普通应答	010	0	0010	0x42	无	12B
选举结果	010	0	0011	0x43	选举结果	12B
出错信息	010	0	0100	0x44	出错信息	12B
重传命令	010	0	0101	0x45	重传帧号	12B
广播类命令						
开始传数据	011	1	0001	0x71	总帧数	1B
传选票数据	011	1	0010	0x72	总帧数 本帧序号 数据	27B
传选举结果	011	1	0011	0x73	总帧数 本帧序号 数据	27B
选举开始	011	1	0100	0x74	无	1B
选举结束	011	1	0101	0x75	无	1B
重新选举	011	1	0110	0x76	无	1B
其他选举	011	1	0111	0x77	无	1B

• 从机发送接入请求 从机初始化后依次在各个信道上监听，当接收到有效的数据帧后，从机就找到了系统工作的主信道。当从机接收到主机的呼叫信息后，延时随机时间发送接入请求。为减小从机之间碰撞的可能性，从机延时的时间以时隙为单位。如果从机发送接入请求后，在本时隙内接收到主机的接入允许，则此从机接入成功，反之则等待下一个呼叫信息，直到接入成功；

• 主机发送接入允许 主机在规定的时间内接收到从机的接入请求后，发送接入允许信息，并将分配给此从机的地址发送给从机，实现地址的动态分配。

网络建立过程的流程图如图 5.13。

当有一个终端接入到无线网络中，主机便开始对从机进行循环查询，在查询一轮结束后继续发送呼叫信息，直到接入的终端数已经达到允许值。

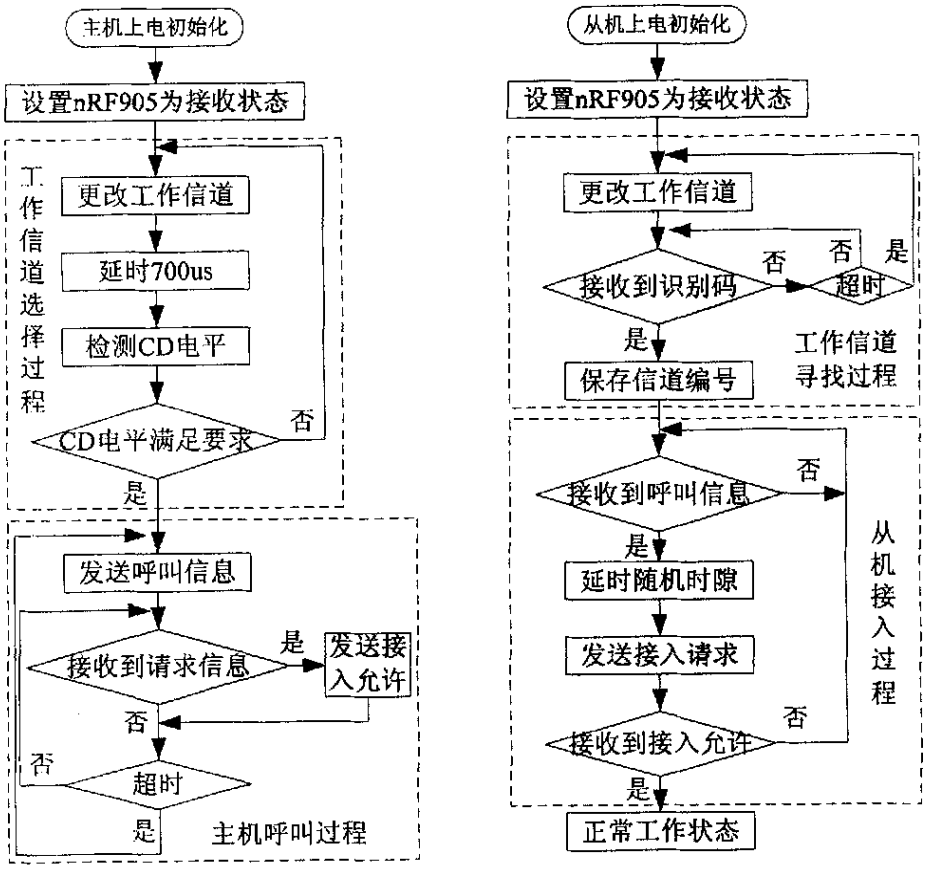


图 5.13 网络建立过程流程图

Fig.5.13 the Flow chart of the network sets up course

5.5.2 网络维护

网络维护包括两部分内容：主机对从机的查询和对地址的管理。

无线网络的通信采用主从方式，即每一次通信都有主机发起。需要与某个终端通信时，主机首先发送一帧数据给此终端，然后此终端再发送数据。主机对从机的查询采用轮询的方式，按照地址依次查询各终端。当主机查询到某个终端时，此终端可以将错误信息、需要重传的数据帧号、投票结果等信息传送给主机。

由于投票活动规模的不同，投票活动中需要的电子表决器数有较大的差别，主机对电子表决器查询一轮所需的时间也有较大差别。如果电子表决器采用固定地址，由于投票活动中使用的电子表决器的未确定性，主机需要对全部（包括投入使用的和未使用的）投票表决器进行查询。在较小规模的投票活动中，这种方式大大降低了查询的效率，采用地址的动态分配则能很好的解决这一问题。

但由于通信过程中误码和丢包现象的存在，有时地址分配不一定能成功：主机已经向从机发送接入允许帧，而从机未接收到正确的数据。或是由于电子表决器的故障需要更换。这时一些已经被分配的地址将被闲置起来，并且占用轮询的时间，为此主机需要对地址进行管理。如果主机连续轮询一定次数后，某个终端始终没有应答，则主机认为此终端无效，回收此终端的地址，在有其它终端接入时分配给其他终端。在终端方面，如果连续一定时间没有接收到主机的查询，则认为本终端已经被主机删除，终端将再次接入网络的过程。

6 数据加密

由于无线下载电子投票表决系统采用无线的通信方式，因此传输的数据容易被窃取，特别是投票结果。为保证投票结果的公正、正确，保障投票人的合法权益，需要对数据进行加密。

6.1 数据加密算法简介

从公元前 400 年人类开始对通信的内容进行加密到现在，密码学经历了一个漫长的历史时期。它的发展大约可分为三个阶段：古代加密方法、古典密码和近代密码[27]。虽然密钥学是一门古老的技术，但自诞生以来到第二次世界大战结束，密码始终应用于军事、机要、间谍等领域。从上个世纪中叶，随着计算机同通信技术的迅猛发展，大量的敏感信息通过公共通信设施或计算机网络进行交换，特别是 Internet 的广泛应用、电子商务和电子政务的迅速发展，越来越多的个人信息需要严格保密，密码学才走进了工作的日常生活当中。

密码的基本思想是对机密信息进行伪装。它主要有四个要素：明文空间、密文空间、密码方案和密钥空间。需要加密的信息称为明文（一般用 M 或者 P 表示），明文的全体称为明文空间。密文（一般用 C 表示）是经过伪装的后的明文，全体可能出现的密文的集合称为密文空间。密码方案是确切地描述加密变换与解密变换的具体规则。密钥是控制加密和解密算法操作的元素，分别称为加密密钥和解密密钥，密钥的全体称为密钥空间。

根据加密算法与解密算法所使用的密钥是否相同，或是否能简单地由加（解）密密钥求得解（加）密密钥，可以将密码体制分为对称密钥密码体制（也叫单钥密钥体制、秘密密钥体制、对称密码体制）和非对称密钥密码体制（也叫作双钥密码体制、公开密钥密码体制、非对称密钥密码体制）[27]。对称密钥密码体制如：DES、IDEA、AES 等，非对称密钥密码体制如 RSA、椭圆曲线密码等。

根据密码算法对明文信息的加密方式可以分为流密码和分组密码。流密码算法逐位地加密明文信息字符，如 A5、SEAL 等。分组密码将明文消息分组，逐组的进行加密，如 DES、IDEA、AES 等。

按照是否能进行可逆的加密变换，又可分为单向函数密码体制以及双向变换密码体制。

无线下载电子投票表决系统中使用的 AES 数据加密算法是一种可逆的对称分组密码体制。

6.2 AES 数据加密算法简介

由于计算机计算能力的日益提高 DES 已经不能满足信息加密的要求，1997 年 1 月 2 日美国国家标准技术研究所（NIST）宣布启动 AES（Advanced Encryption Standard）的开发研究计划，并于 1997 年 9 月 12 日指示发布征集算法的公告[27]。

经过长达三年、历经两轮评估，NIST 于 2000 年 10 月 2 日宣布 Rijndael 为 AES 的最终算法。Rijndael 算法的作者是两位比利时的密码专家：Joan Daemen 和 Vincent Rijmen。

Rijndael 密码在设计中考虑了以下三条准则：

- 抗所有已知的攻击；
- 在多个平台上速度要快和编码紧凑；
- 设计简单。

Rijndael 密码由三个不同的可逆已知变换组成：

- 线性混合层：包括行移位变换与列混合变换，确保多轮之后的高度扩散；
- 非线性层：由 16 个 S-盒并置而成，起到混淆作用；
- 密钥加层：简单地把圈密钥异或到中间状态上。

Rijndael 密码的分组长度和密钥长度是可变的，可以独立的指定为 128 比特、192 比特和 256 比特。

6.3 AES 数据加密算法实现

AES 中的各个不同的变换都在称之为状态的中间结果上运算[27]，它可以用字节的一个矩阵来表示，该矩阵有 4 行，列数根据分组长度不同而不同，分别为：4 列、6 列和 8 列。图 6.1 为分组长度为 128 比特的状态图[28]。

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

图 6.1 AES 中的状态矩阵
Fig.6.1 State array of AES

AES 使用了圈变换，变换的圈数随着密钥长度的不同而不同，圈数分别为：10 圈、12 圈和 14 圈。

6.3.1 AES 迭代步骤

在每一轮迭代中包括以下四步：字节代换运算、行变换运算、列混合变换运算、轮密钥的添加变换 [29]。

字节代换运算（SubByte()） 字节代换运算是一个可逆的非线性字节代换操作，这种变换要对分组中的每个字节进行，对字节的操作遵循一个代换表，即S盒[28]。变换分为两个过程：首先，把每个字节看作为表示了一个系数在{0，1}上的多项式，在有限域$GF(2^8)$中取它的相对于模多项式$m(x)=x^8+x^4+x^3+x+1$的乘法逆；其次，再使用下式所定义的GF(2)域上的仿射变换，作用在所求得的乘法逆上。

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{6.1}$$

上述变换可用矩阵（S盒）的形式表示出来，把S盒记为Sbox[x, y]，其中x表示将进行变换的状态矩阵字节分量的高四位，y表示低四位。

行变换运算（ShiftRows()）行变换是一种线性变换，变换在每一行间进行。变换以字节为单位循环左移，依次为：第 0 行不移位，第 1 行左移一字节，第 2 行左移两字节，第 3 行左移三字节。变换过程如图 6.2。

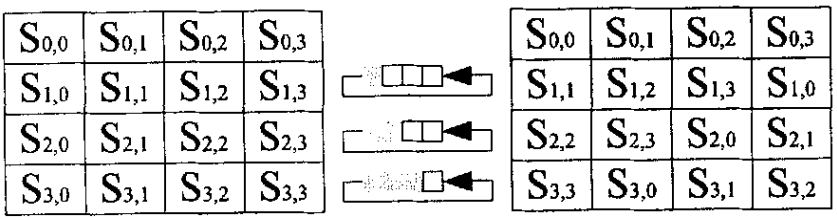


图 6.2 行变换过程

Fig.6.2 Rows shift

列混合运算（MixColumns()） 列混合变换作用在状态矩阵的每一列上。它把每一列视作有限域$GF(2^8)$上的四项多项式。在列混合变换中，每一列所表示的多项式将乘以一个固定的多项式a(x)：

$$a(x)=\{03\}\cdot x^3+\{01\}\cdot x^2+\{01\}\cdot x+\{02\} \tag{6.2}$$

并将所得的结果模x^4+1。这一过程写成矩阵乘法的形式如下：

$$\begin{bmatrix} s'_{0,c} \\ s'_{2,c} \\ s'_{3,c} \\ s'_{4,c} \end{bmatrix}=\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{6.3}$$

其中：$0\le c\le N_b$，N_b为状态的列数。

轮密钥的添加变换（AddRoundKey()） 在轮密钥的混合变换中，就是将轮密钥的各字节与状态中的各字节分别异或，实现密码和密钥的混合。轮密钥由加密密钥通过一定的规则扩展产生，产生的轮密钥总长为$N_b(N_r+1)$个字，N_r为迭代轮数，在每一轮的密钥混合变换中，都要从轮密钥$N_b(N_r+1)$中按顺序依次取出N_b个字与状态的各字节分别异或。若再次把轮密钥和各字节异或便得其逆运算。轮密钥的混合运算可用下面的式子表达：

$$[s'_{0,c},s'_{1,c},s'_{2,c},s'_{3,c}]=[s_{0,c},s_{1,c},s_{2,c},s_{3,c}]\oplus[W_{round\times N_b+c}] \tag{6.4}$$

其中，$0\le c<N_b$，$[W_i]$表示轮密钥中第i个字节，$0\le round\le N_r$。

6.3.2 密码调度过程

密码密钥（密码密钥计算法所接受的密钥）也被看成是下标从 0 到$4\times N_k-1$的一维字节数组$key[4\times N_k]$；当密钥长度分别为 128 比特、192 比特、及 256 比特时，对应地，这样的一维字节数组的下标分别是0..15、0..23、0..31[30]。

圈密钥是通过密钥调度过程从密码密钥中获得的。密钥调度过程首先将输入的密码密钥进行扩展，得到长度为 $N_b(N_r+1)$ 的一维（4 字节）字块扩展密钥数组 $W[N_b\times(N_r+1)]$。整个密钥调度过程的伪 C 代码如下：

```
KeyExpansion(byte key[4*Nb],word W[Nb*(Nr+1)],Nk)
{   word temp;
    int i=0;
    for (i=0;i<Nk;i++)
        W[i]=word(key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
    for (i=Nk;i<Nb*(Nr+1);i++)
    { temp=W[i-1];
      if (i%Nk==0)
         temp=SubWord(RotWord(temp))^Rcon[i/Nk];
      else if (Nk==8 && (i%Nk==4))
         temp=SubWord(temp);
      W[i]=W[i-Nk]^temp;
    }
}
```

其中 word(a,b,c,d)表示将四个单字节参数 a、b、c、d 分别按照从高位至低位的顺序表示为一个 4 字节的字（即 abcd）；函数 RotWord()返回一个字，该字的 4 个字是输入参数字所有的 4 个字节的一个循环移位，例如，字 abcd 作为参数的 RotWord()函数输出 bcda；SubWord()函数以一个字作为输入，返回对该字的每一字节都进行一次 SubBytes()变换所得到的结果。

可以看到，扩展密钥的最前面 N_k 个字是直接由输入的密码密钥填充的，而后面的每个字 W[i]则是由前面的字 W[i-1]与 N_k 各位置之前的字 $W[i-N_k]$ 进行异或得到。

并且，在 N_k 的整数倍位置处的字，在异或前还将对 W[i-1]进行一次变换，并与由字数组 Rcon[]取出的一个常数进行异或。当 N_k=8 时，如果 i-4 是 N_k 的整数倍，则在进行异或之前需要先对 W[i-1]进行 SubWord()变换。

由于密钥扩展时仅使用了前一个字与 N_k 个位置之前的字，因此在计算圈密钥时并不需知道所有以前的圈密钥，而仅须前一圈的圈密钥即可，因此，圈密钥可以具有 N_k 个字的缓存在进行全变换时实时地计算出来。

6.3.3 解密过程

出于 AES 的加密过程的每个步骤都是可逆的，因此可以按相反的过程使用加密过程中变换的反变换得到明文。

6.4 AES 的安全性及效率

6.4.1 AES 的安全性

到目前为止，对于 AES 密码的攻击还没有比穷尽密钥搜索攻击更有效的方法，已经公布的攻击思想不能形成有效的攻击[31]。目前对密码进行攻击的方法主要有差分和线性分析方法、变量法。差分和线性分析方法是到目前为止两种最有用的通用密码分析方法，对于 AES 来说，已经证明对于一个四轮的 AES，差分的分析概率上限是2^{-150}，线性分析方法的概率上限是2^{-75}，结合 AES 实际轮数，这些轮数对于抵抗上述攻击能够提供足够的安全性。在变量法中破解轮数最高的 Spuare 方法能在有 256 位相关密钥的2^{77}个明文、2^{224}个密文时破解 9 轮的 AES-256 密码。

AES 的密钥长度是可变的，当进行穷举密钥攻击时，穷举密钥的期望尝试次数与密码密钥的长度有关：对于 128 比特的密钥，期望尝试执行次数为2^{127}次，对于 192 比特的密钥，期望尝试执行次数为2^{191}次，对于 256 比特的密钥，期望尝试执行次数为2^{225}次。

6.4.2 AES 的效率

无论在有无反馈模式的计算环境下，AES 的硬件、软件实现都表现出了良好的性能。

AES 可以在包括 8 位和 64 位平台在内的各种平台及 DSP 上进行加密和解密。AES 算法的圈变换与 S 盒是完全并行的，它这种固有的高并行性便于有效使用处理器资源，即使不以并行的方法实现该算法，它的软件效能也非常好。另外，AES 对 RAM 和 ROM 的需求量低，非常适合在空间有限的环境单独进行加密或解密。

7 总结与展望

本章通过对已经制成的样机的测试来总结全文，并且对本设计在以后的应用中可以进一步提高的地方提出了一些想法。

7.1 性能测试与总结

电子表决器体积　试制的电子表决器的最小体积：130*67*20（长*宽*高，单位：mm），并且还可以继续缩小，完全满足手持的要求。

电源测试　一节 5 号干电池已能使电子表决器正常工作；工作平均电流在 50mA 左右，一节 5 号干电池能让电子表决器工作 10 小时以上，完全能满足一次会议的需要。

通信距离　测试表明，在室内视距 100m 的范围内，电子表决器和主机都能顺利通信。

电子表决器接入时间　在电子表决器较少时，5s 左右便可以顺利接入到无线网络中；对电子表决器较多时（上千个）时，经计算 1 分钟内可接入到无线网络。

数据传输成功几率　由于在通信协议中加入了提高系统可靠性的校验和重传机制，因此，数据几乎能百分之百的正确传输。

系统的计票速度　系统最快能在最后按下发送按键的选民按下发送按键后立即统计出投票结果；按最坏的情况考虑（3000 选民同时按下发按钮）考虑，系统也可在 1 分钟内完成选票统计。

从测试的结果来看，设计生产的电子表决系统基本满足设计要求。无线下载电子投票表决系统能够解决传统投票表决方式的缺点、弥补现有投票表决系统的不足，为民主政治的发展做出贡献。

7.2 展望

本设计考虑了设计制作成本和当今社会承受能力，在以后的设计中还有一些值得改进的地方，以增加系统的功能、提高系统的安全性。

- 添加触摸屏，以增加手写输入功能，方便有另选人的情况；
- 增加能检验选民身份有效性的功能，以防止冒名顶替的情况出现；
- 增加主机数量，以满足更大的会议场合需要。

参 考 文 献

[1] 胡盛仪，陈小京，田穗生. 中外选举制度比较，第一版. 北京：商务印书馆，2000.

[2] 唐竞新，童彦伶，李庆祥. 计算机控制管理的投票系统. 清华大学学报（自然科学版），1997，37（1）：98-101.

[3] 唐竞新，许欢. 第二代计算机控制和管理的投票系统. 计算机工程与应用 2002（13）：203-205.

[4] Chaum D. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM, 1981, 42(2): 84-88.

[5] Nordic. Single chip 433/868/915 MHz Transceiver nRF905 V1.1.http://www.nordic.no，2004：1-36.

[6] 裕能电子. nRF™系列单片无线收发器的应用设计，http://www.iclist.net，2004：1-13.

[7] 谢瑞和. 串行技术大全，第一版. 北京：清华大学出版社，2003.

[8] 胡大可. MSP430 系列 FLASH 型超低功耗 16 位单片机，第一版. 北京：北京航空航天大学出版社，2001.11.

[9] TI Corporation. MSP430x13x, MSP430x14x MIXED SIGNAL MICROCONTROLLER，http://www.ti.com, 2004:1-65.

[10] ST Corporation. M68AW031A, http://www.st.com, 2004：1-19.

[11] ATMEL.2-Wire Serial EEPROM，http://www.atmel.com, 2004:1-18.

[12] ST Corporation. M29W008AB，http://www.st.com, 1999:1-31.

[13] MAXIM Corporation. MAX3232, http://www.maxim-ic.com.cn, 1999:1-16.

[14] JEAN J.LABROSSE. μC/OS-II-源码公开的实时嵌入式操作系统，第一版. 北京：中国电力出版社，2001.

[15] JEAN J.LABROSSE.μC/OS-II:the Real Time Kernel, USA, 2000.

[16] 倪敏，周怡颓，杨继堂. COS-II 的任务切换机理及中断调度优化. 单片机与嵌入式系统应用 2003（10）：26-29.

[17] MSP430 IARCompil Programming Guide. IAR SYSTEMS, 1996.

[18] ChenJian, Labrosse Jean J. μC/OS port for the MSP430.http://www.ucos-ii.com,2003.

[19] 窦万峰. Delphi5 功能解析，第一版. 北京：电子工业出版社，2000. 4.

[20] 肖 建. Delphi6 编程基础，第一版. 北京：清华大学出版社，2002. 1.

[21] 杨志刚，何志成等. Delphi5 程序设计 基础教学篇，第一版. 北京：中国铁道出版社，2000. 1.

[22] 李朝青. PC 机及单片机数据通信技术，第一版. 北京：北京航空航天大学出版社，2000. 12.

[23] 曲伯涛. 80i86 微型计算机系统原理、接口与组装，第二版. 大连：大连理工大学出版社，1998. 6.

[24] （日）丸山修孝著，王庆译. 通信协议技术，第一版. 北京：科学出版社，2004. 1.

[25] 张友德，赵志英，涂时亮. 单片微型机原理、应用与实验，第二版. 上海：复旦大学出版社，1997. 9.

[26] William Stallings. Data & Computer Communications, 第六版. 北京：高等教育出版社，2001. 5.

[27] 宋震. 密码学，第一版. 北京：中国水利水电版社，2002. 7.

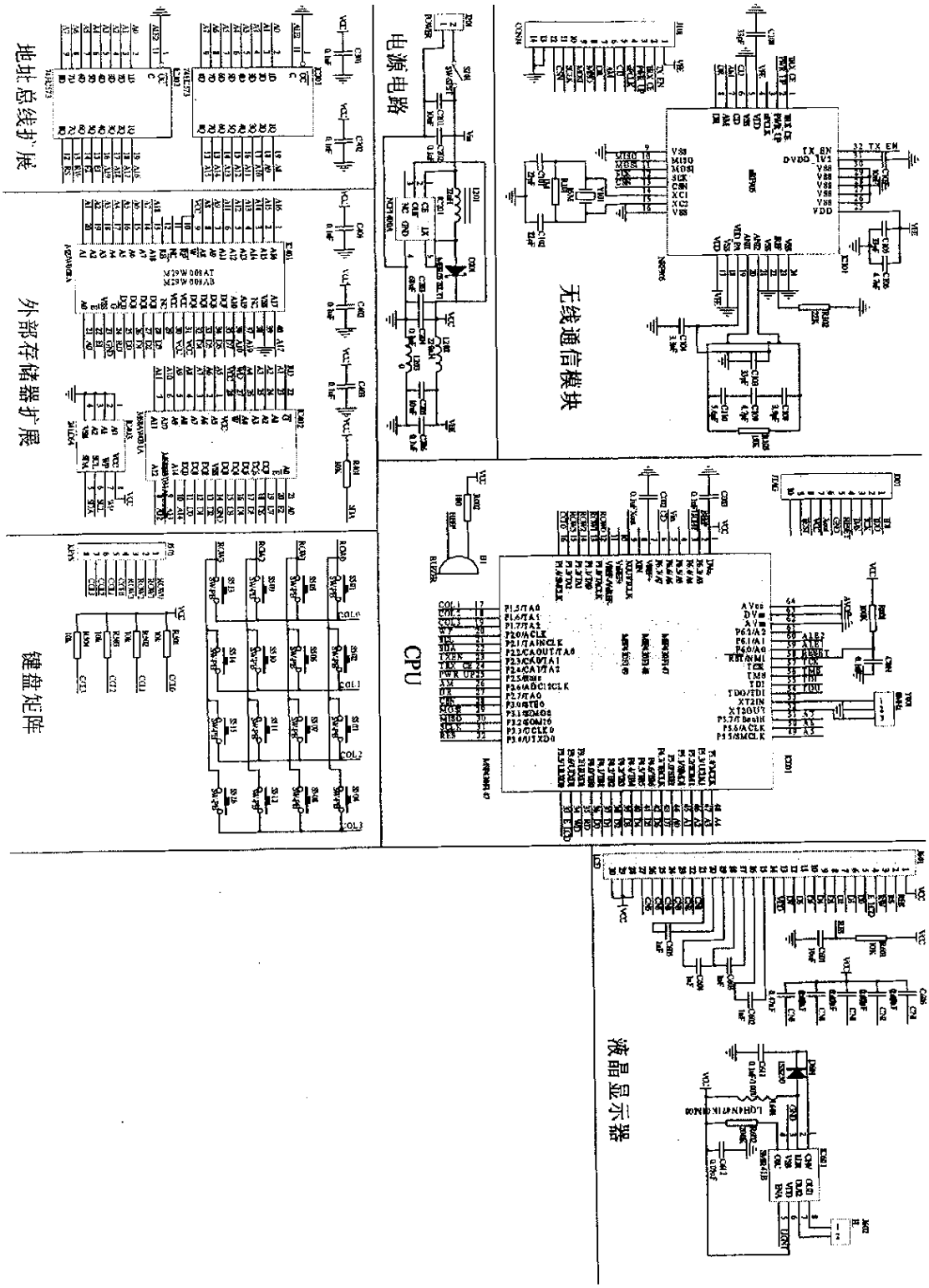
[28] 王先培，张爱菊，熊平，张俊. 新一代数据加密标准——AES 计算机工程. 2003，29（3）：69-70，186.

[29] Daemen J, Rijmen V. AES Proposal: Rijndael. AES Algorithm Submission, 1999, 09(03).

[30] Gladman B. Input and Output Block Conventions for AES Encryption Algorithms. AES Round 2 Public Comment, 1999, 06(06).

[31] 黄文平. AES 安全性分析 微型机与应用 2004（2）：4-6.

附录A　电子表决器原理图



附录B　AES 加密算法源码

```
//字节代替变换
void SubBytes(void)
{   unsigned char r,c,temp;
    for (r=0;r<4;r++)
    {   for (c=0;c<Nb;c++)
        {   temp=State[r][c];
            State[r][c]=Sbox[temp>>4][temp & 0x0F];
        }
    }
}

void InvSubBytes(void)
{   unsigned char r,c,temp;
    for (r=0;r<4;r++)
    {   for (c=0;c<Nb;c++)
        {   temp=State[r][c];
            State[r][c]=InvSbox[temp>>4][temp & 0x0F];
        }
    }
}

//行位移变换
void ShiftRows(void)
{   unsigned char r,c,temp[3];
    for (r=1;r<4;r++)
    {   for (c=0;c<r;c++)
            temp[c]=State[r][c];
        for (c=0;c<Nb-r;c++)
            State[r][c]=State[r][c+r];
        for (;c<Nb;c++)
            State[r][c]=temp[r+c-Nb];
    }
}
```

```
void InvShiftRows(void)
{   unsigned char r,c,temp[3];
    for (r=1;r<4;r++)
    {   for (c=Nb-r;c<Nb;c++)
            temp[c+r-Nb]=State[r][c];
        for (c=Nb-1;c>=r;c--)
            State[r][c]=State[r][c-r];
        for (c=0;c<r;c++)
            State[r][c]=temp[c];
    }
}

//列混合变换
//多项式乘法
unsigned char Multiply(unsigned char a,unsigned char b)
{   unsigned char i,j,result,temp;
    if (b & 0x01) result=a;
    else result=0x00;
    for (j=1;j<8;j++)
    {   if (b & Flag[j])
        {   temp=a;
            for (i=0;i<j;i++)
            {   if (temp & 0x80)
                {   temp<<=1;
                    temp=temp^0x1B;
                }
                else temp<<=1;
            }
            result=result^temp;
        }
    }
    return (result);
}
```

```
void MixColumns(void)
{   unsigned char r,c,temp[4];
    for (c=0;c<Nb;c++)
    {   for (r=0;r<4;r++)
            temp[r]=0;
        temp[0]=Multiply(State[0][c],0x02)^Multiply(State[1][c],0x03)^State[2][c]^State[3][c];
        temp[1]=State[0][c]^Multiply(State[1][c],0x02)^Multiply(State[2][c],0x03)^State[3][c];
        temp[2]=State[0][c]^State[1][c]^Multiply(State[2][c],0x02)^Multiply(State[3][c],0x03);
        temp[3]=Multiply(State[0][c],0x03)^State[1][c]^State[2][c]^Multiply(State[3][c],0x02);
        for (r=0;r<4;r++)
            State[r][c]=temp[r];
    }
}

void InvMixColumns(void)
{   unsigned char r,c,temp[4];
    for (c=0;c<Nb;c++)
    {   for (r=0;r<4;r++)
            temp[r]=0x00;

    temp[0]=Multiply(State[0][c],0x0E)^Multiply(State[1][c],0x0B)^Multiply(State[2][c],0x0D
)^Multiply(State[3][c],0x09);

    temp[1]=Multiply(State[0][c],0x09)^Multiply(State[1][c],0x0E)^Multiply(State[2][c],0x0B
)^Multiply(State[3][c],0x0D);

    temp[2]=Multiply(State[0][c],0x0D)^Multiply(State[1][c],0x09)^Multiply(State[2][c],0x0E
)^Multiply(State[3][c],0x0B);

    temp[3]=Multiply(State[0][c],0x0B)^Multiply(State[1][c],0x0D)^Multiply(State[2][c],0x09
)^Multiply(State[3][c],0x0E);
        for (r=0;r<4;r++)
            State[r][c]=temp[r];
    }
}
```

```
//圈密钥加法
void AddRoundKey(unsigned char round)
{   unsigned char r,c,i;
    for (c=0;c<Nb;c++)
    {   i=round*Nb+c;
        for (r=0;r<4;r++)
            State[r][c]=State[r][c]^W[i].BYTE[r];
    }
}

//密钥扩展
word Word(unsigned char a,unsigned char b,unsigned char c,unsigned char d)
{   word result;
    result.BYTE[0]=a;
    result.BYTE[1]=b;
    result.BYTE[2]=c;
    result.BYTE[3]=d;
    return (result);
}

word SubWord(word a)
{   unsigned char temp;
    temp=a.BYTE[0];
    a.BYTE[0]=Sbox[temp>>4][temp & 0x0F];
    temp=a.BYTE[1];
    a.BYTE[1]=Sbox[temp>>4][temp & 0x0F];
    temp=a.BYTE[2];
    a.BYTE[2]=Sbox[temp>>4][temp & 0x0F];
    temp=a.BYTE[3];
    a.BYTE[3]=Sbox[temp>>4][temp & 0x0F];
    return (a);
}

word RotWord(word a)
{   unsigned char temp;
```

```
    temp=a.BYTE[0];
    a.BYTE[0]=a.BYTE[1];
    a.BYTE[1]=a.BYTE[2];
    a.BYTE[2]=a.BYTE[3];
    a.BYTE[3]=temp;
    return(a);
}

void KeyExpansion(void)
{   word temp;
    unsigned char i,j;
    for (i=0;i<Nk;i++)
    {   j=i<<2;
        W[i]=Word(Key[j],Key[j+1],Key[j+2],Key[j+3]);
    }
    for (;i<Nb*(Nr+1);i++)
    {   temp=W[i-1];
        if (i%Nk==0)
        {
            temp=SubWord(RotWord(temp));
            temp.BYTE[0]=temp.BYTE[0]^RC[i/Nk];
            temp.BYTE[1]=temp.BYTE[1]^0x00;
            temp.BYTE[2]=temp.BYTE[2]^0x00;
            temp.BYTE[3]=temp.BYTE[3]^0x00;
        }
        else if (Nk==8 && (i%Nk==4))
            temp=SubWord(temp);
        W[i].BYTE[0]=W[i-Nk].BYTE[0]^temp.BYTE[0];
        W[i].BYTE[1]=W[i-Nk].BYTE[1]^temp.BYTE[1];
        W[i].BYTE[2]=W[i-Nk].BYTE[2]^temp.BYTE[2];
        W[i].BYTE[3]=W[i-Nk].BYTE[3]^temp.BYTE[3];
    }
}
```

```
//加密过程
void Cipher(void)
{   unsigned char round;
    AddRoundKey(0);
    for (round=1;round<Nr;round++)
    {   SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(round);
    }
    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);
}

//解密过程
void InvCipher(void)
{   unsigned char round;
    AddRoundKey(Nr);
    InvShiftRows();
    InvSubBytes();
    for (round=Nr-1;round>0;round--)
    {   AddRoundKey(round);
        InvMixColumns();
        InvShiftRows();
        InvSubBytes();
    }
    AddRoundKey(0);
}
```

硕士学位期间发表学术论文情况

[1] 王 云,毛德祥. 基于 nRF903 和 MSP430 的低功耗无线网络设计. 辽宁师范大学学报（社会科学版）. 2005 年增刊 2005,5. 内容与本文中的第五章（通信协议设计）相关.

致　谢

首先要感谢我的导师毛德祥老师。多年来，毛老师用丰富的理论和实践知识在工作和专业学习方面给予我悉心的指导，让我受益匪浅，毛老师严谨务实的工作作风更是鞭策着我努力进取。在完成项目和毕业设计的过程中，毛老师耐心的指导和讲解让我能够把握正确的研究方向，少走弯路，所有这些都是我今后工作和学习最宝贵的经验。

感谢大学生创新院的徐循老师、冯林老师和吴振宇老师，在近七年的时间里，你们给我创造了很多学习和工作条件，给了我很多的鼓励和帮助，在此毕业之际，我要衷心的向你们说一声，谢谢。

感谢创新院的同学们，与你们相识、合作是我的荣幸；感谢与我一起研究设计无线投票系统的付忠鑫、林立峰同学，我们曾共同付出过的努力，永远都是我美好的回忆。

感谢我的家人，在我遇到困难的时候，总能得到家庭的温暖和鼓励，在我取得进步的时候，总能得到家人默默的祝福。

大连理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“大连理工大学硕士、博士学位论文版权使用规定”，同意大连理工大学保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权大连理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：王云

导师签名：

2005年3月20日