基于无线传感器网的声音采集传输系统

院　（系）：电子信息与电气工程学院

专　　业：信息工程

学　　号：5020309014

学生姓名：陈卓

指导教师：徐国治

基于无线传感器网的声音采集传输系统

摘要

20 世纪90 年代以来，随着嵌入式系统、无线通信、网络及MEMS(微电子机械系统)等技术的快速发展，具有感知、计算和无线网络通信能力的传感器，以及由其构成的无线传感器网络(Wireless Sensor Network, WSN)引起了人们的极大关注。由于其广泛的应用前景，无线传感器网络受到越来越多研究人员的重视，被誉为21世纪四大高新技术之一。

本设计正是应用这一技术，实现了一个基于无线传感器网的声音采集传输系统。由无线传感器网采集声音数据，通过串口传给嵌入式设备，嵌入式设备将声音数据以及所采集的图像数据，通过以太网传输给远端的服务器上，最后由服务器完成声音数据的分析和图像的显示。无线传感器网络采用建立在 IEEE802.15.4 上的、极低功耗的 ZigBee 协议，作为整个网络的通信协议。传感器网络硬件节点采用 Microchip 公司的 PICDEM Z DEMO KIT，选取 ICOP 公司的 E-Box 作为嵌入式系统硬件平台，服务器为普通的 32 位计算机。

本文首先介绍了无线传感器网络通信的基础：IEEE802.15.4 协议规范，它定义了整个网络通信的物理层和数据链路层的 MAC 层，然后介绍了建立在其上的 ZigBee 网络通信协议。进而介绍了无线传感器网节点的硬件设计与实现，通信协议栈的软件设计与实现，语音采集电路的设计与实现。以及嵌入式设备与传感器网的通信接口，摄像头视频采集与网络传输功能的设计与实现。最后介绍了 PC 服务器网络接收，视频显示，声音处理和分析的软件设计与实现。

关键词： 无线传感器网，ZigBee，嵌入式系统

AUDIO SIGNAL COLLECTING AND TRANSPORTING SYSTEM BASE ON WIRELESS SENSOR NETWORK

ABSTRACT

With the development of embedded system，wireless communication, network, and micro electro mechanical systems (MEMS) from late 20th century, sensors which have the ability to perceive, compute and communicate with others in wireless way, and the wireless sensor network (WSN) which formed, have greatly attract our attentions. The researchers have paid more and more attentions on the researches of the wireless sensor network, because of its bright future in applications, and wireless sensor network is considered as one of the four high-techs in 21st century.

Our designs is just an application using this technology, and make a audio signal collecting and transporting system base on the wireless sensor network. The sensor network collects audio signal and transports them to the embedded system, and the embedded system collects video information and transports them and the audio data to remote server using Ethernet. The server displays the video and does some analyses about the audio data. The wireless communication in the wireless sensor network is base on the low power standard ZigBee transmission standard which is base on the standard IEEE802.15.4. We chose the product of Microchip Company PICDEM Z DEMO KIT as the hardware of the node of our wireless sensor network, and chose the product of ICOP Company E-Box as the hardware of our embedded system, and chose normal PC as the server.

In this thesis, we first introduce the base of communication way of wireless sensor network: IEEE802.14.4, which standard defines the physical layer and MAC layer of the transmissions, and we introduce ZigBee communication protocol base on it. And then introduce design and implementation of the hardware of wireless sensor network, the software of the communication stack, circuit of audio collecting, the interface between sensor network and embedded system, video collecting and Ethernet transporting. Finally we introduce the design and implementation of Ethernet receiver, video display and signal processing and analyzing of audio data on server PC.

KEY WORDS： Wireless Sensor Network, ZigBee, Embedded System

目录

1. 绪论……1
1.1 课题背景、目标和意义……1
1.2 系统整体方案设计……1
2. 传感器网络设计与实现……3
2.1 802.15.4 协议规范简介与特点……3
2.1.1 802.15.4 协议的规范简介……3
2.1.2 802.15.4 协议的特点……4
2.2 ZigBee 技术规范简介与特点……6
2.2.1 ZigBee 技术规范简介……6
2.2.2 ZigBee 与 IEEE802.15.4 的关系……7
2.2.3 ZigBee 技术的优势及与其他通信方式的比较……7
2.3 传感器网络硬件设计与实现……9
2.3.1 系统设计……9
2.3.2 微处理控制器……11
2.3.3 射频芯片……13
2.4 网络通信协议栈的设计与实现……15
2.4.1 物理层协议设计……15
2.4.2 数据链路层协议设计……18
2.5 语音采集设计与实现……22
2.5.1 声电转换及 AD 前级匹配电路的设计与实现……22
2.5.2 PIC 单片机 A/D 转换的软件实现……22
2.6 传感器网络的对外接口……25
2.6.1 UART 通讯协议……25
2.6.2 PIC 单片机 UART 通讯的软件实现……25
2.7 传感器网络节点软件设计……28
2.7.1 协调器节点……28
2.7.2 设备节点……30
2.8 本章小结……31
3. 基于 E-Box 的嵌入式系统设计……32
3.1 E-Box 嵌入式系统简介……32
3.1.1 E-Box 硬件系统简介……32
3.1.2 Windows CE.Net 操作系统简介……33
3.1.3 Platform Builder 和 Embedded Visual C++开发环境简介……34
3.2 传感器网数据的接收和网络传输……36
3.2.1 Windows CE.NET 下的串行通信开发……36

3.2.2 网络传输软件设计与实现 ……38
3.3 摄像头的视频采集与传输 ……41
3.3.1 摄像头与驱动程序简介 ……41
3.3.2 应用程序设计 ……41
3.4 本章小结 ……43
4. 基于 PC 的服务器系统设计 ……44
4.1 语音数据的以太网接收和分析 ……44
4.1.1 语音数据的以太网数据的接收与保存 ……44
4.1.2 节点语言数据显示与频谱分析 ……45
4.2 视频图像的以太网接收与显示 ……48
4.2.1 以太网图像数据的接收 ……48
4.2.2 位图图像的显示 ……49
4.3 本章小结 ……52
5. 总结和展望 ……53
5.1 总结 ……53
5.2 展望 ……54
参考文献 ……56
致谢 ……57
译文及原文 ……58

1. 绪论

1.1 课题背景、目标和意义

近年来禽流感疫情屡次发作，危害严重，为了战胜这场灾难，除了生物与医学方面的努力，电子信息技术在与瘟疫的抗争中也可以起到重要的作用。出于这样的目标，我们准备设计和搭建一个无线传感器网络，可以在条件恶劣的湿地环境工作，采集鸟类叫声和视频图像，并传输到远端的服务器上进行语音分析和图像显示。虽然要通过对音频信号的分析，全面推断出鸟类种群中的疫情分布尚不太现实，但是我们的至少可以通过对频谱和图像的分析，实现对鸟类种类和数量做大致判断，并且一定程度上分析声音中的异常状况。充分利用信息技术协助生物和医学专家，在这场与瘟疫的战役中取得最后的胜利。

1.2 系统整体方案设计

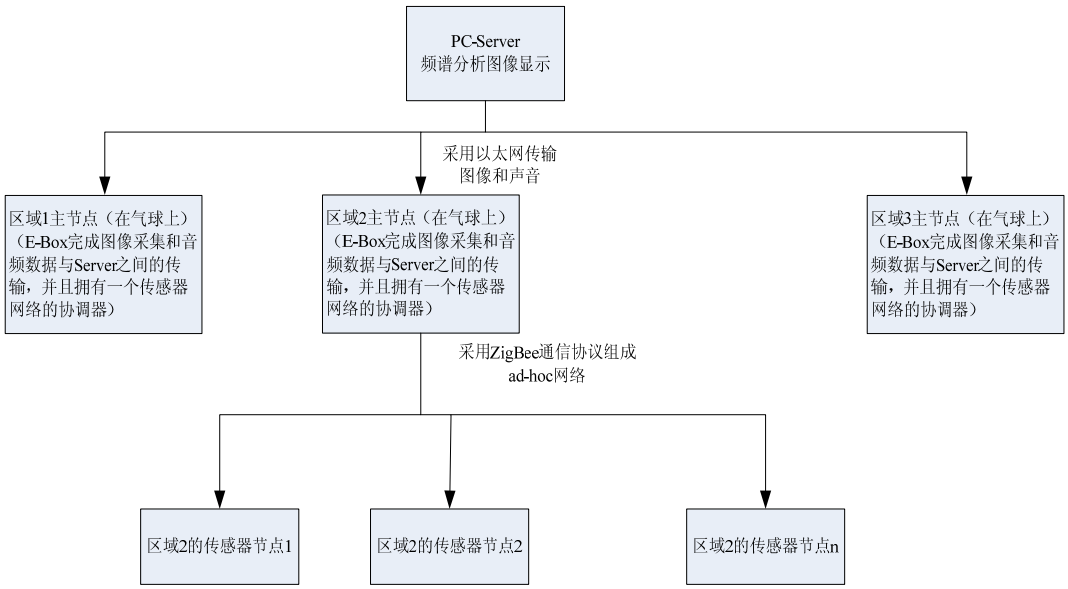

图 1-1 系统设计方案

Figure 1-1 System Design Architecture

我们系统的整体设计方案如图 1-1 所示，系统实物图如图 1-2 所示。

因为我们的无线传感器网络要搭建在野外或者环境恶劣的湿地，因此要把整个网络功耗降到最低，以解决长期监测的电池供电问题。而无线传感器网络的功耗主要集中在数据传输与通信中。所以我们选择了建立在 IEEE802.15.4 上的、低功耗的 ZigBee 传输协议，作为我们无线传感器网通信协议。而且 ZigBee 传输协议支持 250kb/s 的传输速率和至少 10 米的通信距离，符合我们设计的要求。在众多符合 ZigBee 协议通信规范的产品中，我们选择了性价比极高的 Microchip 公司生产的 PICDEM Z DEMO KIT 作为我们无线传感器网络的硬件节点，每个节点由 PIC 单片机与 RF 通信模块组成。

我们选择了 ICOP 公司的 E-Box，作为嵌入式图像采集、声音传输系统的硬件平台，它具有极强的运算处理能力以及丰富的通讯接口。E-Box 通过串口接收无线传感器网络采集的

语音数据，通过 USB 接口的摄像头采集视频信息，并最终通过以太网将语音和视频数据传送给远端在实验室中的 32 位计算机。

放在实验室中 32 位计算机作为整个系统的服务器，通过以太网接口，接收 E-Box 传送过来的声音和图像数据。运用信号处理软件对声音和图像进行存储、显示、处理和分析。

图 1-2a 系统实物图

Figure 1-2a Picture of Our System

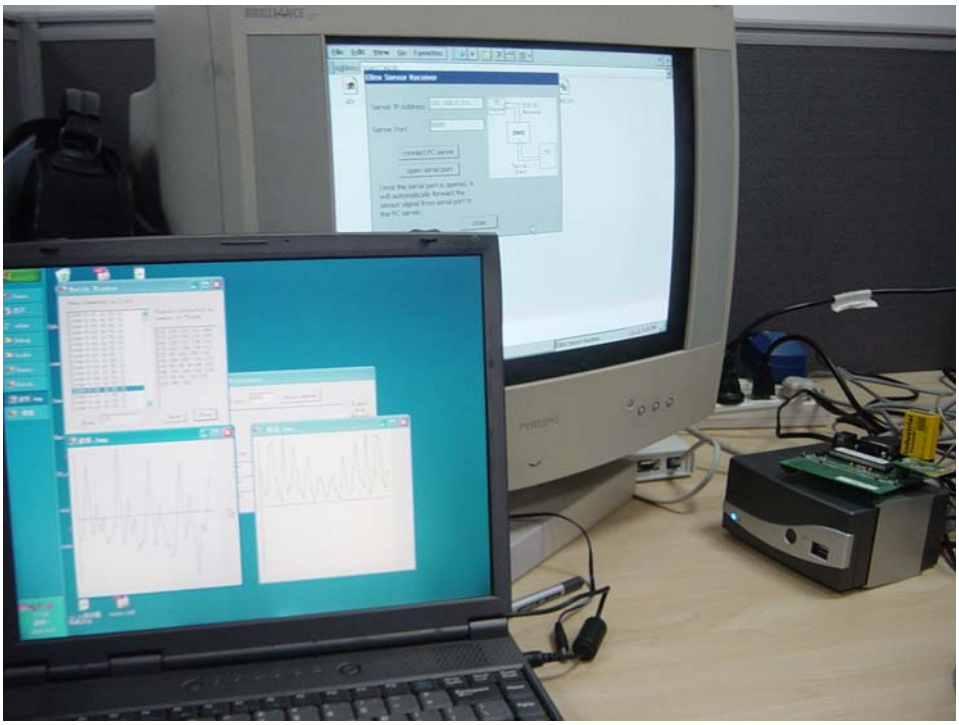

图 1-2b 系统实物图

Figure 1-2b Picture of Our System

2. 传感器网络设计与实现

传感器网络在整个系统中起着数据采集的作用，整个传感器网络为星形结构，如图 2-1 所示。整个传感器网络物理层和数据链路层的 MAC 层建立在 IEEE802.15.4 上，网络层及以上各层建立在 ZigBee 联盟制订的 ZigBee 协议规范上。每个传感器网络的节点有三部分组成，微处理控制器、传感器单元以及射频通信模块。根据节点在网络中的功能和地位不同，分为协调器节点和设备节点。其中协调器节点在本系统中起着协调其它节点工作、数据传输中转的作用。在网络组建时，它将建立绑定表。组建网络后接收传感器网的三个设备节点传来得数据，并将其传输到嵌入式设备 E-BOX。设备节点在本系统中起着数据采集和传输作用。在网络组建时，与协调器建立绑定关系。组建网络后进行 A/D 转换采集数据，将数据发送到协调器节点，并得到确认。

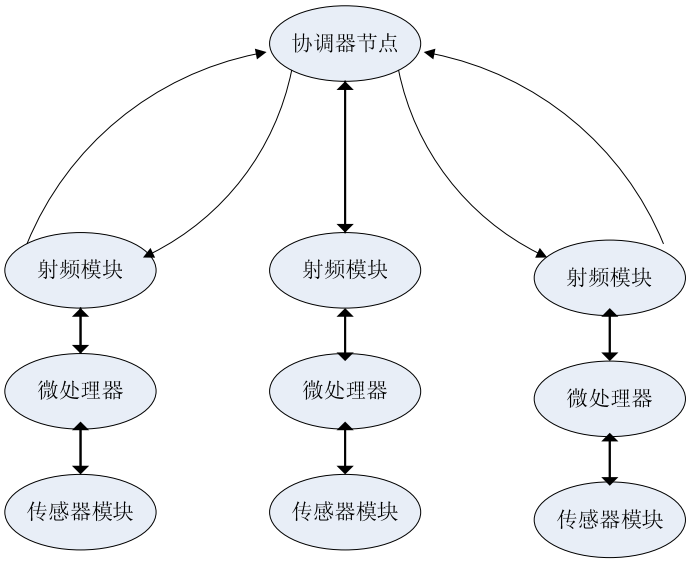

图 2-1 传感器网络结构图

Figure 2-1 Architecture of Sensor Network

2.1 802.15.4 协议规范简介与特点

2.1.1 802.15.4 协议的规范简介

802.15.4 包括用于低速无线个人域网(LR-WPAN)的物理层和媒体接入控制层两个规范。它能支持消耗功率少，一般在个人活动空间(10m 直径或更小)工作的简单器件。支持两种网络拓扑，即单跳星状或当通信线路超过 10m 时的多跳对等拓扑。但是对等拓扑的逻辑结构由网络层定义。LR-WPAN 中的器件既可以使用 64 位 IEEE 地址，也可以使用在关联过程中指配的 16 位短地址。一个 802.15.4 网可以容纳最多 216 个器件。

IEEE802.15.4 满足国际标准组织(ISO)开放系统互连(OSI)参考模式。它定义了单一的 MAC 层和多样的物理层(如图 2-2 所示)。

ZigBee	
网络应用层	
数据链路层	
IEEE802.15.4 LCC	802.2 LCC
IEEE802.15.4 MAC	
868/915MHz	2.4GHz

图 2-2 IEEE802.15.4 协议架构

Figure 2-2 Architecture of IEEE802.15.4

IEEE802.15.4 的 MAC 层能支持多种 LLC 标准，通过 SSCS（Service-Specific Convergence Sub Layer，业务相关的会聚子层）协议承载 IEEE802.2 类型之一的 LLC 标准，同时允许其他 LLC 标准直接使用 IEEE802.15.4 的 MAC 层服务。

IEEE802.15.4 定义了两个物理层标准，分别是 2.4GHz 物理层和 868/915MHz 物理层。它们都基于 DSSS（Direct Sequence Spread Spectrum，直接序列扩频），使用相同的物理层数据包格式，区别在于工作频率、调制技术、扩频码片长度和传输速率。2.4GHz 波段为全球统一的无需申请的 ISM 频段，有助于 ZigBee 设备的推广和生产成本的降低。2.4GHz 的物理层通过采用高阶调制技术能够提供 250kb/s 的传输速率，有助于获得更高的吞吐量、更小的通信时延和更短的工作周期，从而更加省电。868MHz 是欧洲的 ISM 频段，915MHz 是美国的 ISM 频段，这两个频段的引入避免了 2.4GHz 附近各种无线通信设备的相互干扰。868MHz 的传输速率为 20kb/s，916MHz 是 40kb/s。这两个频段上无线信号传播损耗较小，因此可以降低对接收机灵敏度的要求，获得较远的有效通信距离，从而可以用较少的设备覆盖给定的区域。

2.1.2 802.15.4 协议的特点

(1) 工作频段和数据速率

802.15.4 工作在工业科学医疗(ISM)频段，它定义了两个物理层，即 2.4GHz 频段和 868/915MHz 频段物理层。在 802.15.4 中，总共分配了 27 个具有三种速率的信道：在 2.4GHz 频段有 16 个速率为 250k bit/s(或 62.5k symbol/s)的信道，在 915MHz 频段有 10 个 40k bit/s(或 40k symbol/s)的信道，在 868MHz 频段有 1 个 20k bit/s(或 20k symbol/s)的信道。在保持简单性的同时，802.15.4 还试图提供设计上的灵活性。一个 802.15.4 网可以根据可用性、拥挤状况和数据速率在 27 个信道中选择一个工作信道。从能量和成本效率来看，不同的数据速率能为不同的应用提供较好的选择。例如，对于有些计算机外围设备与互动式玩具，可能需要 250k bit/s，而对于其他许多应用，如各种传感器、智能标记和家用电器等，20k bit/s 这样的低速率就能满足要求。

(2) 支持简单器件

802.15.4 低速率、低功耗和短距离传输的特点使它非常适宜支持简单器件。在 802.15.4 中定义了 14 个物理层基本参数和 35 个媒体接入控制层基本参数，总共为 49 个，仅为蓝牙传输协议的三分之一。这使它非常适用于存储能力和计算能力有限的简单器件。在 802.15.4 中定义了两种器件：全功能器件(FFD)和简化功能器件(RFD)。对全功能器件，要求它支持所有的 49 个基本参数。而对简化功能器件，在最小配置时只要求它支持 38 个基本参数。一

个全功能器件可以与简化功能器件和其他全功能器件通话，可以按三种方式工作，即用作个人域网协调器、协调器或器件。而简化功能器件只能与全功能器件通话，仅用于非常简单的应用。

(3) 信标方式和超帧结构

802.15.4 网可以工作于信标使能方式或非信标使能方式。在信标使能方式中，协调器定期广播信标，以达到相关器件同步及其他目的。在非信标使能方式中，协调器不是定期地广播信标，而是在器件请求信标时向它单播信标。在信标使能方式中使用超帧结构，超帧结构的格式由协调器来定义，一般包括工作部分和任选的不工作部分。

(4) 数据传输和低功耗

在 802.15.4 中，有三种不同的数据转移：从器件到协调器；从协调器到器件；在对等网络中从一方到另一方。为了突出低功耗的特点，把数据传输分为以下三种方式：

直接数据传输：这适用于以上所有三种数据转移。采用无槽载波检测多址与碰撞避免(CSMA-CA)或开槽 CSMA-CA 的数据传输方法，视使用非信标使能方式还是信标使能方式而定。

间接数据传输：这仅适用于从协调器到器件的数据转移。在这种方式中，数据帧由协调器保存在事务处理列表中，等待相应的器件来提取。通过检查来自协调器的信标帧，器件就能发现在事务处理列表中是否挂有一个属于它的数据分组。有时，在非信标使能方式中也可能发生间接数据传输。在数据提取过程中也使用无槽 CSMA-CA 或开槽 CSMA-CA。

有保证时隙(GTS)数据传输：这仅适用于器件与其协调器之间的数据转移，既可以从器件到协调器，也可以从协调器到器件。在 GTS 数据传输中不需要 CSMA-CA。

低功耗是 802.15.4 最重要的特点。因为对电池供电的简单器件而言，更换电池的花费往往比器件本身的成本还要高。在有些应用中，更换电池不仅麻烦，而且实际上是不可行的，例如嵌在汽车轮胎中的气压传感器或高密度布设的大规模传感器网。所以在 802.15.4 的数据传输过程中引入了几种延长器件电池寿命或节省功率的机制。多数机制是基于信标使能的方式，主要是限制器件或协调器之收发信机的开通时间，或者在无数据传输时使它们处于休眠状态。

(5) 安全性

安全性是 802.15.4 的另一个重要问题。为了提供灵活性和支持简单器件，802.15.4 在数据传输中提供了三级安全性。第一级实际是无安全性方式，对于某种应用，如果安全性并不重要或者上层已经提供足够的安全保护，器件就可以选择这种方式来转移数据。对于第二级安全性，器件可以使用接入控制清单(ACL)来防止非法器件获取数据，在这一级不采取加密措施。第三级安全性在数据转移中采用属于高级加密标准(AES)的对称密码。

AES 可以用来保护数据净荷和防止攻击者冒充合法器件，但它不能防止攻击者在通信双方交换密钥时通过窃听来截取对称密钥。为了防止这种攻击，可以采用公用密钥加密。

(6) 自配置

802.15.4 在媒体接入控制层中加入了关联和分离功能，以达到支持自配置的目的。自配置不仅能自动建立起一个星状网，而且还允许创建自配置的对等网。在关联过程中可以实现各种配置，例如为个人区域网（PAN）选择信道和识别符(ID)，为器件指配 16 位短地址，设定电池寿命延长选项等。

2.2 ZigBee 技术规范简介与特点

2.2.1 ZigBee 技术规范简介

ZigBee 是一组基于 IEEE 批准通过的 802.15.4 无线标准研制开发的，有关组网、安全和应用软件方面的技术标准。协议定义了两种相互配合使用的物理设备——全功能设备和削减功能设备：

● 全功能设备(Full function device, FFD)，可以支持任何一种拓扑结构，可以作为网络协商者和普通协商者，并且可以和任何一种设备进行通信。

● 削减功能设备(Reduced function device, RFD)，只支持星型结构，不能成为任何协商者，可以和网络协商者进行通信，实现简单。

ZigBee 协议中明确定义了三种拓扑结构：星型结构(Star)、簇状结构(Cluster tree)和网状结构(Mesh)，如图 2-3 所示。

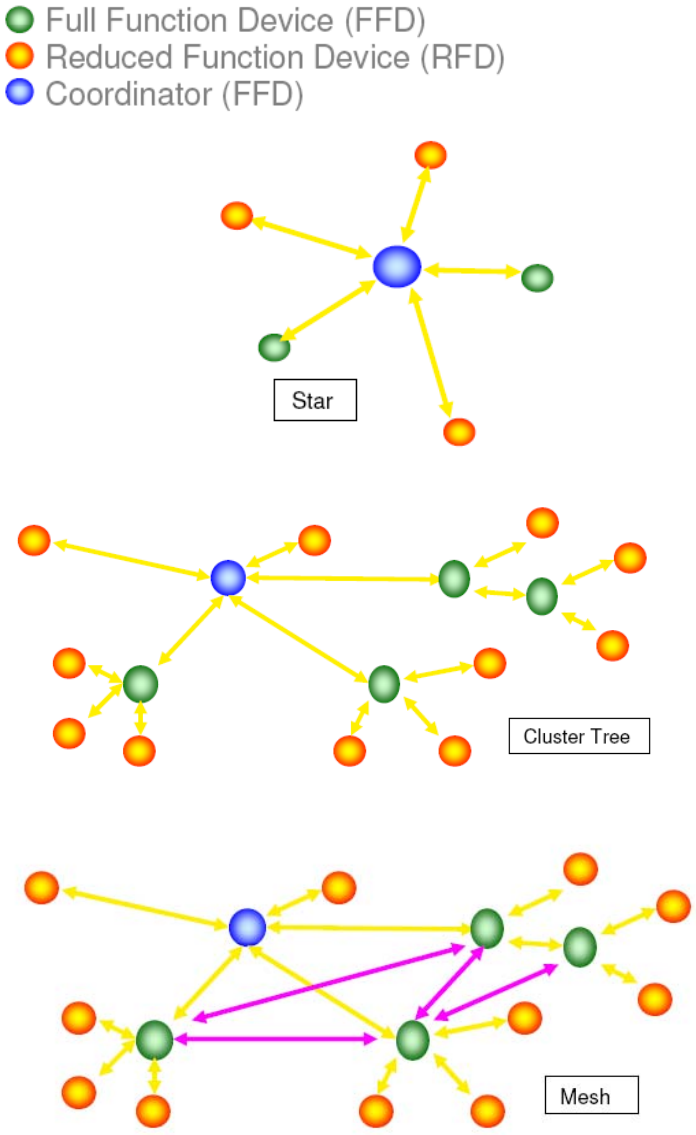

图 2-3 ZigBee 协议中的三种拓扑结构

Figure 2-3 Three Topologies of ZigBee Specification

IEEE 802.15.4/ZigBee 网络需要至少一个全功能设备作为网络协商者，终端节点一般使用削减功能设备来降低系统成本和功耗，提高电池使用寿命。另外所有设备必须使用一个 64 位的 IEEE 地址；可以使用 16 位短地址来减少数据包大小；寻址模式可以为网络增加设备标识符的星型结构，以及源和目标标识符的点到点结构两种。每个协调器可连接多达 255 个节点，而几个协调器则可形成一个网络，对路由传输的数目则没有限制。ZigBee 联盟还开发了安全层，以保证这种便携设备不会意外泄漏其标识，而且这种利用网络的远距离传输不会被其它节点获得。

完整的 ZigBee 协议套件由高层应用规范、应用会聚层、网络层、数据链路层和物理层组成。网络层以上协议由 ZigBee 联盟制定，IEEE802.15.4 负责物理层和链路层标准。

应用会聚层将主要负责把不同的应用映射到 ZigBee 网络上，具体而言包括：

（1） 安全与鉴权；

（2） 多个业务数据流的会聚；

（3） 设备发现；

（4） 业务发现。

网络层将主要考虑采用基于 ad hoc 技术的网络协议，应包含以下功能：

（1） 通用的网络层功能：拓扑结构的搭建和维护，命名和关联业务，包含了寻址、路由和安全；

（2） 同 IEEE802.15.4 标准一样，非常省电；

（3） 有自组织、自维护功能，以最大程度减少消费者的开支和维护成本。

2.2.2 ZigBee 与 IEEE802.15.4 的关系

ZigBee 与 IEEE802.15.4 的关系就好像 WiFi 与 IEEE802.11 的关系。网络层以上协议由 ZigBee 联盟制定，IEEE802.15.4 负责物理层和链路层标准。IEEE802.15.4 仅处理低级 MAC 层和物理层协议（如图 2-4），ZigBee 联盟对其网络层协议和 API 进行了标准化。

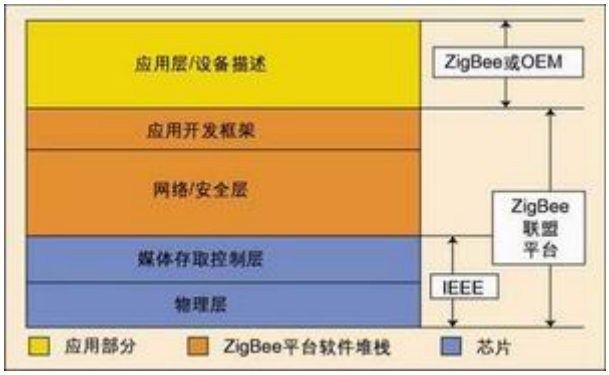

图 2-4 ZigBee 与 IEEE802.15.4 的关系

Figure 2-4 The Relationship Between ZigBee and IEEE802.15.4

2.2.3 ZigBee 技术的优势及与其他通信方式的比较

2.2.3.1 ZigBee 技术的主要优势

IEEE802.15.4 和 ZigBee 从一开始就被设计用来构建无线传感器网络。这是由其主要技术优势决定的：

(1) 数据传输速率低：只有 10k 字节/秒到 250k 字节/秒，专注于低传输应用。

(2) 功耗低：在低耗电待机模式下，两节普通 5 号干电池可使用 6 个月到 2 年，免去了

充电或者频繁更换电池的麻烦。这也是 ZigBee 的支持者所一直引以为豪的独特优势。

(3) 成本低：ZigBee 数据传输速率低，协议简单，所以大大降低了成本。且免收专利费。

(4) 网络容量大：每个 ZigBee 网络最多可支持 255 个设备。

(5) 时延短：通常时延都在 15 毫秒至 30 毫秒之间。

(6) 安全：ZigBee 提供了数据完整性检查功能，采用 AES-128 加密算法。

(7) 有效范围小：有效覆盖范围 10～75 米之间，具体依据实际发射功率的大小和各种不同的应用模式而定，基本上能够覆盖普通的家庭或办公室环境。

(8) 工作频段灵活：使用频段为 2.4GHz、868MHz(欧洲)及 915MHz(美国)，均为免执照频段。

此外，相对于常见的无线通信标准，ZigBee 协议套件紧凑而简单，其具体实现的要求很低，以下是 ZigBee 协议套件的需求估计：

(1) 8 位处理器，如 80c51；

(2) 协议套件软件需要 32kbytes 的 ROM；

(3) 最小协议套件软件大约 4kbytes 的 ROM；

(4) 网络主节点需要更多的 RAM，以容纳网络内所有节点的设备信息、数据包转发表、设备关联表、与安全有关的密钥存储等。

2.2.3.2 ZigBee 技术与蓝牙和 Wi-Fi 的比较

蓝牙技术基本上只是设计作为有线的替代品，经常是为手机和附近的耳机或 PDA 联网用的。它可以在不充电的情况下工作几周，但无法工作几个月，更不用说几年了；

一般情况下，蓝牙设备需要人手动配置和维护网络连接；它可以用来有效地处理 8 个设备（一个主设备和 7 个从设备），如果更多的话，通讯速率则显著下降。

而 802.11，也被称作 Wi-Fi 也有类似的问题。虽然它是将笔记本和桌面电脑接入有线网络的很好的解决方案，但它的功耗却非常高。

虽然他们的速度快，但是功耗方面是不能跟 ZigBee 相比的。这也是我们传感器网络采用 ZigBee 技术的原因。系统需要在只有电池供电的情况下完成长时间采集工作。

2.2.3.3 ZigBee 网络与 GPRS/CDMA1X 的比较

共同点：

(1) 它们都是具有自动数据中转和路由功能的无线数据传输网络；

(2) 用户只需要将他们的远程终端，传感器，以及控制计算机与网络相联即可，而不需要知道他们的数据是如何在网络中传输的；

(3) 在每一个 ZigBee 网络主节点（FFD）的 MCU 中,都有自动路由和网络控制管理软件，它相当于 GPRS/CDMA1X 的一个基站，只不过比起移动网络基站来要简单得多；我们所使用的移动网终端---GPRS/CDMA1XModem,,就相当于 ZigBee 网络的一个子节点（RFD）；

(4) 它们都遵循国际通信标准，可以很容易实现互联互通，例如,我们可以很容易通过 GPRS/CDMA1X 网络，或互联网，将远处的多个 ZigBee 局域网连接在一起成为一个网络进行管理。

不同点：

(1) GPRS/CDMA1X 移动网，是一个大范围内（覆盖全国）的大网络，它们主要是以语音通信为目的而设计的，因而在应用于数据通信时，难免存在这样或那样的不足。而 ZigBee 网络是用户专门根据自己的实际应用需要而布置，建立的一个无线数据传输局域网，因此，用户使用起来更加有效可靠；

(2) 除了专门的移动基站外，GPRS/CDMA1X 网络的基站都是固定的，而 ZigBee 网络主节点由于体积小（香烟盒大小），而且可以动态组网，因而移动十分方便；

(3) GPRS/CDMA1X 移动网基站结构复杂，价格昂贵，维护工作量也很大，而 ZigBee 网络主节点（FFD）由于结构简单，体积小，因而安装维护都十分简单方便，而且每一个主节点自身就可以当作为网络终端使用，承担数据采集和终端控制，以及直接与控制计算机连接的工作。

(4) 现有 GPRS/CDMA1X 网络基站庞大复杂，相互间的距离一般都很远，而 ZigBee 网络主节点由于体积小且成本很低，它们之间的距离可以根据需要布置得很近，因而，它可以用来进行移动目标的网络定位。

经济上：

(1) GPRS/CDMA1X 是付费网络，使用 GPRS/CDMA1X，购买用户终端的价格在 1000 元人民币上下，而 ZigBee 网络节点的价格要低得多，特别是近距离的只承担数据采集，而不承担网络数据中转任务的子节点，例如，星型网络结构的远端节点，移动子节点等；

(2) GPRS/CDMA1X 网络属于移动网络运营商，用户需要长期支付网络使用费，而 ZigBee 网络是用户自建网络，属于用户所有，不需要支付任何费用；

(3) 在 GPRS/CDMA1X 网络覆盖盲区，先建立一个 ZigBee 网络对盲区进行覆盖，再将这个 ZigBee 网络，通过远距离 ZigBee 网络节点（2-10 公里）接入最近的 GPRS/CDMA1X 网，将是一个十分经济而有效的解决方案。而在远端控制终端（数据采集或监控点）离开控制中心，或其它通信网络（例如互联网）接口不远（几公里范围内）时，直接使用 ZigBee 网络无疑将大大降低你的系统建设成本和运营成本，并将极大地提高系统信息传输的可靠性。

从经济和成本方面考虑，我们选取 ZigBee 而不是 GPRS/CDMA1X 来组建我们的传感器网络。

2.3 传感器网络硬件设计与实现

2.3.1 系统设计

传感器结点主要由 4 部分组成，如图 2-5。数据处理和控制模块主要包含微处理器以及存储器，其主要芯片是 Microchip 公司的 PIC18F4620 单片机。通信模块主要包括 CC2420 2.4GH 射频通信芯片，BalUn 电路和天线。其中 CC2420 符合 IEEE802.15.4 标准，为其提供硬件支持。BalUn 电路主要任务是完成双端到单端的转换。数据采集模块为语音采集电路（将在 2.4 节中详细说明）。单个结点的实物图如图 2-6 所示。

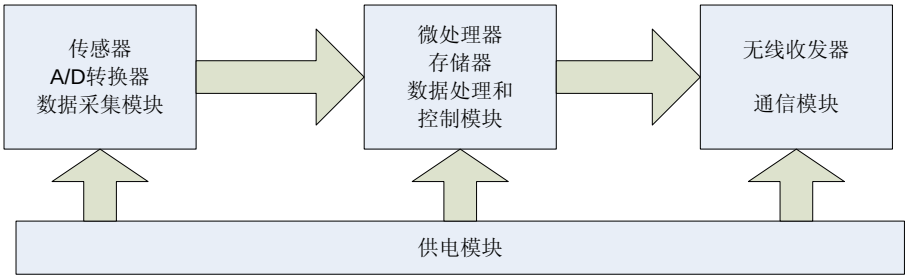

图 2-5 无线传感器网络节点的基本结构

Figure 2-5 The Basic Structure of One Node of Sensor Network

图 2-6 单个节点实物图

Figure 2-6 Picture of one node of sensor network

本系统采用星型网络配置，如图2-7所示。由一个协调器节点（主设备）和三个终端设备（从设备）组成。协调器是实现了一组ZigBee服务的全功能设备（Full Function Device，FFD）。终端设备为简化功能设备（RFD）。RFD是最小而且最简单的ZigBee节点。它只实现了一组最少的ZigBee服务。在星型网络中，所有的终端设备都只与协调器通信。如果某个终端设备需要传输数据到另一个终端设备，它会把数据发送给协调器，然后协调器依次将数据转发到目标接收器终端设备。

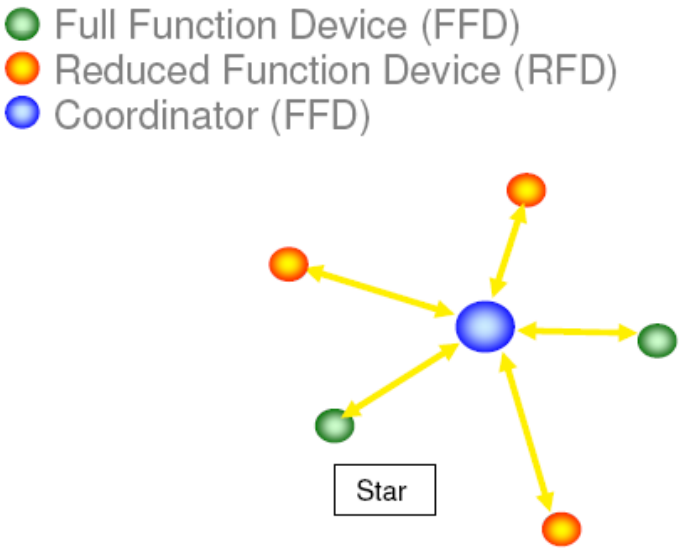

图 2-7 系统网络配置示意图

Figure 2-7 Structure of Network System

2.3.2 微处理控制器

我们对的传感器节点选用的微处理和控制器为 PIC18F4620 单片机，它是 Microchip 公司生产的基于增强型 FLASH 带 10 位 A/D 转换和纳瓦技术的 16 位单片机。芯片内部集成了大容量的存储器（64KFlash，4KSRAM）和丰富强大的硬件接口电路，具有运算速度快，功耗低，功能强大，性价比高的优点。图 2-8 是 PIC18F420 单片机的结构框图，属于哈佛结构。具有多种电源管理模式降低功耗，并支持双速晶振启动：

(1) 运行模式：CPU 工作 外围器件工作

(2) 空闲模式：CPU 关闭 外围器件工作 低电流典型值 5.8uA

(3) 睡眠模式：CPU 关闭 外围 关闭 低电流典型值 0.1uA

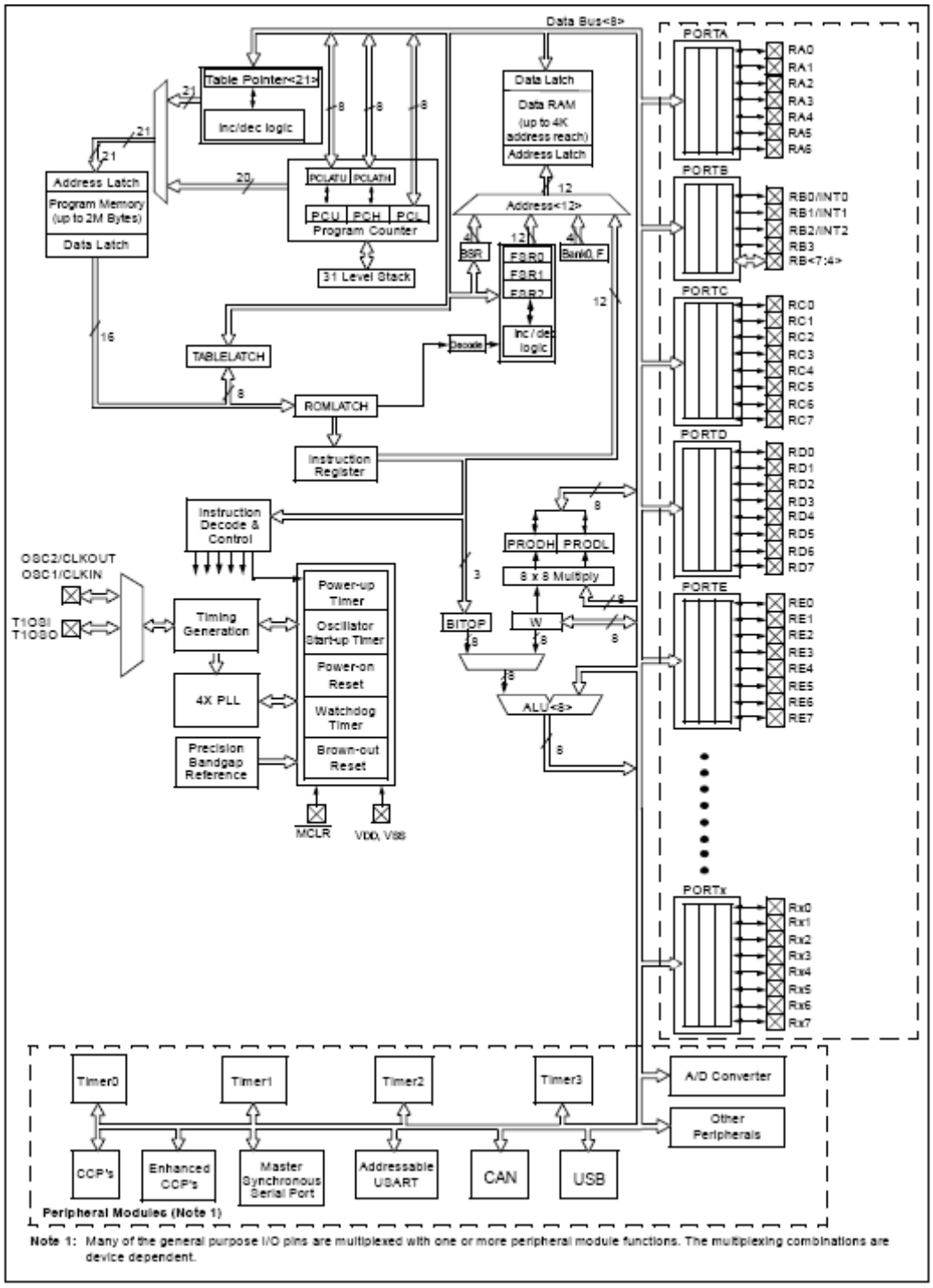

图 2-8 PIC18F4620 内部结构框图

Figure 2-8 General Block Diagram of PIC18F4620

采用柔性振荡结构，最高始终频率可达 40MHz。它拥有 3 个可编程外部中断脚，4 个输入脚转换中断，2 个捕捉器/比较器/PWM 模式 CCP，2 路 PWM 输出，主同步串行口 MSSP 有两种工作模式：3 线 SPITM（支持全部 4 种 SPI 模式）和 I2CTM 主动和从动模式，增强功能的可寻址通用同步异步收发器 USART 模式，10 位，13 路通道 A/D 转换模式和自动采集功能。

此外它还支持一些特殊的功能，如编译器优化结构体系；可扩展指令设置的体系结构；增强功能的 FLASH 可经受 100，000 次擦/写操作；EEPROM 可经受 1，000，000 次擦/写操作；FLASH/数据 EEPROM 数据保存期可超过 100 年；软件控制自动可改编程序；中断优先级；8×8 单周期硬件乘法器；扩展的看门狗定时器等。

2.3.3 射频芯片

CC2420 是 Chipcon AS 公司推出的首款符合 2.4GHz, IEEE802.15.4 标准的射频收发器。该器件包括众多功能，是第一款适用于 ZigBee 产品的 RF 器件。它基于 Chipcon 公司的 SmartRF03 技术，以 0.18um COMS 工艺制成，只需极少外部元器件，性能稳定且功耗极低。CC2420 的选择性和敏感性指数超过了 IEEE802.15.4 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数据传输率高达 250kbps。可以实现多点对多点的快速组网。

CC2420 实现了 IEEE802.15.4 物理层和部分 MAC 层功能。主要性能特点如下：

(1) 工作频带范围：2.400～2.4835GHz；
(2) 采用IEEE802.15.4规范要求的直接序列扩频方式；
(3) 数据速率达250kbps，码片速率达2Mchip/s；
(4) 采用O-QPSK调制方式；
(5) 超低电流消耗(RX：19.7mA, TX：17.4mA)，高接收灵敏度(-94dBm)；
(6) 抗邻频道干扰能力强(39dB)；
(7) 内部集成有VCO、LNA、PA以及电源整流器，采用低电压供电(2.1～3.6V)；
(8) 输出功率编程可控；
(9) IEEE802.15.4MAC层硬件可支持自动帧格式生成、同步插入与检测、16bit CRC校验、电源检测、完全自动MAC层安全保护(CTR, CBC-MAC, CCM)；
(10) 与控制微处理器的接口配置容易(4总线SPI接口)；

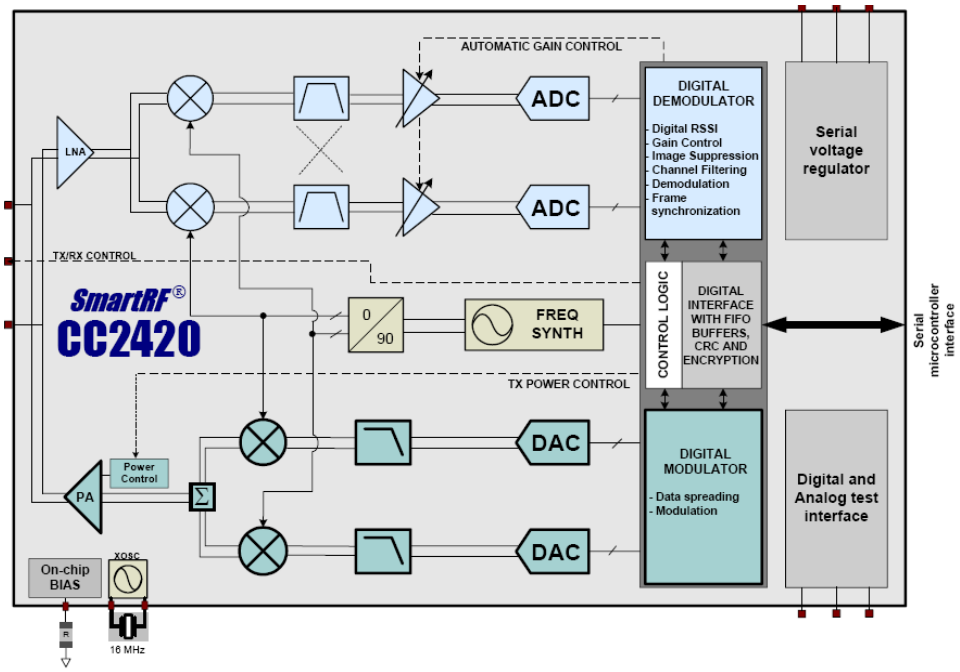

图2-9 CC2420芯片的内部结构

Figure 2-9 Block Diagram of CC2420

CC2420芯片的内部结构如图2-9所示。天线接收的射频信号经过低噪声放大器和I/Q下变频处理后，中频信号只有2MHz，此混合I/Q信号经过滤波、放大、A/D变换、自动增益控制、数字解调和解扩，最终恢复出传输的正确数据。

发射机部分基于直接上变频。要发送的数据先被送入128字节的发送缓存器中，头帧和起始帧是通过硬件自动产生的。根据IEEE802.15.4标准，所要发送的数据流的每4个比特被32码片的扩频序列扩频后送到D/A变换器。然后，经过低通滤波和上变频的混频后的射频信号最终被调制到2.4GHz，并经放大后送到天线发射出去。

CC2420只需要极少的外围元器件，其典型应用电路如图2-10所示。它的外围电路包括晶振时钟电路、射频输入/输出匹配电路和微控制器接口电路三个部分。

芯片本振信号既可由外部有源晶体提供，也可由内部电路提供。由内部电路提供时需外加晶体振荡器和两个负载电容，电容的大小取决于晶体的频率及输入容抗等参数。例如当采用16MHz晶振时，其电容值约为22pF。

射频输入/输出匹配电路主要用来匹配芯片的输入输出阻抗，使其输入输出阻抗为50Ω，同时为芯片内部的PA及LNA提供直流偏置。

PIC18F4620可以通过4线SPI总线(SI、SO、SCLK、CSn)设置CC2420的工作模式，并实现读/写缓存数据，读/写状态寄存器等。通过控制FIFO和FIFOP管脚接口的状态可设置发射/接收缓存器。

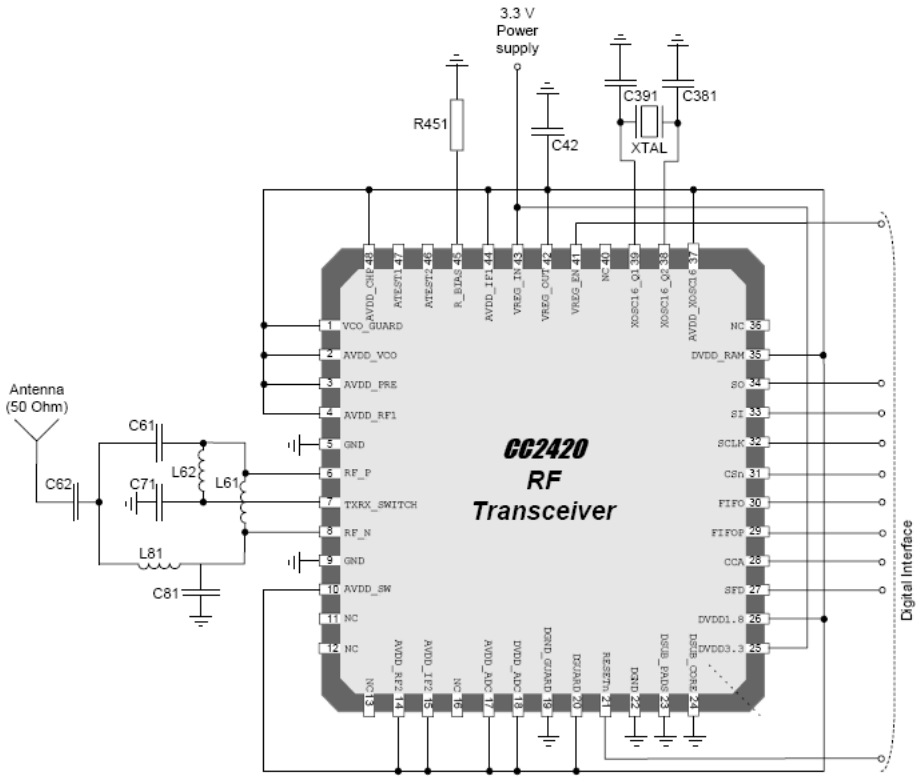

图 2-10 CC2420 芯片的典型应用电路

Figure 2-10 Typical Application Circuit of CC2420

CC2420片内有33个16比特状态设置寄存器，在每个寄存器的读/写周期中，SI总线上共有24比特数据，分别为：1比特RAM/寄存器选择位(0：寄存器，1：RAM)，1比特读/写控制位(0：写，1：读)，6比特地址选择位、16比特数据位。在数据传输过程中CSn必须始终保持低电平。

另外，通过CCA管脚状态的设置可以控制清除通道估计，通过SFD管脚状态的设置可以控制时钟/定时信息的输入。这些接口必须与微处理器的相应管脚相连来实现系统射频功能的控制与管理。CC2420和PIC18F4620的端口链接如表2-1所示。

表 2-1 CC2420 和 PIC18F4620 SPI 端口的连接

Table 2-1 SPI Interface Between CC2420 and PIC18F4620

PIC 单片机 I/O 引脚	RF 收发器引脚
RB0(输入)	CC2420:FIFO
RB1(输入)	CC2420:CCA
RB2(输入)	CC2420:SFD
RB3(输入)	CC2420:FIFOP
RC0(输出)	CC2420:CSn
RC1(输出)	CC2420:VREG_EN
RC2(输出)	CC2420:RESET
RC3(输出)	CC2420:SCK
RC4(输入)	CC2420:SO
RC5(输出)	CC2420:SI

2.4 网络通信协议栈的设计与实现

2.4.1 物理层协议设计

物理层在设计中完全采用了IEEE802.15.4标准的物理层协议。由于CC2420提供了对IEEE802.15.4物理层协议的支持，实际中主要通过对CC2420的寄存器配置来实现物理层的协议功能。CC2420自动把发送或接受的数据帧送入在RAM中的128 字节的缓存区，并进行相应的帧打包和拆包操作，这两个缓存区可以看作物理层和数据链路层交互的通道。

IEEE802.15.4此协议的物理层采用了两个频段：2.4GHz频段和868/915MHz频段物理层。免许可证的2.4GHz ISM频段全世界都有，而868MHz和915MHz的ISM频段分别只在欧洲和北美有。在802.15.4中，总共分配了27个具有三种速率的信道：在2.4GHz频段有16个速率为250k bit/s(或62.5k symbol/s)的信道，在915MHz频段有10个40k bit/s(或40k symbol/s)的信道，在868MHz频段有1个20k bit/s(或20k symbol/s)的信道，如表2-2所示。

表2-2 IEEE802.15.4物理层频段分布

Table 2-2 Frequency Bands and Data Rates

物理层(MHz)	频带宽度(MHz)	传输参数		数据参数		
		速率 kchip/s	调制方式	比特率	码元速率	码元形式
868/915	868-868.6	300	BPSK	20	20	二进制
	902-928	600	BPSK	40	40	二进制
2450	2400－2483.5	2000	O-QPSK	250	62.5	16 进制（正交）

考虑到将来设计的全球化，在我们的设计中，我们采用的全球范围内免许可证的2.4GHz的ISM频段。

2.4.1.1 物理层协议数据单元格式

在物理层传输数据所采用的物理层协议数据单元（PPDU），如图 2-11 所示，主要有三个部分组成：

1) 同步头（SHR），它的作用是使接受端同步并锁定字节流。
2) 物理层头部（PHR），它包含帧长度的信息。
3) 可变长度的负载，它包含 MAC 子层帧。

字节数：4	1	1		长度可变
前导码	帧起始分隔符	帧长度（7bit）	保留位（1bit）	数据包
同步头		物理层头部		物理层数据包

图 2-11 物理层协议数据单元

Figure 2-11 Format of the PPDU

CC2420 通过前导序列从一则消息中得到同步符，因此，前导序列必须由 32 个 0Bit 组成。帧起始分隔符（SFD）是一个 8bit 的序列，用来表明同步序列的结束和数据序列的开始。帧起始分隔符（SFD）必须符合图 2-12 中的格式要求。帧长序列有 7bit，它确定了物理层数据包（PSDU）中所包含的字节数。它的取值范围从 0 到 127。物理层数据包（PSDU）序列包含所要传输的数据。

Bit：0	1	2	3	4	5	6	7
1	1	1	0	0	1	0	1

图 2-12 同步头（SHR）的格式

Figure 2-12 Format of the SFD Field

2.4.1.2 物理层调制方法

2.4GHz 的物理层主要采用了基于 DSSS 方法（16 个状态）的准正交调制技术。传输的数据首先进行比特到码元的变换，再把码元映射成伪随机码片序列，以便传输。经过 DSSS 扩频变换后，码片速率达到 2M chips/s，此码片序列再经过 O-QPSK 调制，每个码片被调制为半个周期的正弦波。码片流通过 I/Q 通道交替传输，两通道延时为半个码片周期。其据的流程如图 2-13 所示。

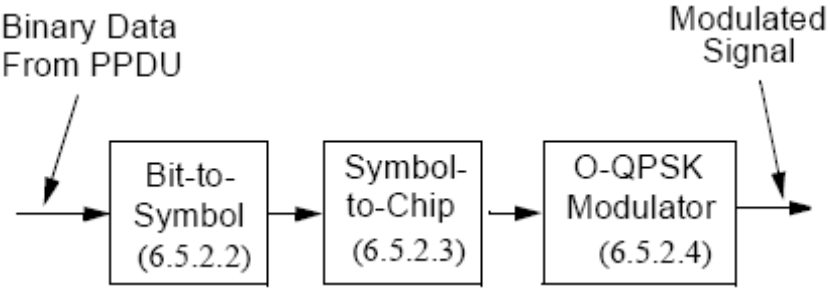

图 2-13 调制和扩频流程

Figure 2-13 Modulation and Spreading Functions

比特到码元的映射：将来自物理层协议数据单元中所有字节进行映射。每个字节的低 4 位映射成一个码元，高 4 位映射成一个码元。其中低 4 位将先进行映射，然后是高 4 位。

码元到码片的转换：每一数据码元被映射成 32 位伪随机码片序列，共有 16 个不同的 32 位码片伪随机序列。码元到码片序列的据铁映射关系如表 2-3 所示。

表 2-3 码元到码片序列的映射关系

Table 2-3 Symbol-to-chip mapping

码元值（10 进制）	码元值（2 进制）	码片值
0	0000	11011001110000110101001000101110
1	1000	11101101100111000011010100100010
2	0100	00101110110110011100001101010010
3	1100	00100010111011011001110000110101
4	0010	01010010000101110110110011100011
5	1010	00110101001000101110110110011100
6	0110	11000011010100100010111011011001
7	1110	10011100001101010010001011101101
8	0001	10001100100101100000011101111011
9	1001	10111000110010010110000001110111
10	0101	01111011100011001001011000000111
11	1101	01110111101110001100100101100000
12	0011	00000111011110111000110010010110
13	1011	01100000011101111011100011001001
14	0111	10010110000001110111101110001100
15	1111	11001001011000000111011110111000

交错正交相移键控（O-QPSK）调制：调制方式采用半正弦脉冲波形的 O-QPSK 调制。O-QPSK 数字调制方式具有较高的带宽利用率和功率利用率，受系统非线性影响小。O-QPSK 是在 QPSK 基础上发展起来的一种恒包络数字调制技术。这里，所谓恒包络技术是指已调波的包络保持为恒定，它与多进制调制是从不同的两个角度来考虑调制技术的。恒包络技术所产生的已调波经过发送带限后，当通过非线性部件时，只产生很小的频谱扩展。这种形式的已调波具有两个主要特点，其一是包络恒定或起伏很小；其二是已调波频谱具有高频快速滚降特性，或者说已调波旁瓣很小，甚至几乎没有旁瓣。

O-QPSK 数字调制方式将同相 I(t)和正交 Q(t)两支路的码流在时间上错开了半个码元周期，如图 2-14 所示。

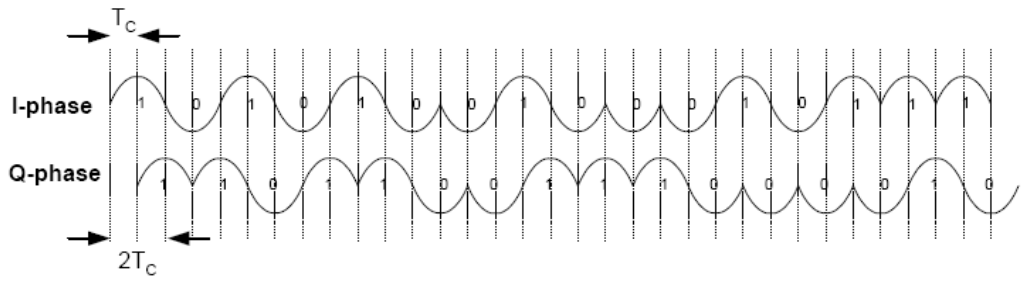

图 2-14 O-QPSK I(t)与 Q(t)的半周期偏移

Figure 2-14 Sample baseband chip sequences with pulse shaping

O-QPSK 信号可表示位：

$$s(t)=\frac{1}{\sqrt{2}}I(t)\cos 2\pi f_c t-\frac{1}{\sqrt{2}}Q(t-T_c)\sin 2\pi f_c t \qquad (4\text{-}1)$$

其中 f_c 为中心频率，正弦和预先载波分量上的比特转移在时间上便宜 T_c 秒。其正弦脉冲波形为：

$$p(t)=\begin{cases}\sin(\pi\dfrac{t}{2T_c}), & 0\le t\le 2T_c\\ 0, & \text{其他}\end{cases} \qquad (4\text{-}2)$$

从表达式可以看出，O-QPSK 趋向育恒包络。

2.4.2 数据链路层协议设计

IEEE802 系列标准把数据链路层分成 LLC（Logical Link Control，逻辑链路控制）和 MAC（Media Access Control，媒介接入控制）两个子层。LLC 子层在 IEEE802.2 标准中定义，为 802 标准系列共用。LLC 子层的主要功能包括：传输可靠性保障和控制；数据包的分段与重组；数据包的顺序传输。而 MAC 子层协议则依赖于各自的物理层。IEEE802.15.4 的 MAC 协议包括以下功能：设备间无线链路的建立、维护和结束；确认模式的帧传送与接收；信道接入控制；帧校验；预留时隙管理；广播信息管理。IEEE802.15.4 的 MAC 层能支持多种 LLC 标准，通过 SSCS（Service-Specific Convergence Sub layer，业务相关的会聚子层）协议承载 IEEE802.2 类型一的 LLC 标准，同时也允许其他 LLC 标准直接使用 IEEE802.15.4 的 MAC 层的服务。我们将重点介绍 MAC 协议的设计与实现。

2.4.2.1 MAC 算法的设计与实现

因为网络节点间的传输速率最大为 250kbps，数据包也较短，所以系统采用非持续的 CSMA 算法（CSMA-CA），即每个节点在发送数据前先侦听信道上其它节点是否在发送数据。如果侦听发现已经有数据在发送，则此节点暂时不发送数据，根据协议的算法延迟一个随机的时间再次侦听信道。一旦发现信道空闲之后，就将数据发送出去。如果反复侦听超过规定次数仍不能成功发送，则认为发送出错，进入出错处理程序。具体流程如图 2-15 所示。

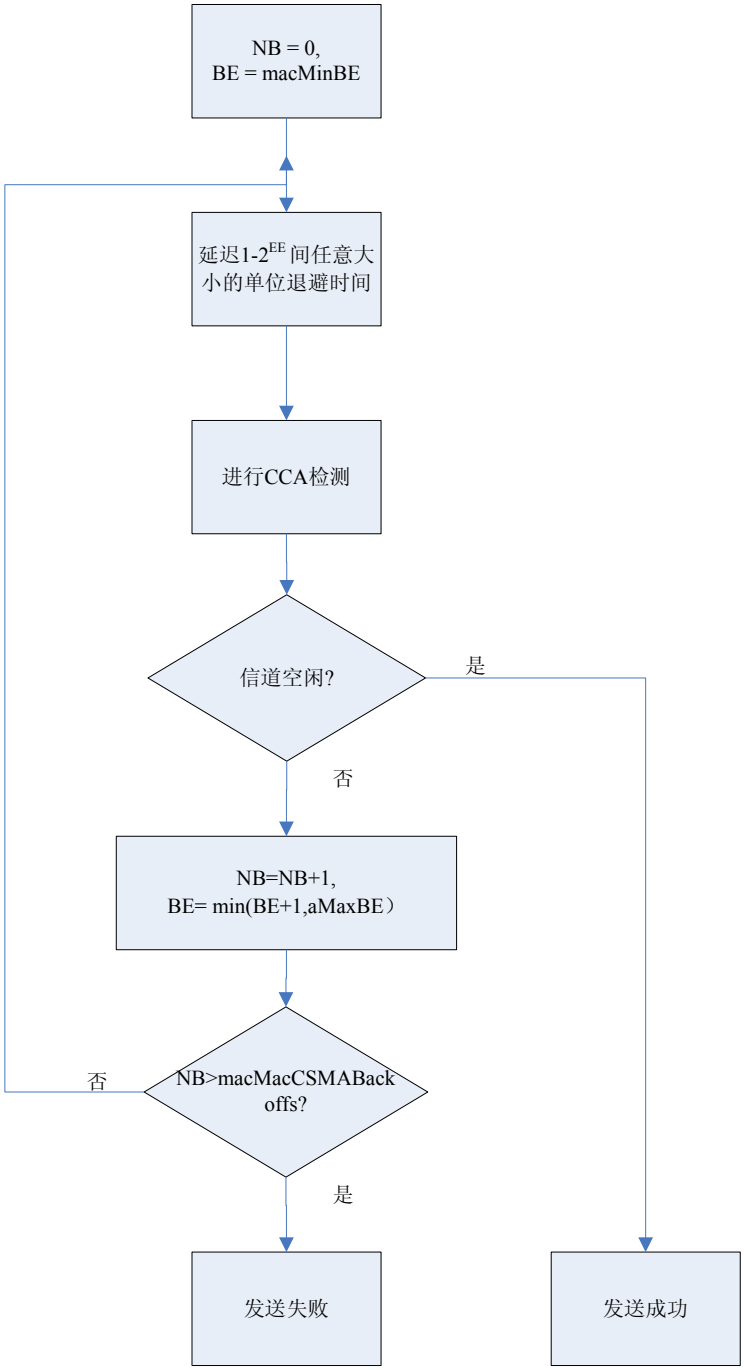

图 2-15 CSMA-CA 算法流程

Figure 2-15 The CSMA-CA algorithm

在 CSMA-CA 算法中，主要有两个关键的变量：

NB：传输冲突时所退避的次数。这个变量在一次传输开始前必须设为 0。

BE：退避指数，即再次尝试访问信道前所要退避的周期数。在一次传输前，BE 的初始化值为 macMinBE。

其中还有几个重要的常量是：

macMinBE：它是 BE 的初始值，在我们所采用的物理层规范 IEEE802.15.4-2003 标准中，它的取值范围为 0-3，显然，如果取 macMinBE 的值为 0，那么 CSMA-CA 冲突处理机制将处于关闭状态。在本系统软件设计中 macMinBE 的取值为 3，这也是 IEEE802.15.4-2003 协议规范中的缺省值。

aMaxBE：它是 BE 的最大值，本系统软件设计中，aMaxBE 值为 5，这也是 IEEE802.15.4-2003 协议规范中所规定的值。

macMaxCSMABackoffs：它是一次传输尝试退避次数的最大值。在我们所采用的物理层规范 IEEE802.15.4-2003 标准中，它的取值范围为 0-3。在本系统软件设计中 macMinBE 的取值为 3，这也是 IEEE802.15.4-2003 协议规范中的缺省值。

2.4.2.2 MAC 帧格式

CC2420 为 IEEE802.15.4 的 MAC 帧格式提供了硬件支持，为此 MAC 帧格式建立在 IEEE802.15.4 标准的 MAC 帧格式基础上。MAC 层协议数据单元（MPDU, MAC Protocol Data Unit）如图 2-16 所示，主要有以下几个部分：

1) MAC头（MHR, MAC Header），它由帧控制域(Frame Control Field)，数据序列号（Data Sequence Number）和地址信息(Address Information)组成。
2) 可变长度的MAC负载，它带有符合帧类型的数据包。
3) MAC尾（MFR, MAC Footer），包括帧校验序列（FCS, Frame Check Sequence）。

<table>
<tr><td>Octets: 2</td><td>1</td><td>0/2</td><td>0/2/8</td><td>0/2</td><td>0/2/8</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame control</td><td rowspan="2">Sequence number</td><td>Destination PAN identifier</td><td>Destination address</td><td>Source PAN identifier</td><td>Source address</td><td rowspan="2">Frame payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td></tr>
<tr><td colspan="6">MHR</td><td>MAC payload</td><td>MFR</td></tr>
</table>

图2-16 MAC层协议数据单元

Figure 2-16 General MAC frame format

MAC头结构：帧控制域有两个字节，定义了帧的类型，地址域，和一些控制标志位。它的格式必须按图2-17所示。

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame type	Security enabled	Frame pending	Ack. request	Intra-PAN	Reserved	Dest. addressing mode	Reserved	Source addressing mode

图2-17 帧控制域的格式

Figure 2-17 Format of the frame control field

其中，帧类型这3个Bits必须按表2-4中列出非保留值。

表2-4 帧类型的取值范围

Table 2-4 Values of the frame type subfield

Frame type value $b_2\,b_1\,b_0$	Description
000	Beacon
001	Data
010	Acknowledgment
011	MAC command
100—111	Reserved

还有一个比较重要的是序列号域，有8 bits长，为MAC唯一的一个标识序号。

帧校验序列（FCS，Frame Check Sequence）：包含一个两个字节的循环冗余校验码。

FCS是对帧进行循环冗余校验（CRC, Cycled Redundant Check）,使接受方可以检测数据包是否出错，协助完成协议层的差错控制。CRC的校验生成多项式为：

$$G(x) = x^{16} + x^{12} + x^5 + 1 \qquad (4\text{-}3)$$

通过对寄存器的配置，FCS值可由CC2420自动产生。其自动加入RF序列中，不放进发送FIFO中。硬件实现描述如图2-18所示。

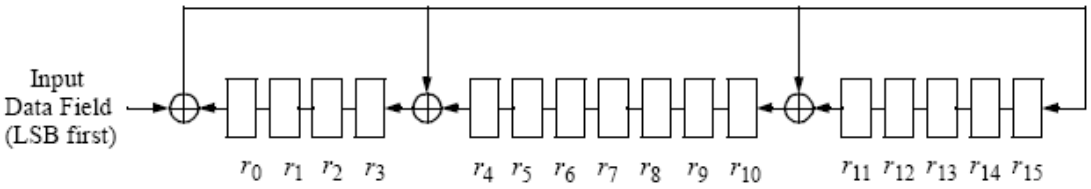

图2-18 帧校验序列的生成

Figure 2-18 Typical FCS implementations

其主要由16级移位寄存器和三个加法器组成。首先将各寄存器初始化为1，信息位随时钟移入。当信息位全部输入后，从寄存器组输出的即为CRC检验和。移位前信息位与bit0异或相加产生临时位，其中bit15移入临时位，bit10，bit3还要加上临时位。当全部信息位输入完成后，从寄存器组取出它们的值，这就是CRC码。电路校验的具体实现过程如下：

1) 由生成多项式初始化16位的各移位寄存器为1；
2) 信息系列从低位开始逐一和移位寄存器的第0位相异或，异或后的结果为y；
3) 寄存器整体向右移移位，将y移入移位寄存器的第15位。移位寄存器中的bit1，bit4分别同y相异或，结果放入bit10与bit3中，这时寄存器中的值就是完成移位CRC校验后的值。
4) 重复1-4步，直到所有的信息序列全部顺序移入移位寄存器，这时移位寄存器的值即为信息序列的16位CRC校验和，在接受端对该信息序列进行相同的CRC检验计算，若CRC校验和相等，则说明此数据传输正确。否则此次数据传输出错。

2.5 语音采集设计与实现

2.5.1 声电转换及 AD 前级匹配电路的设计与实现

我选用的麦克风的特性是，它的电阻值随外界声音震动而改变。因此只要加一个直流偏置可以取得电压随声音变化的波形。用大电容割直后，用运放进行放大并调整到单片机 A/D 所能接受的范围(0-3.3V)。考虑到整个系统的供电，采用单电源高速运放 LM358。为了能够撑满 A/D 的量程，中间地选取在 1.6V。具体电路图如图 2-19 所示。

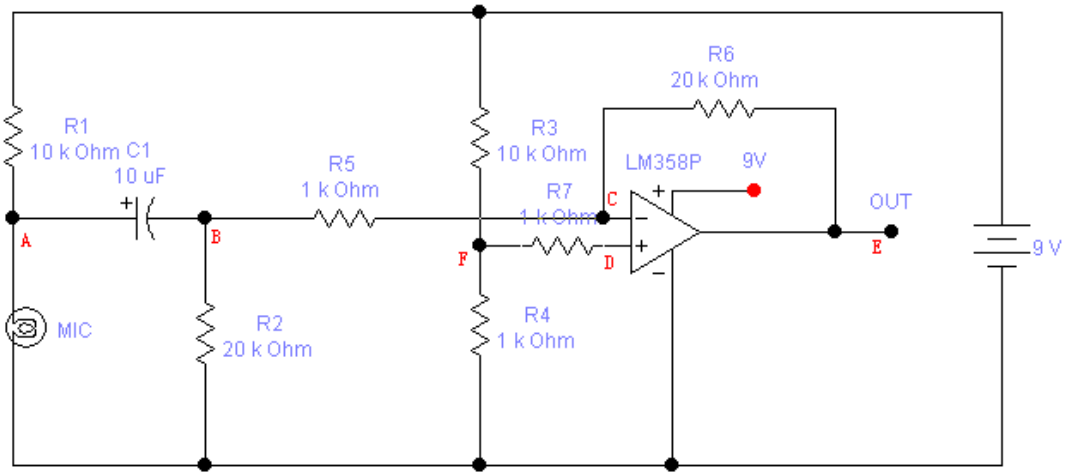

图 2-19 声电转换及 A/D 匹配电路

Figure 2-19 Circuit Designed for A/D Transfer

2.5.2 PIC 单片机 A/D 转换的软件实现

系统微控制器PIC18F4620有13路10位模数（A/D）转换通道。模拟输入对采样保持电容充电，采样保持电容的输出是A/D转换器的输入。其中A/D转换器采用逐次逼近法将模拟电平转换为10位数字结果。模拟参考电压（正电压和负电压）可通过软件选择为器件的电源电压（AVDD、AVss）或AN3/VREF+和AN2/VREF-引脚上的电压。并且A/D转换器具备在器件休眠模式下仍可进行转换的独特特性。

A/D模块包含5个寄存器，分别为：

1) A/D结果高位寄存器（ADRESH）
2) A/D结果低位寄存器（ADRESL）
3) A/D控制寄存器0（ADCON0）
4) A/D控制寄存器1（ADCON1）
5) A/D控制寄存器2（ADCON2）

A/D模块的输入通道由ADCON0寄存器选择。寄存器ADCON0如图2-20所示：

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

图2-20 ADCON0 寄存器

Figure 2-20 ADCON0 Register

端口引脚的功能和A/D模块的参考电压由ADCON1配置。这些端口引脚可配置为模拟输入（其中AN3和AN2也可作为参考电压）或数字I/O。寄存器ADCON1如图2-21所示：

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

图2-21 ADCON1 寄存器

Figure 2-21 ADCON1 Register

ADCON2用于选择A/D转换时钟源以及A/D转换结果的格式。寄存器ADCON2如图2-22所示：

R/W-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	—	ADCS2	ADCS1	ADCS0
bit 7							bit 0

图2-22 ADCON2 寄存器

Figure 2-22 ADCON2 Register

A/D转换的10位结果保存在ADRESH：ADRESL寄存器中。当A/D转换完成后，转换结果被装入A/D结果寄存器对（ADRESH：ADRESL）中，GO/DONE位（ADCON0）清零，而A/D中断标志位ADIF置位。

A/D转换按照以下步骤进行，如图2-23：

1）配置A/D模块：

- 配置模拟引脚 / 参考电压 / 数字I/O （ADCON1）
- 选择A/D输入通道（ADCON0）
- 选择A/D转换时钟（ADCON0）
- 开启A/D模块（ADCON0）

2）需要时，配置A/D中断：

- 将ADIF位清零
- 将ADIE位置1
- 将ADIP位置1/清零
- 将GIE/GIEH或PEIE/GIEL位置1

3）等待所需的采集时间。

4）启动A/D转换：

- 将GO/DONE位置1（ADCON0）

5）等待A/D转换完成，通过以下两种方法之一可判断转换是否完成：

- 查询GO/DONE位是否被清零或ADIF位是否被置1，或
- 等待A/D中断

6）读取A/D结果寄存器对（ADRESH：ADRESL）：需要时将ADIF位清零。

7）要进行下一个转换，根据要求转入步骤1或步骤2。

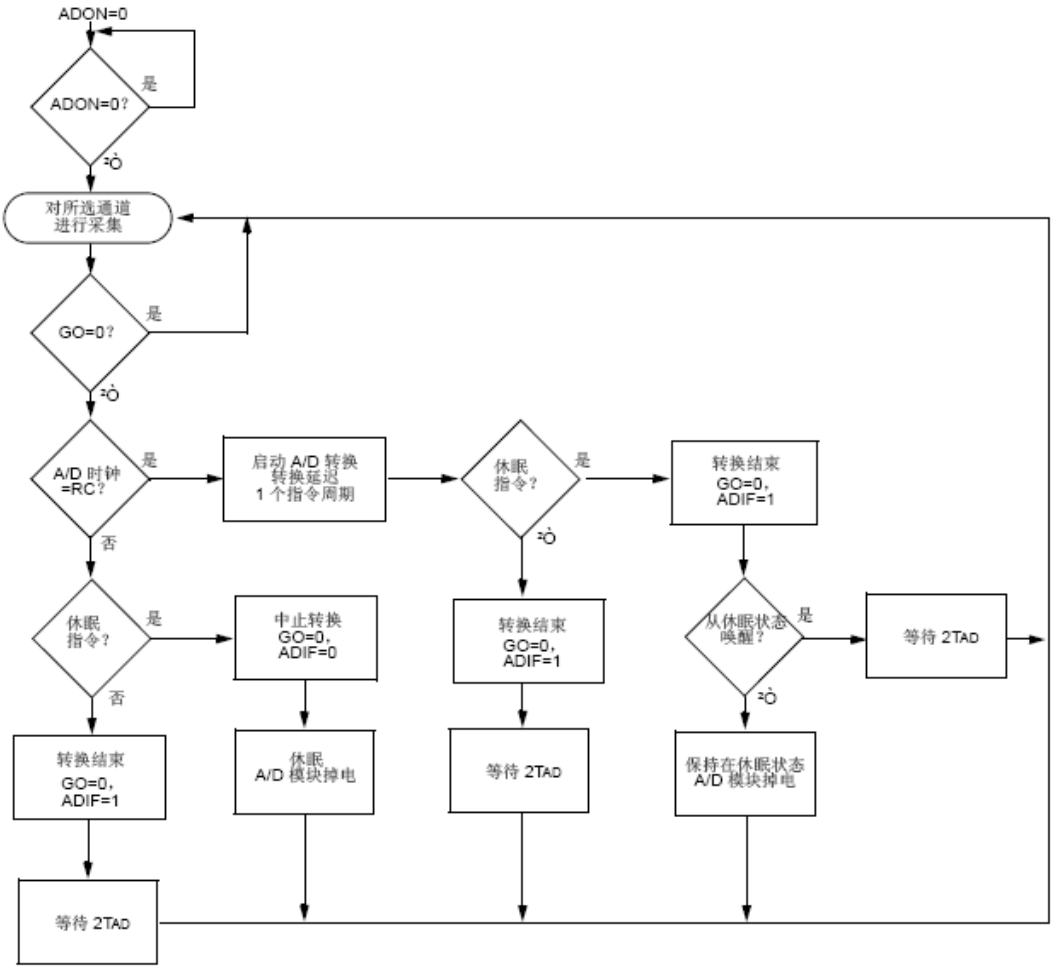

图2-23 A/D转换流程图

Figure 2-23 Flowchart of A/D Operation

程序中A/D转换的程序实现：

```
for(i=0;i<PackageLength;i++)
{

    ConvertADC();
    while(BusyADC());
    result=ReadADC();

    AdData[i]=result>>2;

    for(j=0;j<100;j++)
    {
    CLRWDT();
}
```

配置模拟端口引脚时要注意以下几点：

对A/D端口引脚的操作由ADCON1和TRIS寄存器控制。若希望端口引脚为模拟输入，则必须将相应的TRIS位置1（输入）。如果TRIS位被清零（输出），则此时转换的是数字输出电平（VOH或VOL）。在器件复位后，复用为模拟输入的引脚将被配置为模拟输入，相应的TRIS位被置1。

A/D 转换与 CHS2：CHS0 位及 TRIS 位的状态无关。

2.6 传感器网络的对外接口

2.6.1 UART 通讯协议

串行数据通讯时，数据按位传送，任何时候线上仅有一位数据。因此收、发双方必须同步，以从二进制位流中正确地读出每一位数据。异步串行通讯中，收、发方的同步不采用时钟线来进行，而是由通讯双方约定一个波特率，每一个传送单元通过一个“起始位”来同步。当接收方监测到一个有效起始位，便按照约定的波特率的一个倍频(例如16倍频)对数据进行采样接收。由于每一个传送单元的位数较少(通常不超过11位)，而接收采样的频率要高于通讯波特率，即使收发双方的时基存在一定误差，仍然可以保证准确的通讯。

在空闲状态，传送线为逻辑“1”状态。数据的传送总是以一个“起始位”开始的，接着是要传送的若干数据位，低位先行，最后是一个“1”状态的“停止位”。例如在文档中用“9600 N.8.1”描述一个UART，就表示UART使用9600bps的波特率，帧格式为一个起始位、8个数据位、一个停止位。

当接收方检测到一个“1”向“0”的跳变，便视为可能的起始位。起始位被确认后，依次接收数据位和停止位，若检测不到正确的停止位，可视为传送出错而放弃。

2.6.2 PIC 单片机 UART 通讯的软件实现

PIC18F4620的UART异步发送器的核心是发送移位寄存器（TSR），图2-24所示。移位寄存器从发送缓冲器TXREG获取要发送的数据。TXREG寄存器由软件写入数据。前一次载入的停止位发送出去后，TSR寄存器才会载入数据。停止位一发送出去，TXREG中的新数据（如有）就会立即被载入TSR。一旦把TXREG中的数据送入TSR寄存器（在一个TCY周期内发生），则TXREG寄存器为空，发送中断标志位TXIF被置位。可以通过将TXIE使能位置位或清零来允许或禁止这个中断。不管TXIE位的状态如何，TXIF标志位都会被置位，且不能用软件清零。只有把新数据写入TXREG寄存器后，TXIF位才会复位。TXIF标志位表示TXREG寄存器的状态，而TRMT位（TXSTA寄存器）则表示TSR寄存器的状态。TRMT状态位是一个只读位，当TSR寄存器为空时被置位。任何中断逻辑对TRMT位都没有影响，所以要确定TSR寄存器是否为空，用户就必须对此位进行轮询。

名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	POR、BOR 时的值	所有其他 复位时的值
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 0000x	0000 000x
SPBRG	波特率发生器寄存器								0000 0000	0000 0000

图2-24 与波特率发生有关的寄存器

Figure 2-24 Registers Associated with Baud Rate Generator

将TXEN位（TXSTA寄存器）置位就可以使能发送，但是只有在TXREG寄存器装入数据和波特率发生器（BRG）产生移位时钟之后才能进行数据发送。也可以先载入TXREG寄存器，再置位TXEN启动发送。通常，在第一次开始发送时，TSR寄存器为空，因此送到TXREG寄存器的数据会立即被送到TSR寄存器，导致TXREG寄存器为空，因此可以进行连续的背靠背发送，USART发送器框图如图2-25所示。与波特率发生有关的寄存器如图2-25所所示。在发送过程中将TXEN位清零将导致发送中止，并复位发送器，同时TX/CK引脚会恢复到高阻状态。要选择9位发送模式，TX9位（TXSTA寄存器）应被置位，而且第9位应当写入TX9D位（TXSTA寄存器）。必须先写第9位数据，然后将8位数据写入TXREG寄存器。这是因为如果TSR寄存器为空，向TXREG寄存器写数据会导致数据立即送入TSR寄存器。在这种情况下，装入TSR寄存器的第9位数据可能是错误的。

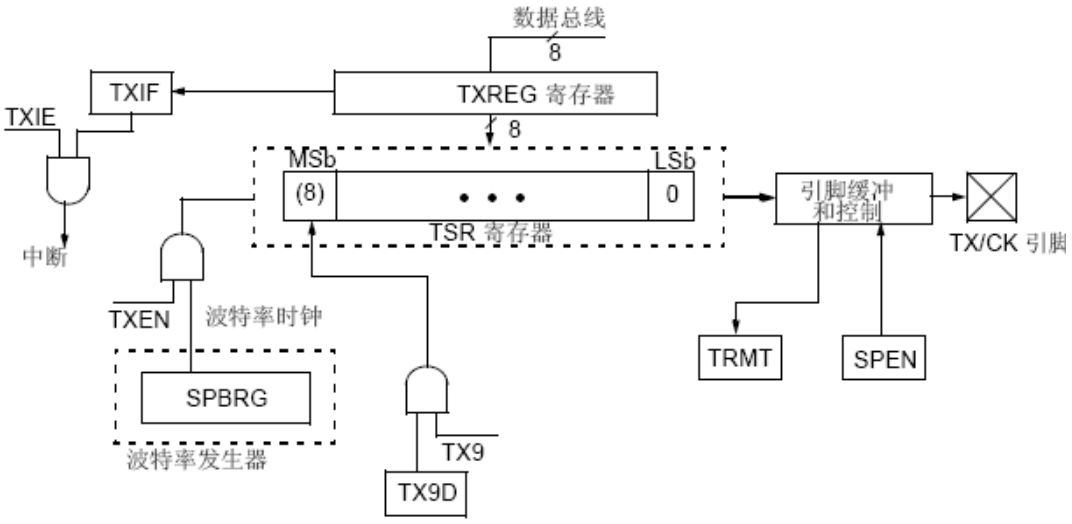

图 2-25 UART 发送框图

Figure 2-25 USART Transmit Block Diagram

建立异步发送模式的步骤：

(1) 选择合适的波特率对SPBRG寄存器进行初始化。

PIC18F4620波特率发生器的计数器有8位计数模式和16位计数模式，BRG16置1时，波特率发生器工作在16位计数模式，置0时波特率发生器工作在8位计数模式。16位工作模式可以使波特率的精度进一步提高。而且有低速波特率和高速波特率两种模式，BRGH置1时，波特率发生器工作在高速模式，BRGH置0时，波特率发生器工作在低速模式。

我们的系统工作在115200bps，单片机串口最大速率可以达到500kbps，但是由于接收端E-BOX的串口允许的最大置为115200bps，限制了传感器网络对外传输数据的速率。

为了提高单片机串口的波特率的精度，我们将波特率发生器设为16位计数，高速模式（BRG16=1，BRGH=1）按表2-5所示

$$\text{Baud Rate} = \text{Fosc}/[4(n+1)]$$

带入Baud Rate = 115200bps Fosc = 16000000　求得n = 34 将n = 34带回，得Baud Rate = 114285bps 误差为0.8%，符合UART得通信规范。

表2-5 波特率计算公式

Table 2-5 Baud Rates Formula for Asynchronous Mode

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n + 1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n + 1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n + 1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

(2) 通过将SYNC位清零并将SPEN位置位，使能异步串行端口。

(3) 通过置TXEN位使能发送，同时也置TXIF位。

(4) 把数据载入TXREG寄存器（开始发送）。

2.7 传感器网络节点软件设计

2.7.1 协调器节点

协调器在本系统中起着协调和数据传输中转的作用。在网络组建时，将建立绑定表。组建网络后接收传感器网的三个设备节点传来得数据，并将其传输到嵌入式设备 E-BOX。图 2-26 中为协调器节点软件流程图。

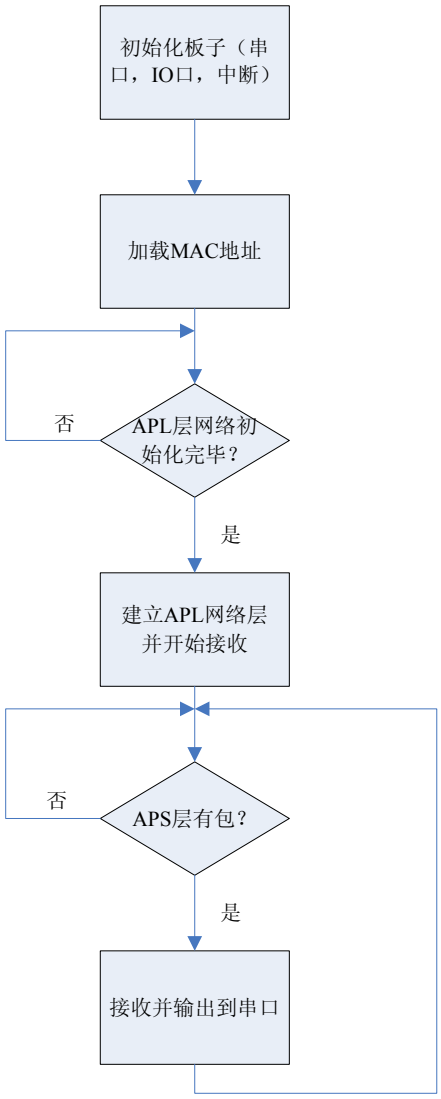

图 2-26 协调器节点软件流程图

Figure 2-26 Flowchart for Software Design of Coordinator

首先初始化板子的硬件设备，加载 MAC 地址，并等待 APL 层网络初始化完毕：

```
InitializeBoard();
TickInit();
SetMAC();
APLInit();      //该函数初始化所有的协议栈模块。同时还初始化 APL 状态机。
APLEnable();//该宏用于使能协议栈模块和 RF 收发器。
APLNetworkInit(); //该宏启动新网络的初始化
while(1)
{
    CLRWDT();
    APLTask();//APLTask 是个协同任务函数，按顺序调用每一个协议栈模块任务
              函数。调用该函数使协议栈获取并处理进入的数据包。
    if(APLIsNetworkInitComplete()) //该宏执行网络初始化状态机并指示网络初始
                                   化是否完成。
    break;
}
```

接着建立 APL 网络层：

```
APLNetworkForm();          //该宏指示网络层在当前的信道上组建新的网络。
APLPermitAssociation();//该宏允许终端设备关联到网络。
ReceiverInit();
```

最后进入程序的主循环，开始接收数据包并发送到串口：

```
while(1)
{
    CLRWDT();
    APLTask();
    ReceiverTask();
}
```

在 ReceiverTask()函数中，首先设定用于接收的端点，如果 APS 层接收数据包，则将数据存放到数组 d[]中，并发送到串口上：

```
APSSetEP(hREC);
if( APSIsGetReady() )
{
    APSGet();
    len=APSGet(); //取得数据包的长度
    APSGetArray(d,len); //将数据接收到数组 d[]中
    for(i=0;i<len;i++)
    {
        ConsolePut(d[i]);   //发送到串口
    }
    ConsolePut(' ');
    APSDiscardRx();
}
```

2.7.2 设备节点

设备节点在本系统中起着数据采集和传输作用。在网络组建时，与协调器建立绑定关系。组建网络后进行 AD 转换采集数据，并发送到从协调器节点，并得到确认。图 2-27 中为协调器节点软件流程图。

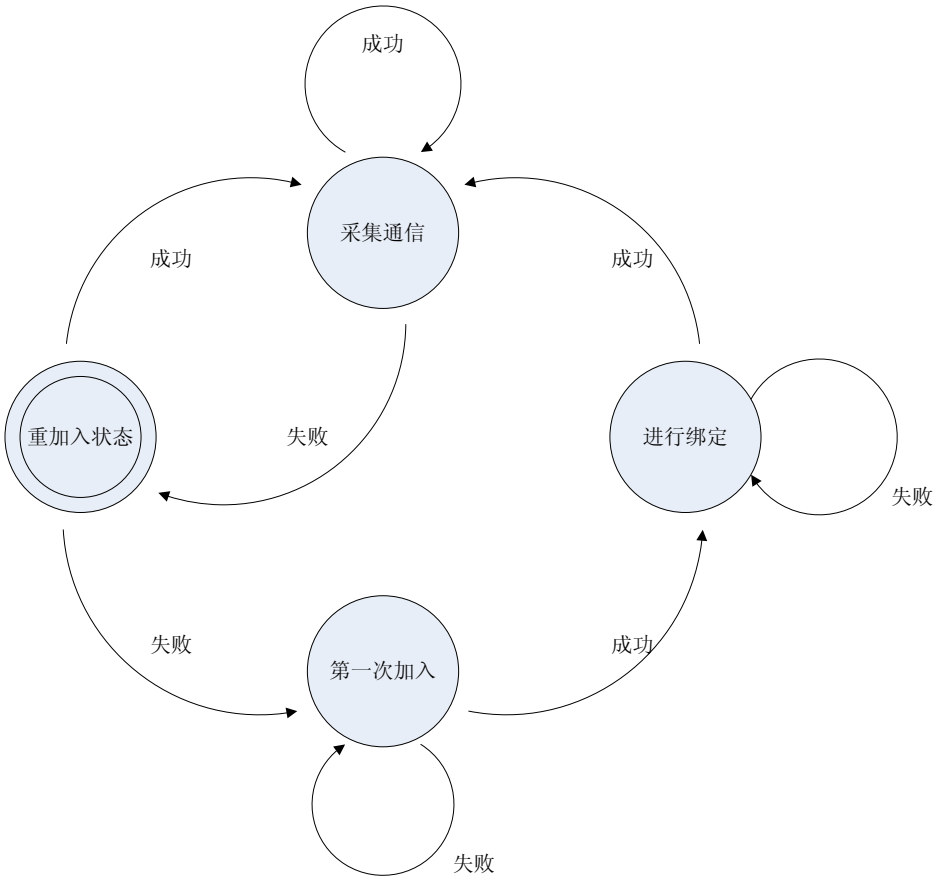

图 2-27 设备节点软件流程图

Figure 2-27 Flowchart for Software Design of Device Node

在设备节点的软件设计中，我们采用了创新性的状态机结构，使整个系统资源得到了最大的利用。整个系统分为重加入状态、第一次加入状态、进行绑定状态、采集通信状态。开始时系统将进入重加入状态，如果重加入成功，则进入采集通信传输状态；如果重试几次失败，则认为节点将加入一个新的网络，将进入第一次加入的状态。进入第一次加入状态后，如果加入成功，则进入绑定状态；如果失败，则再次尝试进行第一次加入。进入绑定状态后，如果绑定成功，则进入采集通信状态；如果失败，将继续停留在进行绑定状态。进入采集通信状态后，如果数据传输成功，则进入下一次数据采集通信；如果传输失败，说明网络连接丢失，进入重加入状态。

2.8 本章小结

我们的无线传感器网络实现了我们的设计要求，三个节点能够各以 1.6kHz 的采样速率对声音进行采样，并将数据传送给嵌入式系统设备，而且采样率还可以进一步提高。

无线传感器网络是我们整个设计的重点，也是整个系统的难点。因此在整个设计和实现过程中，遇到并解决了很多的困难：

首先是软件整体构架的问题，因为无线信道可靠性差，因此软件设计时必须考虑这些问题，不然系统会莫名其妙的死机。我们在软件设计时，除了采用了状态机，保证通信质量外，还充分利用到系统的看门狗功能，当设备节点多次发送数据包而等不到协调器节点的应答时，看门狗可以使系统自动重启，跳过这个死循环，重新建立与协调器的连接，大大增强了系统的鲁棒性。

其次是传输时数据包长对三个设备节点协调通信的影响。因为我们的设备节点是先采集 n 个数据，然后在将 n 个数据打包发送，而采集时设备节点并不占用信道，因此 n 的取值越大越好。但是 n 的取值的变大会导致节点一次发送占用信道时间过长，从而导致其他节点总是侦听到信道忙而产生发送失败，因此 n 的取值并非越大越好。经过我们反复实验尝试，最终选择了 n=50 作为系统的数据包的长度，达到了系统的最优。

最后讨论一下串口通信对整个系统的影响。开始调试通信时，系统采用很低的采样率，因而信道较空闲。当我们逐渐提高系统的采样率后，发现虽然有时信道不是很忙，但是三个设备节点很难协调通信的情况。最后我们发现，因为协调器节点为单线工作程，在串口通信时不侦听数据包，而我们协调器的串口波特率太低，只有 9600bps，自然成为了系统传输的瓶颈。因此我们应采用尽可能高的波特率进行数据传输，占用尽可能少的资源。虽然单片机支持 500kbps 的波特率，因为嵌入式设备 E-Box 的串口最大波特率为 115200bps，因而最后协调器的串口采用 115200bps 的波特率。

3. 基于 E-Box 的嵌入式系统设计

3.1 E-Box 嵌入式系统简介

E-Box 在成个系统中起着将传感器网络采集的声音数据，以及摄像头采集的视频数据进行远程网络传输的作用。E-Box 的串口接收传感器网络传输来的数据，USB 接口接收摄像头传输来的数据，然后通过以太网卡接口，用 TCP/IP 协议将声音和图像数据传输到远端的服务器。应用软件界面如图 3-1 所示。

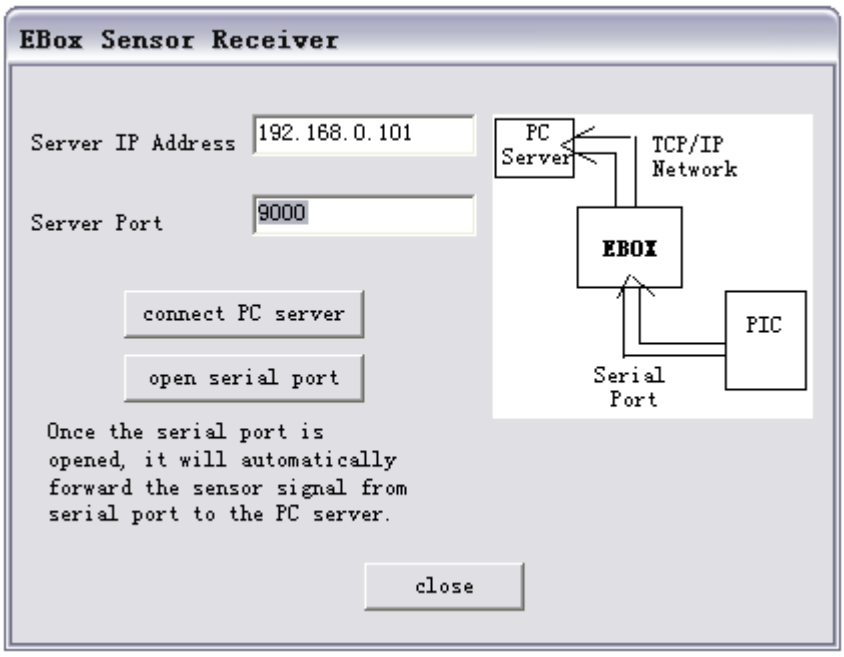

图 3-1 嵌入式应用软件界面

Figure 3-1 Graphic interface of the embedded system application software

3.1.1 E-Box 硬件系统简介

E-Box 是 ICOP 科技有限公司生产的，基于 Vortex86 系列 CPU 的嵌入式设备。作为系统的核心 Vortex86 系列 CPU，是一款性价比极高的 SoC 芯片。不仅集成了 x86 处理器，而且提供了片内的南北桥。以及 USB 接口，显示器接口，以及内存接口。整个 E-BOX 的系统结构框图如图 3-2 所示。

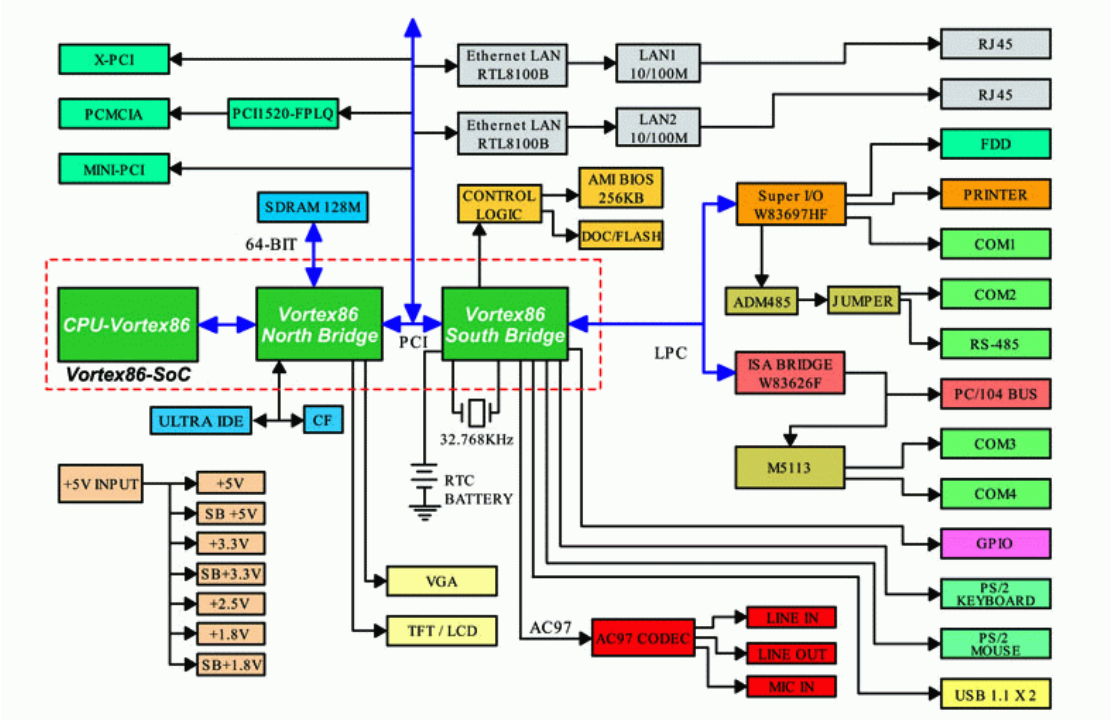

图 3-2 E-Box 系统结构图

Figure 3-2 Block Diagram of E-Box

它的主要特性参数如下：

(1) CPU 主频为 200MHz，以及锂电池供电的实时时钟；

(2) AMI BIOS 支持；

(3) 128MB SDRAM；

(4) 一个 IDE 接口，一个 25 针并口，串口，三个 USB 接口；

(5) 共享显存的片内显示适配器，最高支持 1920×1440 真彩色；

(6) Realtek 8100B 10/100M 以太网卡；

(7) AC97 音频编解码器；

3.1.2 Windows CE.Net 操作系统简介

Windows CE.net 是微软公司在嵌入式操作系统市场上的一个重要产品。Windows CE.net 是一个 32 位，多线程，多任务的操作系统。它的体系结构采用独立于通常的程序设计语言并且和 Windows 兼容的 API 的方式。这样就保证了 Windows CE.net 的组件化和 ROM 化，充分适应有限的存储空间和各种不同芯片的要求。而且 Windows CE.net 是模块型的操作系统，可以选择、组合和配置它的模块和组件来创建用户版的操作系统。

从开发角度看 Windows CE.net，它有一下几个优点：

(1) 定制系统内核

Windows CE.net 提供了一个工具 Platform Builder。是 Windows CE.net 的主要集成开发环境，通过这个可以方便的根据不同的硬件，定制、裁剪出符合不同系统要求的 Windows CE.net 操作系统。

(2) 开发驱动程序

用 Platform Builder 以及 Embedded Visual C++可以方便的编写 Windows CE.net 驱动程序。这对我们编写摄像头驱动有很大的帮助。

(3) 导出 SDK

定制操作系统内核后，如果需要则可以通过 Platform Builder 生成自己的 SDK，生成的 SDK 可以很容易安装到其他的开发环境中去。这样大大提高了我们系统的跨平台性，和可移植性。

(4) 编写应用软件

Windows CE.net 支持 Microsoft .Net Framework，因此开发应用软件十分方便和快速，并且用 Microsoft .Net Framework 编写的软件，可以在 Windows 平台上运行，这就大大降低了软件的调试难度，尤其是通信方面。

(5) 源代码的共享

Windows CE.net 是一个有限开放源代码的软件，Microsoft 对驱动程序，应用软件等均提供了一定数量的源代码，以及开发文档。有了 Microsoft 的强大的技术支持，开发会少走一些弯路。例如我们的摄像头驱动就由微软公司提供了源代码。

基于以上优势，我们选取 Windows CE.net 作为我们的嵌入式操作系统。

3.1.3 Platform Builder 和 Embedded Visual C++开发环境简介

3.1.3.1 Platform Builder 开发环境简介

Platform Builder 是微软提供给 Windows CE.net 开发人员进行基于 Windows CE.net 平台下嵌入式操作系统定制的集成开发环境。它提供了所有进行设计、创建、编译、测试和调试 Windows CE.net 操作系统平台的工具。它运行在 Windows 下，开发人员可以通过交互式的环境来设计和定制内核、选择系统特性，然后进行编译和调试，如图 3-3 所示。同时，开发人员还可以利用 Platform Builder 来进行驱动程序开发和应用程序项目的开发等。

Platform Builder 的主要开发特性有：

(1) 平台开发向导（Platform Wizard）和 BSP 开发向导（BSP Wizard）。例如我们就可以利用 ICOP 的 BSP 进行开发，极大的提高了我们的开发效率。

(2) 特性目录，操作系统可选特性均在特性目录中列出，开发人员可以选择相应的特性来定制操作系统。

(3) 内核调试器，可以对自定义的操作系统映象进行调试，并且向用户提供有关映象性能的信息。

(4) 远程工具，可以执行同基于 Windows CE.net 的目标设备有关的各种调试任务和信息收集任务。

(5) 仿真器，通过硬件仿真加速和简化了系统的开发，使我们可以在 PC 上对平台和应用程序进行测试，大大简化了系统开发流程，缩短了开发时间。

我们在 ICOP 的 BSP 包上对系统进行了开发，并量身订做了自己的 Windows CE.net 操作系统，最终编译出的 Windows CE.net 操作系统映象，只有 14MB 大小。

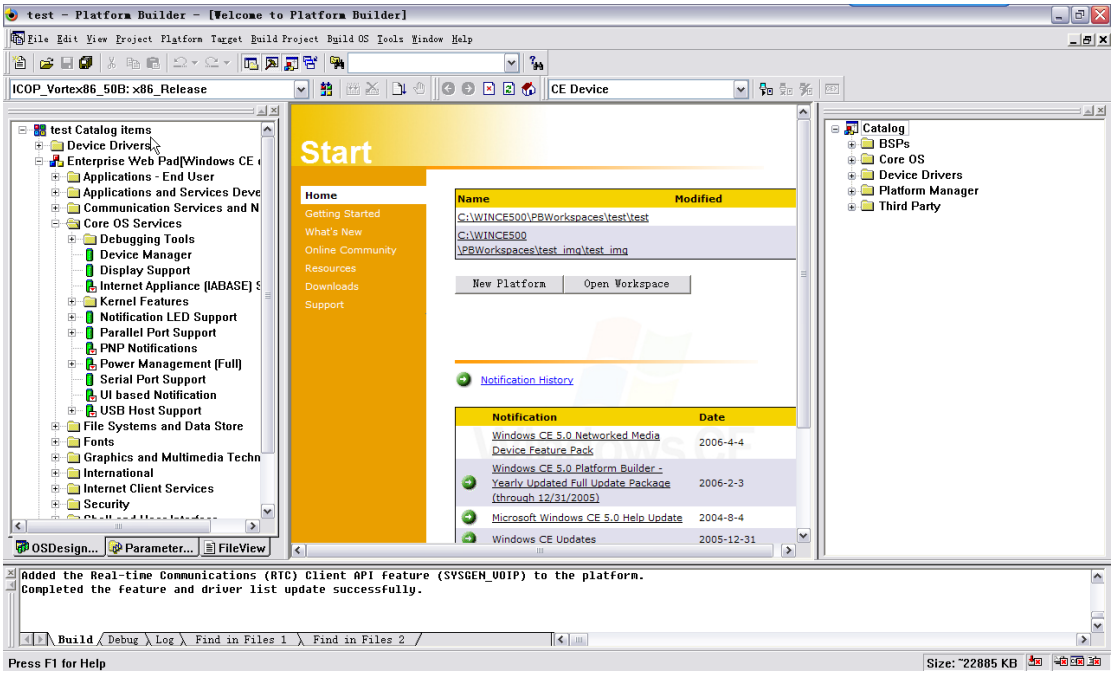

图 3-3 Platform Builder 开发界面

Figure 3-3 Develop Interface of Platform Builder

3.1.3.2 Embedded Visual C++开发环境简介

编写在 Windows CE.net 下运行的应用程序，需要使用专门用于 Windows CE.net 的开发工具。现在应用最广泛的开发工具就是 Embedded Visual C++，如图 3-4 所示。因为嵌入式系统资源有限，而且 C++编译器效率高，性能好，编译出的应用程序结构紧凑。虽然用.Net Framework 相比 Embedded Visual C++编程简便，但是遇到对底层的硬件的操作，如驱动的编写就束手无策了。因此，Embedded Visual C++仍是 Windows CE.net 下开发必不可少的工具。

我们用整个嵌入式设备软件的编写采用了 Embedded Visual C++与.Net Framework 相结合的办法，Embedded Visual C++为.Net Framework 提供下层驱动和动态链接库。比如串口驱动和摄像头驱动。达到了最高的编程效率。

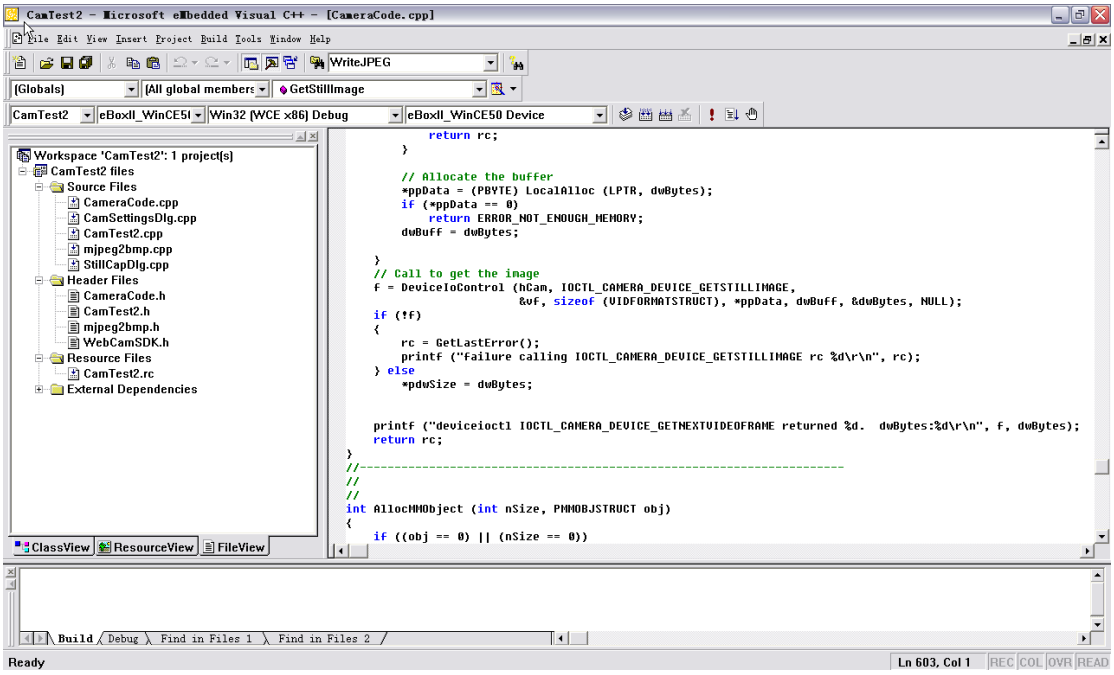

图 3-4 Embedded Visual C++开发界面

Figure 3-4 Develop Interface of Embedded Visual C++

3.2 传感器网数据的接收和网络传输

E-Box 通过串口接收传感器网络中协调器传送来的语音数据，当 E-Box 收到协调器发来的一个包（50Byte）后，将启动 TCP/IP 网络传输协议，通过以太网将数据包发送出去，然后继续接收串口的数据。因为传感器网络送来的数据量非常大速度也非常快，波特率为 115200bps，因此，E-Box 上串口数据接收以及网络传输成为整个系统设计的难点。当进行网络传输时，必须有足够的缓存用于串口的传输（实践证明至少要 4KB）。

在 E-Box 串口的软件开发上，我们采用了应用软件和动态链接库独立的开发方式，即用 Embedded Visual C++开发 E-Box 串口的底层驱动和动态链接库，而用 Microsoft VS.Net 开发上层应用软件，在对串口的操作时调用动态链接库。这样不仅可以极大的提高开发效率，而且避开了串口与网络进程间通信的问题。

3.2.1 Windows CE.NET 下的串行通信开发

串行端口在 Windows CE.NET 下属于流接口设备，他是串行设备接口常规 I/O 驱动程序的调用与通信相关的具体函数的结合。串行设备被视为用于打开、关闭和读写串行端口的常规的可安装的流设备。

(1) 打开和关闭串行端口：

在所有的流设备驱动程序中，均使用 CreateFile 来打开串行端口设备，如果这个端口不存在，CreateFile 返回 ERROR_FILE_NOT_FOUND。因此，用户指定的端口必须是存在并且可用的，而且要遵循 Windows CE.NET 流接口设备驱动程序的命名规则，即 COM 后接要打开的端口号再紧跟一个冒号。

HANDLE hPort = CreateFile (_T("COM1:"), GENERIC_READ|GENERIC_WRITE, 0,NULL, OPEN_EXISTING,0,NULL);

关闭串行端口比较简单，调用 CloseHandle 函数就能关闭一个打开的串行端口。CloseHandle 只有一个参数，即调用 CreateFile 函数打开端口时返回的句柄，方法如下：

```
CloseHandle (hPort);
```

(2) 读写串行端口

正如使用 CreateFile 打开串行端口一样，可以使用 ReadFile 和 WriteFile 函数来读写串行端口。假设已经调用 CreateFile 成功的打开了串行端口，那么只需调用 ReadFile 即可从串行端口读取数据：

```
ReadFile(hPort,buf,dwLength,&dwLength,NULL);
```

由于 Windows CE.NET 不支持重叠的 I/O 操作，所以第 5 个参数必须设置为 NULL。写串行端口也很简单。调用过程如下：

```
DWORD ByteLenWrited=0;
if(hPort==INVALID_HANDLE_VALUE||buf==NULL) return 0;
else
    if(!WriteFile(hPort,buf,ByteLenToWrite*sizeof(BYTE),&ByteLenWrited,NULL))
    return 0;
    else return ByteLenWrited;
```

(3) 配置串行端口

读和写串行端口比较简单，但是还需要对端口配置好正确的波特率、字符长度、奇偶校验、传输模式等，端口才能正确通信。可以调用 I/O 设备控制（IOCTL）来配置串行驱动程序，但此操作需要一些底层的知识，并且要有相应的“嵌入工具包”（ETK），而 SDK 不能实现该操作。除此之外，还有一种更简单的方法，就是调用 GetCommState 和 SetCommState 函数配置串行端口。由于配置端口的 DCB 结构内容较多，所以使用起来比较麻烦。错误的初始化 DCB 结构是配置串行端口常见的问题。如果一个串行通信函数没有产生预期的效果，那么错误很可能是在 DCB 结构体的赋值。在与单片机实现串行通信的时候，由于只用到了 RS 232 串行口的 RXD，TXD 和 GND 三个引脚，而其他的引脚均舍弃不用，所以 DCB 的成员变量应该如下设置，否则不能正常通信：

```
DCB PortDCB;
PortDCB.DCBlength = sizeof (DCB);
PortDCB.BaudRate = CBR_115200;
PortDCB.fBinary = TRUE;
PortDCB.fParity = TRUE;
PortDCB.fOutxCtsFlow = FALSE;
PortDCB.fOutxDsrFlow = FALSE;
PortDCB.fDtrControl = DTR_CONTROL_ENABLE;
PortDCB.fDsrSensitivity = FALSE;
PortDCB.fTXContinueOnXoff = FALSE;
PortDCB.fOutX = FALSE;
PortDCB.fInX = FALSE;
PortDCB.fErrorChar = FALSE;
```

```
PortDCB.fNull = FALSE;
PortDCB.fRtsControl = RTS_CONTROL_ENABLE;
PortDCB.fAbortOnError = FALSE;
PortDCB.ByteSize = 8;
PortDCB.Parity = NOPARITY;
PortDCB.StopBits = ONESTOPBIT;
```

(4) 设置超时值

对于串行端口来说，还必须配置超时值，否则程序可能陷入到一个等待来自串口字符的死循环。通常，配置超时值和配置串口类似。首先用 GetCommTimeouts 函数获取当前串口的超时值，然后修改 CommTimeouts 成员变量的值，最后用 SetCommTimeouts 设置新的超时值。

3.2.2 网络传输软件设计与实现

3.2.2.1 网络传输协议的选择

Internet 在传输层主要有两种主要的协议：一种是面向连接的协议 TCP 协议，一种是无连接的协议 UDP 协议。TCP（Transmission Control Protocol）专门用于在不可靠的 Internet 上提供可靠的、端到端的字节流通信协议。而 UDP(User Datagram Protocol)主要用来传输大量的数据。因为我们的系统将来会放在野外，而 PC 服务器要放在距离很远实验室里。如果用 UDP 传输协议，经过几层网关和路由后，数据包的很可能丢失，或出现次序上的错误。因此必须采用 TCP 这样面向连接可靠的传输协议。

3.2.2.2 软件的设计与开发

开发工具选用了 Microsoft Visual Studio，因为其支持 Microsoft .Net Framework，我们可以调用.Net Framework 中的类库方便的进行编程，而且 Windows CE.Net 也支持.Net Framework1.0。我们选取的编程语言是对.Net Framework 支持最好的 C#语言。

我们编写的负责传感器网数据的接收，网络传输的应用软件 E-Box Sensor Receiver，其软件设计流程图如图 3-5 所示。

首先创建一个 TcpClinet 类，建立一个 socket 接口，并创建一个 NetworkStream 类与 Server 建立连接：

```
another_client=new TcpClient(t_ip.Text,Convert.ToInt32(t_port.Text));
another_stream=another_client.GetStream();
```

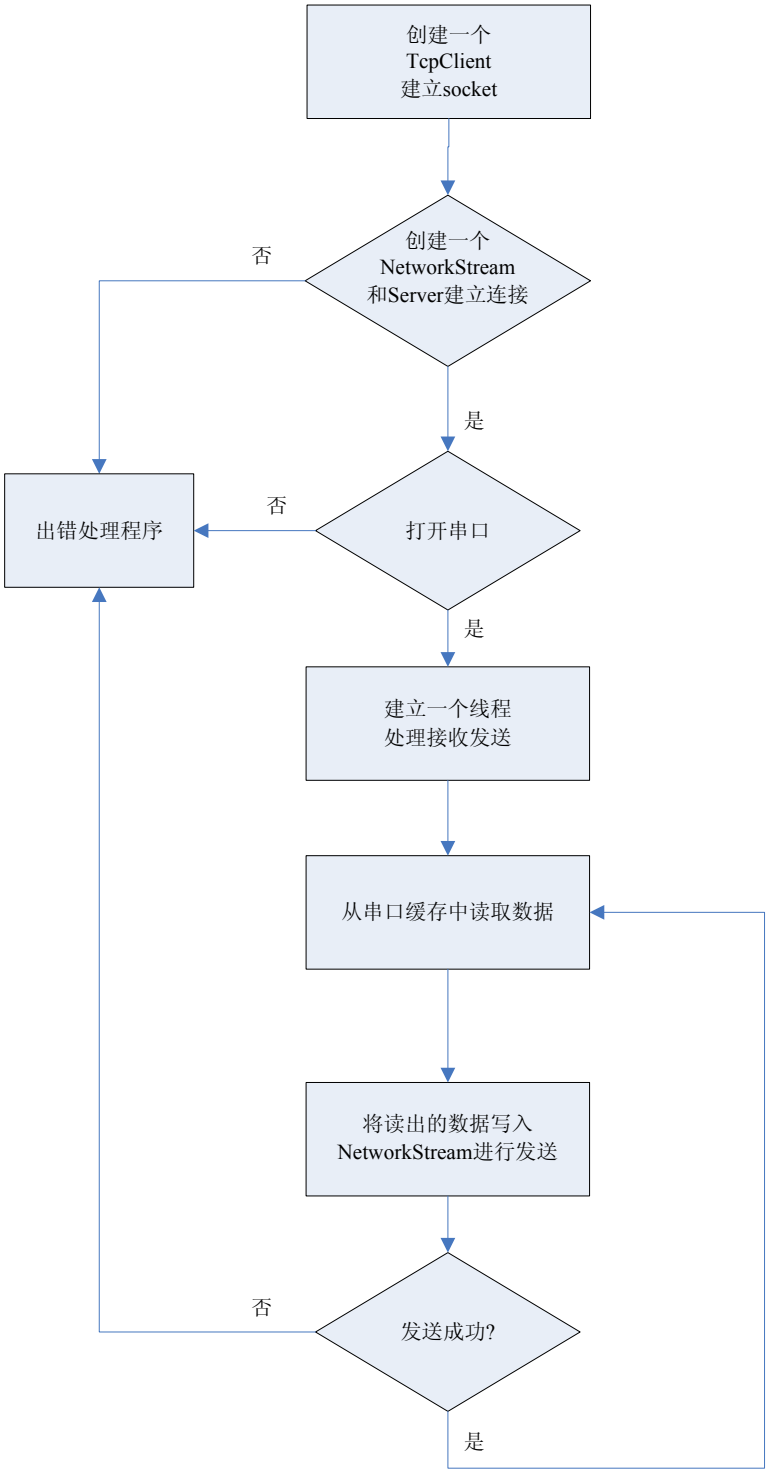

图 3-5 嵌入式系统上应用软件 E-Box Sensor Receiver 流程图

Figure 3-5 Flowchart of application software Architecture for Embedded system

然后打开并配置串口，建立一个新的线程处理数据的接收和发送：

```
if(UseSerial.OpenPort("COM1:")==false)
{
    MessageBox.Show("Failed to open port COM1");
    return;
}
if(UseSerial.SetPort(115200,8,0,1)==false)
{
    MessageBox.Show("Failed to set port");
    return;
}
Thread t=new Thread(new ThreadStart(thread_ReadComm));
t.Start();
```

将串口读入的数据从网络上发送出去，如果发送成功，则继续采集；如果失败，则进入出错处理程序，停止发送：

```
while(stop==false)
{
    if((len=UseSerial.Read(buff,15))>0)
    {
        newbuff=new byte[len];
        for(int i=0;i<len;i++)
        {
            newbuff[i]=buff[i];
        }
        try
        {
            EBox.SendMessage3(another_stream,newbuff);
        }
        catch
        {
            timeoutCount++;
            if(timeoutCount>=100)
            {
                MessageBox.Show("100 times network detected..");
                PCConnected=false;
                UseSerial.ClosePort();
                return;
            }
        }
    }
}
```

3.3 摄像头的视频采集与传输

3.3.1 摄像头与驱动程序简介

我们采用的是罗技公司型号为 Pro5000 的摄像头，如图 3-6 所示。这款摄像头采用的是高品质 CMOS 感应器，其动态捕捉可以达到 640x480 像素。采用 Rightlight 超级弱光补偿技术，在光线不足时性能仍然出色。并且采用了高质量光学广角镜头。

图 3-6 罗技 Pro5000 摄像头

Figure 3-6 Logitech Camera Pro5000

我们选择这款摄像头除了它特性出众外，还因为 Microsoft 为其编写了 Windows CE.Net 的驱动程序。提供了从摄像头读取一帧图像的接口：

DWORD CAM_IOControl (DWORD dwOpen, DWORD dwCode, PBYTE pIn,
DWORD dwIn, PBYTE pOut, DWORD dwOut, DWORD *pdwBytesWritten)

3.3.2 应用程序设计

我们采用 Embedded Visual C++在 E-Box 上开发这个视频采集和传输应用程序，因为摄像头驱动的接口只有 Win32 应用程序才可以访问。摄像头传输过来的数据都是经过 mjpeg 编码的，每一帧 mjpeg 格式图片转换成 bmp 格式，进行显示和传输。Mjpeg 转换成 bmp 格式时调用了 Microsoft 的 GDI(Graphic Digital Interface)库中的函数。由于 Microsoft 在开发这款驱动时，附带开发了一个 Demo 程序，因此我们的应用程序主要是在上面进行了修改，添加了网络传输的功能。

在网络传输方面，我们选择了面向连接的 TCP 协议，因为我们图像格式不大（160*120），因此网络的带宽足以保证传输速率（100M 以太网卡）。最后实验的结果可以保证每秒传输采集 10 帧图像以上。而 TCP 协议可以保证远距离图像的丢失和错续问题。这对于一些远程监控的应用来说是必须的。

网络传输编程采用 Windows Socket 编程，如图 3-7 所示。

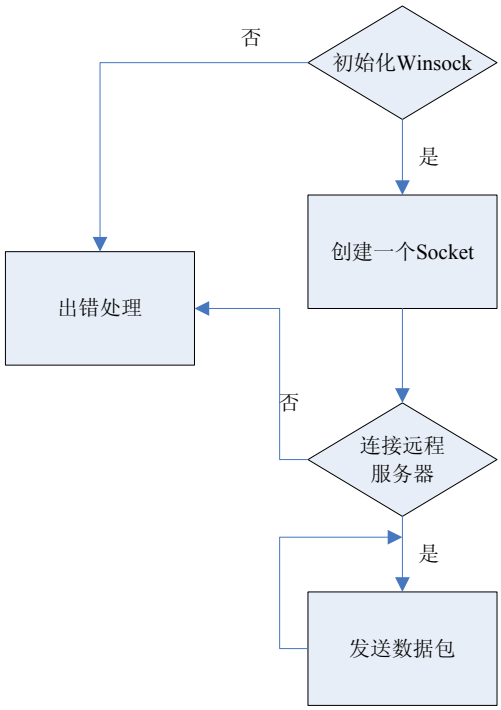

图 3-7 基于 Windows Socket 的网络编程

Figure 3-7 Flowchart of software design base on the Windows Socket

首先初始化 Winsock：

```
// Initialize Winsock.
int iResult = WSAStartup( MAKEWORD(2,2), &wsaData );
if ( iResult != NO_ERROR )
    printf("Error at WSAStartup()\n");
```

然后创建一个 Socket：

```
m_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
```

以及服务器的地址和端口：

```
clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = inet_addr( "192.168.0.101" );
clientService.sin_port = htons( 8000 );
```

然后进行远端服务器的连接尝试：

```
if ( connect( m_socket, (SOCKADDR*) &clientService, sizeof(clientService) )
    == SOCKET_ERROR)
{
    printf( "Failed to connect.\n" );
    WSACleanup();
    return 0;
}
```

调用 send()函数将包含数据的帧发送出去：

```
    bytesSent = send( m_socket, (const char *)(p), bufferSize, 0 );
```

建立连接后，每次从摄像头读取一帧数据并转化成 bmp 格式后，通过以太网发送到远端服务器，如图 3-8 所示。

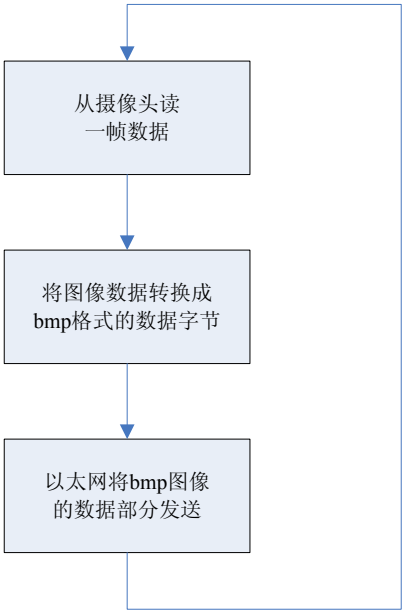

图 3-8 摄像头图像的网络传输

Figure 3-8 Network transmitting of Graphics from camera

3.4 本章小结

本章详细讨论了基于 E-Box 的嵌入式系统的实现过程，系统的数据流量非常大，所以在开发软件的过程中遇到并解决了很多问题：

首先是串口缓存区大小的问题。因为在数据网络发送的时候，不可能从串口缓存区读出数据，所以要分配足够大的内存作为串口的缓存区（在我们的软件设计中为 4KB），从而保证在网络发送的时候串口缓存区不会溢出。否则，程序运行时会莫名其妙的出现错误而自动退出。

其次就是网络传输中 TCP/IP 建立连接的问题。开始设计时，每传送一个数据包之前都要重新建立 TCP 连接，传送后销毁。在调试中我们发现，即使 100M 的网络传输速度，也经常会发生网络拥塞的现象。查阅了相关文档后，发现 TCP 建立连接时要经过三次握手的过程，在销毁连接时要经过四次握手的过程，这样就占用了大量的网络资源。我们在保证程序的健壮性的基础上，对程序进行了优化，实现了仅建立一次连接就可以进行网络传输，大大提高了系统的资源利用率。

经过我们的优化后，程序的健壮性和资源利用率都得到了很大的提高。在 10M 的网络带宽条件下运行了几天，都没有出现异常。

4. 基于 PC 的服务器系统设计

PC 机作为整个系统的服务器，完成了语音数据的接收，显示，频谱分析工作，以及视频图像显示的工作。所有的应用程序都是使用 C#语言在 Microsoft .Net Framework 上开发的。

4.1 语音数据的以太网接收和分析

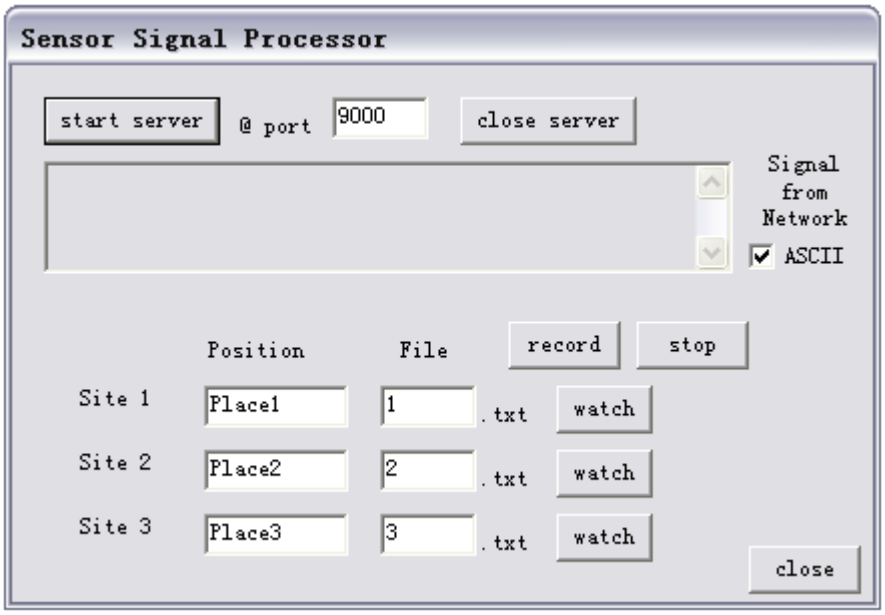

图 4-1 语音数据的以太网接收和分析软件

Figure 4-1 Software for voice data receiving from Ethernet and analyzing

4.1.1 语音数据的以太网数据的接收与保存

如图 4-2 所示，作为 TCP/IP 服务器，首先应建立一个 TcpListener 类绑定端口和 IP，开始侦听连接请求：

```
server=new TcpListener(IPAddress.Any,Convert.ToInt32(textBox1.Text));
server.Start();
```

并新开一个线程，开始接收工作，接收工作任务量比较大，因此新建一个线程：

```
thread_receive=new Thread(new ThreadStart(thread_receiver));
thread_receive.Start();
```

接受连接请求并赋给 NetworkStream 类型的数据：

```
client = server.AcceptTcpClient();
stream = client.GetStream();
```

如果网络中收到发往该端口的数据包，读到一个数组内，判断包的首字节，存到节点各自的文件中。

```
Byte[] bytes=new Byte[1024];
Int32 i= stream.Read(bytes, 0, bytes.Length);
if(node==1)
SaveToFile(ref sw1,placename,cache);
else if(node==2)
   SaveToFile(ref sw2,placename,cache);
   else
   SaveToFile(ref sw3,placename,cache);
```

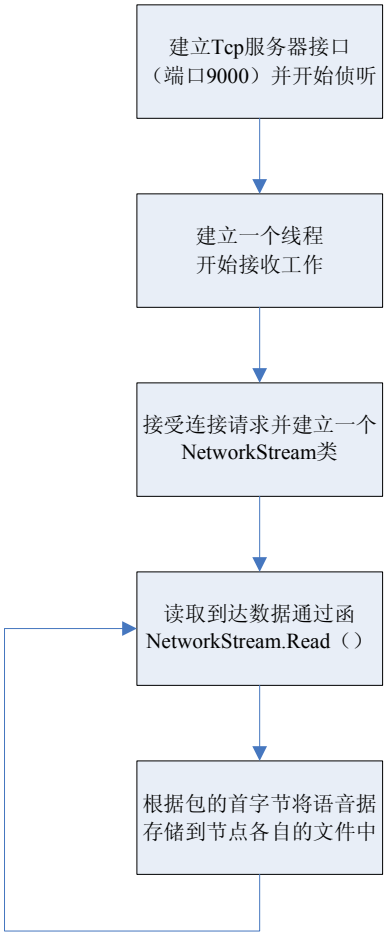

图 4-2 语音数据的以太网数据的接收与保存

Figure 4-2 Flowchart of voice data receive from Ethernet and save

4.1.2 节点语言数据显示与频谱分析

当数据记录到本地文件后，可以点击 watch 按钮查看每个节点的语音数据，如图 4-3 所示。

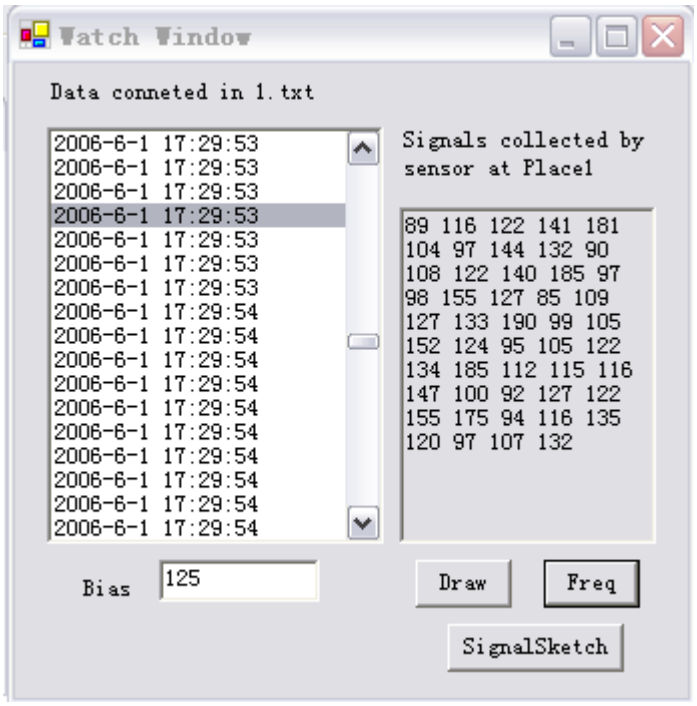

图 4-3 数据观察面板

Figure 4-3 Form for data watching

如果想观看该节点一次采样的波形（49 个点数据点）点击 Draw 按钮，即可看到波形图，如图 4-4 所示。

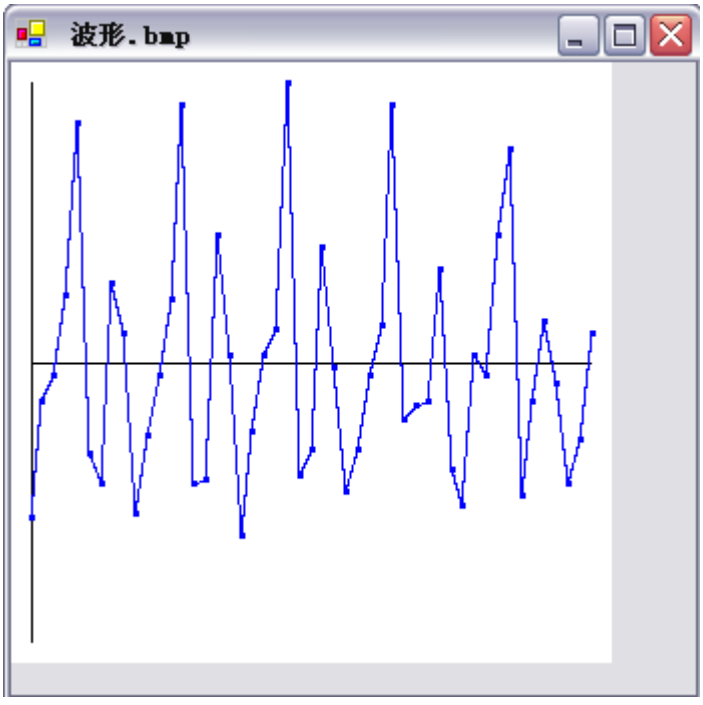

图 4-4 数据波形图

Figure 4-4 Picture of voice data received and saved

如果想观测该数据点的频谱，可以点击 Freq 按钮，即可看到这 49 个数据点进行 DFT 变换后的频谱图。如图 4-5 所示

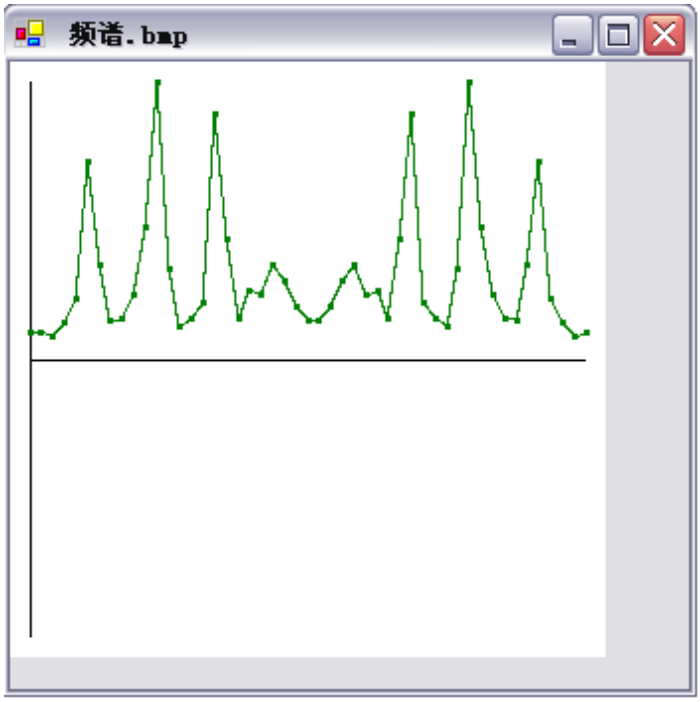

图 4-5 数据 DFT 变换后的频谱图

Figure 4-5 Voice data after DFT transform

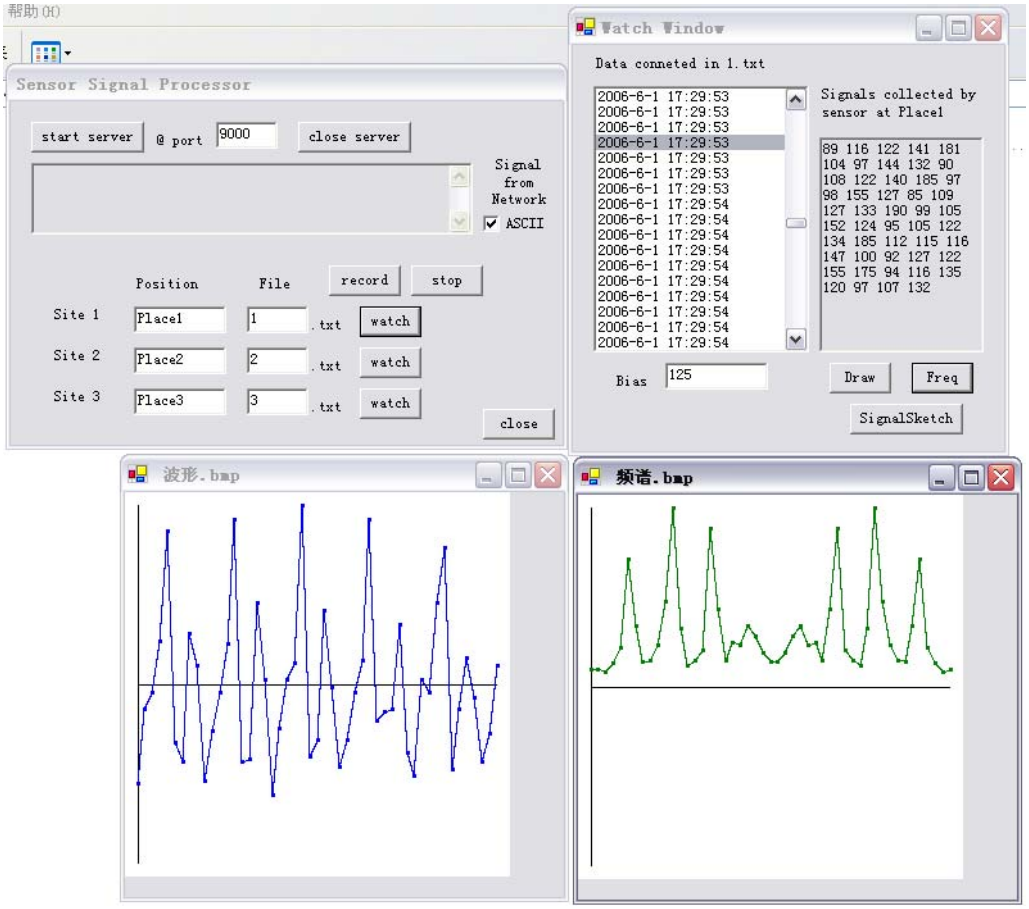

图 4-6 软件的整体结构图

Figure 4-6 Software for voice data receive and analyses

4.2 视频图像的以太网接收与显示

4.2.1 以太网图像数据的接收

网络接收部分与接收语音数据类似，如图 4-7 所示。首先应建立一个 TcpListener 类绑定端口和 IP，开始侦听连接请求：

```
server=new TcpListener(IPAddress.Any,Convert.ToInt32(textBox1.Text));
server.Start();
```

并新开一个线程，开始接收工作，接收工作任务量比较大，因此新建一个线程：

```
thread_receive=new Thread(new ThreadStart(thread_receiver));
thread_receive.Start();
```

接收连接请求并赋给 NetworkStream 类型的数据：

```
client = server.AcceptTcpClient();
stream = client.GetStream();
```

与语音数据接收不同的是，因为图像数据比较大，一帧图片实际上是分成好几个 TCP 包传送过来的，因此首先要进行一个判断是否接收完一帧图像，才能进行显示：

```
revSize += stream.Read(bytes, revSize, bytes.Length - revSize );
if( revSize == bytes.Length )
{
   ByteToBmp(bytes);
   revSize = 0;
}
```

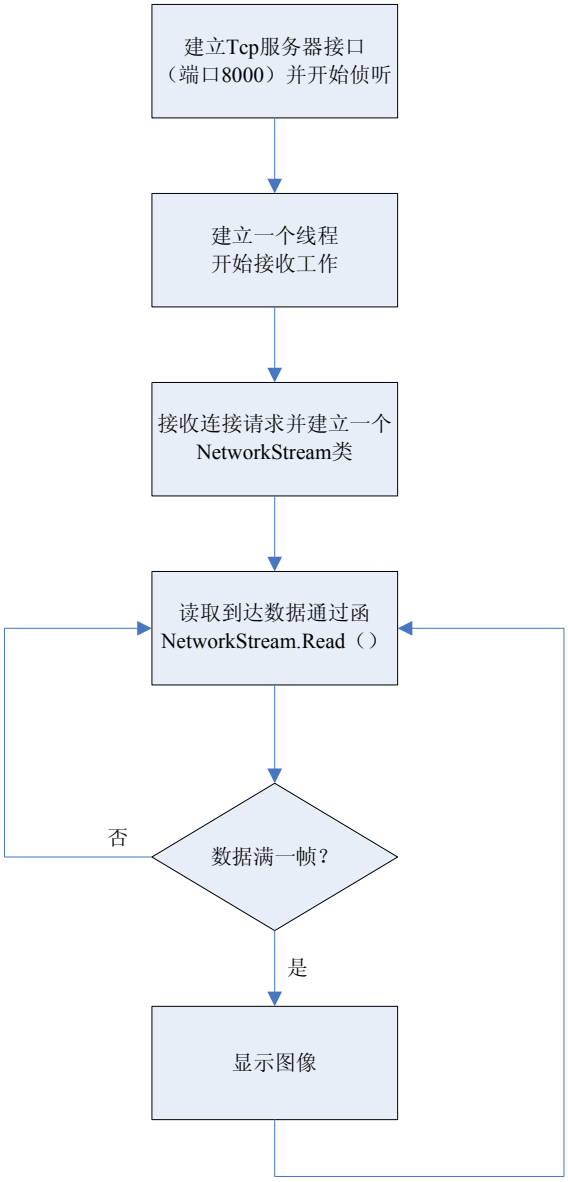

图 4-7 网络图像接收与显示

Figure 4-7 Video receive form Ethernet and display

4.2.2 位图图像的显示

图像显示整体流程如图4-8所示。首先，创建一个空的位图类并定义其宽和高：

```
private Bitmap bmp = null;
private const int WIDTH = 160;
private const int HEIGHT = 120;
```

如果网络中传输过来的图像数据大小符合要求，则将 Bitmap 对象锁定到系统内存中：

```
Debug.Assert( buffer.Length == ( WIDTH * HEIGHT * 3 ) );
BitmapData bd = bmp.LockBits( new Rectangle( 0, 0, WIDTH, HEIGHT ),
ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb );
```

将接收到的数据填入锁定内存区：

```
byte* p = (byte*)(void*)bd.Scan0;
for( int i = 0; i < WIDTH * HEIGHT * 3; ++i )
{
    p[0] = buffer[i];
    p++;
}
```

将Bitmap对象从系统内存中解锁并显示：

```
bmp.UnlockBits( bd );
pb.Image = bmp;
```

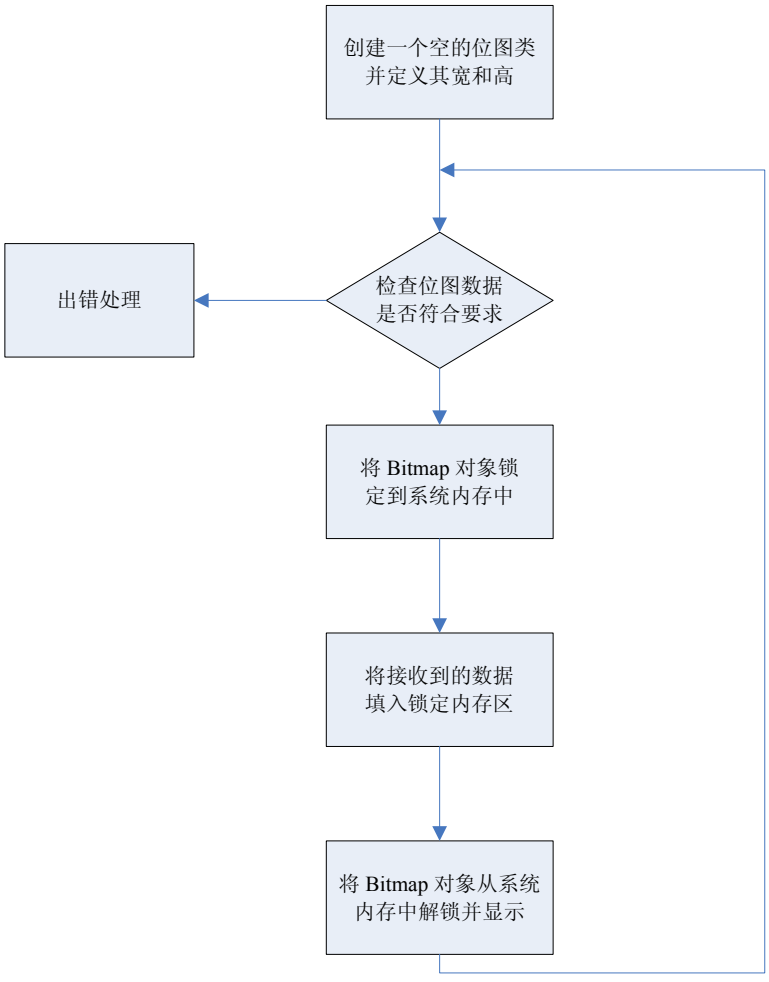

图 4-8 位图显示流程图

Figure 4-8 Bitmap data displayed

整体软件界面图如图 4-9 所示。按下 start server 按钮后，开始接收图像数据，如图 4-10 所示

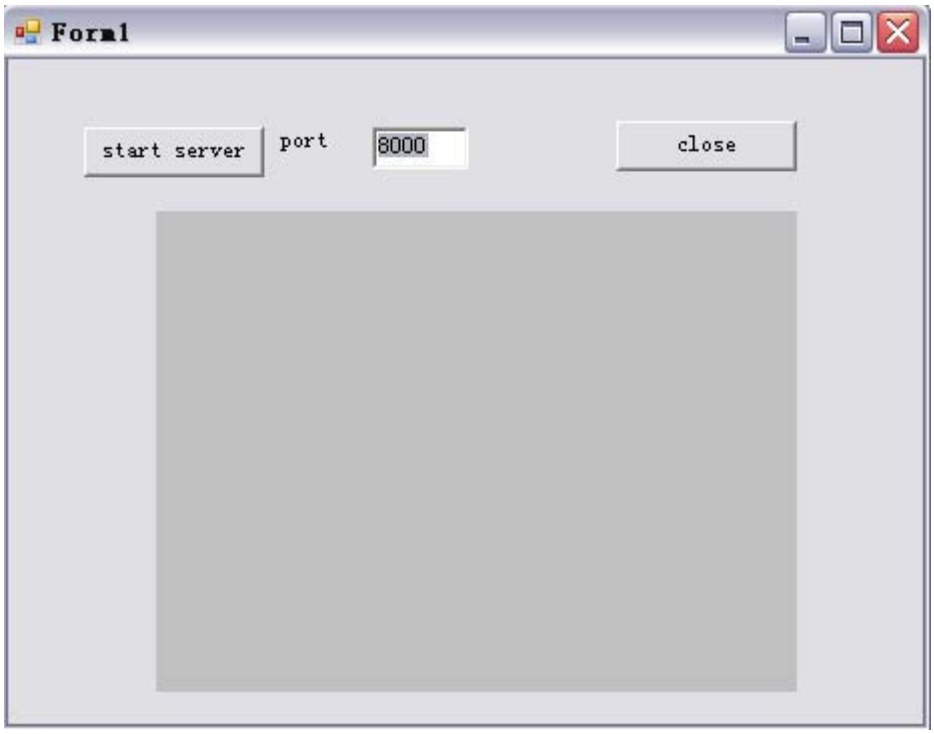

图 4-9 无图像传输时的界面

Figure 4-9 Graphic interface without video transmitting

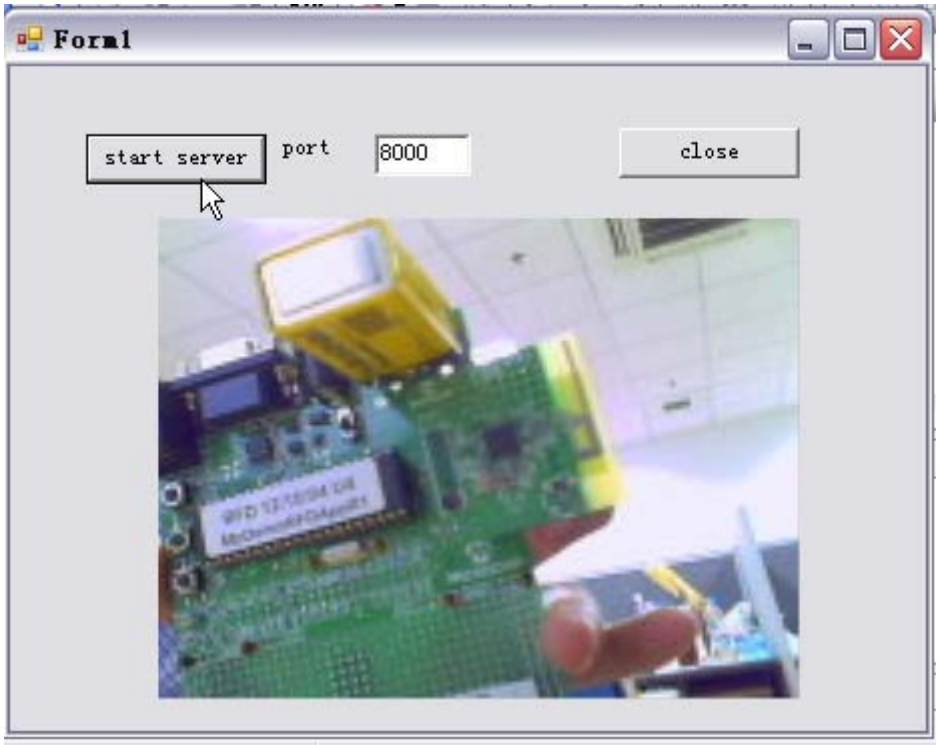

图 4-10 图像传输时的界面

Figure 4-10 Graphic interface with video transmitting

4.3 本章小结

本章详细讨论了基于 PC 的系统服务器的实现过程，尽管 PC 的处理能力非常强大，但是如果在对一些细节的开发处理不当的话，会大大降低系统的资源利用率。因此我们在开发软件的过程中也遇到并解决了很多问题：

首先是语音数据存储格式的问题。我们一开始采用 XML 格式的文件对数据进行存储，因为 XML 文件的读写起来较方便，格式也十分美观。但是后来发现在将数据存储成 XML 格式的文件时，将耗费大量的系统资源，以至于出现了网络数据接收的延迟现象。使我们不得不放弃将数据存成 XML 文件格式的设计想法，采用读写速度较快的文本文件（txt 文件），并自己定义了数据在文件中的存储格式。大大提高了存储速度，从而解决了数据延迟的现象。

其次是位图图像显示不完整的问题，图像往往仅显示 1/3 或一半的大小。后来发现问题的关键就是以太网数据传送和接收机理的问题。虽然 socket 通信中发送数据包的函数 send() 只要求给出发送的数据和长度，但是在网络层，数据可能被拆成几个数据包发送。因此在接收端必须要做好判断是否收满一帧图像的工作，当收满一帧图像后，才可以进行位图图像的显示。

5. 总结和展望

5.1 总结

从今年 1 月份放假前知道课题的方向，到 6 月份把整个系统调试成功，过了半年的时间。可以说整个半年的时间都在为这一个事情而忙碌。寒假里阅读了大量关于传感器网硬件节点文档以及各个生产厂商的产品的特性及价格，并委托成都的同学到厂家按着我给定的指标进行实地测试，开学后开始节点的以及开发工具的购买。终于 3 月中旬拿到了满意的硬件节点和全套开发工具，开始了传感器网络的搭建工作。

因为买的节点比较便宜，每个节点软件部分写的比较差，包括易读性和易用性。我们在网络通信方面遇到了很大的困难，最后决定重写主程序代码和部分主要通信函数，经过一个月的努力终于使传感器网络能够正常的工作。

嵌入式平台开发的开始就遇到了意想不到的问题。原本选择的嵌入式系统平台 Intel Sitsang 板出厂时的 USB 芯片有 BUG，不能支持摄像头这种准实时传输模式的设备。如果我们要实现摄像头功能要另选嵌入式系统平台。

在半个月里，再对比了 E-Box（Windows CE.Net 操作系统），Samsung ARM9 2410（ARM Linux 操作系统）开发板和 Motorola 一块开发板（ARM Linux 操作系统）的性能，实现摄像头功能的复杂度，操作系统的易开发性，以及课题资金以及实验室资源等各个方面。最终选择 E-Box 作为我们嵌入式系统硬件平台。并从淘宝网上以低于市场价 200 元的价格（不想再花徐老师的实验经费，自淘腰包）购买了新的有 Windows CE.Net 操作系统驱动的摄像头。

因为整个设计是个全新的领域，因此在开发工程中每前进一步都要花费很多心血，也接触了许多过去从未接触过的知识和领域。当整个系统连调成功后回头看整个系统，发现整个系统还是相当复杂的：

首先，开发整个系统要使用了 3 种编程语言：

(1) C 语言，用来开发传感器节点，因为要跟许多具体的硬件打交道，不得不阅读和编写很多底层的 C 语言程序

(2) C++语言，用来开发嵌入式设备串口驱动，嵌入式操作系统镜像，摄像头传输程序 C++强大的类库让我们编程终于有了少许的轻松。

(3) C#语言，用来开发嵌入式设备应用程序，以及 PC 机服务器的程序。可以说整个系统的所用的应用程序都是用 C#语言开发的。.Net Framework 类库让我们不用了解底层的实现，就可以直接编写出美观可视的应用程序，极大的加快了我们的开发速度。让我们充分体会到，面向对象的程序设计语言给我们带来的好处。

其次，开发整个系统使用了 4 种开发工具和平台：

(1) Microsoft Visual Studio.Net 开发工具，用来开发整个系统的应用程序，包括 E-Box 上的和 PC 上的。友好的界面，详细的帮助文档，以及完整的调试工具，使我们开发起程序来十分轻松。因此，应用程序的开发在我们整个系统开发中占了很少的时间。

(2) Platform Builder 开发工具，用来开发 Windows CE.Net 操作系统，编译生成我们我

们所需要功能的 Windows CE.Net 操作系统镜像，下载到 E-Box 中，并提供远程硬件调试手段。虽然使用比较繁琐，但提供了调试手段。

(3) Embedded Visual C++开发工具，用来开发 Windows CE.Net 操作系统底层驱动程序，以及 Windows 32 程序。因为运行的平台跟开发所在的平台不一样。所以不支持调试，只能通过在程序中加一些调试信息帮助调试。

(4) Microchip IDE 开发工具，用来开发传感器网络节点单片机的程序。编程环境十分恶劣，不仅不支持许多编译器应用的功能，而且不支持在线调试。我们一般仅用其进行编译和程序的下载。

再次，开发的对象是 3 种完全不同的硬件平台和操作系统

(1) 传感器节点，该单片机上没有操作系统，所有的任务都是单线程，所有的操作都是基于寄存器的读写。因此开发时要特别注意代码的优化，使每个函数执行时间尽可能的短。

(2) E-BOX，基于 x86 系列 CPU，和 Windows CE.Net 操作系统，支持多线程和多任务。因为 Windows CE.Net 操作系统只支持部分 Windows API 函数，因此有时编程时要特别留意。

(3) PC，最强大的处理平台，应此系统设计时应该考虑，将一些复杂度的工作放到 PC 来处理。这样会大大的提高整个系统的效率。

最后，整个系统涉及了很多学科和技术领域：

(1) 无线通信：传感器网络的通信基础。

(2) 传感器：数据采集的核心模块。

(3) 无操作系统的嵌入式开发：传感器网络节点的开发。

(4) 基于 Windows CE.Net 的嵌入式开发：E-Box 嵌入式平台的开发。

(5) 远程网络通信：以太网的数据传输。

(6) 数字图像处理：视频信息的编码和解码。

(7) 数字信号处理：采集声音数据的处理和分析。

通过这次毕业设计，我学到了很多新的知识，接触到很多新的事物。在这次毕业设计中，我学习了 C#语言，第一次编写基于.Net Framework 的程序，第一次接触到 Windows CE.Net 操作系统，以及 Win32 程序设计。学会了使用 Microsoft Visual Studio.Net，Platform Builder，Embedded Visual C++开发工具进行编程和开发，收获非常大。

5.2 展望

正如我 5.1 节总节中所说的，我们的整个系统涉及了很多学科和技术领域。虽然毕设课题已经基本完成，但是随着研究的深入，认识到整个系统还有许多需要改进和完善的地方：

(1) 制作自主知识产权的传感器：可以先绘制制作射频板，然后再制作母板。

(2) 语音采集的编码传输：由于现在经 A/D 转换得到的数据没有经过编码和压缩，所以网络中的数据量比较大，影响每个节点的采样率。可以借鉴 mpeg 等编码方式，进行差分编码。

(3) MESH 网和路由节点的实现：由于时间有限，我们没有尝试 MESH 网和路由节点的实现。实现 MESH 网可以极大的提高系统的鲁棒性，而路由节点的实现可以极大的提高传感器网的覆盖范围。

(4) PC 服务器端声音的播放：重现每个传感器采集的声音，更加形象，并且可以用于一些语音侦听的特殊场合。PC 服务器端将传来的语音数据进行 PCM 编码，再加上 wav 格式头，就可以变成 wav 文件播放出来。

(5) 语音信号的其它数字信号处理方式：不同特征的语音可以采用不同的数字信号处理方式，不一定是 DFT 变换，可以尝试其它如 FFT 变换、经常用于语音处理的小波变换等。

(6) 传感器网络对外数据传输的问题：数据在系统中传输的另一个瓶颈就是在协调器节点与外部通信上。虽然采用一个带有网络接口和高速串口的设备可以缓解一下这个问题。但如果采用一个带有 SPI 接口和网络接口的高速处理器，作为协调器的处理与控制器可以从根本上解决这个问题。难点在于 ZigBee 协议栈在新系统上的移植工作。

(7) 对于某些特殊的应用场合，和对时间要求很严格的场合来说，传感器网中各个节点声音采集的同步是一个很重要的问题。可以在网络通信中使用带信标的通信方式，在一定程度上解决这个问题。传感器节点通过侦听协调器发送的信标来保持声音采集上的同步。

(8) 像在湿地，这种条件非常恶劣的条件下，架设一根网线与外部 Internet 连通是一件非常困难的事情。我们将来可以考虑用能构覆盖几公里的 Wi-Max 技术，代替有线的以太网络。

(9) 这次系统视频图像是在 E-Box 上完成 mjpeg 到 bmp 转换，然后再进行以太网传输的，在图像分辨率增大到一定程度后，会严重影响视频质量和网络传输带宽。如果将来在 PC 上完成 mjpeg 格式图像流的解码，可以保证高清晰视频的采集和传输。

(10) 传感器节点的单片机上没有操作系统，导致资源利用率低。将来在上面可以移植操作系统，如 uc-osII，Tiny-Os 等嵌入式操作系统。这样可以更好的利用系统资源，提高采集传输的效率。

(11) 声音的采集主要靠声电以及模数转换模块来实现的，因此这个模块对外界声音信号的敏感程度，对外界噪声的抗干扰程度，都将制约着采集到语音的质量。可以选择更敏感的麦克风，以及设计更好的放大抑制噪声电路。

这个系统除了用在鸟类声音检测外，将来可以有更广泛的应用。比如可以用于 2010 年世博会的安防工作。在展览馆中布置很多节点，一旦有非法者在非法时段入侵，声音或者震动会将休眠中的传感器网络唤醒，采集他们入侵的声音以及图像资料保存到服务器上。并可以短信等方式通知相关的保卫人员，协助他们进行安防保卫工作。

上面仅仅是我描述我们系统的一个应用场景，除了适应实时高速的采集传输外，还可以胜任大部分传感器网络的应用场景，如智能教室，智能家居，交通流量检测，环境监测等。

参考文献

[1] IEEE std. 802.15.4-2003: Wireless Medium Access Control(MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) http://standards.ieee.org/getieee802/download/80.15.4-2003.pdf

[2] ZigBee Specification: The ZigBee Specification describes the infrastructure and services available to applications operating on the ZigBee platform. Accepted by ZigBee Alliance Board of Directors.

[3] Data sheet of CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver.

[4] A FLASH Boot loader for PIC16 and PIC18 Devices: Microchip Company.

[5] Microchip Stack for the ZigBee Protocol: Microchip Company.

[6] Implementing a PID Controller Using a PIC18 MCU: Microchip Company.

[7] Microchip TCPIP Stack Application Note: Microchip Company.

[8] 8-bit PIC® Microcontroller Solutions: Microchip Company.

[9] An SNMP Agent for the Microchip TCPIP Stack: Microchip Company.

[10] Complete PIC18C Reference Manual: Microchip Company.

[11] Dynamic Memory Allocation for the MPLAB® C18 C Compiler: Microchip Company.

[12] 孙利民等。无线传感器网络。北京：清华大学出版社，2004。

[13] 周毓林等。Windows CE.Net 内核定制及应用开发 北京：电子工业出版社

[14] James W. Cooper C# Design Patterns: A tutorial 北京：电子工业出版社

[15] eBox-II Windows CE 5.0 Jump Start: Build Windows CE 5.0 Image With Platform Builder end ICOP_Vortex86_50 Board-Support-Package. ICOP Technology Inc.

致谢

本设计是在徐国治教授的悉心指导下完成的。徐老师虽然有繁忙的工作，但仍抽出时间对我们的毕业设计进行关怀和指导，而且还给我们提供了良好的实验环境和先进的硬件设备。徐老师对学生认真负责的态度、严谨的科学研究方法、敏锐的学术洞察力、勤勉的工作作风以及勇于创新、勇于开拓的精神永远是我学习的榜样。在此，谨向徐老师致以深深的敬意和由衷的感谢。

在毕业设计的前期、中期和后期，还得到了周玲玲老师，蒋乐天老师，应忍冬老师的热心指导与帮助，在此对他们表示衷心的谢意。同时还要感谢跟我同组的雷暴和宋恺同学，整个团队的团结合作，保证了这个课题设计按期完成。从他们两个身上学到了很多东西，让我受益匪浅。此外还要感谢张骋元，段云同学，从他们那里得到了许多关于 E-Box 的资料，特别是段云同学关于串口驱动的动态链接库，让我们在开发的时候节省了很多时间。

还要感谢我的父母和我的女朋友，他们在生活上给予我很大的支持和鼓励，是他们给予我努力学习的信心和力量。

最后，再次感谢所有关心我，支持我和帮助过我的同学、朋友、老师和亲人！

译文及原文

ZigBee 协议概述

ZigBee 是一个为低速控制网络设计的无线传感器网络标准。ZigBee 网络的一些应用包括建筑物自动化网，建筑物安防，工业网络控制，远程抄表以及 PC 外设等。为了更好的理解 Microchip 协议栈，在下面的几节将提供一个关于 ZigBee 简要的概述。有兴趣的读者还可以访问 ZigBee 的网站（www.zigbee.org）来获取更多的资料。

IEEE 802.15.4

ZigBee协议使用IEEE 802.15.4 规范作为介质访问层（MAC）和物理层（PHY）。IEEE802.15.4总共定义了3个工作频带：2.4GHz、915MHz和868MHz。每个频带提供固定数量的信道。例如，2.4GHz频带总共提供16个信道（信道11-26）、915MHz频带提供10个信道（信道1-10）而868MHz频带提供1个信道（信道0）。

协议的比特率由所选择的工作频率决定。2.4GHz频带提供的数据速率为250kbps，915MHz频带提供的数据速率为40kbps而868MHz频带提供的数据速率为20kbps。由于数据包开销和处理延迟，实际的数据吞吐量会小于规定的比特率。

IEEE802.15.4MAC数据包的最大长度为127字节。每个数据包都由头字节和16位CRC值组成。

16位CRC值验证帧的完整性。此外，IEEE802.15.4还可以选择使用应答数据传输机制。使用这种方法，所有特殊ACK标志位置1的帧均会被它们的接收器应答。这就可以确定帧实际上已经被传递了。如果发送帧的时候置位了ACK标志位而且在一定的超时期限内没有收到应答，发送器将重复进行固定次数的发送，如仍无应答就宣布发生错误。注意接收到应答仅仅表示帧被MAC层正确接收，而不表示帧被正确处理，这是非常重要的。接收节点的MAC层可能正确地接收并应答了一个帧，但是由于缺乏处理资源，该帧可能被上层丢弃。因此，很多上层和应用程序要求其他的应答响应。

网络配置

ZigBee无线网络可采用多种类型的配置。星型网络配置由一个协调器节点（主设备）和一个或多个终端设备（从设备）组成。协调器是实现了一组很多ZigBee服务的一种特殊的全功能设备（Full Function Device，FFD）。终端设备可能是FFD或简化功能设备（RFD）。RFD是最小而且最简单的ZigBee节点。它只实现了一组最少的ZigBee服务。在星型网络中，所有的终端设备都只与协调器通信。如果某个终端设备需要传输数据到另一个终端设备，它会把数据发送给协调器，然后协调器依次将数据转发到目标接收器终端设备。

除了星型网络之外，ZigBee还可以采用点对点网络、群集或网状（mesh）网络配置。由于群集和网状网络具有在多个网络之间路由数据包的功能，因而被称为多跳网络，而星型网络则被称为单跳网络。

和任何网络一样，ZigBee网络也是多点接入网络，这意味着网络中的所有节点对通信介质的访问是同等的。有两种类型的多点接入机制。在没有使能信标的网络中，只要信道是空闲的，在任何时候都允许所有节点发送。在使能了信标的网络中，仅允许节点在预定义的时隙内进行发送。协调器会定期以一个标识为信标帧的超级帧开始发送，并且希望网络中的所有节点与此帧同步。在这个超级帧中为每个节点分配了一个特定的时隙，在该时隙内允许节点发送和接收数据。超级帧可能还含有一个公共时隙，在此时隙内所有节点竞争接入信道。

Microchip协议栈的当前版本仅支持无信标的星型网络配置。

网络关联

ZigBee网络可以是ad-hoc网络，即可以根据需要组建或不组建新的网络。在星型网络配置中，终端设备在可以执行任何数据传输之前将总是搜索网络。新的网络首先由协调器建立。启动时，协调器会搜索附近的其他协调器，如果没有找到协调器，它就会建立一个自己的网络并选择一个惟一的16位PANID。一旦新网络建立，就会允许一个或多个终端设备与此网络相关联。具体是允许还是不允许新关联由协调器决定。

一旦组建了网络，就可能由于物理更改而发生多个网络重叠和PANID冲突。在这种情况下，协调器会启动PANID冲突解决过程并且会有一个协调器将更改其PANID和/或信道。受到影响的协调器会指示它所有的终端设备进行必要的更改。Microchip协议栈的当前版本不支持PANID冲突解决。根据系统需求，协调器会在非易失性存储器中存储所有网络关联，称为邻接表。为了连接到网络，终端设备可能执行孤立通知过程来查找先前与之关联的网络或者执行关联过程来加入一个新网络。在执行孤立通知过程的情况下，协调器将通过查找其邻接表来识别先前与之关联的终端设备。

一旦关联到网络，终端设备就可选择通过执行解除关联过程与该网络解除关联。如果需要的话，协调器本身也会启动解除关联过程来强制节点离开网络。Microchip协议栈的当前版本支持新的关联和孤立通知过程。它仅支持由终端设备启动的离开网络过程。

端点、接口、群集、属性和配置文件

典型的ZigBee节点可支持多种特性和功能。例如，I/O节点可能有多种数字和模拟输入/输出。一些数字输入可能被一个远程控制器节点用到，而其他数字输入可能被另一个远程控制器节点使用。这种分配将创建一个真正的分布式控制网络。为了便于在I/O节点和两个控制器节点之间进行数据传输，所有节点中的应用程序必须保存多个数据链路。为了减少成本，ZigBee节点仅使用一个无线信道来和多个端点/接口来创建多条虚拟链路或信道。

一个ZigBee节点支持31个端点（编号为0-31）和8个接口（编号为0-7）。端点0被保留用于设备配置而端点31被保留仅用于广播。剩下的总共30个端点用于应用。每个端点总共有8个接口。因此，实际上，应用在一个物理信道中最多可能有240条虚拟信道。

一个典型的ZigBee节点也将有很多属性。例如，I/O节点包含称为数字输入1、数字输入2、模拟输入1等的属性。每个属性都有自己的值。例如，数字输入1属性可能有值1或0。属性的集合被称为群集。在整个网络中，每个群集都被分配了一个惟一的群集ID。每个群集最多有65,536个属性。

ZigBee协议还定义了一个称为配置文件的术语。配置文件就是指对分布式应用的描述。它根据应用必须处理的数据包和必须执行的操作来描述分布式应用。使用描述符对配置文件进行描述，描述符仅仅是各种值的复杂结构。此配置文件使ZigBee设备可以互操作。ZigBee联盟已经定义了很多标准的配置文件，比如远程控制开关配置文件和光传感器配置文件等。任何遵循某一标准配置文件的节点都可以与其他实现相同配置文件的节点进行互操作。Microchip协议栈的当前版本不提供任何标准的配置文件功能。如果需要的话，可以编写一个实现所需要的配置文件的协同任务函数。也可以创建自己的自定义配置文件（或分布式应用程序）仅与专有节点配合工作。和此应用笔记一起提供的演示应用程序实现了远程控制LED的自定义分布式应用程序，其中一个节点的LED由另一个节点上的开关控制。每个配置文件可以定义最多256个群集，而且和我们在前面所看到的一样，每个群集可以最多有65,536

个属性。此灵活性允许节点有大量的属性（或I/O点）。

端点绑定

前面提到过，星型网络中的终端设备总是只与协调器通信。协调器负责将端点发送的数据包从一个节点转发到接收终端设备的相应端点。您可能会猜到，当建立一个新的网络时，必须告知协调器如何创建源端点和目标端点之间的链路。ZigBee协议定义了一个称为端点绑定的特殊过程。作为绑定过程的一部分，一个远程网络或一个类似于设备管理器的节点会请求协调器修改其绑定表。协调器节点维护一个基本上包含两个或多个端点之间的逻辑链路的绑定表。每个链路根据其源端点和群集ID来惟一定义。

例如，如果需要将来自我们的I/O节点的数字输入1的数据发送到控制器节点的控制信道1，我们就必须请求协调器创建一个绑定表项，此绑定表项将I/O节点的数字输入1端点作为源端点，将控制器节点的控制信道1作为目标端点。一旦创建了绑定表项，任何时候I/O节点从数字输入1端点发送数据时，协调器节点就会查找它的绑定表，并将数据包转发到控制器节点的控制信道1端点。数字输入1和控制信道1将共享同一个群集ID。根据绑定表的创建方法，可能将数据从一个端点组播到多个节点上的多个端点。Microchip协议栈的当前版本不支持这种组播绑定表项。

ZigBee协议定义了称为ZigBee设备对象（ZigBeeDeviceObject，ZDO）的特殊软件对象，它在其他服务中提供绑定服务。只有在协调器上运行的ZDO才会提供绑定服务。远程网络/设备管理器将直接将特殊绑定请求发送给ZDO（端点0）以创建或修改绑定表项。根据ZigBee规范，运行特殊ZigBee节点软件的PC或其他高端控制器可以作为网络管理器。如果不想创建或使用特殊网络管理器节点，可以编写自定义的绑定服务来简化绑定过程。和此应用笔记一起提供的演示应用程序实现了简单的自定义绑定方法。它使能每个节点将各自的对协调器节点的绑定请求发送给端点0并定义各自的自定义群集ID。欲知更多信息，请参阅ZDO.c文件中的CUSTOM_DEMO_BIND群集ID。根据此自定义的绑定过程，终端设备必须处于配置模式才能发送绑定请求。在正常执行模式下，协调器可以接收和发出自己的绑定请求。

当检测到某个按开关的过程时，终端设备将使用CUSTOM_DEMO_BIND作为群集ID发出特殊的二进制数据结构（欲知有关配置模式和绑定过程的更多信息，请参阅“配置演示应用程序”）。它直接将绑定请求数据包发送到协调器的端点0。演示协调器中的ZDO接收群集ID为CUSTOM_DEMO_BIND的数据包并将此过程委派给ProcessCustomBind函数。此函数实际上是在DemoCoordApp.c文件中实现的。可以简单地按照ProcessCustomBind函数的执行逻辑来充分理解自定义绑定的概念。你可以直接使用此自定义绑定逻辑或将其作为编写自己的绑定逻辑时的参考。

数据传输机制

传输数据到终端设备和从终端设备传输数据的确切机制随网络类型的不同而有所不同。在无信标的星型网络中，当终端设备想要发送数据帧时，它只需等待信道变为空闲。在检测到空闲信道条件时，它将帧发送到协调器。如果协调器想要将此数据发送到终端设备，它会将数据帧保存在其发送缓冲器中，直到目标终端设备明确地来查询该数据为止。此方法确保终端设备的接收器是被开启的，而且可从协调器接收数据。

在点对点网络中，每个节点必须一直保持它们的接收器为开启状态或者同意在一个时间段内开启它们的接收器。这将允许节点发送数据帧并确保数据帧会被其它节点接收。

终端设备必须查询协调器以获取其数据，而不是保持接收器开启，从而允许终端设备降低其功耗要求。根据应用的要求，在绝大部分时间内终端设备都处于休眠状态，而仅定期地唤醒设备来发送或接收数据。此方法的一个缺点就是协调器必须将所有数据帧保存在内部缓冲器中，直到目标终端设备唤醒并查询数据。如果网络包含很多休眠时间很长的终端设备，协调器就必须将数据帧保存很长时间。根据节点的数量和交换数据帧的速率，这将大幅增加协调器对RAM的需求。协调器可以根据终端设备的设备描述符有选择地决定将一个特定的帧保持一段长时间或短时间。ZigBee协议要求所有终端设备都保存描述它们的特性和功能的不同方面的各种描述符。Microchip协议栈的当前版本不支持描述符。

协议栈架构

Microchip协议栈是采用C语言编写的，可用MPLABC18和Hi-TechPICC-18编译器进行编译。源文件会自动根据所使用的编译器进行必要的更改。Microchip协议栈设计为仅在MicrochipPIC18F系列单片机上运行。Microchip协议栈使用内部闪存程序存储器来存储可配置的MAC地址、网络表和绑定表。因此，必须使用可自编程的闪存存储器单片机。如果需要的话，可以修改非易失性存储器（NVM）程序来支持任何其他类型的NVM而不使用可自编程的单片机。此外，该协议栈旨在在PICDEMZ演示板上运行。但是，它可很容易地移植到任何使用兼容单片机的硬件中。

协议栈层

Microchip协议栈根据ZigBee规范的定义将其逻辑分为多个层。实现每个层的代码位于一个独立的源文件中，而服务和应用程序接口（ApplicationProgrammingInterfaces，API）则在头文件中定义。

协议栈的当前版本不实现安全层。每个层为紧接着的上一层定义一组容易理解的函数。要实现抽象性和模块性，顶层总是通过定义完善的API和紧接着的下一层进行交互。特定层的C头文件（如zAPS.h）定义该层所支持的所有API。必须切记，用户应用程序总是与应用编程支持（ApplicationProgrammingSupport，APS）层和应用层（ApplicationLayer，APL）交互。由每层提供的很多API都是简单的C语言宏，调用下一层中的函数。此方法可以避免与模块化相关的典型开销。

协议栈API

Microchip协议栈由很多模块组成。典型的应用程序总是与应用层（APL）和应用支持子层（APS）接口。但是，如果需要的话，也可以简单地将应用程序与其他模块接口并且/或者根据需要对它们进行自定义。以下部分仅提供对APL和APS模块的详细API描述。如果需要的话，可以在它们各自的头文件中了解其他模块的API详细信息。如需最新信息，可以参考实际的源文件。

应用层（APL）

APL模块提供高级协议栈管理功能。用户应用程序使用此模块来管理协议栈功能。zAPL.c文件实现了APL逻辑，而zAPL.h文件定义APL模块支持的API。用户应用程序将包含zAPL.h头文件来访问其API。

网络层

网络层（NWK）负责建立和维护网络连接。它独立处理传入数据请求、关联、解除关联和孤立通知请求。典型的应用不需要直接调用NWK层。如有需要，可以参阅NWK.c和NWK.h文件了解对NWK层API的详细描述。

ZigBee设备对象

ZigBee设备对象（ZDO）打开和处理EP0接口。ZDO负责接收和处理远程设备的不同请求。不同于其他的端点，EP0总是在启动时就被打开并假设绑定到任何发往EP0的输入数据帧。

ZDO对象允许远程设备管理服务。远程设备管理器会向EP0发出请求，ZDO会处理这些请求。下面是一些远程服务实例：NWK_ADDR_REQ（给出网络地址），NODE_DESC_REQ（给出节点描述符）和BIND_REQ（绑定源EP、群集ID和目标EP）。一些请求仅适用于协调器。参阅ZDO.c获取完整列表。还应参阅ZigBee规范了解更多信息。

应用程序除非在启动时进行初始化，否则无需直接调用任何ZDO函数。如有需要，可以参阅ZDO.c和ZDO.h文件了解对ZDOAPI的详细描述。

ZigBee设备配置层

ZigBee设备配置层提供标准的ZigBee配置服务。它定义和处理描述符请求。远程设备可能通过ZDO接口请求任何标准的描述符信息。当接收到这些请求时，ZDO会调用配置对象以获取相应的描述符值。在目前的版本中，还没有完全实现设备配置层。

应用程序不需要直接调用任一配置函数。如有需要，可以参阅zProfile.c和zProfile.h文件了解对API的详细描述。

介质访问控制层

介质访问控制（MediumAccessControl，MAC）层实现了IEEE802.15.4规范所要求的功能。MAC层负责同物理（Physical，PHY）层进行交互。为支持不同类型的RF收发器，Microchip协议栈将不同的PHY交互归类到不同的文件中。每个支持的收发器都有一个独立的文件。注意由于RF收发器功能上的差异，MAC和PHY文件不能完全独立。MAC文件根据当前的RF收发器调整自身的某些逻辑。

在当前版本的协议栈中，zPHYCC2420.c文件实现ChipconCC24202.4GHz收发器特定的功能。将来，随着对RF收发器支持的添加，也将会相应地添加新的PHY文件。所有的RF收发器PHY文件使用zPHY.h文件作为它们与MAC层的主接口。

典型应用不需要直接调用MAC层。如有需要，可以参阅MAC.c、MAC.h和PHY.h文件了解对MAC/PHYAPI的详细描述。

总结

MicrochipZigBee协议栈提供了一种易于使用的不依赖于应用和RTOS的函数库。该函数库是专为仅需对上层软件做极小更改就可支持多个RF收发器而设计的。应用程序能容易地从一个RF收发器移植到另一个收发器。此外，它还自动支持MPLABC18和Hi-TechPICC-18C编译器。如有需要，能很容易地修改该协议栈以支持其他编译器。

USART 异步发送器

图 21-1 所示是 USART 发送器框图。发送器的核心是发送移位寄存器（TSR）。移位寄

存器从发送缓冲器 TXREG 获取要发送的数据。TXREG 寄存器由软件写入数据。前一次载入的停止位发送出去后，TSR 寄存器才会载入数据。停止位一发送出去，TXREG 中的新数据（如有）就会立即被载入 TSR。一旦把 TXREG 中的数据送入 TSR 寄存器（在一个 TCY 周期内发生），则 TXREG 寄存器为空，发送中断标志位 TXIF 被置位。可以通过将 TXIE 使能位置位或清零来允许或禁止这个中断。不管 TXIE 位的状态如何，TXIF 标志位都会被置位，且不能用软件清零。只有把新数据写入 TXREG 寄存器后，TXIF 位才会复位。TXIF 标志位表示 TXREG 寄存器的状态，而 TRMT 位（TXSTA 寄存器）则表示 TSR 寄存器的状态。TRMT 状态位是一个只读位，当 TSR 寄存器为空时被置位。任何中断逻辑对 TRMT 位都没有影响，所以要确定 TSR 寄存器是否为空，用户就必须对此位进行轮询。

将 TXEN 位（TXSTA 寄存器）置位就可以使能发送，但是只有在 TXREG 寄存器装入数据和波特率发生器（BRG）产生移位时钟之后才能进行数据发送（图 21-1）。也可以先载入 TXREG 寄存器，再置位 TXEN 启动发送。通常，在第一次开始发送时，TSR 寄存器为空，因此送到 TXREG 寄存器的数据会立即被送到 TSR 寄存器，导致 TXREG 寄存器为空，因此可以进行连续的背靠背发送（图 21-3）。在发送过程中将 TXEN 位清零将导致发送中止，并复位发送器，同时 TX/CK 引脚会恢复到高阻状态。

要选择 9 位发送模式，TX9 位（TXSTA 寄存器）应被置位，而且第 9 位应当写入 TX9D 位（TXSTA 寄存器）。必须先写第 9 位数据，然后将 8 位数据写入 TXREG 寄存器。这是因为如果 TSR 寄存器为空，向 TXREG 寄存器写数据会导致数据立即送入 TSR 寄存器。在这种情况下，装入 TSR 寄存器的第 9 位数据可能是错误的。

USART 异步接收器

图 21-4 为接收器框图。在 RX/DT 引脚上接收数据，并驱动数据恢复模块。数据恢复块实际是以 16 倍波特率高速工作的移位器，而主要的接收串行移位器是以位速率或 FOSC 工作的。此模式常用在 RS-232 系统中。

USART 模块具备特殊功能，以使多处理器进行通讯。当 RCSTA 寄存器中的 RX9 位被置位时，接收到 9 位数据，并且第 9 位写入 RSTA 寄存器中的 RX9D 状态位。可对端口进行编程，以使接收到停止位时，如果 RX9D 位置位，只激活串行端口中断。该功能通过置位 RCSTA 寄存器中的 ADDEN 位来使能，并可以下述方式用于多处理器系统。

在多处理器系统中要传输数据块，主处理器首先发送标识目标从处理器的一个地址字节。RX9D 位为 1 标识为地址字节，（而为 0 标识为数据字节）。如果在从处理器的 RCSTA 寄存器的 ADDEN 位被置位，所有的数据字节将被忽略。然而，如果接收到第 9 位为 1，即表示接收到的字节是地址，将中断从处理器，并将接收移位寄存器的内容传递到接收缓存器。这样，从处理器仅可被地址中断。因此，从处理器可以分析接收到的字节，查看自己的地址是否与该地址匹配。之后，地址匹配的从处理器将清零它的 ADDEN 位，并准备接收来自主处理器的数据字节。

当 ADDEN 位置 1 时，将忽略所有的数据字节。在停止位后，数据将不会载入到接收缓存器，且不产生中断。如果另一个字节移入到了 RSR 寄存器，以前的数据将丢失。

只有在接收器配置为 9 位模式时，ADDEN 位才生效。

一旦选择异步模式，将 CREN 位置 1 使能接收。

接收器的核心是接收（串行）移位寄存器（RSR）。在 RX/TX 引脚上采样到停止位之后，RSR 中接收到的数据被送到 RCREG 寄存器（如果 RCREG 寄存器为空）。数据传送完后，RCIF 标志位被置 1。将 RCIE 位置位（或清零）可允许（或屏蔽）接收中断。RCIF 标志位

是只读位，并可由硬件清零。它在 RCREG 寄存器被读取之后且寄存器为空时被硬件清零。RCREG 寄存器是一个双缓冲寄存器（即两级深 FIFO），因此可以实现接收两个字节的数据并传送到 RCREGFIFO，同时第三个字节开始移位到 RSR 寄存器。在检测到第三个字节的停止位后，如果 RCREGFIFO 仍然是满的，则溢出错误标志位 OERR 会置 1，RSR 寄存器中的数据丢失。可以对 RCREG 寄存器读两次，从而获得 FIFO 中的两个字节。OERR 位必须由软件清零，这可以通过复位接收逻辑（将 CREN 位清零后再置 1）来实现。如果 OERR 位置 1，则禁止 RSR 中的数据传送到 RCREG 寄存器，因此若 OERR 位被置 1，必须将它清零。如果检测到的停止位为低电平，帧出错标志位 FERR 将被置 1。FERR 位和接收到的第 9 位数据的缓冲方式与接收数据相同。读 RCREG 将使用新值写入 RX9D 和 FERR 位。因此，用户读 RCREG 寄存器前必须读取 RCSTA 寄存器，防止 FERR 和 RX9D 位先前（旧）的信息丢失。

ZigBee PROTOCOL OVERVIEW

ZigBee is a standard wireless network protocol designed for low rate control networks. Some of the applications for the ZigBee protocol include building automation networks, building security systems, industrial control networks, remote meter reading and PC peripherals. The following sections provide a brief overview of the ZigBee protocol that is relevant in understanding Microchip Stack functionality. Interested readers should refer to the ZigBee web site (www.zigbee.org) for more information.

IEEE 802.15.4

The ZigBee protocol uses IEEE 802.15.4 specifications as its Medium Access Layer (MAC) and Physical Layer (PHY). The IEEE 802.15.4 defines a total of three frequency bands of operations: 2.4 GHz, 915 MHz and868 MHz. Each frequency band offers a fixed number of channels. For example, the 2.4 GHz frequency band offers a total of 16 channels (channel 11-26), 915 MHz offers 10 channels (channel 1-10) and 868 MHz offers1 channel (channel 0).

The bit rate of the protocol depends on the selection of frequency of operation. The 2.4 GHz band provides250 kbps, 915 MHz provides 40 kbps and 868 MHz provides a 20 kbps data rate. The actual data throughput would be less than the specified bit rate due to the packet overhead and processing delays.

The maximum length of IEEE 802.15.4 MAC packets 127 bytes. Each packet consists of header bytes and a 16-bit CRC value.

The 16-bit CRC value verifies the frame integrity. In addition, IEEE 802.15.4 optionally uses an acknowledged data transfer mechanism. With this method, all frames with a special ACK flag set are Acknowledged by its receiver. This makes sure that a frame is in fact delivered. If the frame is transmitted with an ACK flag set and the Acknowledgement is not received within a certain time-out period, the transmitter will retry the transmission for a fixed number of times before declaring an error. It is important to note that the reception of an Acknowledgement simply indicates that a frame was properly received by the MAC layer. It does not, however, indicate that

the frame was processed correctly. It is possible that the MAC layer of the receiving node received and acknowledged a frame correctly, but due to the lack of processing resources, a frame might be discarded by upper layers. As a result, many of the upper layers and application require additional Acknowledgement response.

Network Configurations

A ZigBee wireless network may assume many types of Configurations. A star network configuration consists of one coordinator node (a "master") and one or more end devices ("slaves"). The coordinator is a special variant of a Full Function Device (FFD) that implements a larger set of ZigBee services. The end devices may be FFD or a Reduced Function Device (RFD). An RFD is the smallest and simplest ZigBee node. It implements only a minimal set of ZigBee services. In a star network, all end devices communicate to the coordinator only. If an end device needs to transfer data to another end device, it sends its data to the coordinator and the coordinator, in turn, forwards the data to the intended receiver end device.

In addition to the star network, a ZigBee network may also assume peer-to-peer, cluster, or mesh network configurations. The cluster and mesh network are also known as a multi-hop network, due to their abilities to route packets between multiple networks, while the star network is called a single-hop network.

As with any network, a ZigBee network is a multi-access network, meaning that all nodes in a network have equal access to the medium of communication. There are two types of multi-access mechanisms. In a non-beacon enabled network, all nodes in a network are allowed to transmit at any time as long as the channel is Idle. In a beacon enabled network, nodes are allowed to transmit in predefined time slots only. The coordinator periodically begins with a superframe identified as a beacon frame and all nodes in the network are expected to synchronize to this frame. Each node is assigned a specific slot in the superframe during which it is allowed to transmit and receive its data. A superframe may also contain a common slot during which all nodes compete to access the channel. he current version of the Microchip Stack supports non-beacon star network configuration only.

Network Association

ZigBee networks can be ad-hoc, meaning that a new network is formed and unformed as needed. In a star network configuration, end devices would always search for a network before they can perform any data transfer. A new network is first established by a coordinator. On start-up, a coordinator searches for other coordinators nearby and if none is found, it establishes its own network and selects a unique 16-bit PAN ID. Once a new network is established, one or more end devices are allowed to associate with the network. The exact decision to allow or disallow new associations depends on the coordinator. Once a network is formed, it is possible that due to the physical changes, more than one network may overlap and a PAN ID conflict may arise. In that situation, a coordinator may initiate a PAN ID conflict resolution procedure and one of the coordinators would change its PAN ID and/or channel. The affected coordinator would instruct all of its end devices to make the necessary changes. The current version of the Microchip Stack does not support PAN ID conflict resolution. Depending on system requirements, a coordinator may

store all of the network associations in nonvolatile memory, called a neighbor table. In order to connect to a network, an end device may either execute the orphan notification procedure to locate its previously associated network or execute the association procedure to join a new network. In the case of the orphan notification procedure, the coordinator will recognize a previously associated end device by looking up its neighbor table. Once associated to a network, an end device may choose to disassociate from the network by performing the disassociate procedure. If required, a coordinator itself may also initiate a disassociate procedure to force a node to leave the network. The current version of the Microchip Stack supports new association and orphan notification procedures. It only supports a network leave procedure initiated by an end device.
Endpoints, Interfaces, Clusters, Attributes and Profiles

A typical ZigBee node may support multiple features and functionality. For example, an I/O node may have multiple digital and analog inputs/outputs. Some of the digital inputs may be used by a remote controller node and others may be used by another remote controller node. This arrangement creates a truly distributed control network. To facilitate data transfer between the I/O node and two controller nodes, the applications in all nodes must maintain multiple data links. In order to reduce cost, a ZigBee node uses only one radio channel and multiple endpoint/interfaces to create multiple virtual links or channels.

One ZigBee node supports 31 endpoints (numbered 0-31) and 8 interfaces (numbered 0-7). The endpoint 0 is reserved for device configuration and endpoint 31 is reserved for broadcasts only. This leaves a total of 30 endpoints for application use. For each endpoint, there can be a total of 8 interfaces. Thus, in reality, an application may have up to 240 virtual channels in one physical channel.

A typical ZigBee node would also have many attributes. For example, our I/O node contains attributes called digital input #1, digital input #2, analog input #1, etc. Each attribute would have its own value. For example, the digital input #1 attribute may have a value of '1' or '0'. A collection of attributes is called a cluster. Each cluster is assigned a unique cluster ID in the entire network. Each cluster may have up to 65,536 attributes.

The ZigBee protocol also defines a term called profile. A profile is synonymous to the description of a distributed application. It describes a distributed application in terms of the packets it must handle and actions it must perform. A profile is described using a descriptor, which is nothing but a complex structure of various values. It is the profile that makes ZigBee devices interoperable. The ZigBee Alliance has defined many standard profiles, such as remote control switch profile, light sensor profile, etc. Any node that conforms to one of these standard profiles will be interoperable with other nodes implementing the same profile. The current version of the Microchip Stack does not provide any standard profile functionality. If required, you may easily write a cooperative task function that implements the desired profile. You may also create your own custom profile (or distributed application) that works with your proprietary nodes only. The demo application provided with this application note implements a custom distributed application of a remote controlled LED, where an LED of one node is controlled by a switch on another node. Each profile can define up to 256 clusters and as we saw earlier, each cluster can have up to 65,536 attributes.

This flexibility allows a node to have a very large number of attributes (or I/O points).

Endpoint Binding

As mentioned earlier, end devices in a star network always communicate to the coordinator only. The coordinator is responsible for forwarding the data packet sent from an endpoint from one node to the appropriate endpoint(s) in the receiving end device. As you might have guessed, when a new network is established, the coordinator must be told how to create source and destination endpoint links. The ZigBee protocol defines a special procedure called endpoint binding. As a part of the binding process, a remote network/device manager-like node may ask the coordinator to modify its binding table. The coordinator node maintains a binding table that essentially contains a logical link between two or more endpoints. Each link is uniquely defined by its source endpoint and cluster ID.

For example, if the data from digital input #1 of our I/O node needs to be sent to control channel #1 of the controller node, we must ask the coordinator to create a binding table entry that consists of digital input #1 endpoint of the I/O node as a source and control channel #1 of the controller node as a destination. Once a binding table entry is created, any time the I/O node sends data from its digital input #1 endpoint, the coordinator node will look up its binding table and forward the packet to the control channel #1 endpoint of the controller node. Both digital input #1 and control channel #1 will share a common cluster ID. Depending on how the binding table is created, it is possible to multicast data from one endpoint to multiple endpoints on multiple nodes. The current version of the Microchip Stack does not support such multicast binding table entries.

The ZigBee protocol defines a special software object, called the ZigBee Device Object (ZDO), that provides binding services among other services. Only the ZDO running on the coordinator will provide the binding services. A remote network/device manager would issue a special binding request directed to the ZDO (endpoint 0) to create or modify a binding table entry. As per the ZigBee specifications, a PC or other highend controller running special ZigBee node software may act as a network manager.

If you do not want to create or use a special network manager node, you may write your own custom binding services that simplify the binding procedure. The demo applications included with this application note implement a simple custom binding method. It enables each node to send its own binding request to the coordinator node to the endpoint 0 and defines its own custom cluster ID. For more information, please see the CUSTOM_DEMO_BIND cluster ID in the ZDO.c file. According to this custom binding procedure, the end device must be in Configuration mode to send binding requests. The coordinator can receive and originate its own binding requests in normal mode of execution.

When a certain sequence of switch presses is detected, the end device sends out a special binary data structure using the CUSTOM_DEMO_BIND as a cluster ID (refer to “Configuring Demo Applications” to learn about the Configuration mode and binding sequence). It directly sends the binding request packet to the endpoint 0 of the coordinator. The ZDO in the demo coordinator receives the packet identified as the CUSTOM_DEMO_BIND cluster ID and delegates the processing to the ProcessCustomBind function. This function is actually implemented in the DemoCoordApp.c file. You may easily follow the execution logic of the ProcessCustomBind function to fully understand the custom binding concept. You may use this

custom binding logic as it is or write your own using it as a reference.

Data Transfer Mechanism

Depending on the type of network, exact mechanisms to transfer data to and from the end device differ. In a non-beacon star network, when an end device wants to send a data frame, it simply waits for the channel to become idle. Upon detecting an Idle channel condition, it transmits its frame to the coordinator. If a coordinator wants to send data to an end device, it holds the data frame in its transmit buffer until the intended end device explicitly polls for the data. This method ensures that the receiver of the end device is turned ON and it is capable of receiving data from the coordinator.

In a peer-to-peer network, each node must either keep their receiver ON all the time or agree upon an interval period during which they will switch ON their receivers. This will allow a node to transmit a data frame and ensure that the frame will be received by the other node.

The fact that the end device must poll the coordinator for its data, rather than keep its receiver ON, allows the end devices to lower their power requirement. Depending on the application requirements, an end device may spend most of its time sleeping and only periodically wake-up to transmit or receive data. The one disadvantage to this approach is that the coordinator must hold all data frames in its internal buffer until the intended end device wakes up and polls for the data. If a network contains many end devices that sleep for long time, the coordinator must keep the data frame for that long period. Depending on the number of nodes and rate of data frame exchanges, this would drastically increase the coordinator RAM requirements. A coordinator may selectively decide to hold a specific frame for a long time, or a short time, based on the device descriptor for the end device. The ZigBee protocol requires that all end devices will maintain various descriptors that describe various aspects of their features and capabilities. The current version of the Microchip Stack does not support descriptors.

STACK ARCHITECTURE

The Microchip Stack is written in the C programming language, intended for both MPLAB C18 and Hi-Tech PICC-18 compilers. Depending on which is used, the source files automatically make the required changes. The Microchip Stack is designed to run on Microchip's PIC18F family of microcontrollers only. The Microchip Stack uses internal Flash program memory to store its configurable MAC address, network table and binding table. Consequently, you must use a selfprogrammable Flash memory microcontroller. If required, you may modify the Nonvolatile Memory (NVM) routines to support any other type of NVM and not use a self-programmable microcontroller. In addition, the Stack is targeted to run on the PICDEM Z demonstration board. However, it can be easily ported to any hardware equipped with a compatible microcontroller.

Stack Layers

The Microchip Stack divides its logic into multiple layers as defined by the ZigBee specification. The code implementing each layer resides in a separate source file, while the services and Application Programming Interfaces (APIs) are defined in the include file.

The current version of the Stack does not implement a security layer. Each layer defines a set

of easy to understand functions to its immediate upper layer. To create an abstraction and modularity, a top-level layer always interacts with a layer immediately below it through well-defined APIs. A C header file for a specific layer (zAPS.h, for example) defines all APIs supported by that specific layer. With that in mind, a user application would always interact with the Application Programming Support (APS) layer and Application Layer (APL). Many of the APIs provided by each layer are simply C macros that call functions one layer down. This method avoids the typical overhead associated with modularization.

Stack APIs

The Microchip Stack consists of many modules. A typical application would always interface to the Application Layer (APL) and the Application Support Sublayer (APS). However, if required, you may easily interface to other modules and/or customize them as needed. The following sections provide detailed API descriptions of the APL and APS modules only. If required, you may learn the details of APIs for other modules in their respective header files. For up-to-date information, you should refer to the actual source files.

APPLICATION LAYER (APL)

The APL module provides the high-level Stack management functions. A user application would use this module to manage the Stack functionality. The zAPL.c file implements the APL logic and the zAPL.h file defines the APIs supported by the APL module. A user application would include the zAPL.h header file to access its APIs.

NETWORK LAYER

The Network Layer (NWK) is responsible to establish and maintain network connection. It independently handles incoming data request, association, disassociation and orphan notification requests.

A typical application would not need to make direct calls to the NWK layer. If required, you may refer to the NWK.c and NWK.h files for a detailed description of NWK APIs.

ZigBee DEVICE OBJECT

The ZigBee Device Object (ZDO) opens and handles the EP 0 interface. The ZDO is responsible for receiving and processing various requests from a remote device. Unlike other endpoints, EP 0 is always opened on start-up and assumed to be bound to any incoming data frames that are directed to EP 0.

The ZDO object allows remote device management services. A remote device manager would issue requests to EP 0 and the ZDO would process those requests. Some examples of remote services are NWK_ADDR_REQ (give me your network address), NODE_DESC_REQ (give me your node descriptor) and BIND_REQ (bind this source EP, cluster ID and destination EP). Some of the requests are available to the coordinator device only. Refer to the ZDO.c file for the complete list. You should also refer to the ZigBee specification for more information.

An application would not need to call any of the ZDO functions directly except to initialize it on start-up. If required, you may refer to the ZDO.c and ZDO.h files for detailed descriptions of

ZDO APIs.

ZigBee DEVICE PROFILE LAYER

The ZigBee device profile layer provides standard ZigBee profile services. It defines and processes descriptor requests. A remote device may request any of the standard descriptor information via the ZDO interface. Upon receiving such requests, ZDO would call a profile object to retrieve the corresponding descriptor value. In the current version, the device profile layer is not fully implemented.

An application would not need to call any of the profile functions directly. If required, you may refer to the zProfile.c and zProfile.h files for detailed description of APIs.

MEDIUM ACCESS CONTROL LAYER

The Medium Access Control (MAC) layer implements functions required by the IEEE 802.15.4 specification. The MAC layer is responsible for interacting with the Physical Layer (PHY). In order to support different types of RF transceivers, the Microchip Stack separates PHY interactions in its separate file. There is a separate file for each supported transceiver. Note that due to the differences in RF transceiver capabilities, MAC and PHY files are not completely independent. The MAC file adjusts some of its logic based on the current RF transceiver. In the current version of the Stack, the zPHYCC2420.c file implements Chipcon CC2420, 2.4 GHz transceiver specific functions. In the future, as support for new RF transceivers is added, a new PHY file will be added. All RF transceiver PHY files use the zPHY.h file as their main interface to the MAC layer. A typical application would not need to make direct calls to the MAC layer. If required, you may refer to the MAC.c, MAC.h, and PHY.h files for detailed descriptions of MAC/PHY APIs.

CONCLUSION

The Microchip Stack for ZigBee Protocol provides a modular, easy to use library that is application and RTOS independent. It is specifically designed to support more than one RF transceiver with minimal changes to upper layer software. Applications can be easily ported from one RF transceiver to another. In addition, it automatically supports MPLAB C18 and Hi-Tech PICC-18 C compilers. If required, it can be easily modified to support other compilers.

USART Asynchronous Transmitter

The USART transmitter block diagram is shown in Figure 21-1. The heart of the transmitter is the Transmit Shift Register (TSR). The shift register obtains its data from the transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one TCY), the TXREG register is empty and the TXIF flag bit is set. This interrupt can be enabled/disabled by setting/clearing the TXIE enable bit. The TXIF flag bit will be set, regardless of the state of the TXIE enable bit and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While

the TXIF flag bit indicated the status of the TXREG register, the TRMT bit (TXSTA register) shows the status of the TSR register. The TRMT status bit is a read only bit, which is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty.

Transmission is enabled by setting the TXEN enable bit (TXSTA register). The actual transmission will not occur until the TXREG register has been loaded with data and the Baud Rate Generator (BRG) has produced a shift clock (Figure 21-1). The transmission can also be started by first loading the TXREG register and then setting the TXEN enable bit. Normally when transmission is first started, the TSR register is empty, so a transfer to the TXREG register will result in an immediate transfer to TSR, resulting in an empty TXREG. A back-to-back transfer is thus possible (Figure 21-3). Clearing the TXEN enable bit during a transmission will cause the transmission to be aborted and will reset the transmitter. As a result, the TX/CK pin will revert to hi-impedance.

In order to select 9-bit transmission the TX9 bit (TXSTA register) should be set and the ninth bit should be written to the TX9D bit (TXSTA register). The ninth bit must be written before writing the 8-bit data to the TXREG register. This is because a data write to the TXREG register can result in an immediate transfer of the data to the TSR register (if the TSR is empty). In such a case, an incorrect ninth data bit may be loaded in the TSR register.

USART Asynchronous Receiver

The receiver block diagram is shown in Figure 21-4. The data is received on the RX/DT pin and drives the data recovery block. The data recovery block is actually a high speed shifter operating at x16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at FOSC. This mode would typically be used in RS-232 systems.

The USART module has a special provision for multi-processor communication. When the RX9 bit is set in the RCSTA register, 9-bits are received and the ninth bit is placed in the RX9D status bit of the RSTA register. The port can be programmed such that when the stop bit is received, the serial port interrupt will only be activated if the RX9D bit is set. This feature is enabled by setting the ADDEN bit in the RCSTA register and can be used in a multi-processor system in the following manner.

To transmit a block of data in a multi-processor system, the master processor must first send an address byte that identifies the target slave. An address byte is identified by the RX9D bit being a ‘1’ (instead of a ‘0’ for a data byte). If the ADDEN bit is set in the slave’s RCSTA register, all data bytes will be ignored. However, if the ninth received bit is equal to a ‘1’, indicating that the received byte is an address, the slave will be interrupted and the contents of the Receive Shift Register (RSR) will be transferred into the receive buffer. This allows the slave to be interrupted only by addresses, so that the slave can examine the received byte to see if it is addressed. The addressed slave will then clear its ADDEN bit and prepare to receive data bytes from the master.

When the ADDEN bit is set, all data bytes are ignored. Following the STOP bit, the data will not be loaded into the receive buffer and no interrupt will occur. If another byte is shifted into the RSR register, the previous data byte will be lost.

The ADDEN bit will only take effect when the receiver is configured in 9-bit mode.

Once Asynchronous mode is selected, reception is enabled by setting the CREN bit.

The heart of the receiver is the Receive (serial) Shift Register (RSR). After sampling the RX/TX pin for the STOP bit, the received data in the RSR is transferred to the RCREG register (if it is empty). If the transfer is complete, the RCIF flag bit is set. The actual interrupt can be enabled/disabled by setting/clearing the RCIE enable bit. The RCIF flag bit is a read only bit that is cleared by the hardware. It is cleared when the RCREG register has been read and is empty. The RCREG is a double-buffered register (i.e., it is a two-deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte begin shifting to the RSR register. On the detection of the STOP bit of the third byte, if the RCREG register is still full, overrun error bit, OERR, will be set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. The OERR bit has to be cleared in software. This is done by resetting the receive logic (the CREN bit is cleared and then set). If the OERR bit is set, transfers from the RSR register to the RCREG register are inhibited, so it is essential to clear the OERR bit if it is set. Framing error bit, FERR, is set if a stop bit is detected as a low level. The FERR bit and the 9th receive bit are buffered the same way as the receive data. Reading the RCREG will load the RX9D and FERR bits with new values. Therefore, it is essential for the user to read the RCSTA register before reading the next RCREG register, in order not to lose the old (previous) information in the FERR and RX9D bits.