物联网基础实验平台

E-BOX300 实验指导书

大唐移动通信设备有限公司

目 录

第1章 物联网实验概述

1.1 物联网基础知识

1.1.1 物联网的概念

1.1.1.1 物联网的定义

物联网，英文名：Internet of Things（IOT），也称为 Web of Things。它是指通过各种信息传感设备，如传感器、射频识别（RFID）技术、全球定位系统、红外感应器、激光扫描器、气体感应器等各种装置与技术，实时采集任何需要监控、连接、互动的物体或过程，采集其声、光、热、电、力学、化学、生物、位置等各种需要的信息，与互联网结合形成的一个巨大网络。如图 1-1 所示。

图1-1 物联网

必须注意的是，物联网中的“物”不是普通意义上的万事万物，这里的“物”要满足一定条件才能够被纳入物联网范围：

1. 要有数据传输通路；
2. 要有相应信息的接收器；
3. 要有一定的存储功能；
4. 要有处理运算单元 CPU；

5. 要有操作系统；

6. 要有专门的应用程序；

7. 要有数据发送器；

8. 遵循物联网的通信协议；

9. 在世界网络中有可被识别的唯一编号。

物联网作为新一代信息技术的重要组成部分，自诞生以来，已经引起了巨大关注，被认为是继计算机、互联网、移动通信网之后的又一次信息产业浪潮。物联网是中国人的发明，整合了美国 CPS（Cyber-Physical Systems）、欧盟 IOT（Internet of Things）和日本 U-Japan 等概念，是一个基于互联网、传统电信网等信息载体，让所有能被独立寻址的普通物理对象实现互联互通的网络。它被视为互联网的应用扩展，应用创新是物联网的发展的核心，以用户体验为核心的创新是物联网发展的灵魂。物联网将是下一个推动世界高速发展的“重要生产力“！

1.1.1.2 物联网的分类

1. 私有物联网（Private IoT）： 一般面向单一机构内部提供服务；

2. 公有物联网（Public IoT）：基于互联网向公众或大型用户群体提供服务；

3. 社区物联网（Community IoT）：向一个关联的“社区”或机构群体（如一个城市政府下属的各委办局：公安局、交通局、环保局、城管局等）提供服务；

4. 混合物联网（Hybrid IoT）：是上述的两种或以上的物联网的组合，但后台有统一运维实体。

1.1.1.3 物联网的技术架构

从技术架构上来看，物联网可分为三层：感知（互动）层、网络（传输）层和应用（服务）层（如图 1-2）：

图1-2　　物联网技术架构图

1.感知（互动）层由各种传感器以及传感器网关构成，包括二氧化碳浓度传感器、温度传感器、湿度传感器、二维码标签、RFID 标签和读写器、摄像头、GPS 等感知终端。感知层的作用相当于人的眼耳鼻喉和皮肤等神经末梢，它是物联网识别物体、采集信息的来源，其主要功能是识别物体，采集信息，并对信息进行初步的融合。

2.网络（传输）层包括接入层和核心层两个子层，由各种私有网络、互联网、有线和无线通信网、网络管理系统和云计算平台等组成，相当于人的神经中枢和大脑，负责传递和处理感知层获取的信息。

3.应用（服务）层是物联网和用户（包括人、组织和其他系统）的接口，它与行业需求结合，实现物联网的智能应用。

1.1.2　物联网的应用

1.1.2.1 物联网的应用模式

根据其实质用途可以归结为三种基本应用模式：

A.　对象的智能标签。

通过二维码、RFID 等技术标识特定的对象，用于区分对象个体，例如在生活中我们使用的各种智能卡，条码标签的基本用途就是用来获得对象的识别信息；此外通过智能标签还可以用于获得对象物品所包含的扩展信息，例如智能卡上的金额余额，二维码中所包含的网址和名称等。

B、环境监控和对象跟踪。

利用多种类型的传感器和分布广泛的传感器网络，可以实现对某个对象的实时状态的获

取和特定对象行为的监控，如使用分布在市区的各个噪音探头监测噪声污染，通过二氧化碳传感器监控大气中二氧化碳的浓度，通过 GPS 跟踪车辆位置，通过交通路口的摄像头捕捉实时交通流量等。

C、 对象的智能控制。

物联网基于云计算平台和智能网络，可以依据传感器网络获取的数据进行决策，改变对象的行为，进行控制和反馈。例如根据光线的强弱调整路灯的亮度，根据车辆的流量自动调整红绿灯间隔等。

物联网是一项综合性技术，一般来讲，物联网的开展步骤主要如下 ：

1.对物体属性进行标识，属性包括静态和动态的属性，静态属性可以直接存储在标签中，动态属性需要先由传感器实时探测；

2.需要识别设备完成对物体属性的读取，并将信息转换为适合网络传输的数据格式；

3.将物体的信息通过网络传输到信息处理中心（处理中心可能是分布式的，如家里的电脑或者手机，也可能是集中式的，如中国移动的 IDC），由处理中心完成相关计算处理。

1.1.2.2 物联网的应用领域

物联网用途广泛，遍及智能交通、环境保护、政府工作、公共安全、平安家居、智能消防、工业监测、环境监测、老人护理、个人健康、花卉栽培、水系监测、食品溯源、敌情侦查和情报搜集等多个领域。

国际电信联盟于 2005 年的报告曾描绘“物联网”时代的图景：当司机出现操作失误时汽车会自动报警；公文包会提醒主人忘带了什么东西；衣服会“告诉”洗衣机对颜色和水温的要求等等。物联网在物流领域内的应用则比如：一家物流公司应用了物联网系统的货车，当装载超重时，汽车会自动告诉你超载了，并且超载多少，但空间还有剩余，告诉你轻重货怎样搭配；当搬运人员卸货时，一只货物包装可能会大叫“你扔疼我了”，或者说“亲爱的，请你不要太野蛮，可以吗？”；当司机在和别人扯闲话，货车会装作老板的声音怒吼“笨蛋，该发车了！”。

物联网把新一代 IT 技术充分运用在各行各业之中，具体地说，就是把感应器嵌入和装备到电网、铁路、桥梁、隧道、公路、建筑、供水系统、大坝、油气管道等各种物体中，然后将“物联网”与现有的互联网整合起来，实现人类社会与物理系统的整合，在这个整合的网络当中，存在能力超级强大的中心计算机群，能够对整合网络内的人员、机器、设备和基础设施实进行时的管理和控制，在此基础上，人类可以以更加精细和动态的方式管理生产和生活，达到“智慧”状态，提高资源利用率和生产力水平，改善人与自然间的关系。

人们正走向“物联网”时代，这个过程可能需要很长的时间，但毫无疑问，如果“物联网”时代来临，人们的日常生活将发生翻天覆地的变化。

1.2 传感器基础知识

1.2.1 传感器的概念

传感器（transducer / sensor）是一种物理装置或生物器官，能够探测、感受外界的信号、物理条件（如光、热、湿度）或化学组成（如烟雾），并将探知的信息传递给其他装置或器官。传感器是通过敏感元件，把外部物理世界的信息按适当比例变换成电信号的一种重要的电子部件。“传感器”在新韦式大词典中定义为：“从一个系统接受功率，通常以另一种形式将功率送到第二个系统中的器件”。根据这个定义，传感器的作用是将一种能量转换成另一种能量形式，它是实现自动检测和自动控制的首要环节。

1.2.2 传感器的作用

人们为了从外界获取信息，必须借助于感觉器官。而单靠人们自身的感觉器官，在研究自然现象和规律以及生产活动中它们的功能就远远不够了。为适应这种情况，就需要传感器。因此可以说，传感器是人类五官的延长，又称之为电五官。

新技术革命的到来，世界开始进入信息时代。在利用信息的过程中，首先要解决的就是要获取准确可靠的信息，而传感器是获取自然和生产领域中信息的主要途径与手段。

在现代工业生产尤其是自动化生产过程中，要用各种传感器来监视和控制生产过程中的各个参数，使设备工作在正常状态或最佳状态，并使产品达到最好的质量。因此可以说，没有众多的优良的传感器，现代化生产也就失去了基础。

传感器早已渗透到诸如工业生产、宇宙开发、海洋探测、环境保护、资源调查、医学诊断、生物工程、甚至文物保护等等极其之泛的领域。可以毫不夸张地说，从茫茫的太空，到浩瀚的海洋，以至各种复杂的工程系统，几乎每一个现代化项目，都离不开各种各样的传感器。

1.2.3 传感器的组成与分类

1.2.3.1 传感器的组成

传感器由两个基本原件组成：敏感元件与转换元件。

在非电量到电学物理量的转换过程中，并非所有的非电量参数都能一次性直接变换为电学物理量，往往是先变换成一种易于变换成电学物理量的非电量（如位移、应变等），然后再通过适当的方法变换成电学物理量。所以把能够完成预变换的单元称为敏感元件。在传感器中，建立在力学结构分析上的各种类型的弹性元件（如梁、板等）统称为弹性敏感元件。而转换元件是能将感觉到的被测非电量转换成电学物理量的，如应变计、压电晶体、热电偶等。转换元件是传感器的核心部分，是利用各种物理、化学、生物效应等原理制成的。新发现的

物理、化学、生物效应也常被用到新型传感器上，使其品种与功能日益增多应用领域更加广泛。

应该指出的是并不是所有的传感器都包括敏感元件与转换元件，有一部分传感器不需要起变换作用的敏感元件如热敏电阻、光敏器件等。

1.2.3.2 传感器的分类

传感器种类繁多，按照其敏感原理、敏感材料、工艺设备和计测技术都会衍生出不同种类的传感器，而不同行业对同类传感器的性能要求也是千差万别。

常将传感器的功能与人类 5 大感觉器官相比拟：

- 光敏传感器——视觉
- 声敏传感器——听觉
- 气敏传感器——嗅觉
- 化学传感器——味觉
- 压敏、温敏、流体传感器——触觉

我们可以用不同的观点对传感器进行分类，通常据其基本感知功能可分为热敏元件、光敏元件、气敏元件、力敏元件、磁敏元件、湿敏元件、声敏元件、放射线敏感元件、色敏元件和味敏元件等十大类（还有人曾将敏感元件分 46 类）。我国的传感器有近 6000 个品种，而国外品种更多（美国约有 1.7 万种）。

1. 传感器按照其用途分类：

- 压力敏和力敏传感器、位置传感器
- 液面传感器、能耗传感器
- 速度传感器、加速度传感器
- 射线辐射传感器、热敏传感器
- 24GHz 雷达传感器

2. 传感器按照其原理分类：

- 振动传感器
- 湿敏传感器
- 磁敏传感器
- 气敏传感器
- 真空度传感器
- 生物传感器等

3. 传感器按照其输出信号为标准分类：

- 模拟传感器——将被测量的非电学量转换成模拟电信号。
- 数字传感器——将被测量的非电学量转换成数字输出信号(包括直接和间接转换)。
- 开关传感器——当一个被测量的信号达到某个特定的阈值时，传感器相应地输出一个设定的低电平或高电平信号。

4. 传感器根据测量目的不同分类：

- 物理型传感器是利用被测量物质的某些物理性质发生明显变化的特性制成的。
- 化学型传感器是利用能把化学物质的成分、浓度等化学量转化成电学量的敏感元件制成的。
- 生物型传感器是利用各种生物或生物物质的特性做成的，用以检测与识别生物体内化学成分的传感器。

1.2.4 传感器的基本特性

1.2.4.1 传感器静态特性

传感器的静态特性是指对静态的输入信号，传感器的输出量与输入量之间所具有相互关系。因为这时输入量和输出量都和时间无关，所以它们之间的关系，即传感器的静态特性可用一个不含时间变量的代数方程，或以输入量作横坐标，把与其对应的输出量作纵坐标而画出的特性曲线来描述。表征传感器静态特性的主要参数有：线性度、灵敏度、迟滞、重复性、漂移等。

A、线性度

指传感器输出量与输入量之间的实际关系曲线偏离拟合直线的程度。定义为在全量程范围内实际特性曲线与拟合直线之间的最大偏差值与满量程输出值之比。

B、灵敏度

灵敏度是传感器静态特性的一个重要指标。其定义为输出量的增量与引起该增量的相应输入量增量之比。用 S 表示灵敏度。

C、迟滞

传感器在输入量由小到大（正行程）及输入量由大到小（反行程）变化期间其输入输出特性曲线不重合的现象成为迟滞。对于同一大小的输入信号，传感器的正反行程输出信号大小不相等，这个差值称为迟滞差值。

D、重复性

重复性是指传感器在输入量按同一方向作全量程连续多次变化时，所得特性曲线不一致的程度。

E、漂移

传感器的漂移是指在输入量不变的情况下，传感器输出量随着时间变化，此现象称为漂移。产生漂移的原因有两个方面：一是传感器自身结构参数；二是周围环境（如温度、湿度等）。

1.2.4.2 传感器动态特性

所谓动态特性，是指传感器在输入变化时，它的输出的特性。在实际工作中，传感器的动态特性常用它对某些标准输入信号的响应来表示。这是因为传感器对标准输入信号的响应容易用实验方法求得，并且它对标准输入信号的响应与它对任意输入信号的响应之间存在一定的关系，往往知道了前者就能推定后者。最常用的标准输入信号有阶跃信号和正弦信号两种，所以传感器的动态特性也常用阶跃响应和频率响应来表示。

1.2.5 几种常用传感器介绍

1.2.5.1 温湿度传感器

1. 温湿度传感器的定义：

温湿度传感器是指将温度量和湿度量转换成容易被测量处理的电信号的设备或装置。市场上的温湿度传感器一般是测量温度量和相对湿度量。

2. 温度传感器的分类：

2.1 接触式温度传感器：接触式传感器直接与被测物体接触进行温度测量，由于被测物体的热量传递给传感器，降低了被测物体温度，特别是被测物体热容量较小时，测量精度较低。因此采用这种方式要测得物体的真实温度的前提条件是被测物体的热容量要足够大。

2.2 非接触式温度传感器：非接触式温度传感器主要是利用被测物体热辐射而发出红外线，从而测量物体的温度，可进行遥测。其制造成本较高，测量精度却较低。优点是：不从被测物体上吸收热量；不会干扰被测对象的温度场；连续测量不会产生消耗；反应快等。

2.3 此外，还有微波测温传感器、噪声测温传感器、温度图测温传感器、热流计、射流测温计、核磁共振测温计、穆斯保尔效应测温计、约瑟夫逊效应测温计、低温超导转换测温计、光纤温度传感器等。这些温度传感器有的已获得应用，有的尚在研制中。

3. 温湿度传感器的应用领域：

温湿度传感器广泛应用于工控行业（如暖通空调 HVAC、自动控制等）、测试及检测设备、汽车、数据记录器、消费品（如食品、烟草、药品等）的储存、气象站、家电、湿度调节器、医疗、档案管理（与排风机、除湿器、加热器等配套 ）、温室大棚、动物养殖等各个领域。

1.2.5.2 光传感器

1. 光传感器的定义：

光传感器是一种能够把光信号转换成可用输出信号的设备。

2. 光传感器的分类：

2.1 环境光传感器：环境光传感器可以感知周围光线情况，并告知处理芯片自动调节显示器背光亮度，降低产品的功耗。另一方面，环境光传感器有助于显示器提供柔和的画面。当外界环境亮度较高时，使用环境光传感器的液晶显示器会自动调成高亮度。当外界环境较暗时，显示器就会调成低亮度。环境光传感器需要在芯片上贴一个红外截止膜，甚至直接在硅片上镀制图形化的红外截止膜。环境光传感器提供了广泛环境亮度条件下的精确光度检测，广泛应用于手机、液晶电视面板和笔记本电脑应用中的 LCD 背光控制。

2.2 红外光传感器：红外线传感器使用充电的热电堆与溴碘化铊（KRS-5）窗口来感应 580 到 40000nm 的波长。

2.3 太阳光传感器：太阳光传感器可识别水平，垂直各 360 度，太阳所在的位置；识别阴天，多云天，半阴天，晴天，白天以及晚上，跟踪方位识别。

2.4 紫外光传感器：紫外光传感器使用一个过滤片测量紫外光波段（315nm-400nm）。除去滤光片，传感器可同时感应可见光。

2.5 光传感器中还有一类接近传感器。接近传感器是代替限位开关等接触式检测方式，以无需接触监测对象为目的的传感器的总称。它能检测对象的移动信息和存在信息转换为电气信号。

3. 光传感器的应用领域：

- 背光调节：电视机、电脑显示器、LCD 背光、手机、数码相机、MP4、PDA、GPS；
- 节能控制：室外广告机、感应照明器具、玩具；
- 仪器、仪表：测量光照度的仪器及工业控制；
- 环保替代：替代传统光敏电阻、光敏二极管、光敏三极管。
- 接近传感器在设计上主要用来检测目标物是否出现，或者进行各种工业、汽车、电子产品的运动检测。

1.2.5.3 烟雾传感器

1. 烟雾传感器的定义：

烟雾传感器（smoke transducer）是指将烟雾浓度变量转换成有一定对应关系的输出信号的装置。

2. 烟雾传感器的分类：

2.1 离子式烟雾传感器：离子式烟雾传感器是一种技术先进，工作稳定可靠的传感器，

2.2 光电式烟雾传感器：该探测器的检测室内装有发光器件及受光器件。光电感烟探测器可分为减光式和散射光式。

2.3 气敏式烟雾传感器：气敏传感器是一种检测特定气体的传感器。它主要包括半导体气敏传感器、接触燃烧式气敏传感器和电化学气敏传感器等，其中用得最多的是半导体气敏传感器。气敏传感器还可以分为以下几种类型：

- 可燃性气体气敏元件传感器，包含各种烷类和有机蒸汽气体，目前大量应用于抽油烟机、泄露报警器和空气清新机；
- 一氧化碳气敏元件传感器，可用于工业生产、环保、汽车、家庭等一氧化碳泄漏和不完整燃烧检测报警；
- 氧传感器，在环保、医疗、冶金、交通等领域需求很大；
- 毒性气体传感器，主要用于检测烟气、尾气、废气等环境污染气体。

3. 烟雾传感器的应用领域：

- 离子式烟雾传感器被广泛运用到各消防报警系统中，应用在城市安防、小区、工厂、公司、学校、家庭、别墅、仓库、资源、石油、化工、燃气输配等众多领域，性能远优于气敏电阻类的火灾报警器。
- 光电烟雾报警器对稍大的烟雾粒子的感应比较灵敏，所以在容易发生闷烧火灾的场所比较适用。
- 气敏式烟雾传感器主要应用在：一氧化碳气体的检测、瓦斯气体的检测、煤气的检测、氟利昂的检测、呼气中乙醇的 检测、人体口腔口臭的检测等等。

1.2.5.4 压力传感器

1. 压力传感器的定义：

压力传感器（pressure transducer）是指能感受压力并转换成可用输出信号的传感器。

2. 压力传感器的分类：

压力传感器是工业实践中最为常用的一种传感器，而我们通常使用的压力传感器主要是利用压电效应制造而成的，这样的传感器也称为压电传感器。除了压电传感器之外，还有利用压阻效应制造出来的压阻传感器，利用应变效应的应变式传感器等，这些不同的压力传感器利用不同的效应和不同的材料，在不同的场合能够发挥它们独特的用途。

3. 压力传感器的应用领域：

压力传感器是工业实践中最为常用的一种传感器，其广泛应用于各种工业自控环境，涉及水利水电、铁路交通、智能建筑、生产自控、航空航天、军工、石化、油井、电力、船舶、机床、管道等众多行业，另外还有医用的压力传感器。

1.2.5.5 加速度传感器

1. 加速度传感器的定义：

加速度传感器（acceleration transducer）是指能够感受到加速度并将其转换成可用输出信号的传感器。

2. 加速度传感器的分类：

加速度计有两种：一种是角加速度计，是由陀螺仪（角速度传感器）改进的。另一种就是线加速计。

3. 加速度传感器的应用领域：

它应用在控制，手柄震动和摇晃、仪器仪表、汽车制动启动检测、地震监测、玩具、环境监视、地质勘探、铁路、桥梁、大坝的振动测试与分析、高层建筑结构动态特性和安全保卫振动侦查上。

1.2.5.6 红外传感器

1. 红外传感器的定义：

红外传感系统是用红外线为介质的测量系统。

2. 红外传感器的分类：

红外系统的核心是红外探测器，按照探测机理的不同，可以分为热探测器（基于热效应）和光子探测器（基于光电效应）两大类。

3. 红外传感器的应用领域：

红外探测器应用可以用于非接触式的温度测量，气体成分分析，无损探伤，热像检测，红外遥感以及军事目标的侦察、搜索、跟踪和通信等。

1.3 RFID 射频识别基础知识

1.3.1 RFID 射频识别的概念

射频识别即 RFID（Radio Frequency Identification），又称电子标签、无线射频识别，是一种通信技术，可通过无线电讯号识别特定目标并读写相关数据，而无需识别系统与特定目标之间建立机械或光学接触。RFID 对于计算机自动识别技术而言是一场革命，极大地提高了信息处理效率和准确度。RFID 利用射频信号通过空间耦合（交变磁场或电磁场）实现无接触信息传递并通过所传递的信息达到识别目的。射频识别技术改变了条形码依靠“有形”的一维或二维几何图案来提供信息的方式，通过芯片来提供存储在其中的数量巨大的“无形”信息。

1.3.2 RFID 射频识别的作用

RFID 的典型应用：

1. 物流和供应管理
2. 生产制造和装配
3. 航空行李处理
4. 邮件/快运包裹处理
5. 文档追踪/图书馆管理
6. 动物身份标识
7. 运动计时
8. 门禁控制/电子门票
9. 道路自动收费
10. 城市一卡通的应用
11. 高校手机一卡通的应用

1.3.3 RFID 射频识别的基本特征

1.3.3.1 RFID 的基本组成

RFID 系统由五个组件构成，包括：传送器、接收器、微处理器、天线、标签。传送器、接收器和微处理器通常都被封装在一起，又统称阅读器（Reader），所以工业界经常将 RFID 系统分为阅读器、天线和标签三大组件：

- 阅读器（Reader）：读取（有时还可以写入）标签信息的设备，可设计为手持式或固定式，因其工作模式一般是主动向标签询问标识信息，所以有时又被称为询问器（Interrogator）（如图 1-3）；

图1-3　　阅读器

- 天线（Antenna）：在标签和阅读器之间传递射频信号（如图 1-4）；

图1-4　　天线

- 标签（Tag）：由耦合元件及芯片组成，每个标签具有唯一的电子编码，附着在物体上标识目标对象，有时又称为应答器（如图 1-5）。

图1-5　　标签

1.3.3.2　RFID 的工作原理

RFID 技术的基本工作原理并不复杂：阅读器通过天线发送电子信号，标签接收到信号后发射内部存储的标识信息，阅读器在通过天线接收并识别标签发回的信息，最后阅读器再将识别结果发送给主机。以 RFID 卡片阅读器以及电子标签之间的通讯及能量感应方式来看大致上可以分成：感应耦合（Inductive Coupling）及后向散射耦合（Backscatter Coupling）两种。一般低频的 RFID 大都采用第一种方式，而较高频大多采用第二种方式。

1.3.3.3　RFID 标签的分类

按工作方式把标签分为：

- 被动式标签（Passive Tag）：因内部没有电源设备又被称为无源标签。被动式标签内部的集成电路通过接收由阅读器发出的电磁波进行驱动，向阅读器发送数据。目前市场上的标签主要是被动式的。

- 主动标签（Active Tag）：因标签内部携带电源又被称为有源标签。电源设备和与其相关的电路决定了主动式标签要比被动式标签体积大、价格昂贵。但主动式标签通信距离更

远，可达上百米远。

- 半主动标签（Semi-active Tag）：这种标签兼有被动标签和主动标签的所有优点，内部携带电池，能够为标签内部计算提供电源。这种标签可以提供携带传感器，可用于检测环境参数，如温度、湿度、是否移动等。然而和主动式标签不同是它们的通信并不需要电池提供能量，而是像被动式标签一样通过阅读器发射的电磁波获取通信能量。又称半被动标签。

1.3.3.4 RFID 的特点

RFID 是未来物联网技术的发展趋势，其特点可以概括为：

- 体积小且形状多：RFID 标签在读取上并不受尺寸大小与形状限制，不需要为了读取精度而配合纸张的固定尺寸和印刷品质。
- 可重复使用：标签具有读写功能，电子数据可被反复覆盖，因此可以被回收而重复使用。
- 穿透性强：标签在被纸张、木材和塑料等非金属或非透明的材质包裹的情况下也可以进行穿透性通讯。
- 数据安全性：标签内的数据通过循环冗余校验的方法来保证标签发送的数据准确性。

1.4 无线通信基础知识

1.3.1 ZIGBEE 和 IEEE 802.15.4 标准

ZigBee是IEEE 802.15.4协议的代名词，可工作在2.4GHz(全球流行)、868MHz(欧洲流行)和915 MHz(美国流行)3个频段上，分别具有最高250kbit/s、20kbit/s和40kbit/s的传输速率，它的传输距离在10-75m的范围内且可以继续增加。ZigBee这一名称来源于蜜蜂的八字舞，由于蜜蜂(bee)是靠飞翔和“嗡嗡”(zig)地抖动翅膀的“舞蹈”来与同伴传递花粉所在方位信息，也就是说蜜蜂依靠这样的方式构成了群体中的通信网络。ZigBee适用于数据流量小、需要数据采集或监控的网点多、地形复杂、低成本和可靠性高的业务中；用于自动控制和远程控制领域，可以嵌入各种设备。它是近几年研究和应用较多的协议，在无线传感器网络领域占有重要地位。

ZigBee作为一种无线通信技术，具有如下特点：

(1) 低功耗：由于ZigBee的传输速率低，发射功率仅为1mW，而且采用了休眠模式，功耗低。因此ZigBee设备非常省电。据估算ZigBee设备仅靠两节5号电池就可以维持长达6个月到2年左右的使用时间，这是其它无线设备望尘莫及的。

(2) 成本低： ZigBee的处理核心是工业级8051单片机，并且ZigBee协议是免专利费的。所以低成本也是ZigBee的一个重要优势。

(3) 时延短：通信时延和从休眠状态激活的时延都非常短，典型的搜索设备时延为30ms，休眠激活的时延是15ms，活动设备信道接入的时延为15ms。因此ZigBee技术适用于对时延要求苛刻的无线控制(如工业控制场合等)应用。

(4) 网络容量大：一个星型结构的Zigbee网络最多可以容纳254个从设备和一个主设备，一个区域内可以同时存在最多100个ZigBee网络，而且网络组成灵活。

(5) 可靠：采取了碰撞避免策略，同时为需要固定带宽的通信业务预留了专用时，避开了发送数据的竞争和冲突。MAC层采用了完全确认的数据传输模式，每个发送的数据包都必须等待接收方的确认信息。如果传输过程中出现问题可以进行重发。

(6) 安全： ZigBee提供了基于循环冗余校验(CRC)的数据包完整性检查功能，支持鉴权和认证，采用了AES-128的加密算法，各个应用可以灵活确定其安全属性。

ZigBee无线组网通信的基础是IEEE 802.15.4协议以及在此基础上建立的ZigBee协议栈，所以要深入理解ZigBee技术，就要对ZigBee的协议栈进行学习和研究。以下是对ZigBee协议栈进行的简单介绍。

1.3.2 ZIGBEE 协议栈简介

如下图所示：ZigBee协议栈主要由应用层、网络层、MAC层和物理层组成的，以下是对ZigBee各层进行简单介绍。

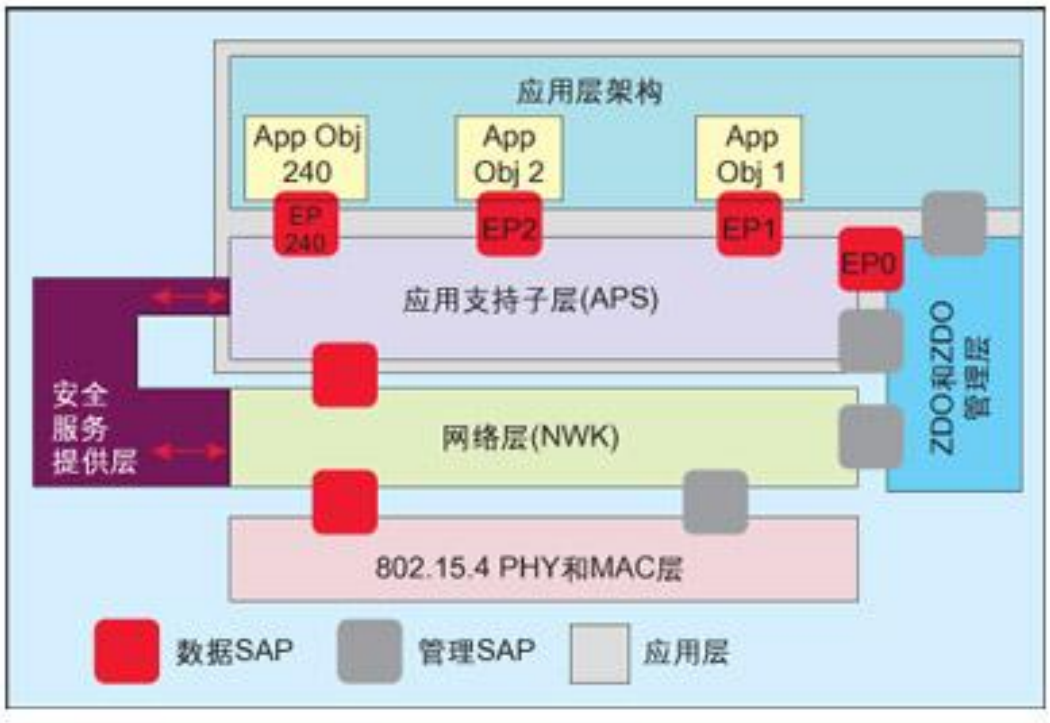


图1-6 ZigBee 协议栈框架

1.3.2.1 ZigBee 应用层

ZigBee应用层包括应用支持子层APS、应用框架AF、ZigBee设备对象ZDO。他们共同为应用开发者提供统一的接口。

1. 应用支持子层APS

APS层主要功能：

1.1 APS层协议数据单元APDU的处理。

1.2 APSDE提供在同一个网络中的应用实体之间的数据传输机制。

1.3 APSME提供多种服务给应用对象，这些服务包括安全服务和绑定设备，并维护管理对象的数据库，也就是我们常说的AIB。

2. 应用框架AF

应用框架为各个用户自定义的应用对象提供了模板似的活动空间，为每个对象提供了键值对KVP服务和报文MSG服务两种服务供数据传输使用。每个节点除了64位的IEEE地址，16位的网络地址，每个节点还提供了8位的应用层入口地址，对应于用户应用对象。端点0为ZDO接口，端点1至240供用户自定义对象使用，端点255为广播地址，端点241至254保留将来使用。每一个应用都对应一个配置文件，配置文件包括：设备ID，事务集群，属性ID等，AF可以通过这些信息来决定服务类型。

3. ZigBee设备对象ZDO

ZDO是一个特殊的应用层的端点。它是应用层其他端点与应用子层管理实体交互的中间件。它主要提供的功能如下：

3.1 初始化应用支持子层，网络层。

3.2 发现节点和节点的功能。在无信标的网络中，加入的节点只对其父节点可见。而其他节点可以通过ZDO的功能来确定网络的整体拓扑结构以及节点所能提供的功能。

3.3 安全加密管理：主要包括安全Key的建立和发送，已经安全授权。

3.4 网络的维护功能。

3.5 绑定管理：绑定的功能由应用支持子层提供，但是绑定功能的管理却是由ZDO提供，它确定了绑定表的大小，绑定的发起和绑定的解除等功能。

3.6 节点管理：对于网络协调器和路由器，ZDO提供网络监测、获取路由和绑定信息、发起脱离网络过程等一系列节点管理功能。

ZDO实际上是介于应用层端点和应用支持子层中间的端点，其主要功能集中在网络管理和维护上。应用层的端点可以通过ZDO提供的功能来获取网络或者是其他节点的信息，包括网络的拓扑结构、其他节点的网络地址和状态以及其他节点的类型和提供的服务等信息。

1.3.2.2 ZigBee 网络层

网络层需要在功能上保证与IEEE 802.15.4标准兼容，同时也需要上层提供合适的功能接口。它负责设备到设备的通信，并负责拓扑的搭建、设备寻址、消息路由和网络发现，属于通用的网络层功能范畴。

对于网络层，其完成和提供的主要功能如下：

1. 产生网络层的数据包：当网络层接收到来自应用子层的数据包，网络层对数据包进行解析，然后加上适当的网络层包头向MAC传输。

2. 网络拓扑的路由功能：网络层提供路由数据包的功能，如果包的目的节点是本节点的话，将该数据包向应用子层发送。如果不是，则将该数据包转发给路由表中下一结点。

3. 配置新的器件参数：网络层能够配置合适的协议，比如建立新的协调器并发起建立网络或者加入一个已有的网络。

4. 建立PAN网络

5. 连入或脱离PAN网络：网络层能够提供加入或脱离网络的功能，如果节点是协调器或者路由器，还可以要求节点脱离网络。

6. 分配网络地址：如果本节点是协调器或者路由器，则接入该节点的子节点的网络地址由网络层控制。

7. 邻居节点的发现：网络层能发现维护网络邻居信息。

8. 建立路由：网络层提供路由功能。

9. 控制接收：网络层能够控制接收器的接收时间和状态。

为了向应用层提供接口，网络层提供了两个功能实体，分别为数据服务实体NLDE和管理服务实体NLME。NLDE通过NLDE-SAP为应用层提供数据传输服务，NLME通过NLME-SAP为应用层提供管理服务，并且，NLME还完成对网络信息库NIB的维护和管理。

1.3.2.3 ZigBee MAC 层

MAC层包括管理实体(MLME)和数据实体(MLDE)。MAC层管理实体提供可以唤醒MAC层管理服务的服务接口，同时也维护一个与MAC层相关的管理对象数据库(MIB)。MAC层与物理层之间通过PLME-SAP和PD-SAP进行通信，通过MAC数据实体服务点(MLDE-SAP)和MAC层管理实体服务接入点(MLME-SAP)向业务相关子层提供MAC层数据和管理服务。另外，MAC层能支持多种LLC标准，通过业务相关会聚子层(SSCS)协议承载802.2类型的LLC标准。

MAC层功能如下：

- 当节点为网络协调器时，产生信标(beacon)帧；
- 在信标帧之间进行同步；

- 支持个人区域网(PAN)的关联与解关联；
- 支持节点安全机制；
- 对信道接入使用CSMA-CA机制；
- 处理和维护有保证的时隙(GTS)机制；
- 在两个对等的MAC实体间提供可靠的链接。

Zigbee中的MAC和物理层协议是网状网络的应用基础，高容错和低功耗的特点能保证网状网络所必须考虑基于拓扑控制和功率控制的网络自组特性。而且对于经典的隐藏终端和暴露终端问题、协议的接入公平性问题、服务质量问题等都有良好的解决。在网状网络中，MAC层的传输调度策略会影响数据包延迟、带宽等性能，影响网络层路由性能，所以网络层必须感知MAC层性能的变化，才可以自适应的方式改变路由，改善网络性能。

1.3.2.4 ZigBee 物理层

物理层提供的服务是由硬件和软件共同实现的，定义了物理无线信道(对于2.4 GHz频段，有16个信道，编号为11-26)和MAC子层之间的接口，提供物理层数据服务(PLDE)和物理层管理服务(PLME)。通过该接1：3可以唤醒层管理服务功能，同时也负责维护与物理层相关的一些管理对象的数据库(PIB)。

物理层通过物理层数据服务接入点(PD-SAP)和物理层管理服务接入点（PLME-SAP)与MAC层通信，PD-SAP支持在对等的MAC层实体间进行MAC协议数据单元传送，PLME-SAP则在MAC层管理实体之间提供管理命令的传送。

物理层主要完成如下任务：

- 无线收发机的激活与关闭；
- 当前信道的能量检测(Energy Detect，ED)；
- 接收数据包的链路质量标识(LQI)；
- 为载波侦听多路访问／冲突防止(CSMS-CA)提供空闲信道评估(CCA)；
- 工作信道选择；
- 数据发送和接收。

信道能量检测为网络层提供信道选择依据，其值取值范围是0x00-0xFF。它主要测量目标信道中接收信号的功率强度，链路质量标识为网络层或应用层提供接受数据帧无线信号的强度和质量信息。

1.3.3 WIFI 概述

1.3.1.1 Wifi 简介

Wifi 是一种可以将个人电脑、手持设备（如 PDA、手机）等终端以无线方式互相连接的技术。它是一个无线网路通信技术的品牌，由 Wifi 联盟所持有，目的是改善基于 IEEE 802.11 标准的无线网路产品之间的互通性。它是一种短程的无线传输技术，能够在数百英尺范围内支持互联网接入的无线电信号。随着技术的发展，以及 IEEE 802.11a 及 IEEE 802.11g 等标准的出现，现在 IEEE 802.11 这个标准已被统称作 Wifi。

1.3.1.2 Wifi 的组成

一个 Wifi 联接点网络成员和结构站点（Station），是网络最基本的组成部分。

基本服务单元（Basic Service Set，BSS）：网络最基本的服务单元。最简单的服务单元可以只由两个站点组成。站点可以动态地联接（associate）到基本服务单元中。

分配系统（Distribution System，DS）：分配系统用于连接不同的基本服务单元。分配系统使用的媒介（Medium）逻辑上和基本服务单元使用的媒介是截然分开的，尽管它们物理上可能会是同一个媒介，例如同一个无线频段。

接入点（Access Point，AP）：接入点既有普通站点的身份，又有接入到分配系统的功能。

拓展服务单元（Extended Service Set，ESS）：由分配系统和基本服务单元组合而成。这种组合是逻辑上的，并非物理上的。不同的基本服务单元物有可能在地理位置上相去甚远。分配系统也可以使用各种各样的技术。

关口（Portal）：也是一个逻辑成分。用于将无线局域网和有线局域网或其他网络联系起来。

1.3.1.3 Wifi 的特点

IEEE 802.11 只负责在站点使用的无线媒介上的寻址（Addressing），分配系统和其他局域网的寻址不属于无线局域网的范围。

IEEE 802.11 没有具体的定义分配系统，只是定义了分配系统应该提供的服务（Service）。整个无线局域网定义了 9 种服务：

- 5 种服务属于分配系统的任务，分别为：联接（Association），结束联接（Dissociation），分配（Distribution），集成（Integration），再联接（Reassociation）。
- 4 种服务属于站点的任务，分别为：鉴权（Authentication），结束鉴权（Deauthentication），隐私（Privacy），MAC 数据传输（MSDU delivery）。

1.3.1.4 Wifi 的应用

由于 Wifi 的频段在世界范围内是无需任何电信运营执照的，因此 WLAN 无线设备提供

了一个世界范围内可以使用的，费用极其低廉且数据带宽极高的无线空中接口。用户可以在Wifi 覆盖区域内快速的浏览网页，随时随地的接听拨打电话。而其他一些基于 WLAN 的宽带数据应用，如流媒体、网络游戏等功能更是值得用户期待。有了 Wifi 功能，我们可以打长途电话（包括国际长途），浏览网页、收发电子邮件、音乐下载、数码照片传递等，而无需担心速度慢和花费高的问题。

Wifi 在掌上设备上应用越来越广泛，而智能手机就是其中一份子。与早前应用于手机上的蓝牙技术不同，Wifi 具有更大的覆盖范围和更高的传输速率，因此 Wifi 手机成为了目前移动通信业界的时尚潮流。

1.3.4 3G 概述

第三代移动通信技术（3rd-generation，3G），是指支持高速数据传输的蜂窝移动通信技术。3G 服务能够同时传送声音及数据信息，速率一般在几百 kbps 以上。目前 3G 存在四种标准：CDMA2000（美国版），WCDMA（欧洲版），TD-SCDMA（中国版），WiMAX。

CDMA 是 Code Division Multiple Access (码分多址)的缩写，是第三代移动通信系统的技术基础。CDMA 系统以其频率规划简单、系统容量大、频率复用系数高、抗多径能力强、通信质量好、软容量、软切换等特点显示出巨大的发展潜力。下面分别介绍一下 3G 的几种标准：

1. W-CDMA

也称为 WCDMA，全称为 Wideband CDMA，也称为 CDMA Direct Spread，意为宽频分码多重存取，这是基于 GSM 网发展出来的 3G 技术规范，是欧洲提出的宽带 CDMA 技术，它与日本提出的宽带 CDMA 技术基本相同，目前正在进一步融合。W-CDMA 的支持者主要是以 GSM 系统为主的欧洲厂商，日本公司也或多或少参与其中，包括欧美的爱立信、阿尔卡特、诺基亚、朗讯、北电，以及日本的 NTT、富士通、夏普等厂商。 该标准提出了 GSM(2G)-GPRS-EDGE-WCDMA(3G)的演进策略。这套系统能够架设在现有的 GSM 网络上，对于系统提供商而言可以较轻易地过渡。预计在 GSM 系统相当普及的亚洲，对这套新技术的接受度会相当高。因此 W-CDMA 具有先天的市场优势。

2. CDMA2000

CDMA2000 是由窄带 CDMA(CDMA IS95)技术发展而来的宽带 CDMA 技术，也称为 CDMA Multi-Carrier，它是由美国高通北美公司为主导提出，摩托罗拉、Lucent 和后来加入的韩国三星都有参与，韩国现在成为该标准的主导者。这套系统是从窄频 CDMAOne 数字标准衍生出来的，可以从原有的 CDMAOne 结构直接升级到 3G，建设成本低廉。但目前使用 CDMA 的地区只有日、韩和北美，所以 CDMA2000 的支持者不如 W-CDMA 多。不过 CDMA2000 的研发技术却是目前各标准中进度最快的，许多 3G 手机已经率先面世。该标准提出了从 CDMA IS95(2G)-CDMA20001x-CDMA20003x(3G)的演进策略。CDMA20001x 被

称为 2.5 代移动通信技术。CDMA20003x 与 CDMA20001x 的主要区别在于应用了多路载波技术，通过采用三路载波使带宽提高。目前中国电信正在采用这一方案向 3G 过渡，并已建成了 CDMA IS95 网络。

3. TD-SCDMA

全称为 Time Division - Synchronous CDMA(时分同步 CDMA)，该标准是由中国大陆独自制定的 3G 标准，1999 年 6 月 29 日，中国原邮电部电信科学技术研究院（大唐电信）向 ITU 提出，但技术发明始于西门子公司，TD-SCDMA 具有辐射低的特点，被誉为绿色 3G。该标准将智能无线、同步 CDMA 和软件无线电等当今国际领先技术融于其中，在频谱利用率、对业务支持具有灵活性、频率灵活性及成本等方面的独特优势。另外，由于中国内地庞大的市场，该标准受到各大主要电信设备厂商的重视，全球一半以上的设备厂商都宣布可以支持 TD—SCDMA 标准。 该标准提出不经过 2.5 代的中间环节，直接向 3G 过渡，非常适用于 GSM 系统向 3G 升级。军用通信网也是 TD-SCDMA 的核心任务。

4. WiMAX

WiMAX 的全名是微波存取全球互通(Worldwide Interoperability for Microwave Access)，又称为 802·16 无线城域网，是又一种为企业和家庭用户提供“最后一英里”的宽带无线连接方案。将此技术与需要授权或免授权的微波设备相结合之后，由于成本较低，将扩大宽带无线市场，改善企业与服务供应商的认知度。2007 年 10 月 19 日，在国际电信联盟在日内瓦举行的无线通信全体会议上，经过多数国家投票通过，WiMAX 正式被批准成为继 WCDMA、CDMA2000 和 TD-SCDMA 之后的第四个全球 3G 标准。

1.5 实验箱基础概述

1.4.1 实验箱布局和实物图

E-Box300 物联网教学实验系统集成了多种传感器模型以及多种无线组网模式，可实现多种物联网构架，面向高校、大专、高职等院校的专业教学及创新竞赛，提供了众多实验例程，便于学生熟悉和掌握物联网的构成及实际应用。图 1-7 是实验箱的实物图。

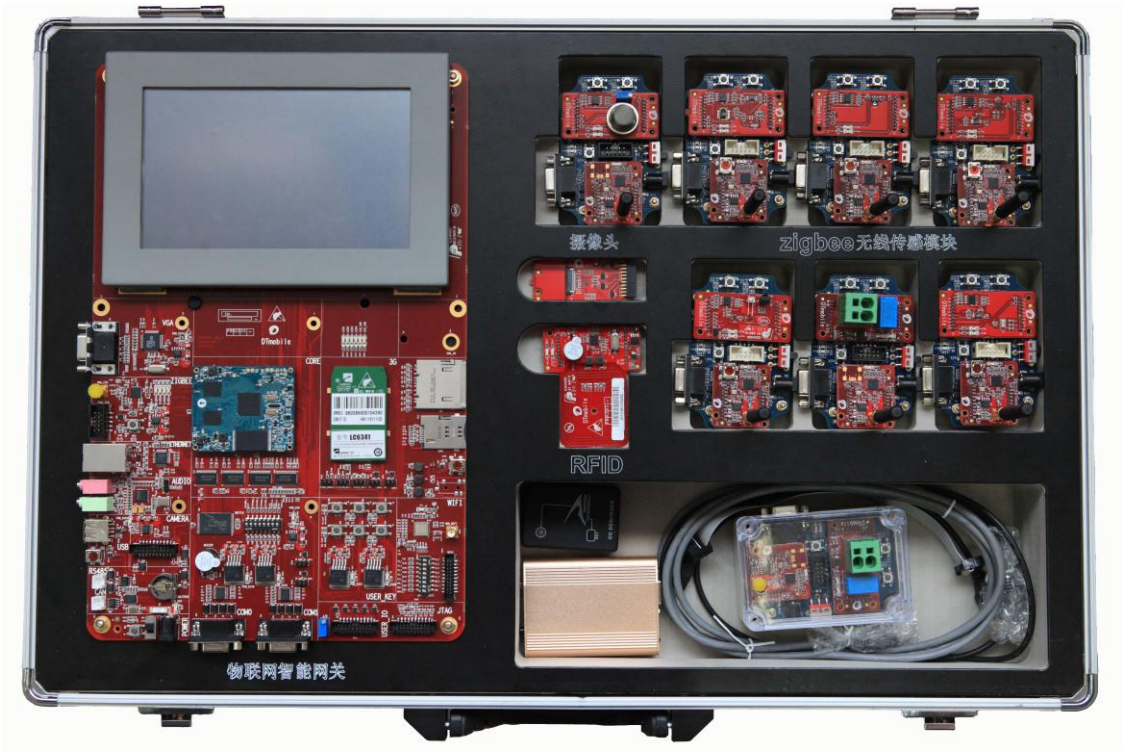


图1-7　　E-Box300

E-Box300 物联网教学实验套件包括硬件设备、软件资源、实验资源三大部分。硬件设备标准配置包括个 7 个 zigbee 无线节点模块、6 个传感器模块、RFID 模块、ARM 网关和其他配套设备。zigbee 无线节点模块使用的是目前市场上主流使用的 CC2531 芯片。软件资源包括无线传感器网络软件、ARM 网关软件和实验平台管理软件，实验资源采用由简单到复杂的设计模式，引导学生很好地掌握物联网开发要领，为物联网工程应用打下坚实的基础，并能通过不同传感器的特性、不同网络的组成形式、Zigbee 无线定位技术以及 RFID 技术，开发出更多实用性强的互联网应用模式。

1.4.2　实验箱硬件构成

物联网试验箱 E-Box300 各硬件模块功能特点以及性能参数：

种类	数量	功能、性能参数	备注
ARM 网关	1	功能特点： 1、ARM 网关处理器采用 S3C6410X　ARM11 嵌入式整体平台，相比简单的 ARM9、XSCALE、单片机核心板式的网关通信能力强大。 2、支持一键烧写系统，支持 Linux、Wince、Andriod； 3、丰富的接口资源，支持 3G、wifi、CMOS 摄像头、ZigBee、RFID 模块的接入，提供各模块的测试程序；	

		4、客户端程序能够完成对传感网络的参数配置，并通过 wifi、以太网完成向远端服务器进行数据传输； 5、支持 7 寸触摸屏； 性能参数： 1、Samsung S3C6410 处理器，ARM1176JZF-S 内核，主频 533MHz/667MHz； 2、256M 字节 DDR 内存，2GNand Flash; 3、12MHz、48MHz、27MHz、32.768KHz 时钟源； 4、支持 5V 电压供电； 5、一个复位按键，采用专用芯片进行复位，稳定可靠； 6、系统启动方式设置开关，采用 8 位拨码开关； 7、一片 2M 字节 NOR Flash； 8、三路串口，包括两个五线 RS_232 串口和一个三线 RS_232 串口； 9、一个 RS_485 总线接口； 10 一个 CAN 总线接口，2.0 标准； 11、一个 100M 网口，采用 DM9000AE，带联接和传输指示灯； 12、一个 USB HOST 插口，支持 USB1.1 协议，使用侧插 USB A 型接口； 13、一个 USB Slave 接口，支持 USB 2.0 协议，使用 mini-USB AB 型接口； 14、一个高速 SD 卡座。可以实现 SD Memory 功能和 SDIO 功能； 15、一个 WIFI 扩展接口； 16、一路立体声音频输出接口可接耳机；另一路音频输入可接麦克风； 17、LCD 和触摸屏接口支持 7 寸触摸显示屏； 18、一个 VGA 接口，支持 800x600 分辨率； 19、一个 CMOS 摄像头接口，支持 ITU-R BT601/656 8 位模式，使用 10X2 插针连接器； 20、内部实时时钟，带有后备锂电池接口； 21、一个 JTAG 接口，使用 10*2 插针连接器； 22、六个用户按键 、四个 LED 23、一个蜂鸣器	

ZigBee+ 传感器采集模块	7	功能特点： 1、 采用广泛应用的德州仪器（TI）ZigBee Soc 射频芯片 CC2531/1-F128 作为 ZigBee 射频开发芯片，集成高性能工业级 8051 内核（高达 128kB 的闪存、8kB 的 RAM）、ADC、UART 等资源，支持 ZigBee 协议栈 Zstack-1.4.3-1.2.1，符合工业级标准要求； 2、 实现了对各种传感器模块，包含温度、湿度、光感、三轴加速度、压力、烟雾、红外对管传感信号的采集； 3、 各种传感器模块统一接口与 ZigBee 模块底板连接，通过 ZigBee 自组网完成对各种传感器的信号采集、传输以及控制； 4、 使用 IAR for MCS s-51 集成开发环境，用户可在线进行调试仿真； 5、 三种供电模式：电池盒、仿真器和外接电源； 性能参数： 1、 工作频率带宽：2.400~2.4835GHz； 2、 数据速率达 250kbps，码片速率达 2Mchips/s； 3、 低功耗：接收 RX：27mA、发送 TX：25mA，休眠电流<10μA； 4、 工作电压：2.9V~3.4V；	
RFID 模块	1	功能特点： 1、采用飞利浦高度集成读写卡芯片 MFRC522 射频基站 2、符合 ISO/IEC 14443 TYPE A/Mifare 标准。 3、具有寻卡、读取、写入、初始化电子钱包以及增值、减值、查询余额等基本功能，全面支持校园一卡通、公交一卡通、非接触智能水、电、气三表、考勤机、各种防伪系统等二次开发。 4、天线和模块一体。 性能参数： 1、 射频基站的射频频率为 13.56MHz； 2、 读写距离 0mm~50mm； 3、 电源：4.1-5.5V，典型值 5V，USB 供电接口； 4、 工作温度范围：-25 ~ +85 ℃； 5、 通讯接口：UART 接口；	
光感传感器 ISL29028A	1	功能特点： 1、可以在所有光源（包括太阳光）下工作。	

		2、双通道模数转换器可以同时独立测量环境光和近距离红外光。 3、智能中断方案：可以分别配置环境光和接近感应的中断阈值。可以配置触发中断之前需要达到阈值的次数：1/4/8/16 次达到阈值后再触发中断。 4、非线性：0.5%。 5、环境光检测的输出数据直接和光强成正比。 6、环境光检测可以选择 125 或 2000 勒克斯（lux）的量程。 7、消除环境中红外噪声（包括阳光）。 8、超低功耗：典型工作电流为 138μA。 9、易用性：I2C 通讯接口。 10、内部温度补偿。 11、小巧的 ODFN8 2.0x2.1x0.7（毫米）封装。 12、I2C 通讯接口匹配电压为 1.7V--3.63V。 13、传感器供电电压为 2.25V--3.63V。 14、符合 RoHS 标准。	
加速度传感器 MMA8452Q	1	功能特点： 1、电源电压：1.95 伏--3.6 伏。 2、通讯接口电压：1.6 伏--3.6 伏。 3、可动态选择±2g/±4g/±8g 量程。 4、输出数据速率：1.56 赫兹--800 赫兹。 5、12 位和 8 位数字输出。 6、I2C 通讯接口（当管脚连接 4.7K 上拉电阻时可达 2.25MHz 速率）。 7、6 个中断源对应 2 个可编程中断引脚。 8、3 个运动检测嵌入式通道：1 通道自由落体或运动检测；1 通道脉冲检测；1 通道震动检测。 9、电流消耗：6uA—165uA。 10、符合 RoHS 标准。 11、典型应用：指南针、静态方位检测、实时运动分析。	
气压传感器 MPL3115A2	1	功能特点： 1、电源电压：1.95 伏--3.6 伏。 2、通讯接口电压：1.6 伏--3.6 伏。 3、补偿内部。 4、直接读取数据。数据已经经过补偿。 压力：20 位数据（帕斯卡）；	

		海拔高度：20 位数据（米）； 温度：12 位数据（摄氏度）。 5、测量范围： 压力：测量 20--110kPa 范围。 温度：测量-40--+85 摄氏度范围。 6、分辨率。 压力：1.5 帕斯卡。 海拔高度：30 厘米。 温度：0.0625 摄氏度。 7、可编程事件。 8、自动数据采集。 9、使用 FIFO，最多能够记录 12 天的数据。 10、IIC 接口（数据速率达 400 千赫）。 11、应用实例：高精度高度计；智能手机；导航；气象站装备。	
温湿度传感器	1	功能特点： 1、相对湿度和温度传感器 2、工作电压范围 2.4 伏--5.5 伏。 3、通讯接口：串行接口，一根数据线，一根时钟线。 4、测温范围：-40—123.8 摄氏度。 5、完全校准，数字输出。 6、卓越的长期稳定性。 7、无需任何外部元件。 8、超低功耗。 9、小尺寸。	
烟雾传感器 MQ-2	1	性能特点： 1、大探测范围。 2、高灵敏度和快速响应。 3、优异的稳定性；寿命长。 4、简单的驱动电路。 5、应用：可用于家庭和工厂的气体泄漏监测装置， 适宜于液化气、丁烷、丙烷、甲烷、酒精、氢气、烟雾等的探测。 6、标准工作条件 温度： 20℃±2℃ 相对湿度： 65%±5% 7、在标准工作条件下，探测浓度范围：	

		液化气和丙烷：100ppm-10000ppm 丁烷：300ppm-5000ppm 甲烷：5000ppm-20000ppm 氢气：300ppm-5000ppm 酒精：100ppm-2000ppm 8、预热时间不少于 24 小时。	
红外安防	1	性能特点： 1、 红外安防装置有效距离可达 10 米。 2、红外发射管连续工作电流 150mA。 3、红外发射管在脉冲控制方式下，最大工作电流可达 3A。 4、红外接收端中心频率：38KHz。 5、对环境光具有高性能抗干扰性。 6、工作环境温度：-25 -- +85 摄氏度。	

1.4.3 实验箱选配模块

物联网试验箱 E-Box300 各选配模块功能特点以及性能参数：

种类	数量	功能、性能参数	备注
Wifi 模块	选配	1、WM-G-MR-09 无线局域网模块 2、符合 IEEE802.11 b/g，高达 54Mbps 数据率 3、支持 802.11i 安全标准 4、支持 802.11e 服务质量 5、3 线，支持硬件信号 BT wifi 共存 6、电压：3.0-3.6V	
摄像头模块	选配	1、OV9650 低电压 CMOS 摄像头； 2、提供完整功能的单芯片 SXGA（1280*1024）小封装处理器。提供全帧，次采样或窗 8-10bit 各式图像 SCCB 控制总线接口； 3、高达 15 帧每秒的 SXGA 分辨率 4、电压：内核部分：1.8V+-10% I/O 口： 2.5V 5、输出格式：YUV/YCbCr：4:2:2 RGB ： 4:2	

		Raw RGB	
3G 模块	选配	基本功能: 串口输入符合 AT 指令协议，可实现通话控制及短信业务、3G 拨号上网，视频传输等 功能特点： 1. 制式:TD-SCDMA(HSPA)& GGE 双模 2. 工作频带: TD-SCDMA:2010～2025MHz，1880～1920MHz Tri-Band EDGE/GPRS/GSM 900/1800/1900MHz 3. 最大发射功率: TDSCDMA 2GHz：+24dBm GSM/GPRS 900MHz： +33dBm GSM/GPRS 1800MHz/1900MHz： +30dBm EDGE900MHz：27dBm EDGE1800MHz/1900MHz：26dBm 4. 遵循的规范: TD-SCDMA:Compliant with 3GPP TS 25.102(R7) GSM/GPRS/EDGE900MHz/1800MHz/1900MHz: Compliantwith 3GPP TS 45.005 PCI_Express_Mini_CEM_1.2 5. 标准 AT 指令: 3GPP 27.007 Support G3-EWALK AT SPEC V2.0 6. 电源:单电源供电 3.0～3.6V 7. 温度范围: -10℃～+70℃（工作温度） -40℃～+85℃（存储温度） 8. 湿度范围:5％～95％ 9. 结构尺寸 Length*Width*Height: 50*30*5mm(Full Size) Bottom Height Limit: _1.35mm Top Height Limit: _2.4mm	

第2章　物联网基础实验平台

2.1　物联网 ZIGBEE 模块单片机基础实验平台

单片机在早期的物联网，特别是传感网相关的硬件方案中扮演着重要的角色。作为物联网神经末梢的无线传感器网络，其典型的特点在于低功耗、低成本、小体积、自组织等。在目前的工艺条件下，单片机是搭建原型系统和产品的主要工具。

2.1.1　硬件环境简介

单片机按照 CPU 处理数据的位宽可分为 4 位、8 位和 16 位机。其中，8 位单片机由于内部构造简单、体积小、成本低等优势，应用最为广泛。4 位单片机主要应用于工业控制领域。随着工艺的发展，由于 4 位单片机性能比较低，目前已逐步退出市场。而 16 位单片机虽然性能比 8 位强得多，但由于成本和应用场合的限制，尤其是近年来 ARM 嵌入式技术的发展，导致它的应用空间也不如 8 位单片机那么广泛。

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

2.1.2　软件环境简介

对于单片机的开发环境，软件方面涉及对编程语言、编译和调试环境的选择问题。根据应用对象的特点选择合适的开发编程语言和工具，是解决问题的首要任务。

2.1.2.1 编程语言

单片机编程语言一般有两种：汇编语言和 C 语言。无论是采用 C 语言，还是汇编语言，都是各有利弊。虽然对汇编语言的娴熟使用需要一定的时间，并且调试起来困难很大，但其程序执行效率高是不争的事实。C 语言虽易学易用，但对于一些底层和重复性操作，采用 C 语句实现起来效率偏低。所以在开发过程中，推荐采用 C 语言和汇编语言相结合的编程方式，以充分发挥这两者的优势。例如，通常用汇编语言来编写底层对硬件的操作，把与硬件无关或相关性较小的部分用 C 代码实现。当然，要充分发挥这两者的性能优势，需要大家对 C 编译器有一定的了解，并注重平时的积累。

2.1.2.2 开发平台

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

可以说，单片机的开发环境品种繁多。用户可以根据使用习惯和应用需求，综合进行开发平台的选择。我们的实验平台选用的是 IAR Embedded Workbench 开发环境。

2.1.3 操作系统

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

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

2.2 在主机上搭建 IAR 开发环境

2.2.1 安装 IAR 软件

2.2.1.1 ZigBee开发环境简介

我们的实验平台选用IAR Embedded Workbench作为ZigBee的开发环境。IAR Embedded Workbench（简称 EW）的C/C++交叉编译器和调试器是目前世界上最完整的和最容易使用的专业嵌入式应用开发工具之一。EW对不同的微处理器提供相同的直观用户界面。EW今天已经支持35种以上的8位/16位/32位的微处理器结构。

EW包括：嵌入式C/C++优化编译器、汇编器、连接定位器、库管理员、编辑器、项目管理器和C-SPY调试器中。IAR编译器使代码更加紧凑和优化，节省硬件资源，最大限度地降低产品成本，提高产品竞争力。

IAR Embedded Workbench集成的编译器主要产品特征：

- 高效PROMable代码

- 完全兼容标准C
- 内建对应芯片程序速度和大小的优化器
- 目标特性扩充
- 版本控制和扩展工具支持良好
- 便捷的中断处理和模拟
- 瓶颈性能分析
- 高效浮点支持
- 内存模式选择
- 工程中相对路径支持

IAR Systems的C/C++编译器可以生成高效可靠的可执行代码，并且应用程序规模越大，效果明显。

忽略项目的最终期限，开发者需要依靠一些可靠的开发工具来完成任务。未能按时完成进度会给项目带来不便，而恶性循环将会导致所有进度安排的拖延，后果会十分严重。IAR Embedded Workbench被认为是一款稳定可靠、高效的开发工具，可以提高项目开发效率。

IAR Embedded Workbench 是一套完整的集成开发工具集合：包括从代码编辑器、工程 建立到C/C++编译器、连接器和调试器的各类开发工具。它和各种仿真器、调试器紧密结合， 使用户在开发和调试过程中，仅仅使用一种开发环境界面，就可以完成多种微控制器的开发工作。

本节所需文件路径：E-Box300\02-开发环境搭建\ 02-WSN开发调试环境

2.2.1.2 ZigBee开发环境安装

IAR Embedded Workbench的安装如同Windows操作系统其它软件一样，单击setup.exe进行安装（路径：E-Box300\02-开发环境搭建\02-WSN开发调试环境\IAR程序编译软件），出现如图2-1的界面。

图2-1　　IAR 软件安装起始界面

单击“Next”至下一步，分别填写你的名字、公司以及认证序列，如图2-2所示。

注：图2-2中的认证序列（License number）和图2-3中的Lisence key由注册机生成，注册机的路径为：“E-Box300\02-开发环境搭建\02-WSN开发调试环境\IAR程序编译软件\7.51注册机”。

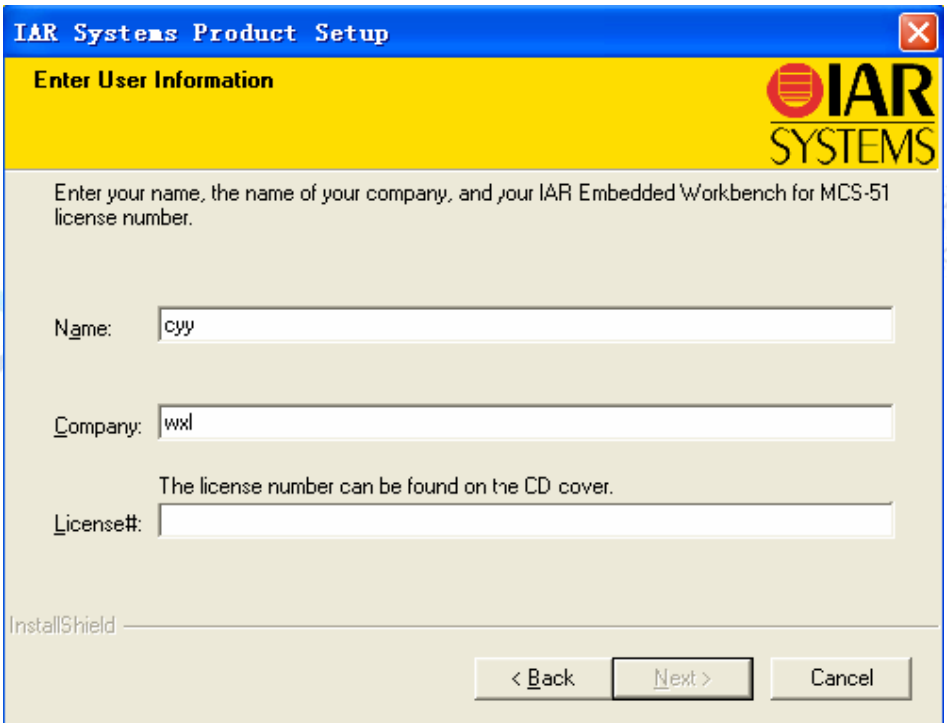


图2-2　　IAR 软件安装界面

正确填写后，单击“Next”至下一步，填写由本计算机的机器码和认证序列生成的序列密钥，如图2-3所示：

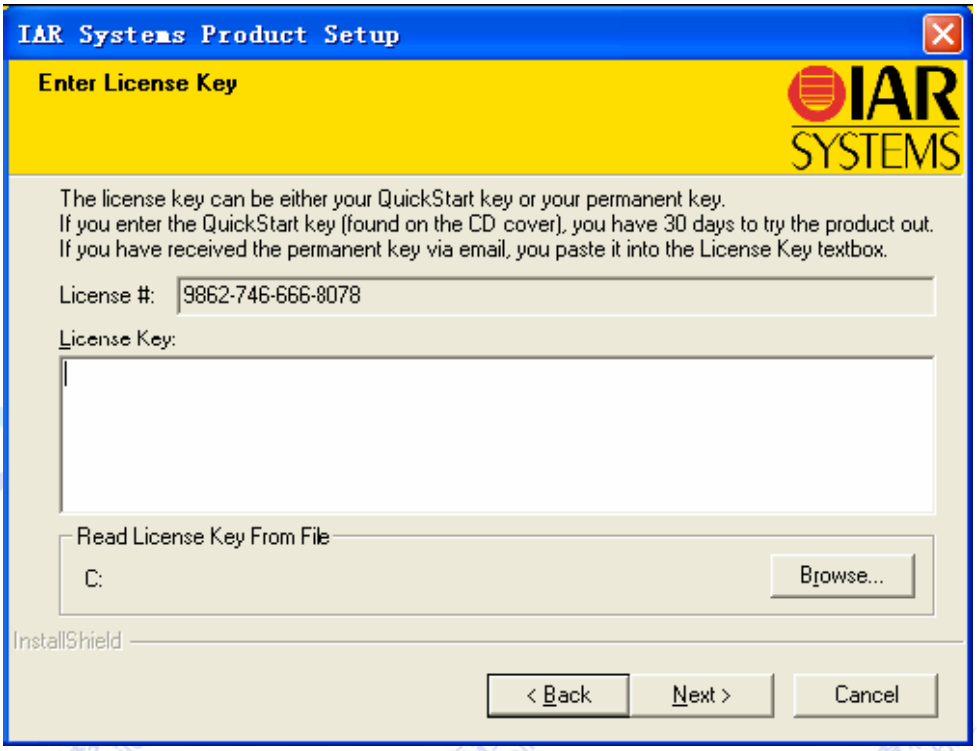


图2-3　　输入安装信息界面

输入的认证序列以及序列密钥正确后，单击“Next”到下一步。如图2-4所示，可以选择完全安装或是典型安装，在这里我们选择完全安装。

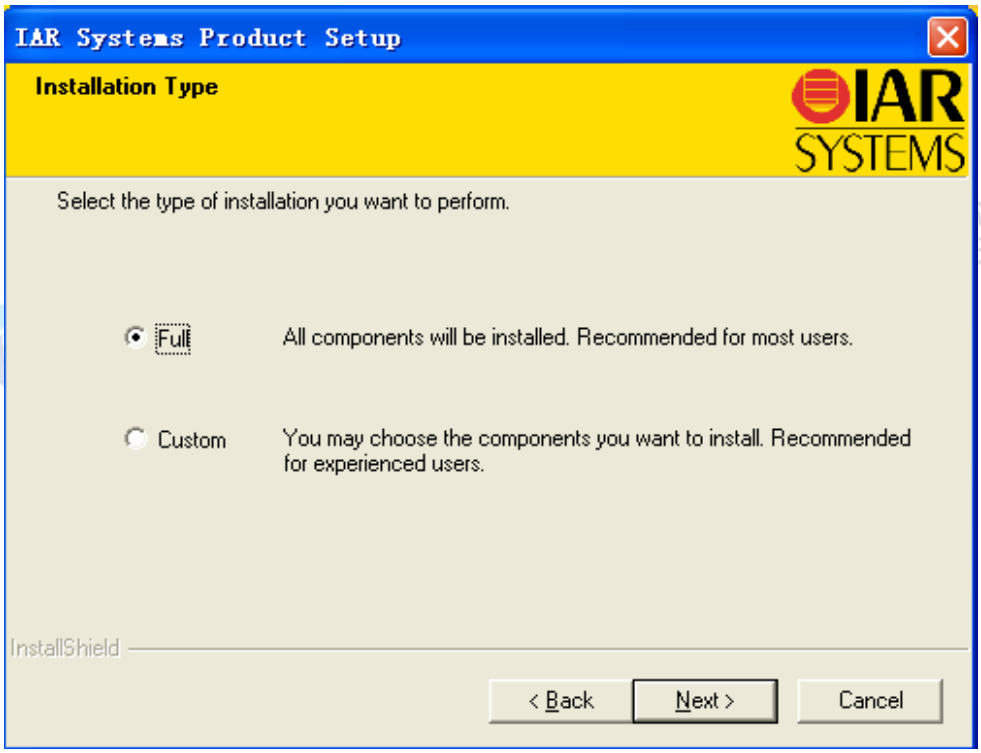


图2-4　　选择安装类型界面

单击“Next”到下一步，在这里可以查证之前输入的信息是否正确，如图2-5所示。如果需要修改，单击“Back”返回修改。

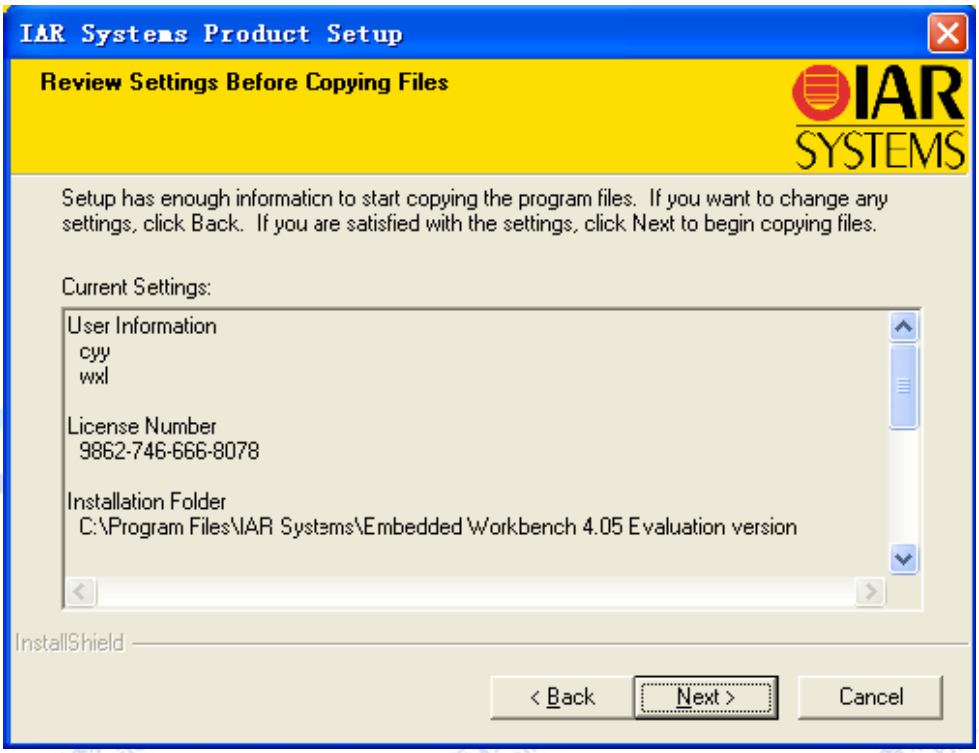


图2-5　　安装信息确认界面

单击“Next”正式开始安装，你可以看到安装进度，如图2-6所示。这将需要几分钟的时间，请耐心等待。

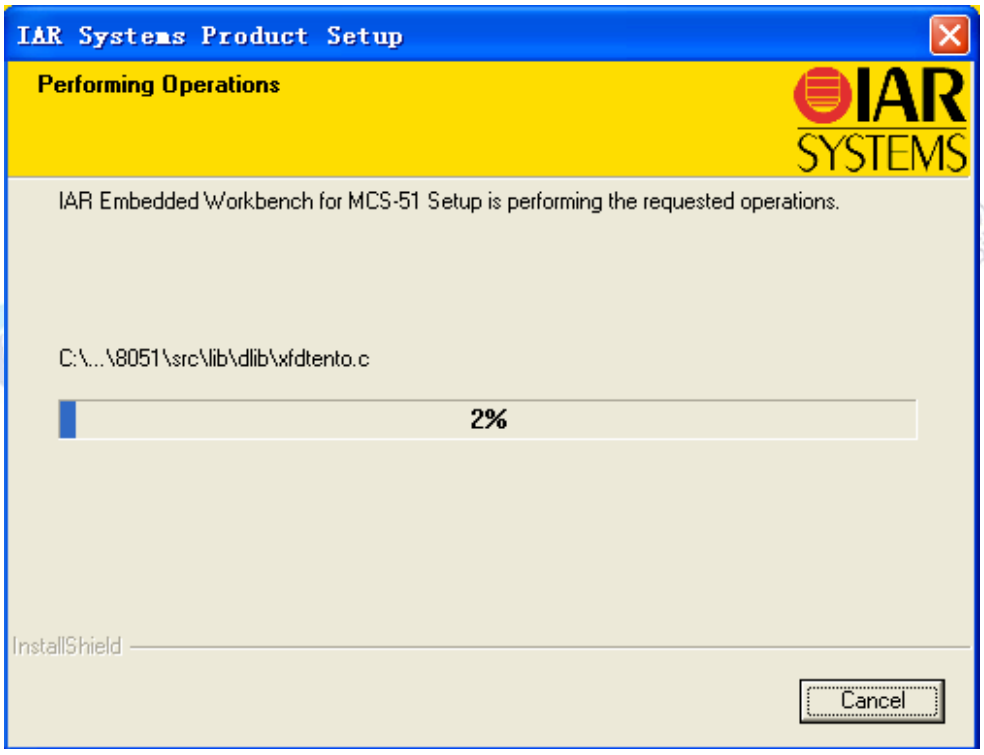


图2-6　　安装进度界面

当进度到100%时，显示如图2-7所示界面。可选择查看IAR的介绍以及是否立即运行 IAR开发集成环境，并单击“Finish”来完成安装。

图2-7　　安装完成界面

完成安装后，可以从“开始”菜单找到刚刚安装的IAR软件，如图2-8所示。

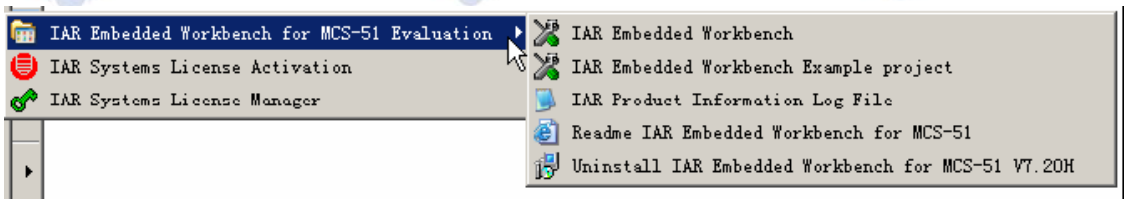


图2-8　　运行 IAR 软件

2.2.2　安装仿真器驱动程序

2. 2. 2. 1　　自动安装仿真器的驱动程序

成功安装IAR软件后，由于IAR安装软件中含有仿真器的驱动，所以连接仿真器与PC后可以自动安装仿真器的驱动程序。具体操作如下：

将仿真器通过附带的USB电缆连接到PC机，在Windows XP系统下，系统发现新硬件后提示如下对话框，选择自动安装软件，点下一步，如图2-9：

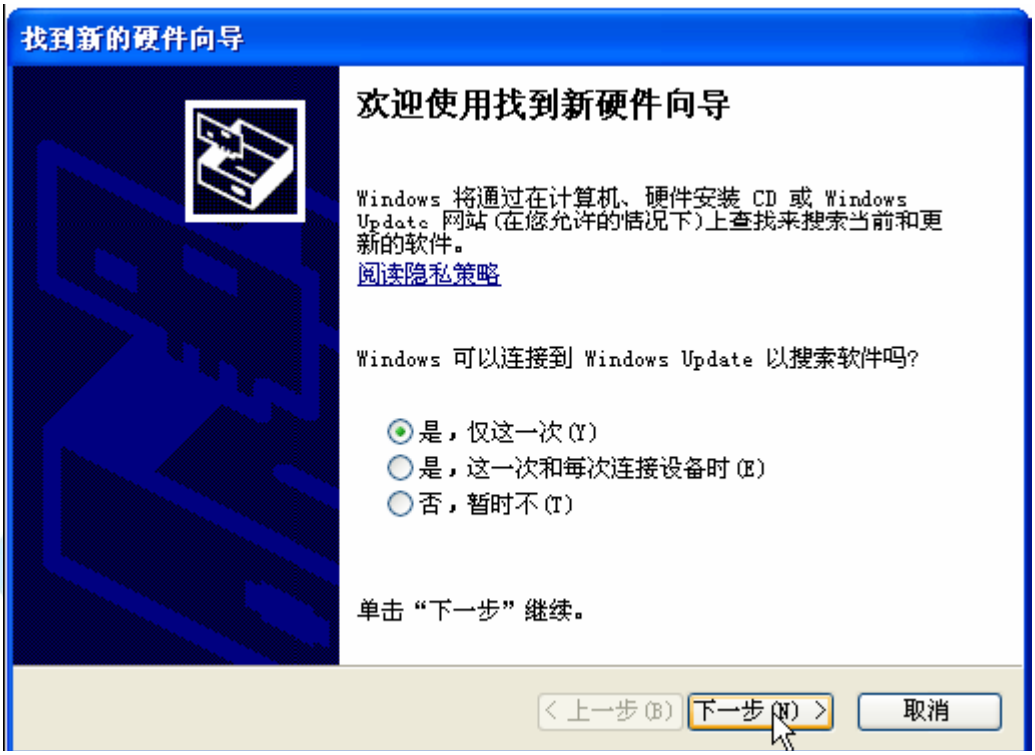


图2-9　　硬件安装向导

系统识别出仿真器，如图2-10：

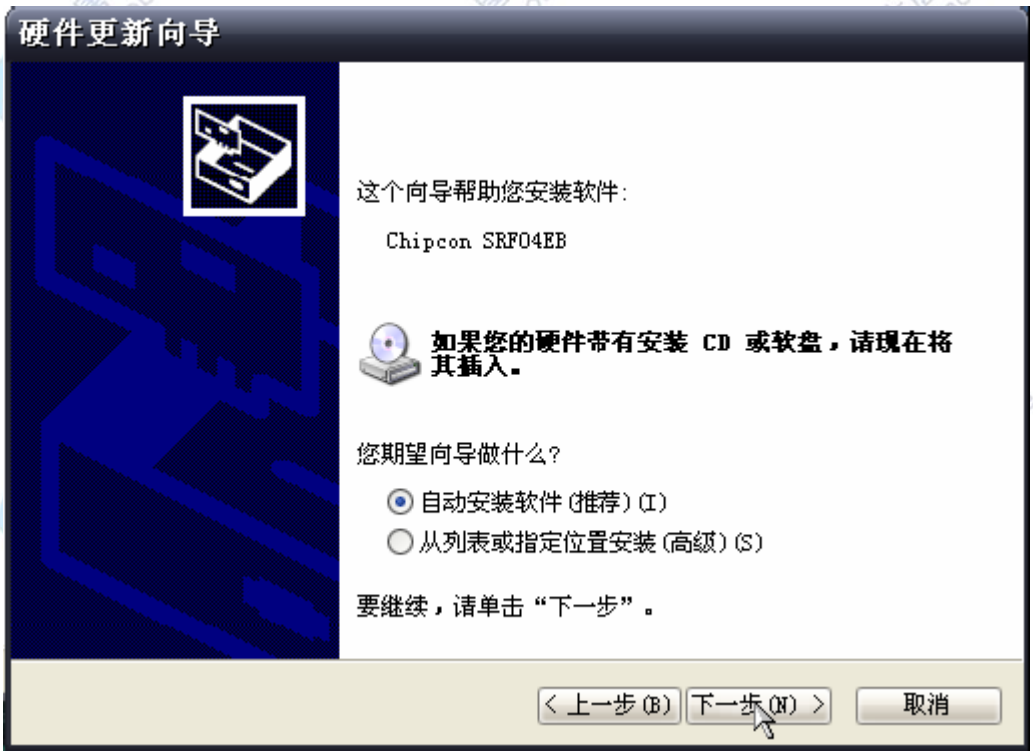


图2-10　　自动安装

向导会自动搜索并复制驱动文件到系统，如 2-11 所示。

图2-11　安装驱动文件

系统安装完驱动后提示完成对话框，点击“完成”退出安装，如图 2-12：

图2-12　仿真器驱动安装完成

2.2.2.2　手动安装仿真器的驱动程序

如果向导未能自动搜索到驱动文件，驱动程序可以在 IAR 的安装文件中找到。选择“从

列表或指定位置安装”，如图2-13所示：

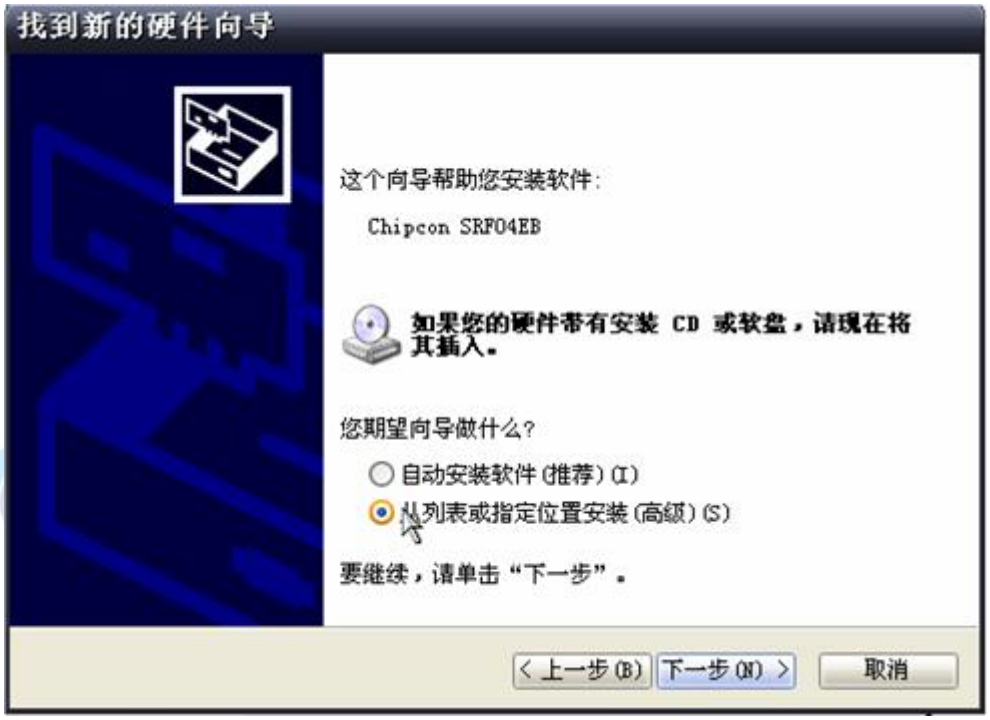


图2-13　手动安装

选择“在搜索中包括的位置”，如图2-14：

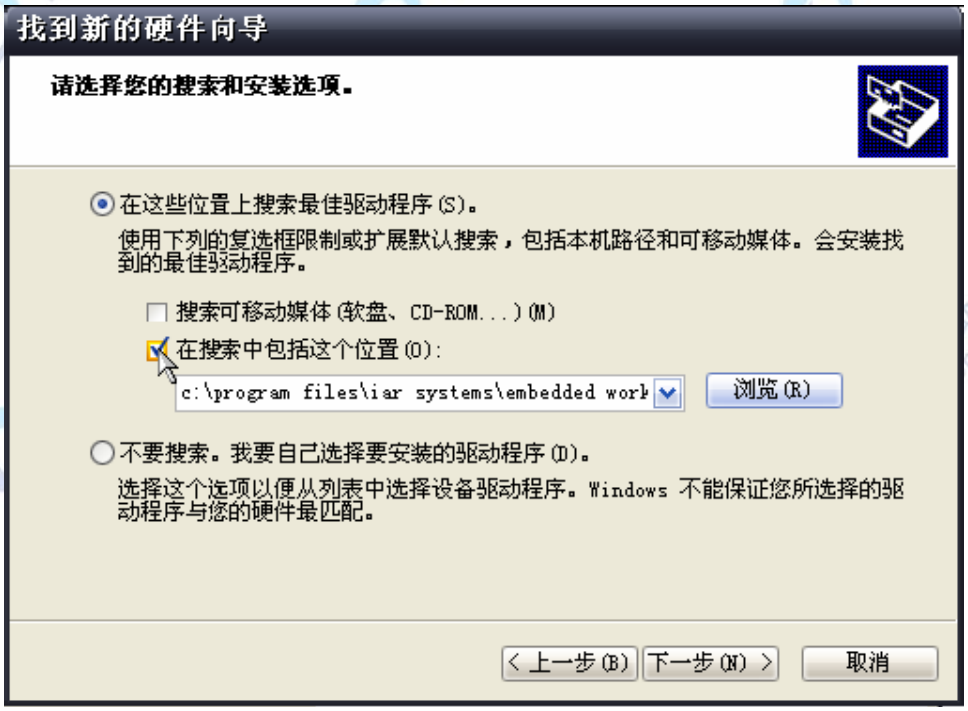


图2-14　搜索位置

选择“在搜索中包括的位置”，如图2-15：

图2-15　搜索位置路径

在 IAR 的安装路径中找到Texas Instruments文件夹（路径：“C:\Program Files\IAR Systems\Embedded Workbench 5.3 Evaluation version\8051\drivers\Texas Instruments”），按系统提示直至完成安装，如图2-16所示。

图2-16　完成安装

注：如果电脑中没有安装 IAR 或者是仿真器的驱动丢失，可以直接安装仿真器驱动，仿

真器驱动的目录：“E-Box300\02-开发环境搭建\02-WSN 开发调试环境\ZigBeeJTAG 仿真器驱动”运行 ebinstaller.exe 即可。 即完成了仿真器驱动的安装。

2.2.3 安装物理地址烧写软件

打开物理地址烧写软件安装程序 Setup_SmartRFProgr_1.6.2.exe（物理地址烧写软件的目录位置：“E-Box300\02-开发环境搭建\02-WSN 开发调试环境\物理地址烧写软件”），显示如图 2-17 所示界面。

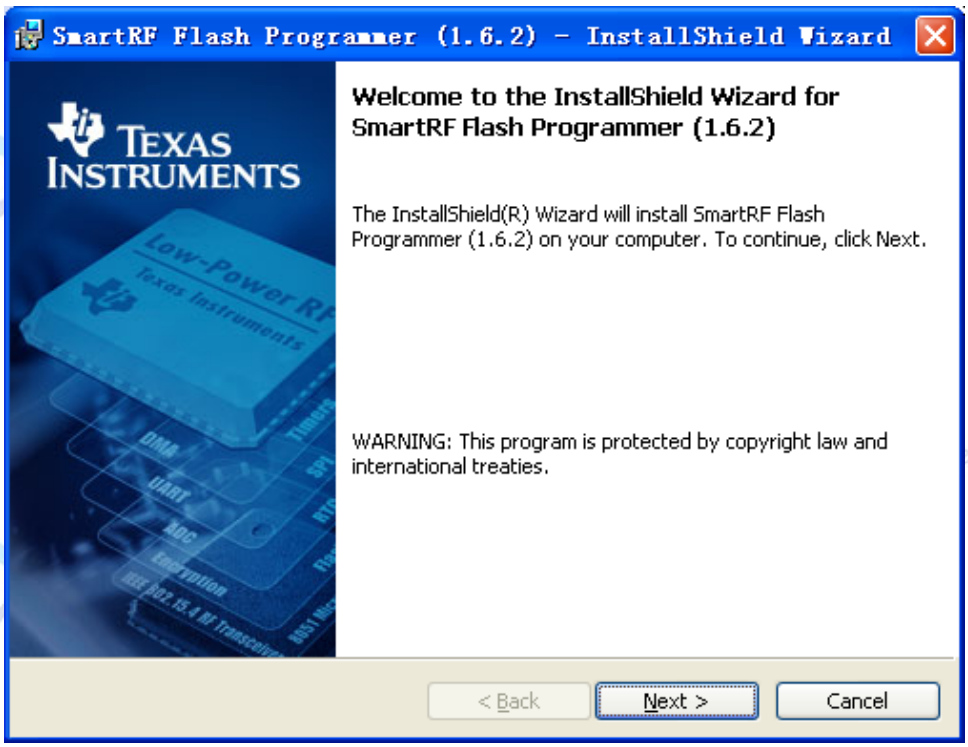


图2-17 物理地址烧写软件安装

点击“next”继续，选择安装路径（默认即可），显示如图 2-18 所示界面：

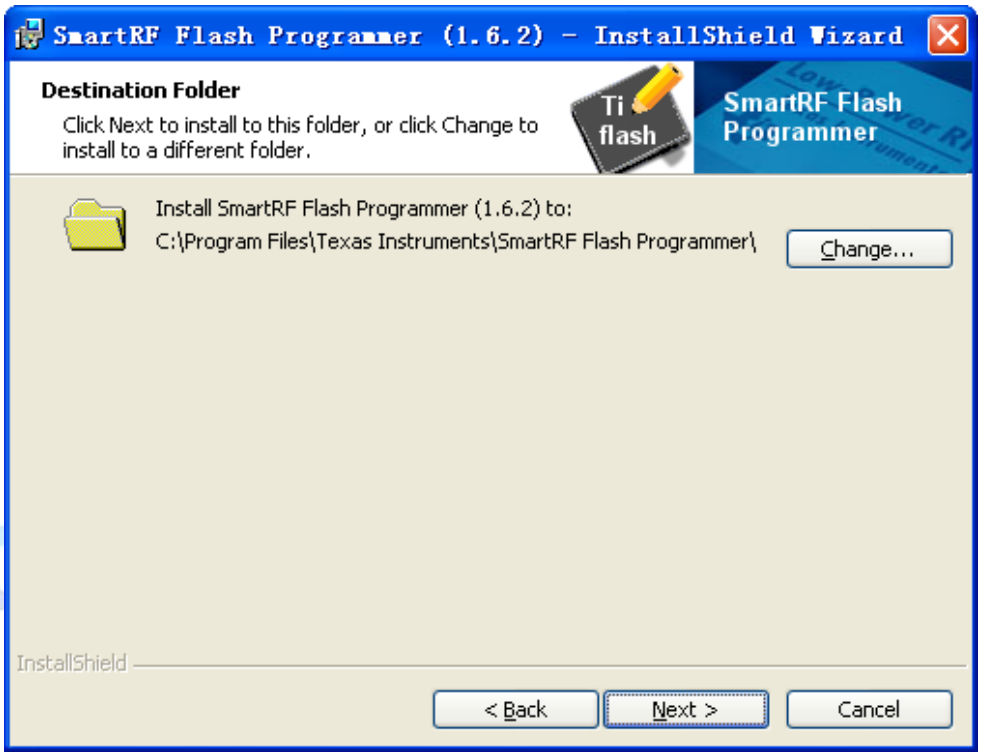


图2-18　安装路径

继续点击“next”，显示图 2-19 所示界面。

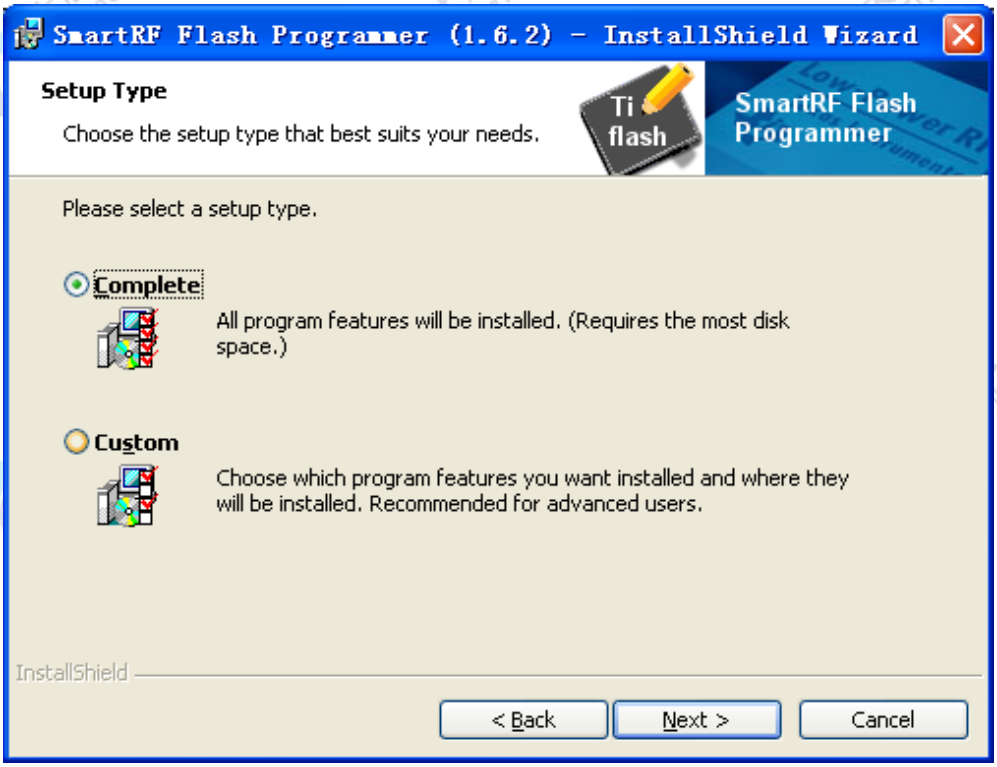


图2-19　安装界面

点击“next”显示图 2-20 所示界面，点击“Install”，开始安装。

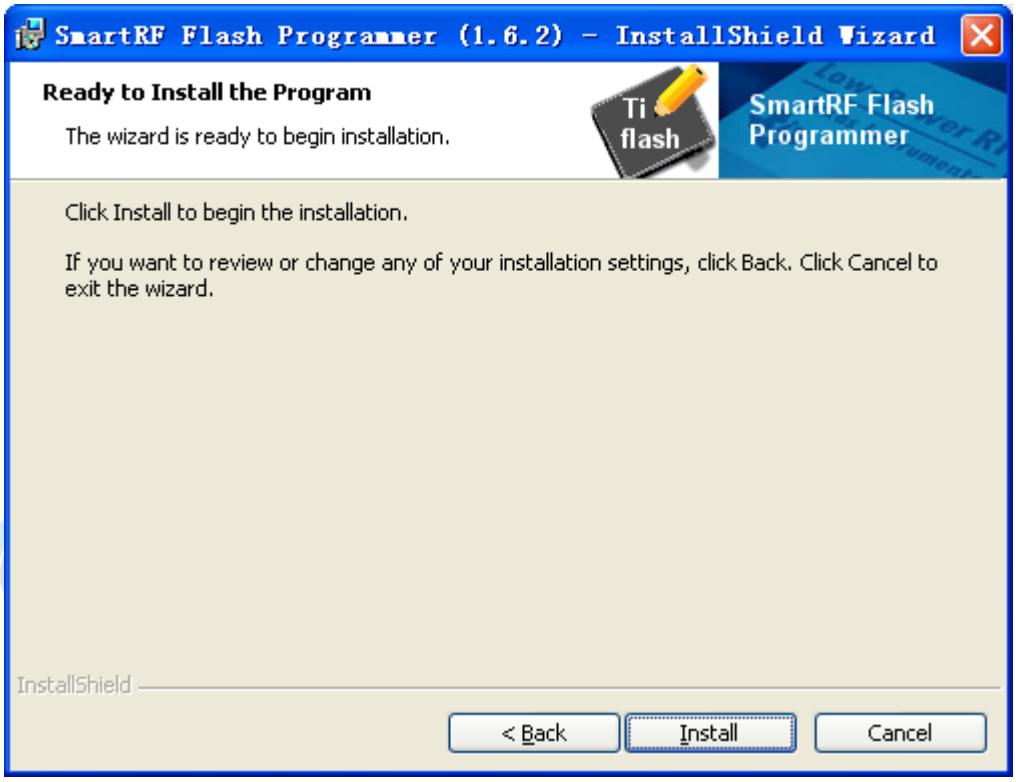


图2-20　安装界面

安装完成，显示图 2-21 所示界面。

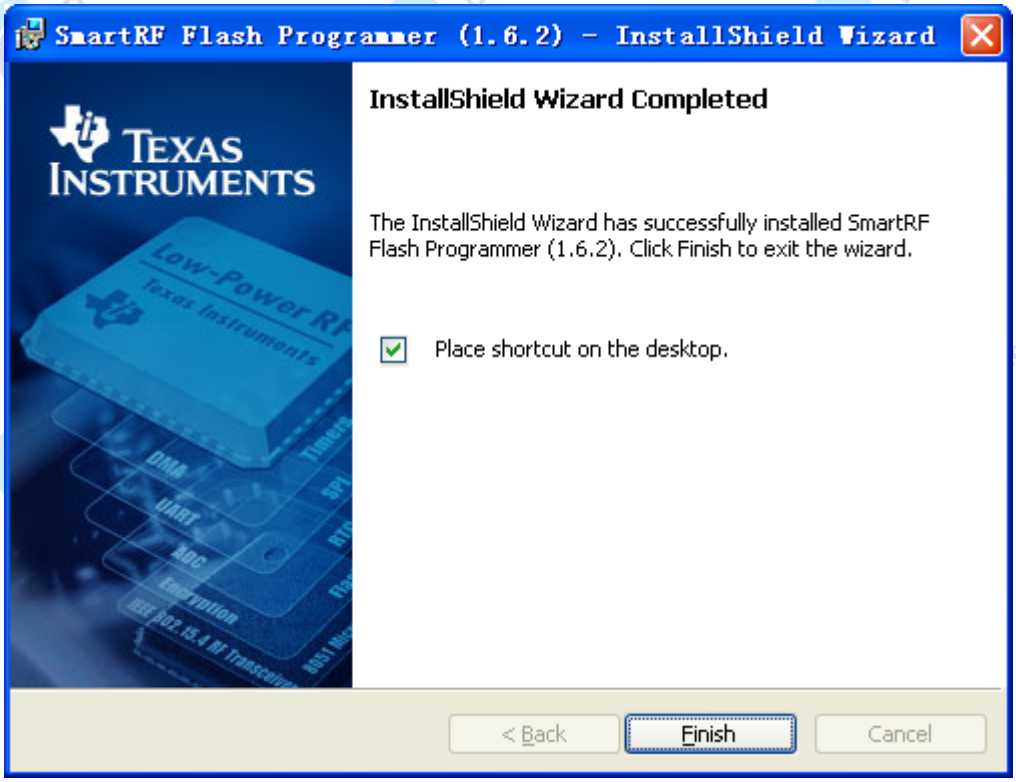


图2-21　物理地址烧写软件安装完成

点击“finish”，退出安装程序。物理地址烧写软件位置：开始菜单中选择 Texas Instruments

→SmarRF Flash Programmer，如图 2-22 所示：

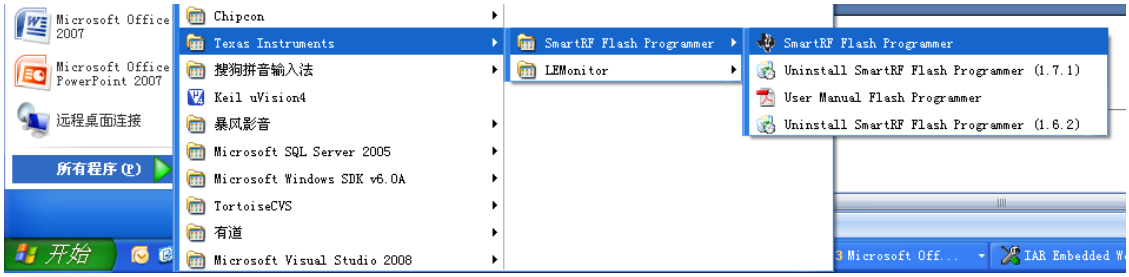


图2-22　物理地址烧写软件位置

2.2.4　IAR 的使用方法

2.2.4.1　新建一个工程

打开 IAR Embedded Workbench 软件，选择 Project→Create New Project…图 2-23 ：

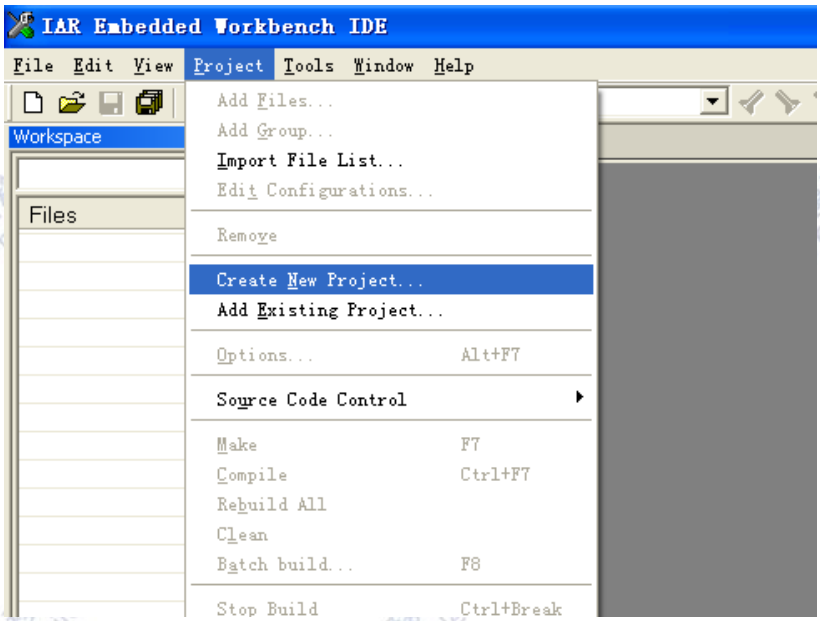


图2-23　新建一个工程

选择 Empty project 默认配置，如图2-24所示：

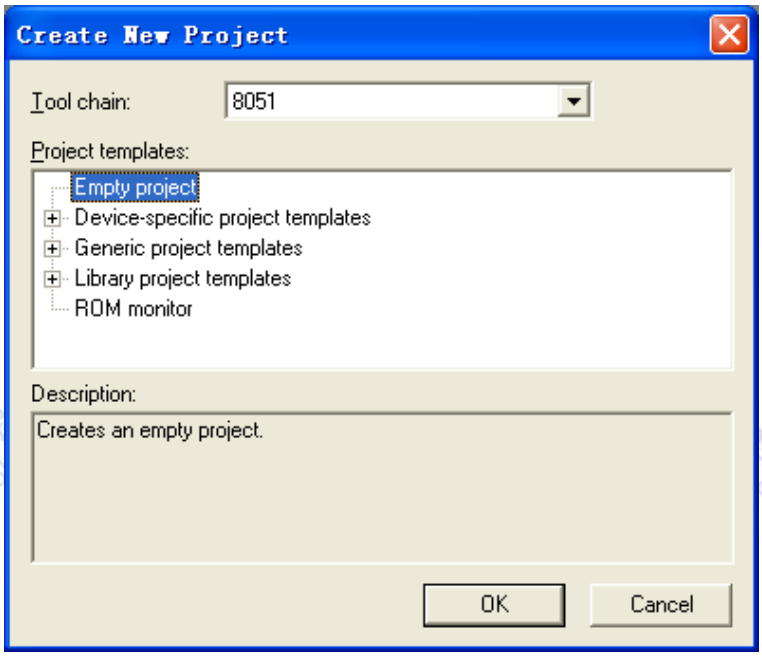


图2-24　选择配置

单击 0K 弹出保存对话框，如图2-25所示：

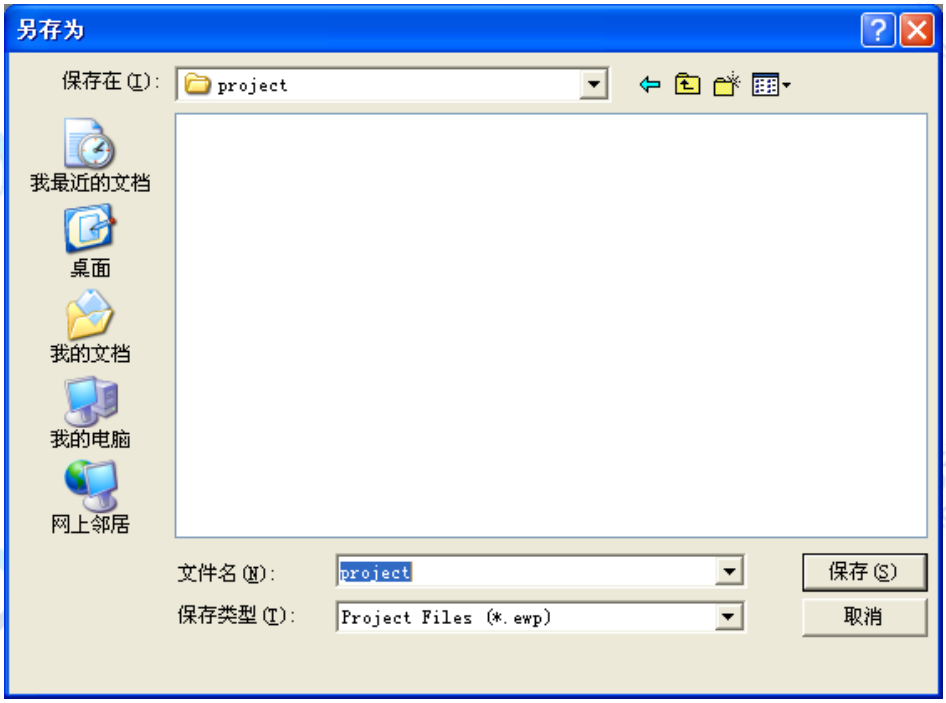


图2-25　保存对话框

这个时候我们选择将其保存在之前已在桌面上建立的一个名为project的文件夹中，并将项目也取名为“project”，会产生一个ewp后缀的文件。

然后选择File→Save Workspace，如图2-26所示，保存工程，弹出保存工程对话框，如图2-27所示：

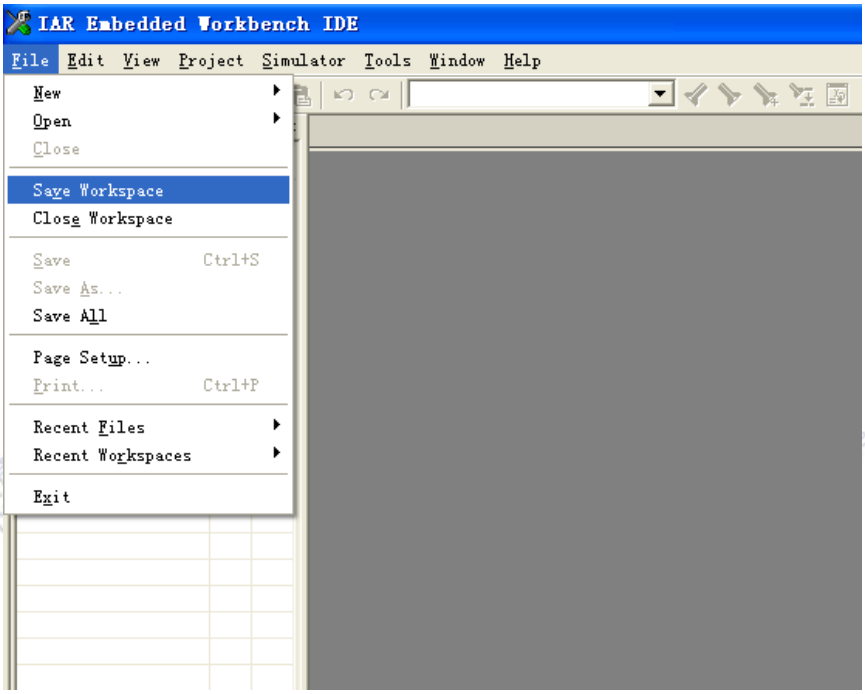


图2-26　选择保存工程

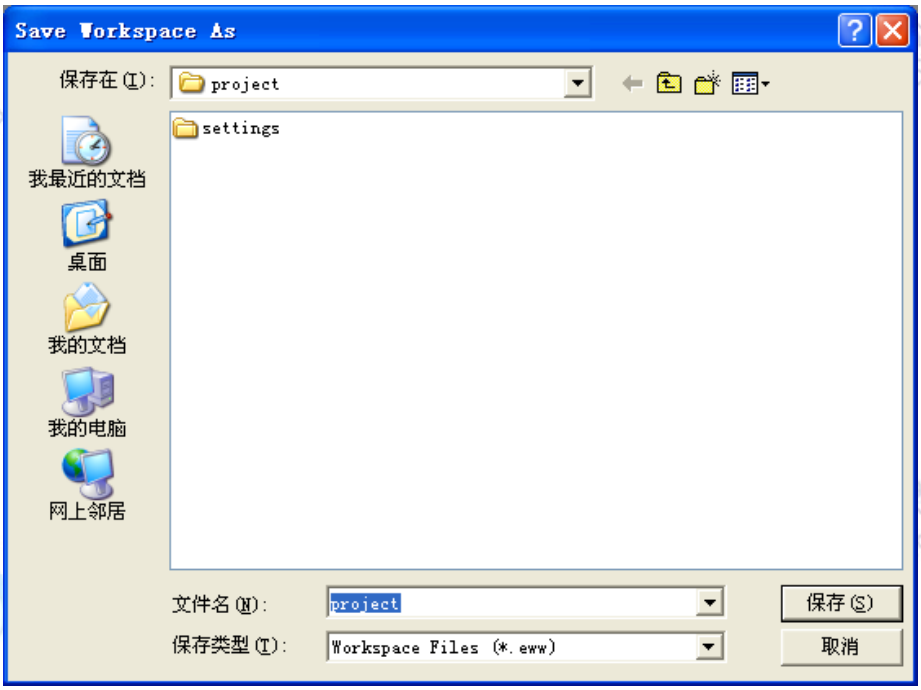


图2-27　保存工程对话框

输入工程文件名，单击保存退出，系统将产生一个以eww为后缀的文件。这样，我们就建立了IAR的一个工程文件。

2.2.4.2　参数设置

接下来，我们对这个工程加入一些特有的配置。 选择Project→Options..。如图2-28所

示：

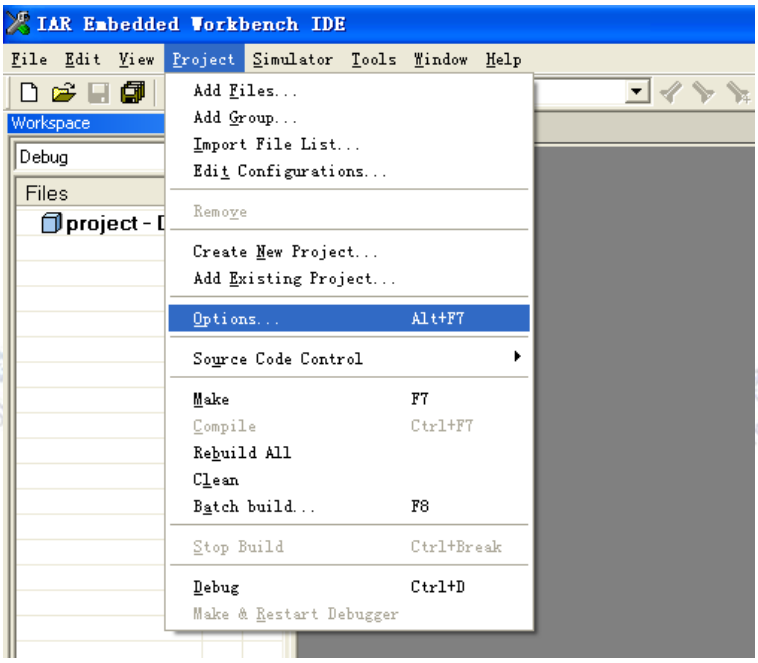


图2-28　打开工程选项

显示工程选项页面，如图2-29所示：

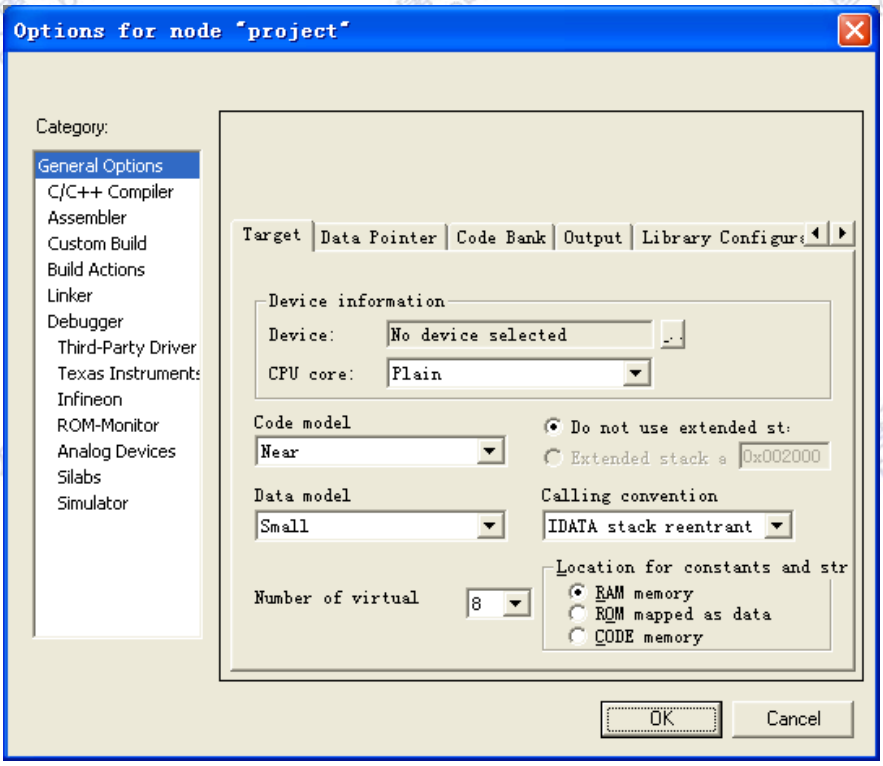


图2-29　工程选项页面

工程选项页面中需要设置很多必要的参数，下面我们针对CC2531来配置这些参数。

1. General Options 设置

在 General Options→Target 选项中 Device 选择为CC2530，如图2-30、图2-31所示。（由于ZigBeePRO协议栈是以CC2530为基准的，所以这里我们将Device选择为CC2530，CC2531与CC2530区别很小。）

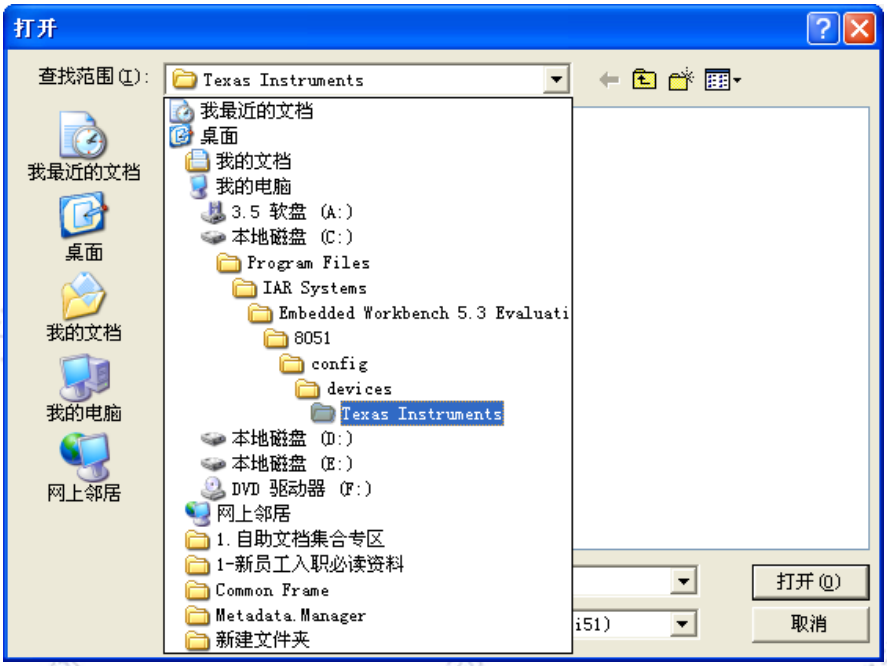


图2-30　找到 Texas Instruments 文件夹

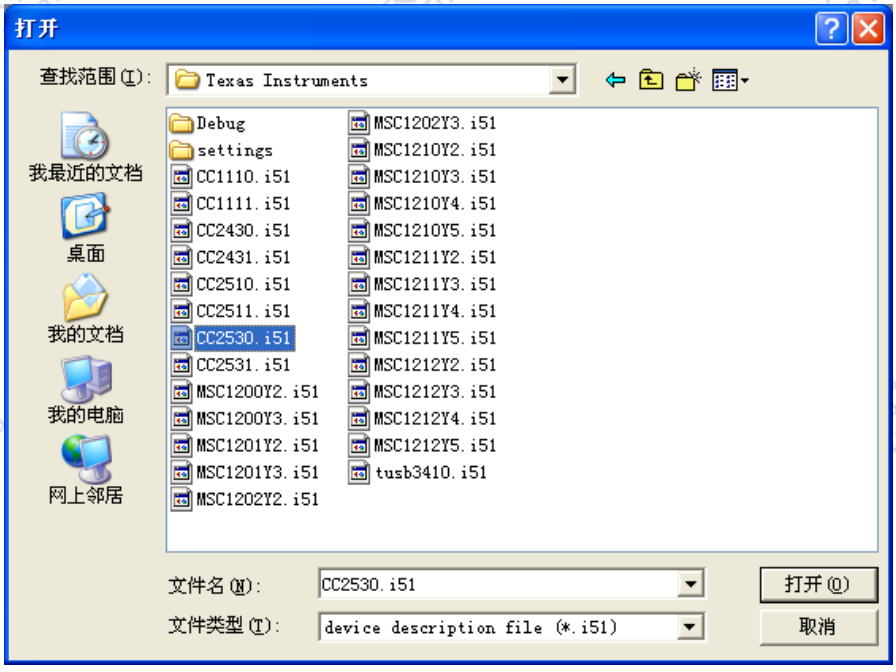


图2-31　选择需要的芯片

在General Options->Target 选项中Data model选择为Large，Calling cinvention选择为XDATA，如图2-32所示：

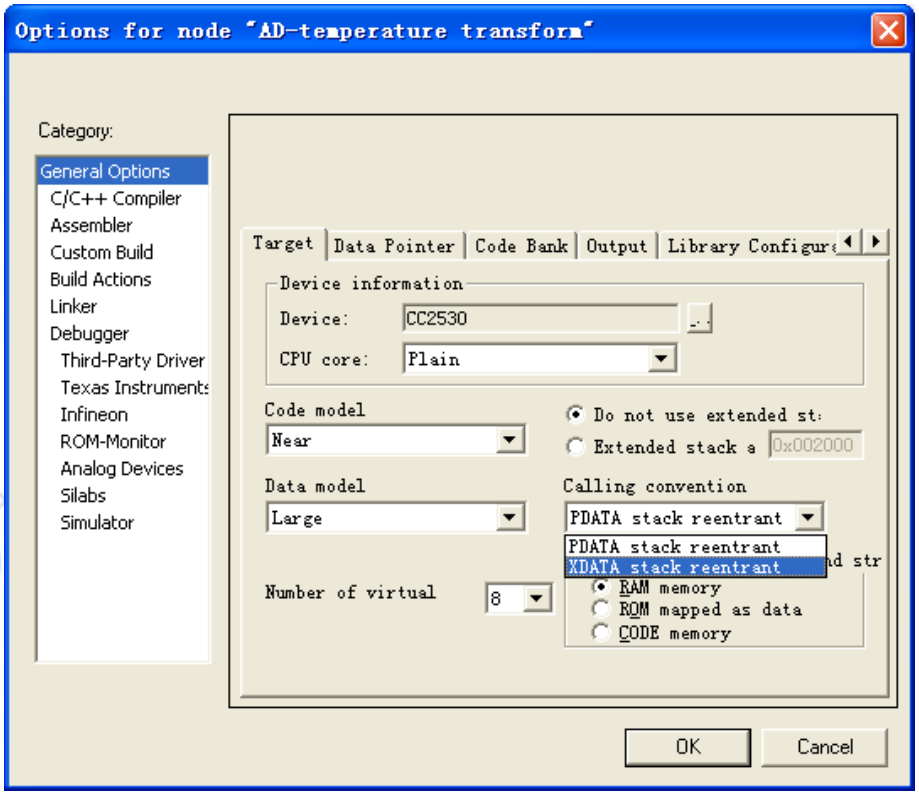


图2-32　修改 Calling cinvention

2. C/C++ Compiler 设置

在 C/C++ Compile->Preprocessor 选项中有两个很重要的选项，它们分别是 Include paths和Defined symbols。 Ignore Standard include directories表示是否忽略在工程中包含文件的路径（选择默认不勾选即可），Defined symbols表示在工程中的宏定义。如图2-33所示：

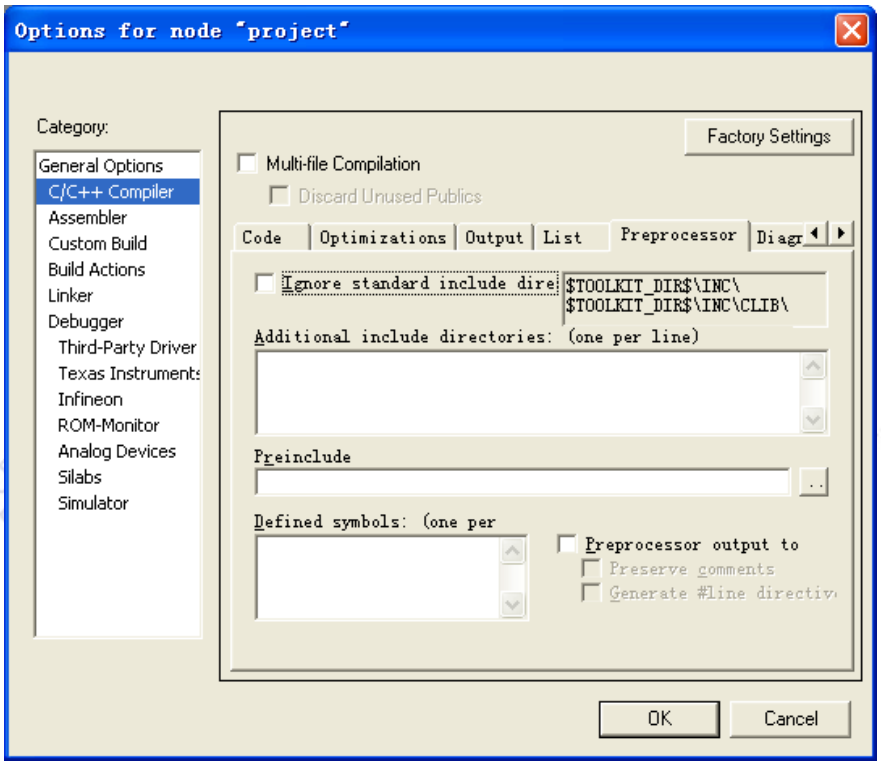


图2-33　C/C++ Compiler 设置

2.1 在定义包含文件路径的文本框中，定义包含文件的路径有两种很重要的语法：

一是$TOOLKIT_DIR$，这个语法表示包含文件的路径在IAR安装路径的8051文件夹下，也就是说如果IAR安装在C盘中，那么它就表示C:\Program Files\IAR Systems\Embedded Workbench 4.05 Evaluation version\8051这个路径。

二是$PROJ_DIR$，这个语法表示包含文件的路径在工程文件中，也就是和eww文件和ewp文件相同的目录。我们刚才建立的project项目中，如果使用了这个语言，那么就表示现在这个文件指向了C:\Documents and Settings\Administrator\桌面\project 这个文件夹。

和这两个语言配合使用的还有两个很重要的符号，这就是“\..”和“\文件夹名”。

“\..”：表示返回上一级文件夹。

“\文件夹名”：表示进入名为“文件夹名”的文件夹。

我们来具体看两个例子：

$TOOLKIT_DIR$\inc\ ：这句话的意思是包含文件指向C:\Program Files\IAR Systems\Embedded Workbench 4.05 Evaluation version\8051\inc。

$PROJ_DIR$\..\Source ：这句话的意思是包含文件指向工程目录的上一级目录中的Source文件夹中。例如：假设我们的工程放在D:\project\IAR中，那么$PROJ_DIR$\..\就将路径指向了D:\project中，再执行\Source，就表示将路径指向了D:\project\Source中。

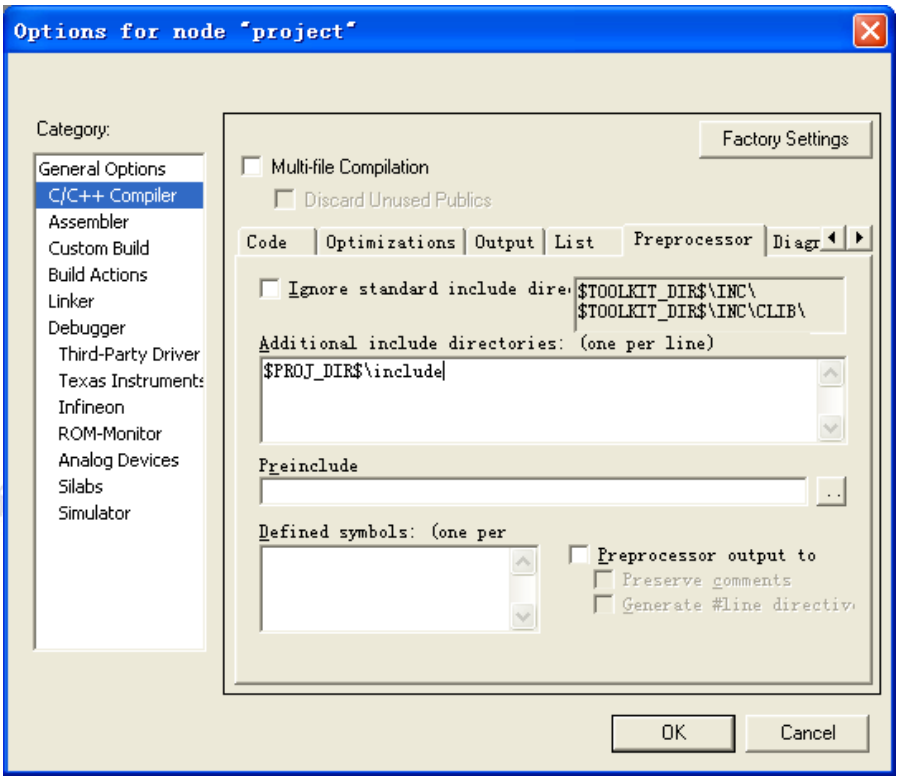


图2-34　设置工程路径

继续回到我们的工程，下面我们通过上面的方法设定一些必要的路径。如上图2-34所示，$TOOLKIT_DIR$\INC\中存放了CC2531的h文件，$TOOLKIT_DIR$\inc\clib\中有很多常用的h文件。一般这两个路径是必须要添加的。还有$PROJ_DIR$\include，是一个包含在工程中的include文件夹，这个文件夹需要自己在工程文件中创建，一般自定义的头文件可以放在这个文件夹中，编程时只要在main函数中用#include声明即可。

2.2　在宏定义文件的文本框中，是用于用户自定义的一些宏定义，他的功能和#define相似，在这里就不多讲，在后面的应用中，会根据具体的使用给出使用方法。

3. linker 设置

Linker-> Output 中是关于输出文件格式的设置。选择如图2-35所示的选项即可实现IAR的在线调试。

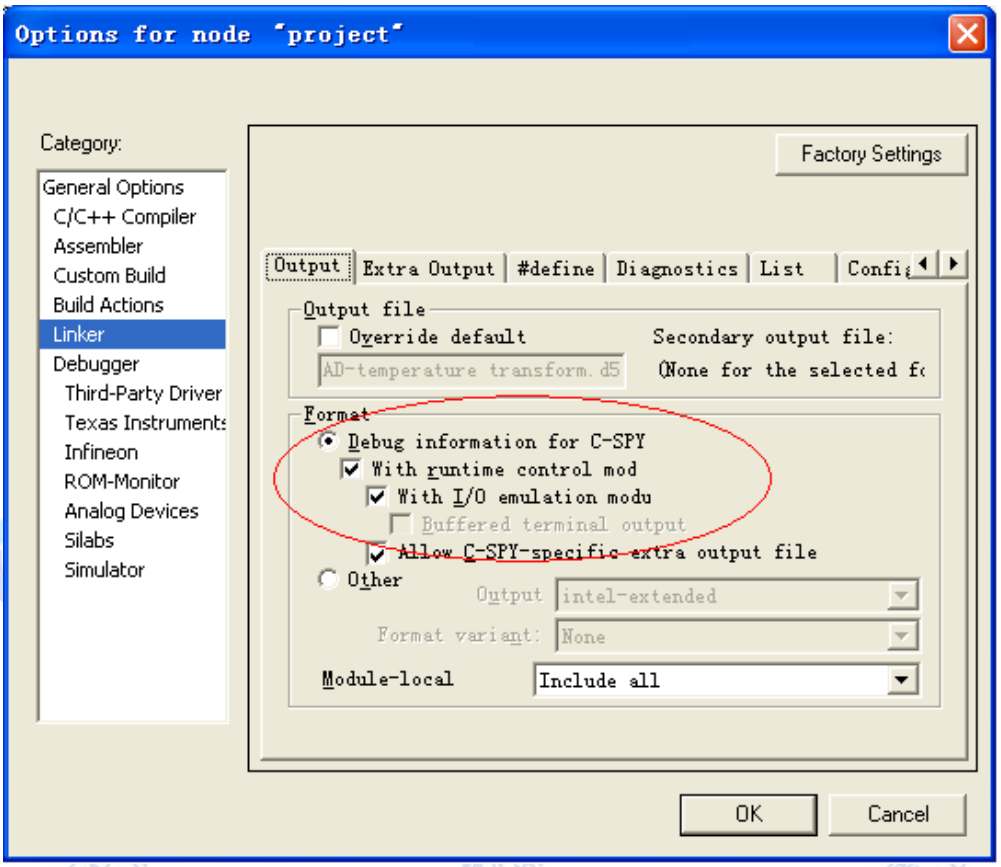


图2-35　Output

在 Linker->Config 中 linker command file 选择lnk51ew_CC2530.xcl，如图2-36所示：

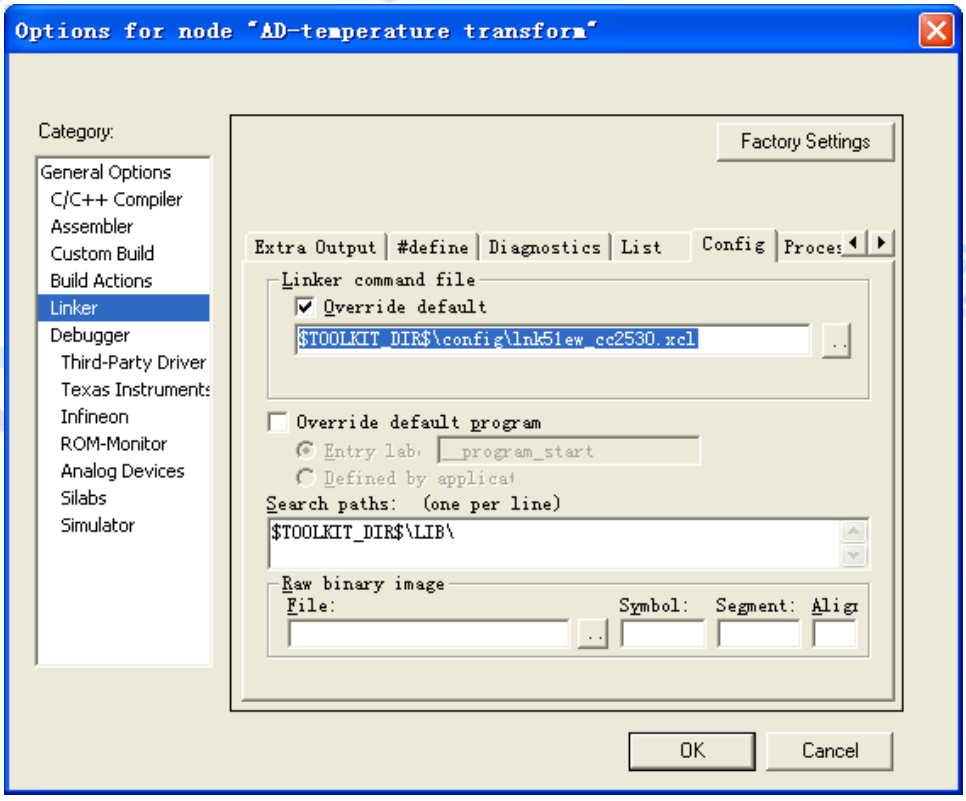


图2-36　设置 linker command file

4. Debugger 设置

如图2-37所示，在Debugger->Setup中Driver项中选择Texas Instruments。

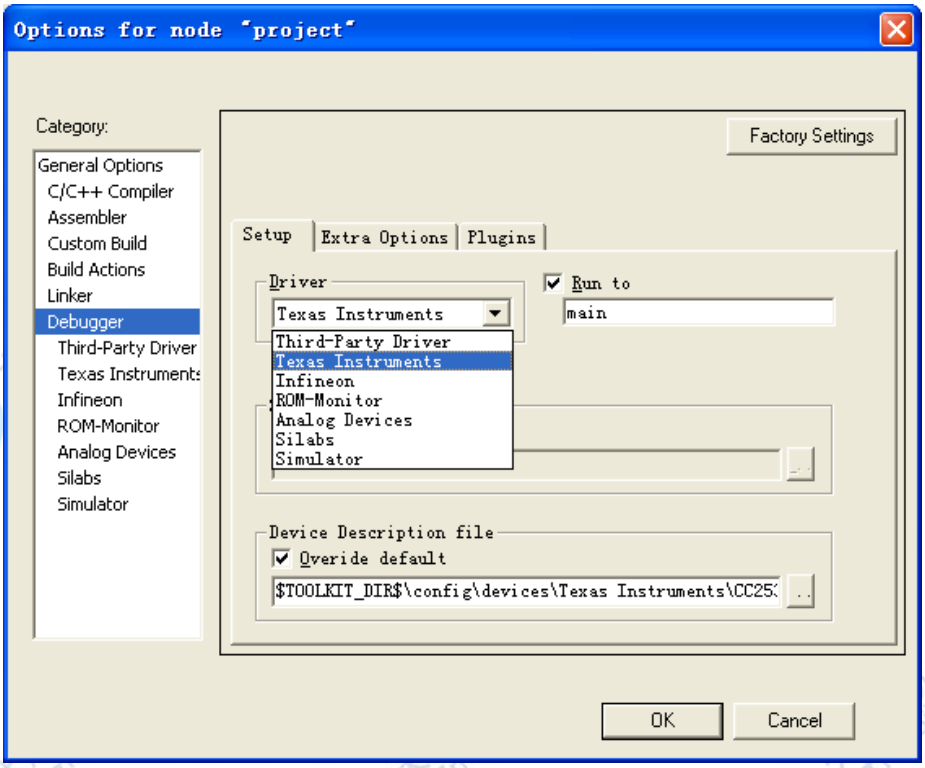


图2-37　设置 Debugger

到此对于整个项目的基本设置就完成了，现在开始第一个项目开发。

2.2.4.3 第一个项目

新建一个C文件，选择New→file，并保存，如图2-38、图2-39所示：

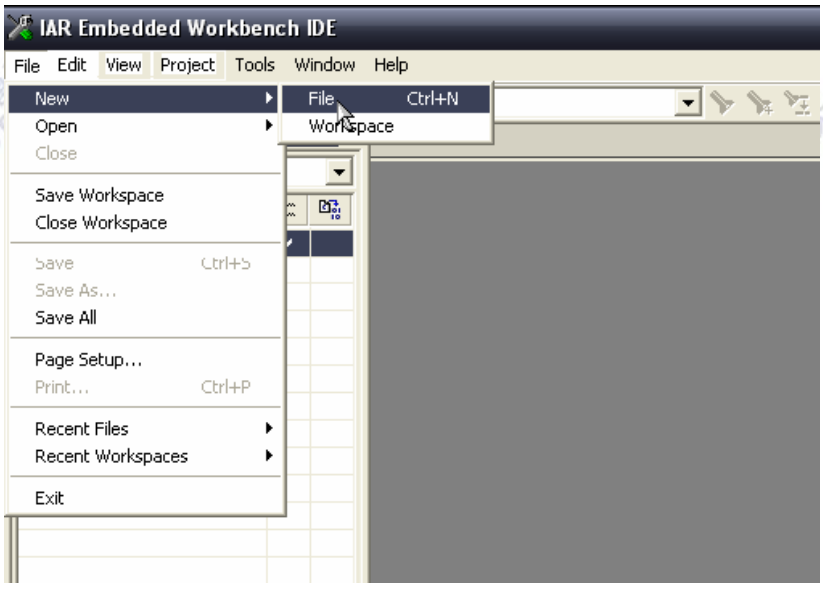


图2-38　新建一个文件

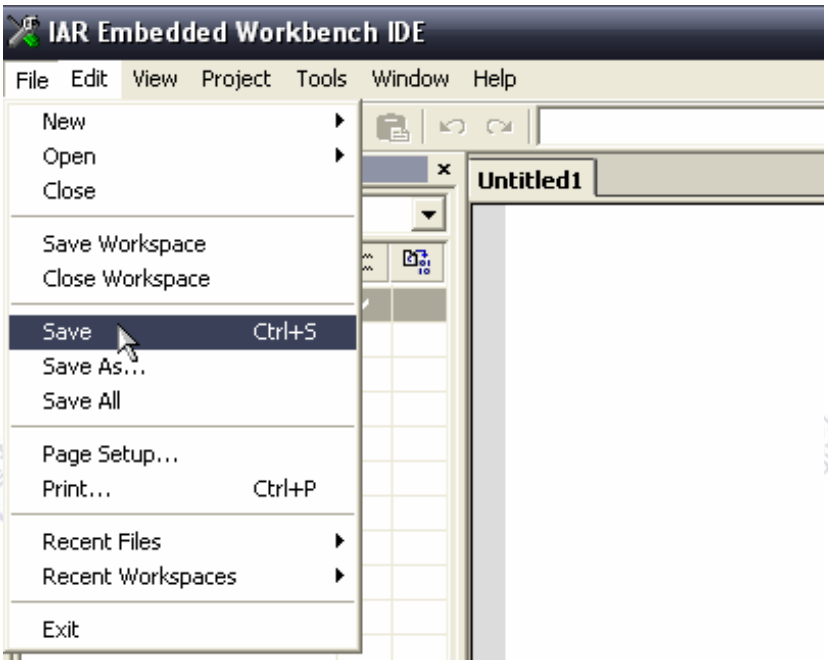


图2-39　保存文件

输入文件名后点击保存，如果是C文件请务必添加“.c”后缀，否则会以文本文件存档。如图2-40：

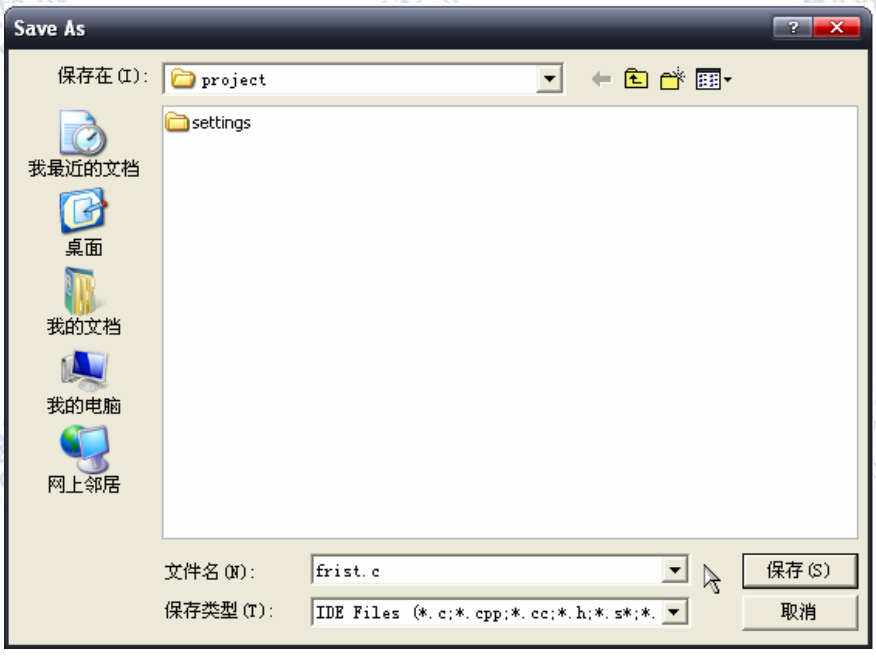


图2-40　输入文件名并保存

右击所建的工程 Project，在弹出的工具栏中选择 Add Group，创建一个文件组。如图 2-41 所示：

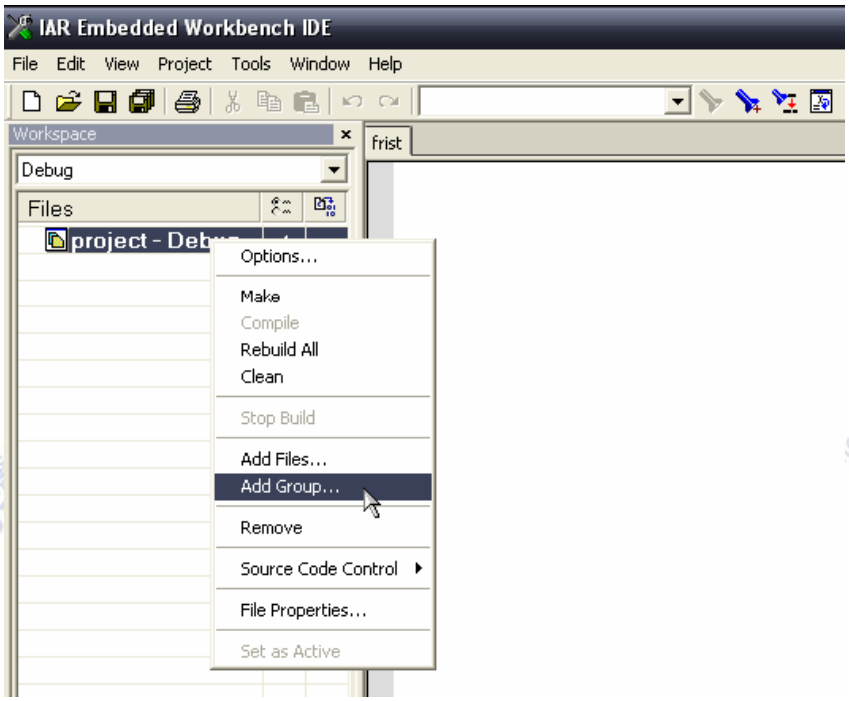


图2-41　创建一个文件组

输入文件组名，如图2-42所示：

图2-42　输入文件组名

选中刚创建的文件组main右击，在弹出的工具栏中选择Add Files，加入”first,c”文件。如图2-43、图2-44所示：

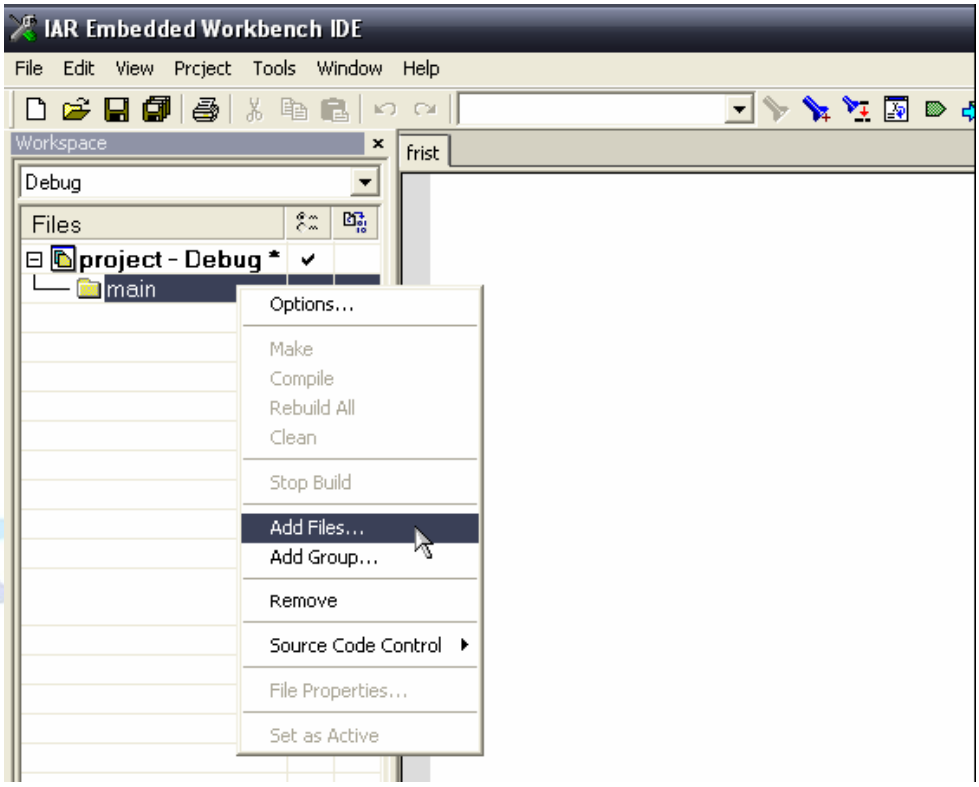


图2-43　加入文件

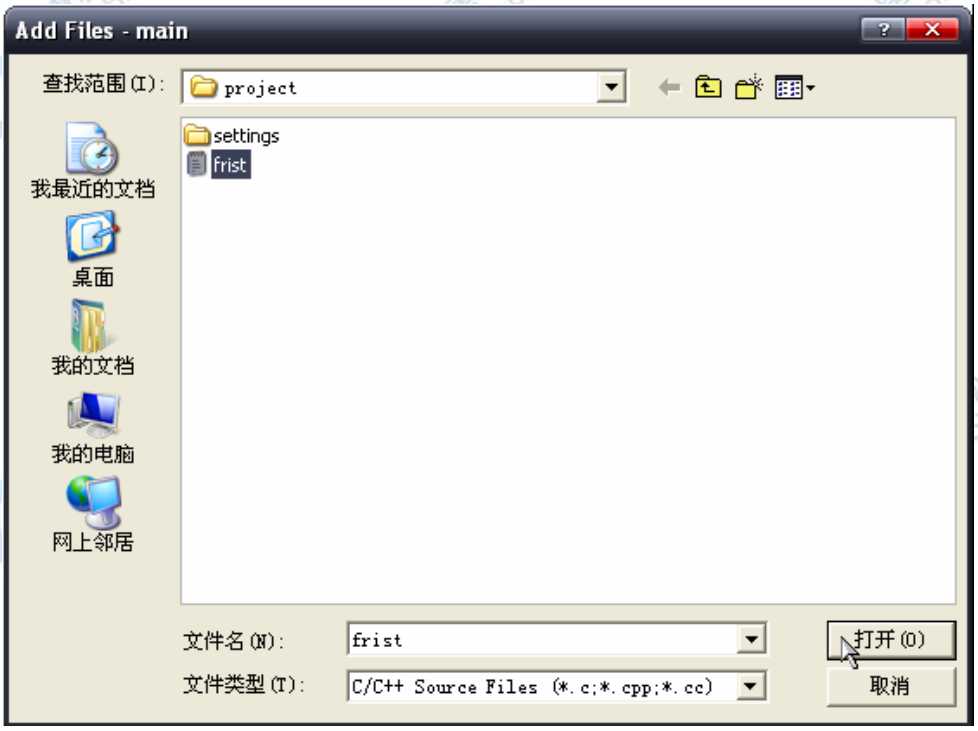


图2-44　选择新建的 C 文件

文件已经加入工程中，如图2-45所示，双击打开文件

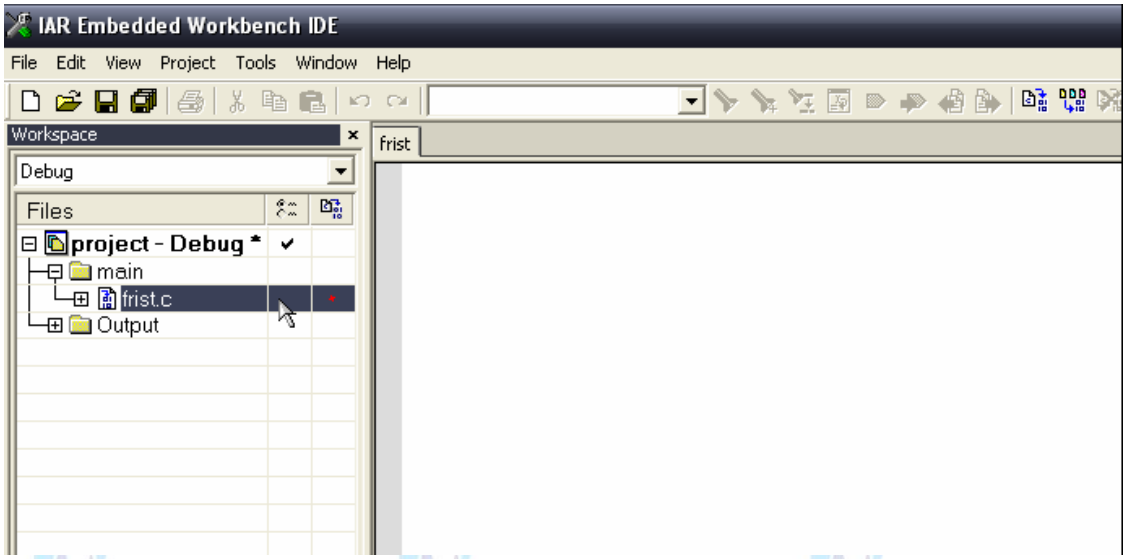


图2-45　打开文件

接下来，在”first.c”中加入第一个代码，如图 2-46 所示。这个代码的意思是将 P1 口设置为输出，将 P1 口置 0，在模块和开发板中有小灯在 P1 口上，当执行这个代码的时候，小灯会点亮。

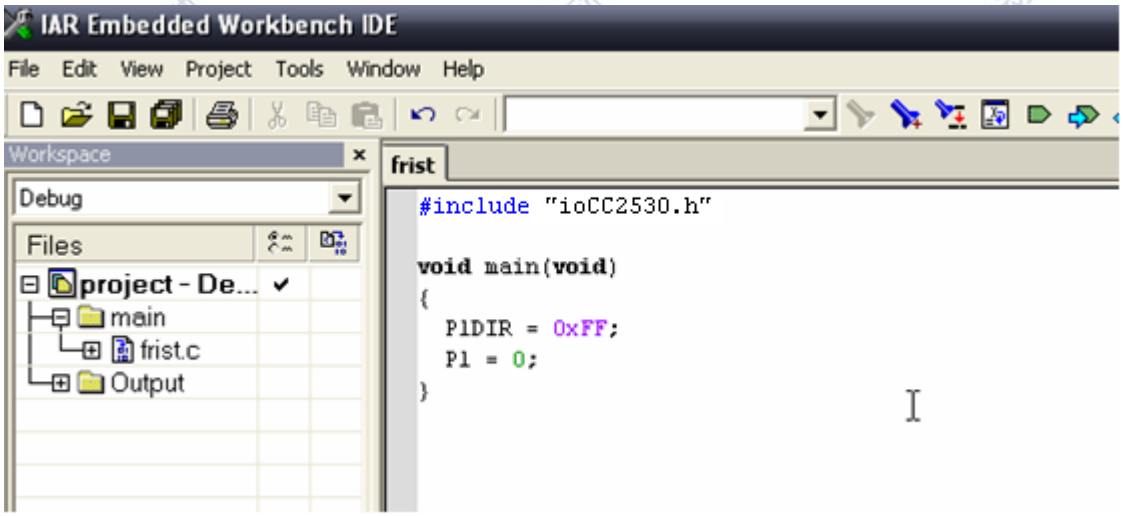


图2-46　代码

在实际的使用中如果IAR的工程路径有中文路径，有可能在调试的时候，设置断点的时候会不能生效，所以为了方便在线调试，我们将建立的工程拷贝到磁盘根目录中。然后打开工程执行Project→Make，如图2-47所示。

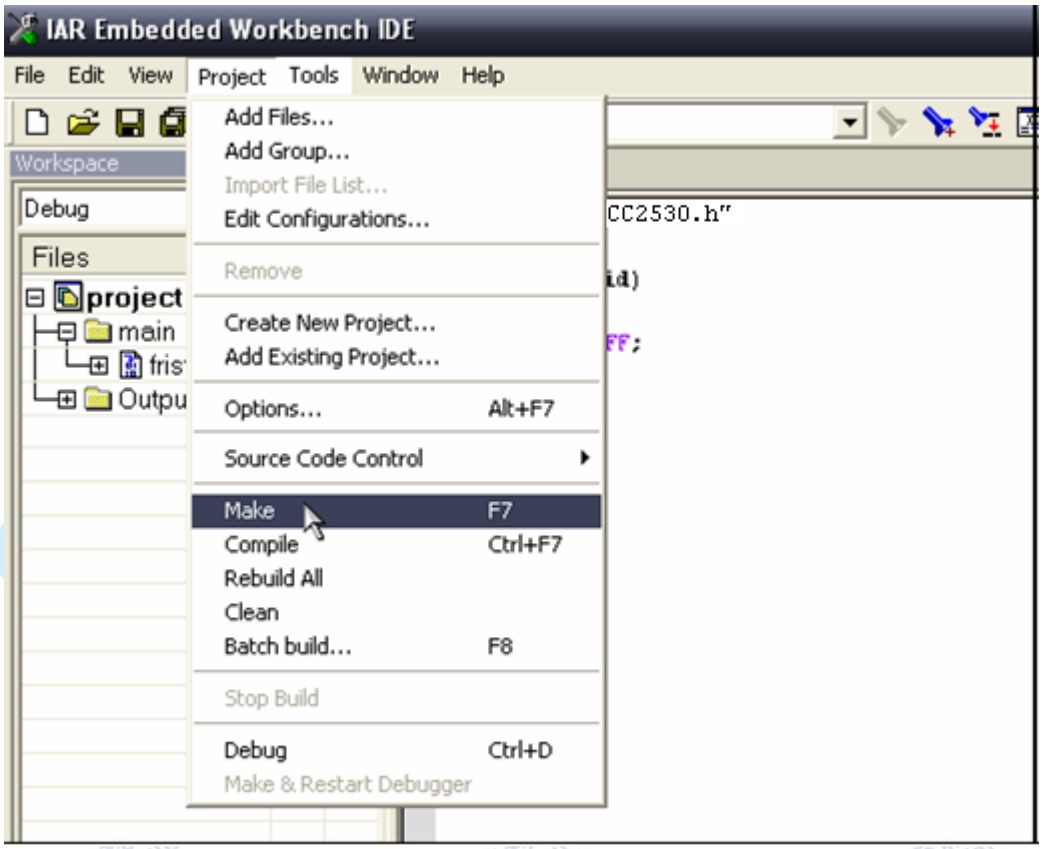


图2-47　编译

可以通过“make”编译，也可以通过Rebuild All全部编译（用 make 只会编译修改过的文件）。编译后只要没有错误就可以使用了，一般警告我们可以放过。有关错误和警告的提示信息如图2-48所示：

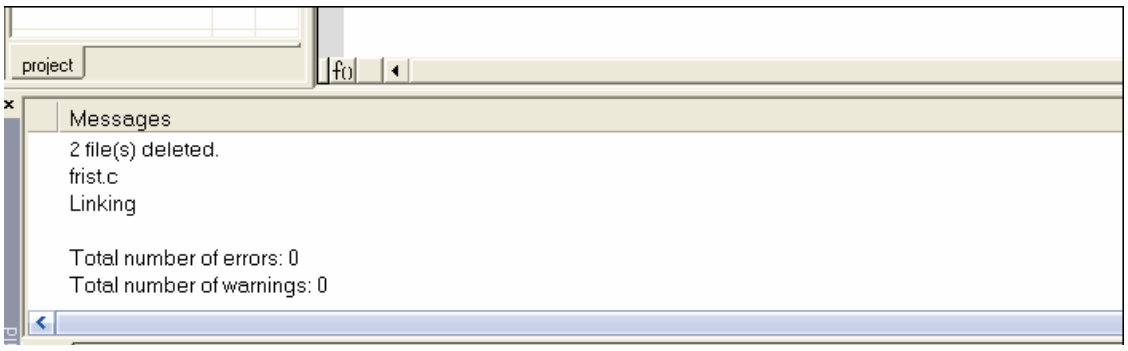


图2-48　提示信息

在编译没有错误后，就可以下载程序了，点击Debug，就现在程序了，下载程序后，软件进入在线仿真模式。如图2-49所示：

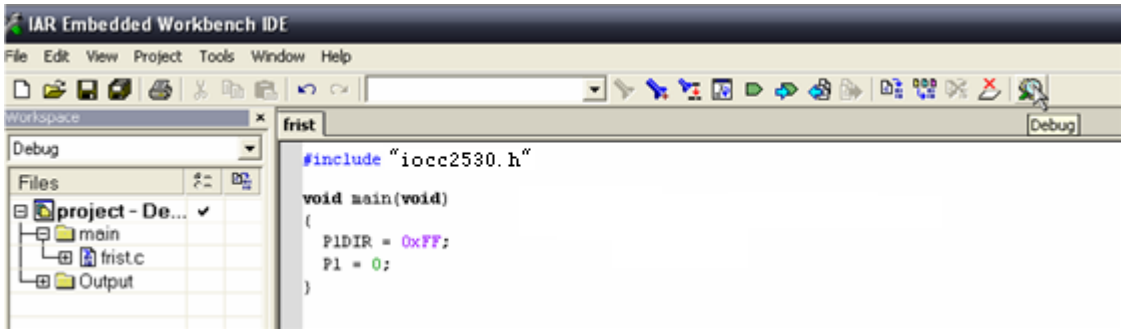


图2-49　Bebug

在仿真模式中，可以对这个文件设置断点，断点的设置方法是首先选择需要设置断点的行，然后单击Toggle Breakpoint设置断点。设置好以后，这行代码会变为红色，这样就表示断点设置已经完成，如图2-50所示：

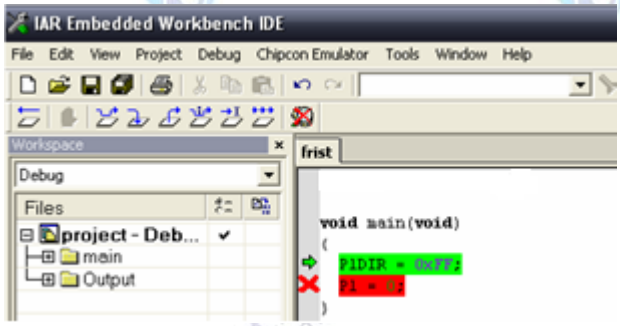


图2-50　设置断点

然后执行全速运行  ，当执行到断点出会停止在断点处，如图 2-51 所示：

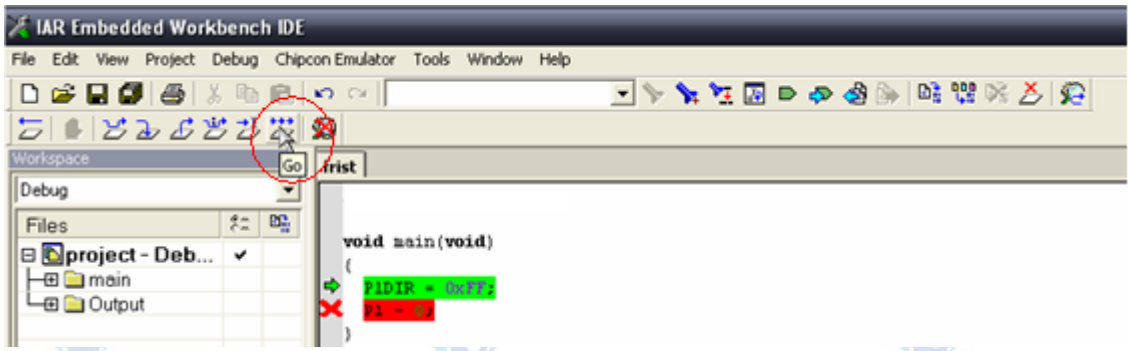


图2-51　运行

然后用鼠标选中“P1DIR”，单击右键，选择Add to Watch或者Quick Watch。如图2-52所示：

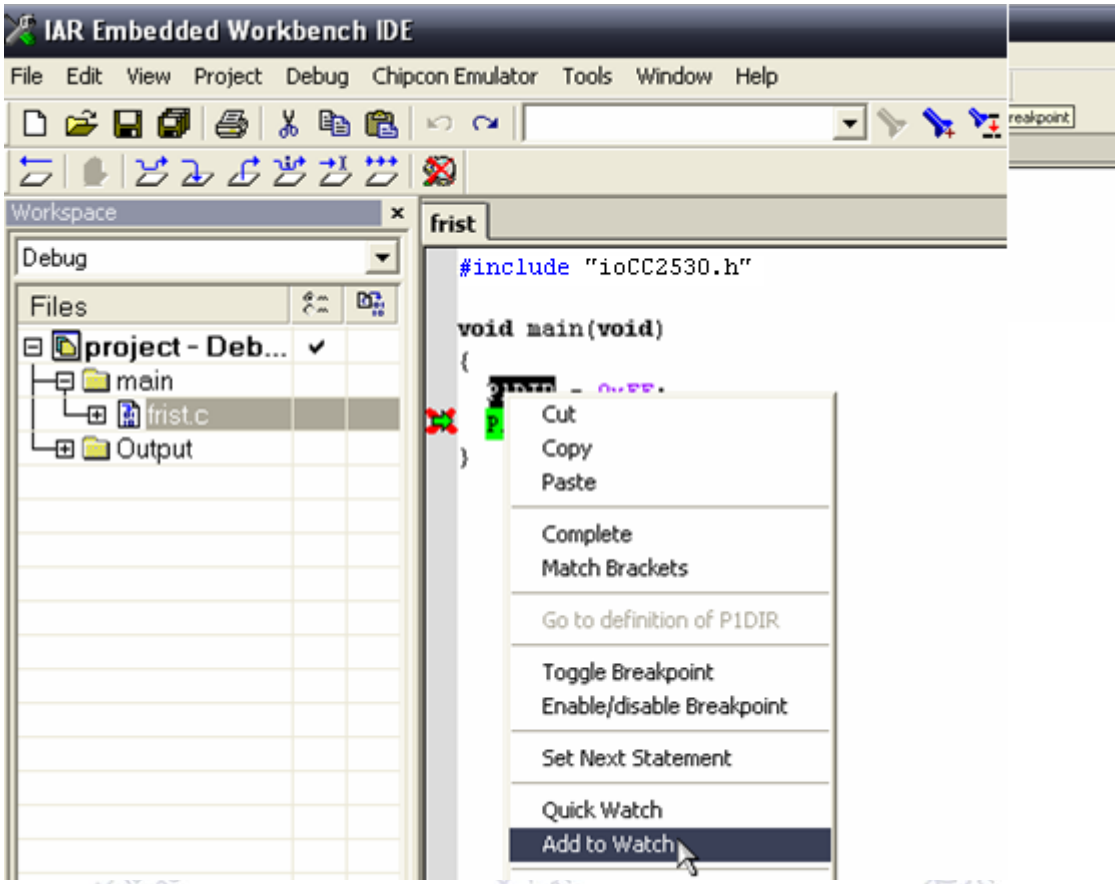


图2-52　watch 查看

这个步骤的作用是查看这个寄存器中的值，如果是一个变量的话，就可以查看一个变量的值。该值在Watch中可以看到，如图2-53所示：

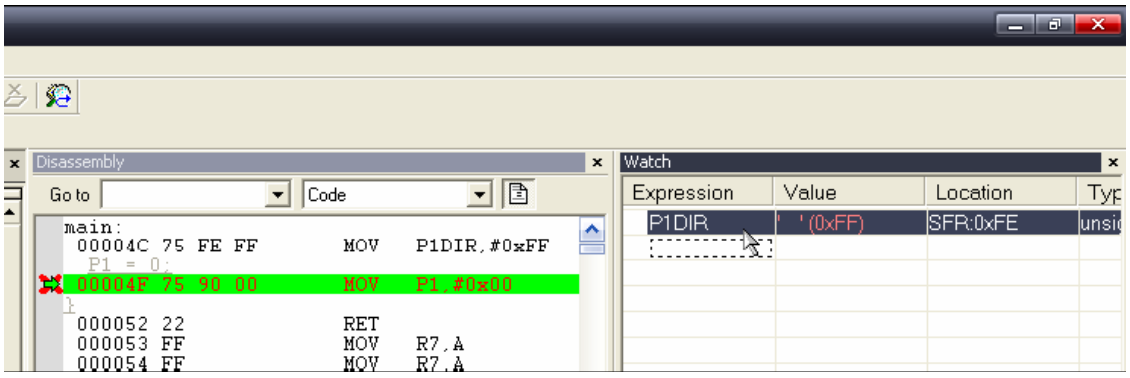


图2-53　查看寄存器值

2.3　物联网网关基础实验平台

物联网的网关主要指的是嵌入式系统。嵌入式系统包括了功能较强的 CPU、DSP、FPGA 及其他专用芯片。它们在性能上的优势可以弥补在功耗、成本等方面的不足，特别适用于能量不十分敏感，对性能及可靠性要求较高的场景，但又不适合使用体积更大的通用计算机或

服务器。嵌入式系统是单片机系统的有力补充。

2.3.1 硬件环境简介

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

按照嵌入式处理器的用途，嵌入式处理器可以分成下面几类：嵌入式微控制器、嵌入式微处理器、嵌入式数字信号处理器、嵌入式片上系统及其他专用处理器。

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

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

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

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

2.3.2 软件环境简介

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

用的嵌入式板卡上运行基于 Linux 的应用，软件可以分为以下 4 个层次：

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

2. Linux 内核。特定于嵌入式板子的定制内核以及内核的启动参数。

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

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

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

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

2.3.3 操作系统简介

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

嵌入式操作系统作为嵌入式系统中极为重要的组成部分，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。嵌入式操作系统具有通用操作系统的基本特点，能够有效管理越来越复杂的系统资源；能够把硬件虚拟化，使得开发人员从繁忙的驱动程序移植和维护中解脱出来；能够提供库函数、驱动程序、工具集以及应用程序。与通用操作系统相比，嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有独特之处。

2.4 在主机上搭建 LINUX 开发环境

开发环境是开发人员在开发过程当中，所需的软硬件。开发环境并不是一个固定的样式，在这里，我们详细讲解一个嵌入式 Linux 开发环境搭建的方法。您已经对嵌入式开发非常了解的话，可以按照自己的需求来搭建环境。如果和本书环境不一样而产生报错，您可以从国内一些大 Linux 论坛和网站搜索相关的信息来解决。本节介绍的环境经过大唐公司的测试，如果对嵌入式开发不是非常熟悉的朋友，希望您按照大唐提供的方法来搭建环境。本节所需

文件路径：E-Box300\02-开发环境搭建\04-主机搭建 linux 环境

2.4.1 安装 UBUNTU10.04

Ubunt 是一个以桌面应用为主的 Linux 操作系统。Ubuntu 拥有很多优点。相对于其他版本的 Linux，Ubuntu 也有着自己的优势。首先，安装系统非常简单，只需要非常少的设置即可，完全可以和 Windows 桌面系统相媲美；其次，图形界面很人性化，模仿了在 xp 下常用的快捷键；还有，安装和升级程序时，可以通过网络，由系统自行安装依赖的文件包，从此不必再为 Linux 系统的依赖关系大伤脑筋。综合考虑大家的使用习惯和学习的需要，我们选用 Ubuntu Linux。

Linux 桌面系统版本众多，目前所有实验和源码在 Ubuntu10.04 版本测试可以通过。使用其他版本 Linux 桌面系统，可能会出现 gcc 编译器和库文件相关的问题。碰到类似问题，可以在 Linux 系统发行商的官方论坛上咨询和查询。如果对 Linux 不熟悉的用户，强烈建议使用大唐介绍的方法。

请注意：在本章中，如没有特殊说明，所执行命令以及设置环境均为 PC 机的 Linux。在每条命令开头加符号“#”以表明命令的开始，并且命令会以灰色背景描出。

现在开始安装：

步骤 1：下载 Ubuntu10.04 的安装程序。

步骤 2：安装程序启动后，会提示选择安装语言种类。使用 PC 键盘的方向键选择在安装过程中显示的语言，在这里我们选择简体中文。如图 2-54 所示：

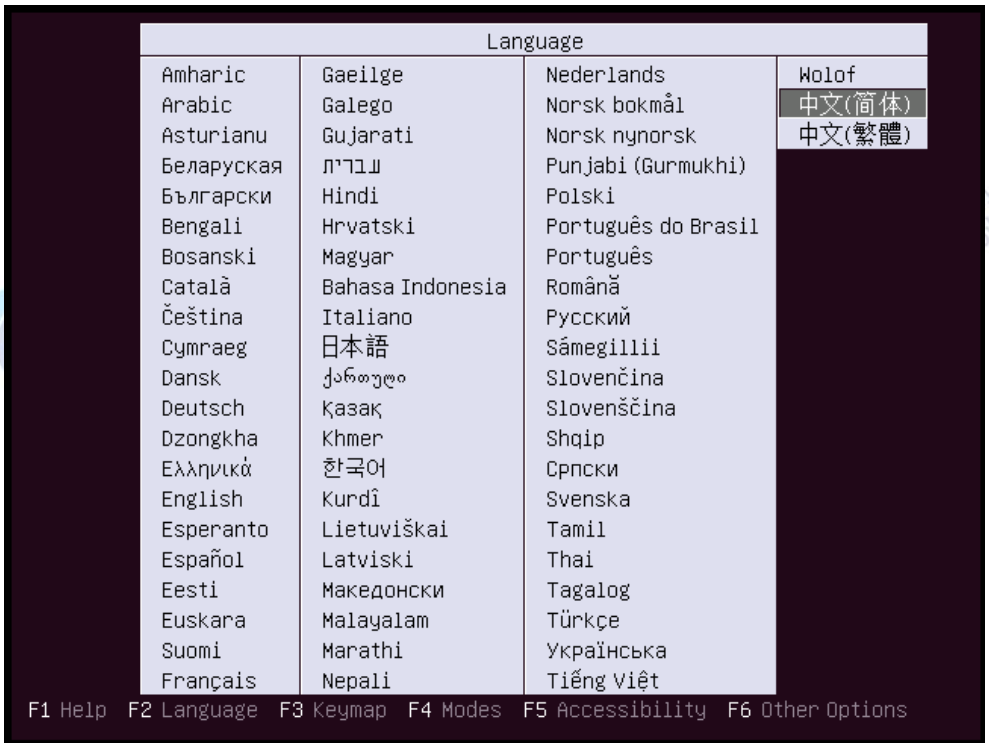


图2-54　选择语言种类

步骤 3：　选择“安装 Ubuntu”，如图 2-55 所示：

图2-55　安装 ubuntu

步骤 4：选择操作系统语言，点击前进。可根据自己的需求选择语言，Ubuntu 支持多种语言，也可以在安装 Ubuntu 完成后更新语言包。如图 2-56 所示：

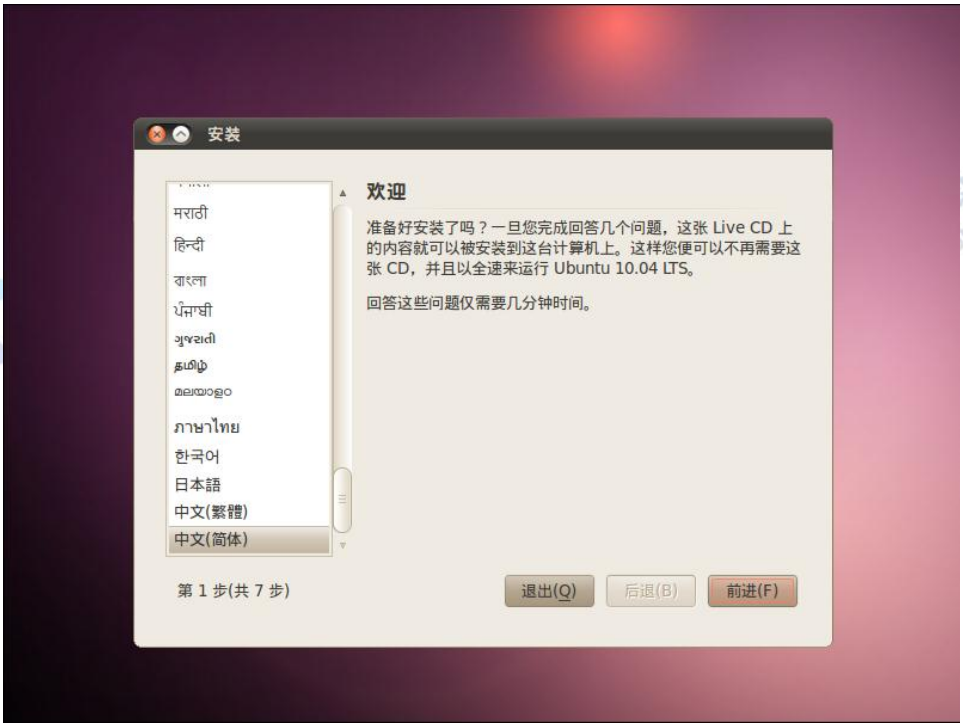


图2-56　选择操作系统语言

步骤 5：系统自动同步操作系统时间。如果不能同步时间，可以点击“跳过”，进入后面的操作。如图 2-57 所示：

图2-57　同步系统时间

步骤 6：选择所在地，因为只有上海可选，所以选择了上海。如图 2-58 所示：

图2-58　选择所在地

步骤 7：选择键盘布局。选择默认。如图 2-59 所示：

图2-59　选择键盘布局

步骤 8：硬盘空间以及挂载点的分配。这里选择默认。也可根据个人需要来进行设置。如图 2-60 所示：

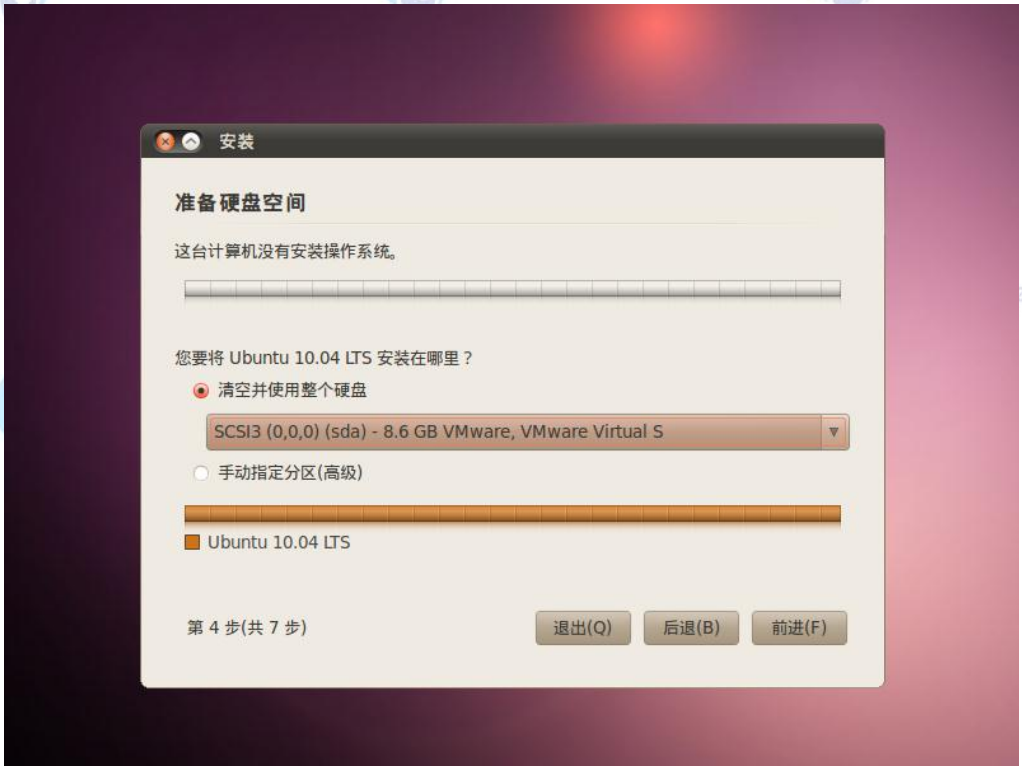


图2-60　准备硬盘空间

步骤 9：填写用户名及用户密码，将启动方式选择为“自动登录”。如图 2-61 所示：

图2-61　填写信息

步骤 10：最后是显示一些安装的配置信息。点击“安装”。如图 2-62 所示：

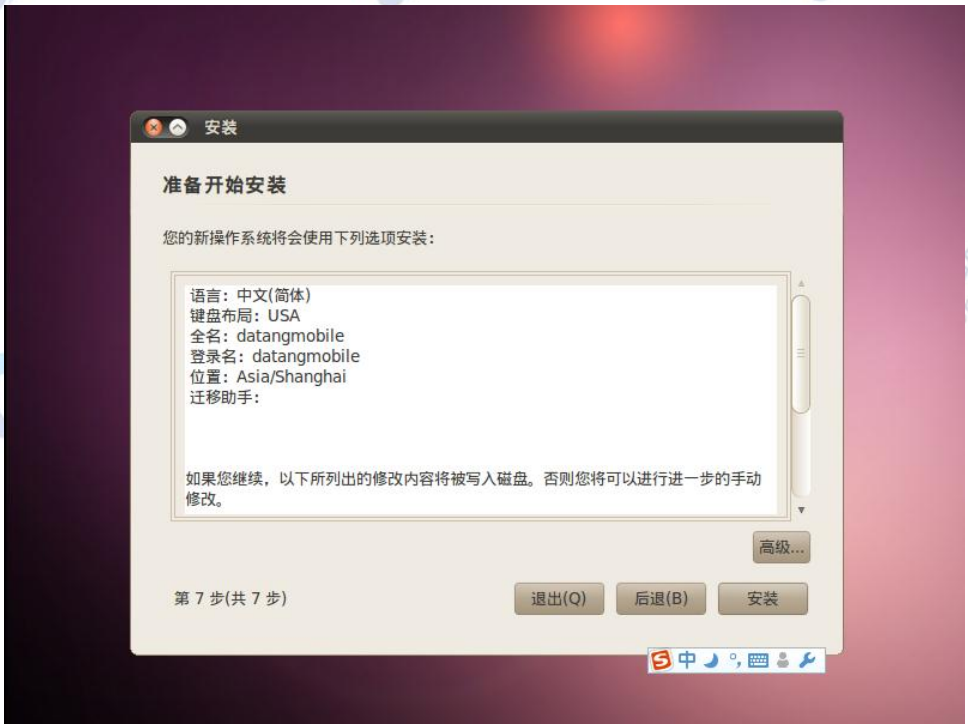


图2-62　开始安装

这样，就开始安装系统了。如图 2-63 所示：

图2-63　正在安装系统

步骤 11：安装完成，点击“现在重启”。如图 2-64 所示：

图2-64　重启计算机

2.4.1.1 LINUX 终端

在 Ubuntu 中，点击应用程序->附件->终端是启动 Linux 终端的一种方法。Linux 终端的作用不再介绍，网络上很多，而且在后面的实验中，会经常用到终端。终端的使用方法，

还是需要掌握的。如图 2-65 所示：

图2-65　终端

2.4.1.2 设置 ROOT 用户自动登录

嵌入式交叉编译，经常需要 root 用户的权限。把登录系统后的终端的默认用户改为 root。如果编译过程当中出现类似权限的问题，要注意查当前用户权限。

在 Ubuntu 中新建一个终端，输入：

```
#sudo　　passwd root
```

输入要设置的密码，这样以后我们就可以用 root 用户登录了。

在终端中输入：

```
#sudo　　gedit　　/etc/gdm/custom.conf
```

这时会弹出文本编辑器，将“custom.conf ”内容修改成下面所示内容（若原来文件为空的话就输入这些内容），如图 2-66 所示。保存关闭，重新启动 Ubuntu 就会发现已经自动用 root 用户登录了。

```
[daemon]
TimedLoginEnable=true
AutomaticLoginEnable=true
TimedLogin=root
```

```
AutomaticLogin=root

TimedLoginDelay=15
```

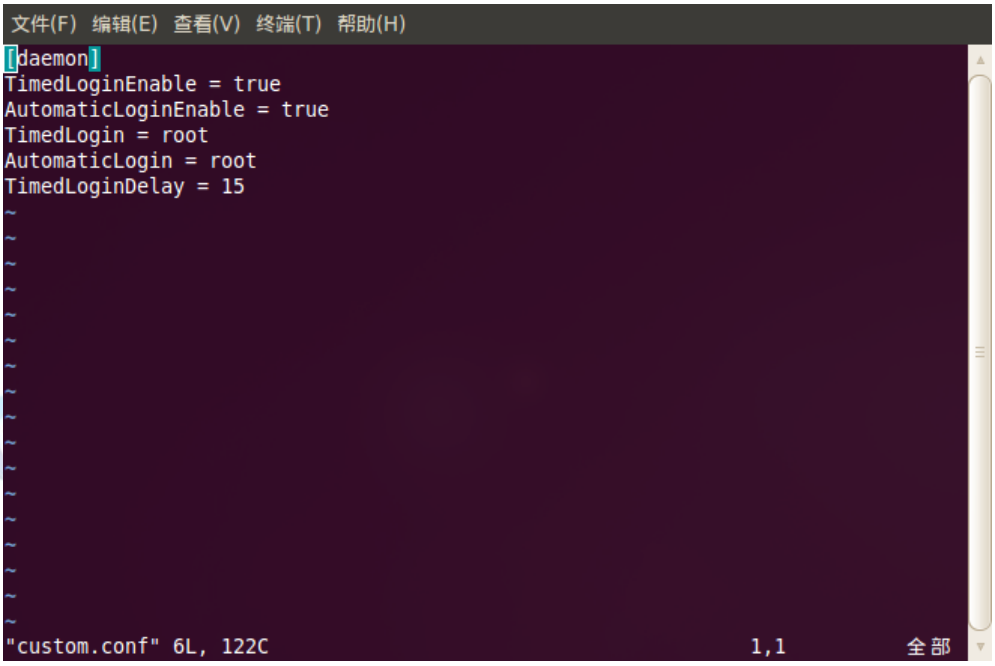


图2-66 编辑内容

2.4.1.3 设置 UBUNTU 网络参数

因为每个 PC 的网络环境都不一定一样，所以需要您根据自己的实际情况来设置 Ubuntu 的网络，如果设置不成功，可以去 Ubuntu 的官方论坛上咨询。

接下来，为您提供一种网络的设置方法，仅供参考。

步骤 1：启动 Ubuntu，在桌面上端选择“系统—>首选项—>网络连接”，弹出网络连接设置窗口。如图 2-67 所示：

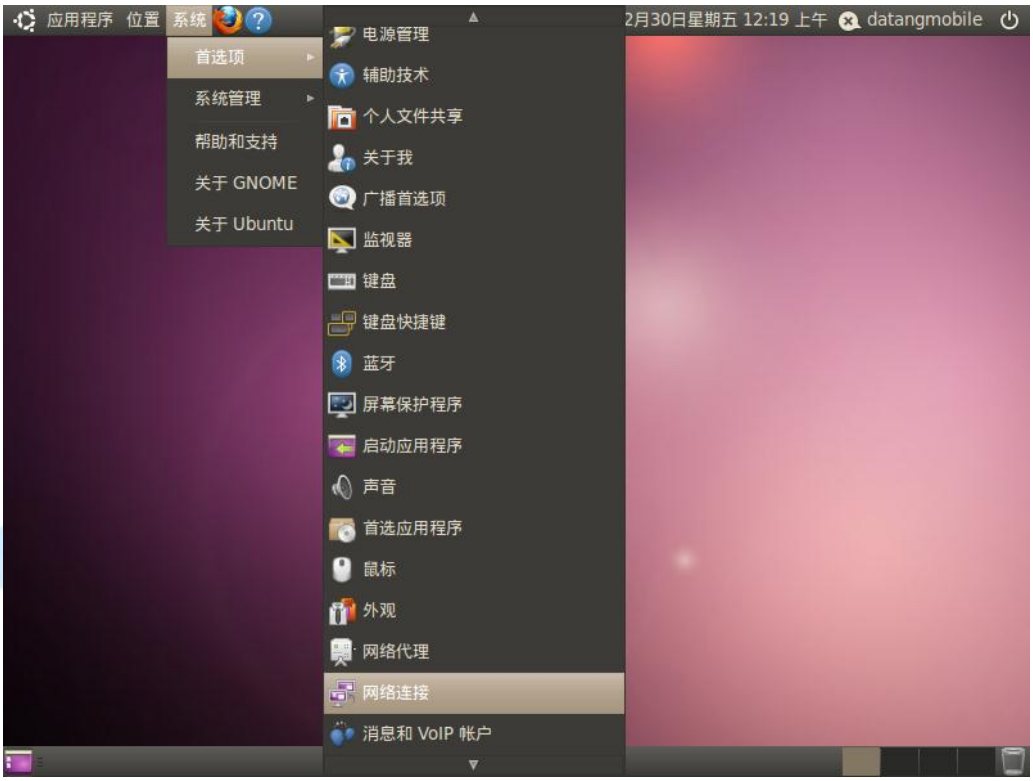


图2-67　网络连接

步骤 2：在其“网络连接”面板上选中 Auto eth0 然后点击右边的“编辑”。接下来在“正在编辑 Auto eth0”窗口上的“Ipv4 设置”面板上设置网络参数。如图 2-68 所示：

图2-68　设置网络连接

图中设置是一个例子，地址、子网掩码、网关、DNS 等信息需要根据个人的实际环境来设置。设置好以后点击“应用”。如图 2-69 所示：

图2-69　设置网络连接参数

步骤 3：修改配置后，需要输入 root 密码来授权，如图 2-70 所示：

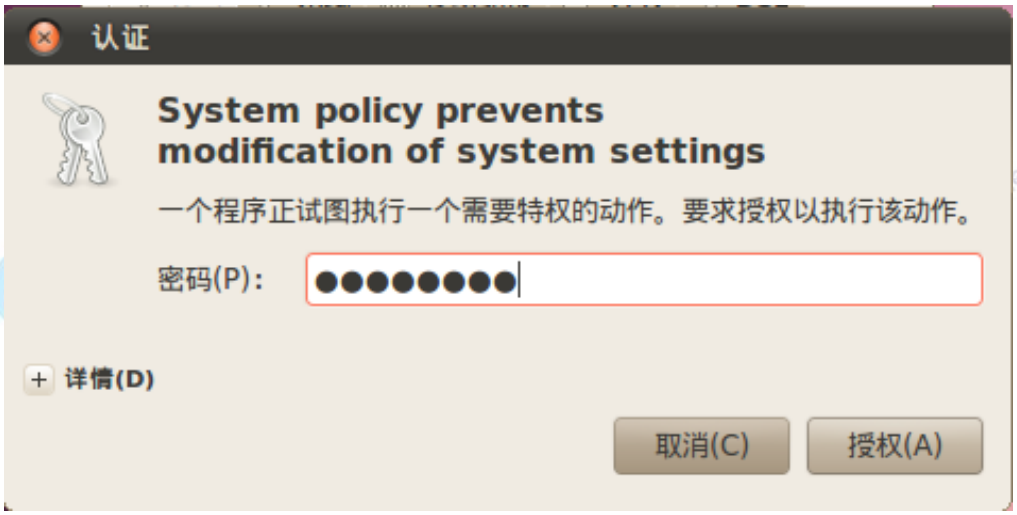


图2-70　输入密码

至此，就完成了 Ubuntu 的网络连接设置。

2.4.2　安装交叉编译器

将 arm-linux-gcc-4.3.2.tgz 文件拷贝到 Ubuntu 的/usr/local 目录下，在 Ubuntu 中新建一个终端，输入下面的命令安装交叉编译器：

```
#cd      /usr/local    （进入/usr/local 目录）
#mkdir      /usr/local/arm （创建目录，若目录已存在会提示错误，跳过即可）
#tar  zxvf  arm-linux-gcc-4.3.2.tgz  –C  arm/  （编译器解压到/usr/local/arm）
```

完成后将在/usr/local/arm/目录下生成“4.3.2”目录

这样，内核或其他应用程序均/usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc 来指定使用该交叉编译器。例如，我们查看这个编译器的版本，可以运行：

```
#/usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc  -v
```

编译器安装正确，会显示编译器的版本等信息，如图 2-71 所示：

```
root@datang-desktop:/usr/local/arm# /usr/local/arm/4.3.2/bin/arm-none-linux-gnue
abi-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/configure --build
=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi --en
able-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --wit
h-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-shared --enable-symvers
=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2008q3-72' --wit
h-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/
opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --
with-build-sysroot=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnue
abi/libc --with-gmp=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-ar
m-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/lite-resp
in/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --
disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scra
tch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin --with-build-tim
e-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
root@datang-desktop:/usr/local/arm#
```

图2-71　显示编译器版本信息

2.4.3　建立 QtCreator 开发环境

您可以使用 QtCreator 集成开发环境开发 Qt 应用，QtCreator 很类似于 Windows 上的 VC++ 集成开发环境，可以进行控件的可视化，所见即所得，使用该软件可以开发 PC 上的 Qt 应用和嵌入式环境下的 Qt 应用。

步骤 1： 将 qt-sdk-linux-x86-opensource-2010.04.bin 文件拷贝到 Ubuntu 的 root 目录下。以下步骤执行的命令效果如图 2-72 所示：

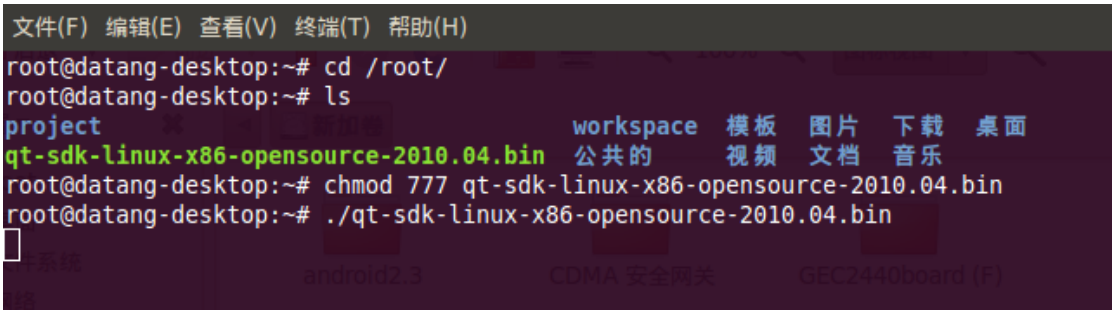


图2-72　拷贝文件效果图

步骤 2：　打开终端，进入 root 目录

#cd　/root

步骤 3：　查看文件（可跳过）

#ls

步骤 4：　修改文件属性，使文件为可执行文件

#chmod　777　qt-sdk-linux-x86-opensource-2010.04.bin

步骤 5：　开始安装，会出现如图 2-73 所示界面。

#./qt-sdk-linux-x86-opensource-2010.04.bin

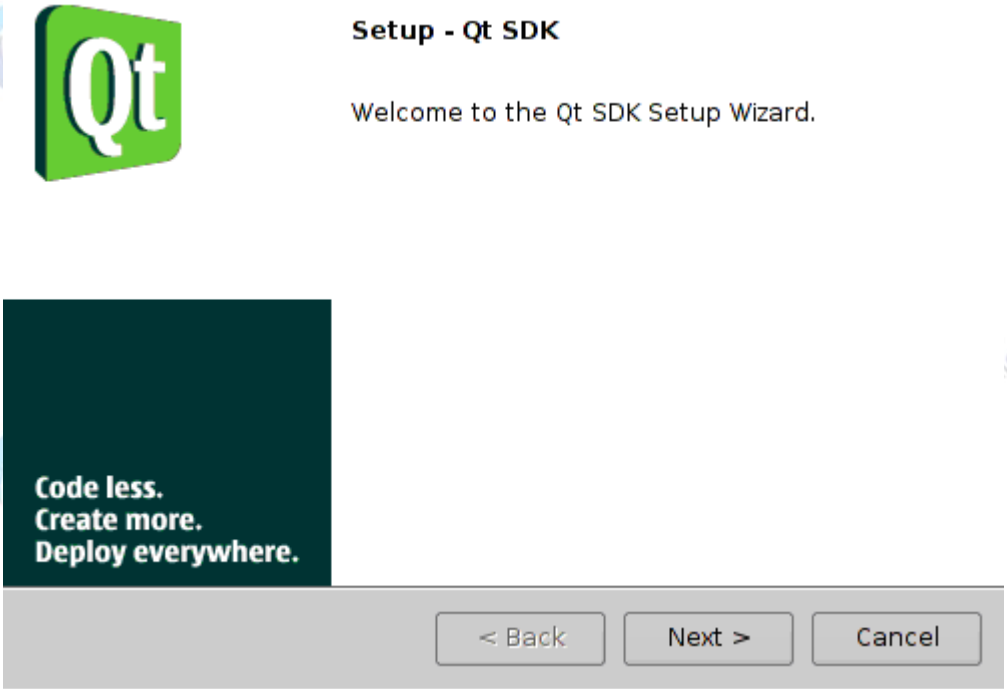


图2-73　安装 Qt 软件

步骤 6：　在界面上选择“Next”直到如 2-74 图界面，可以修改文件安装路径，这里选择默认。

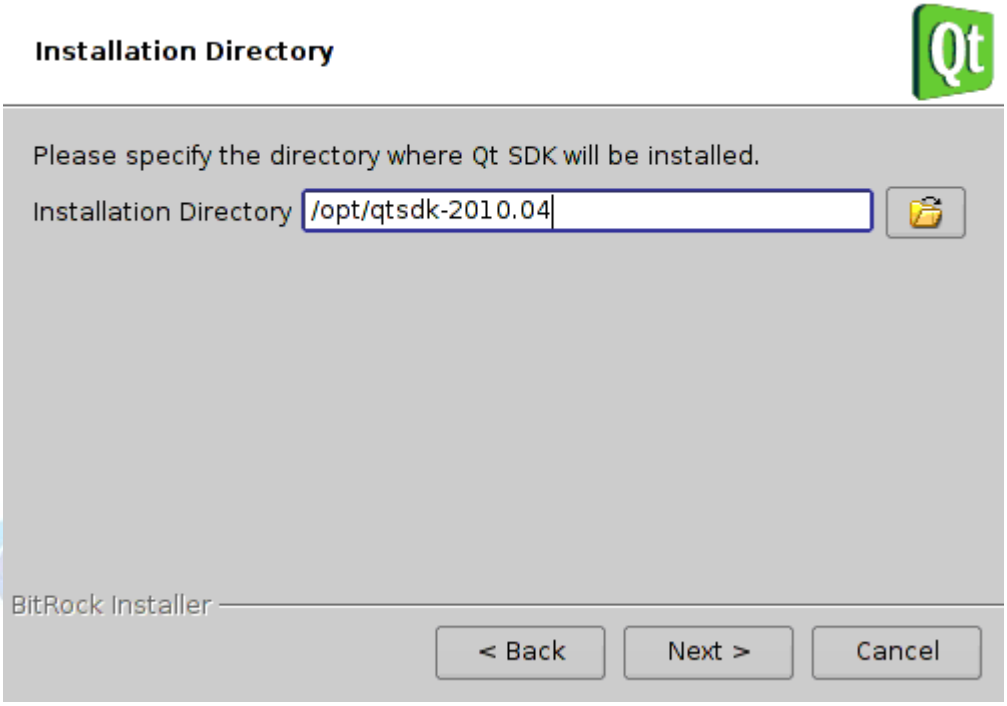


图2-74　文件安装路径

步骤 7：　一直选择默认“Next”，直到安装完成。

这样，QT 就安装成功了。执行#qmake　–v 可以查看到 qmake 的版本信息，如图 2-75 所示。

```
root@datang-desktop:~# qmake -v
QMake version 2.01a
Using Qt version 4.6.3 in /opt/qtsdk-2010.04/qt/lib
root@datang-desktop:~#
```

图2-75　查看版本信息

2.4.4　安装编译工具 g++

步骤 1：　将 g++-4.4_4.4.3-4ubuntu5_i386.deb、libstdc++6-4.4-dev_4.4.3-4ubuntu5_i386.deb、libncurses5-dev_5.7+20090803-2ubuntu3_i386.deb 三个文件拷贝到/usr/local 目录下。

步骤 2：同时安装 3 个文件。

#cd　/usr/local

#dpkg　-i　g++-4.4_4.4.3-4ubuntu5_i386.deb libncurses5-dev_5.7+20090803-2ubuntu3_i386.deb libstdc++6-4.4-dev_4.4.3-4ubuntu5_i386.deb

步骤 3：　安装完成后，创建软连接

#cd /usr/bin

#ln -s g++-4.4 g++

步骤 4： 查看 g++版本信息，如图 2-76 所示，则 g++安装完成。

#g++ -v

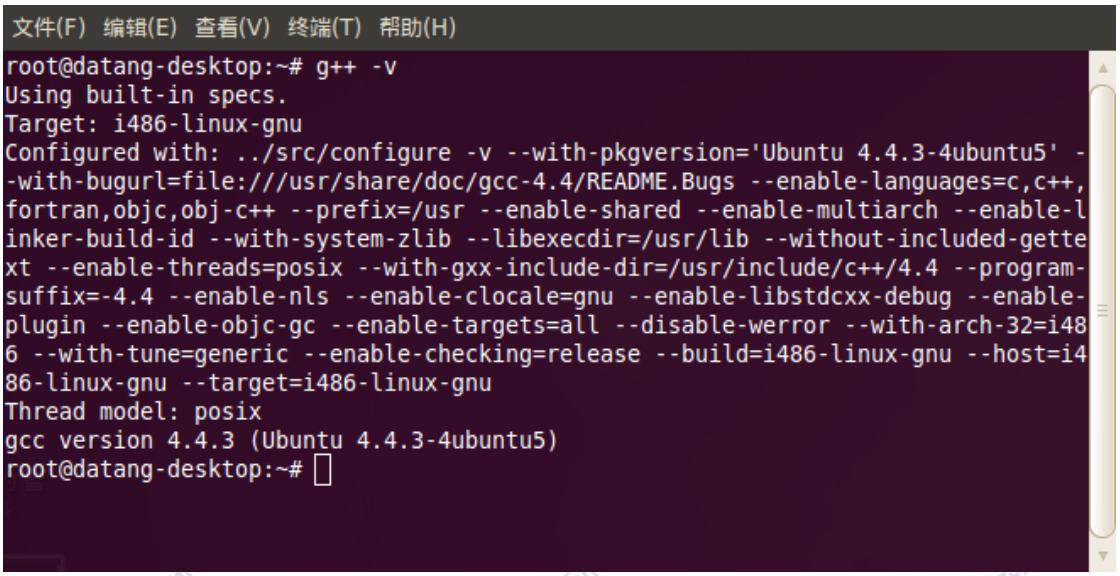


图2-76 查看 g++版本信息

至此，集成开发环境 QT 安装完成，由于以上方法是手动安装的（即没有网络连接的状态下），所以还需根据个人需要安装一些库文件。以下提供在有网络的状态下安装部分库文件。

#sudo apt-get install libglib2.0-dev libSM-dev libxrender-dev libfontconfig1-dev libxext-dev libglui-dev

2.4.5 QT 实例 HelloWorld

在此以 HelloWorld 应用程序来验证基础开发环境 QT 搭建成功。创建 HelloWorld 应用程序过程比较简单，用户根据以下图示操作即可。

步骤 1： 运行 QT，创建工程 File—>New File or Project。如图 2-77 所示：

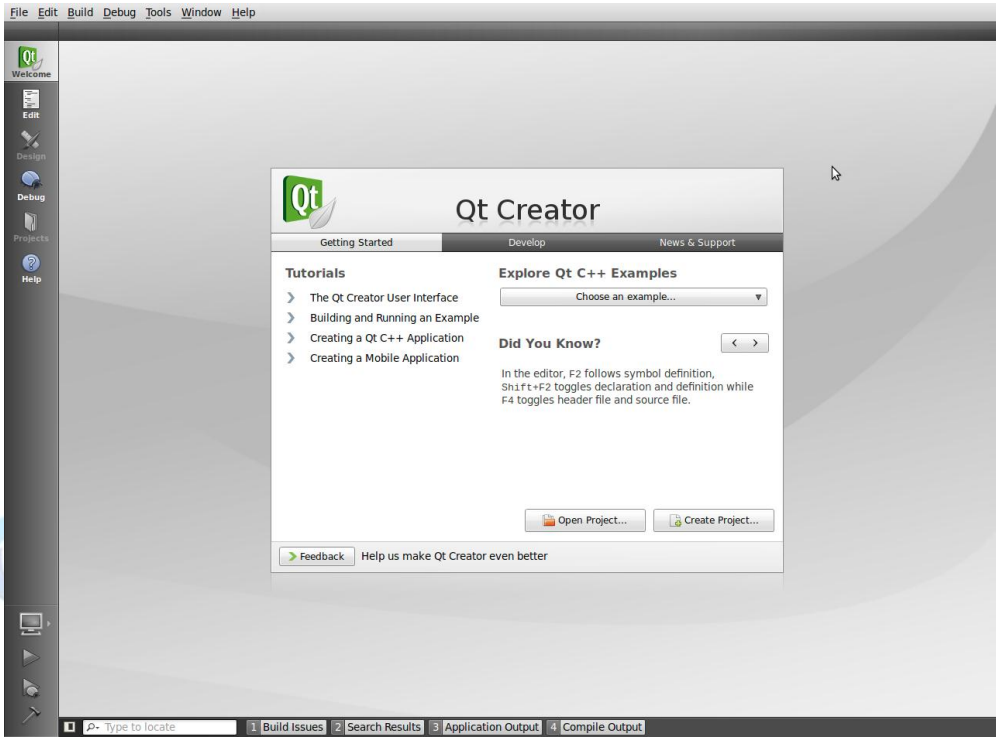


图2-77　　Qt 启动界面

步骤 2：选择“Qt Gui Application”，如图 2-78 所示：

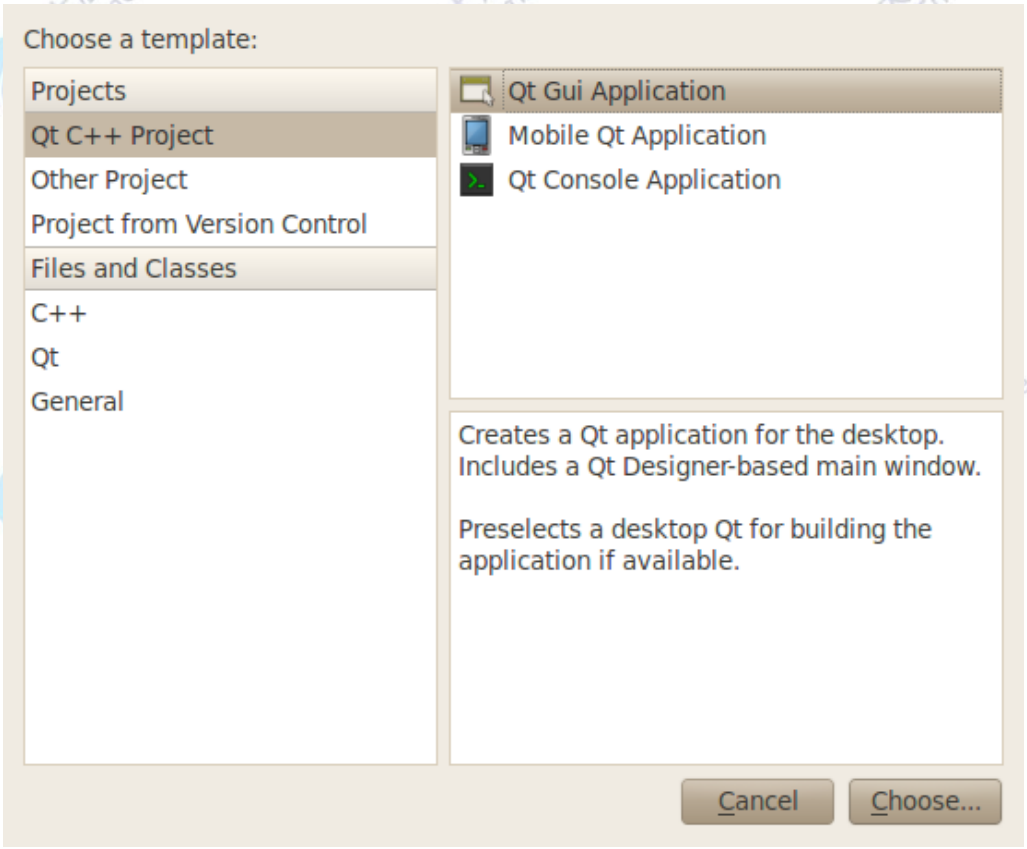


图2-78　　选择 Qt Gui Application

步骤 3： 填写工程名称 HelloWorld 并选择工程存储路径，如图 2-79 所示：

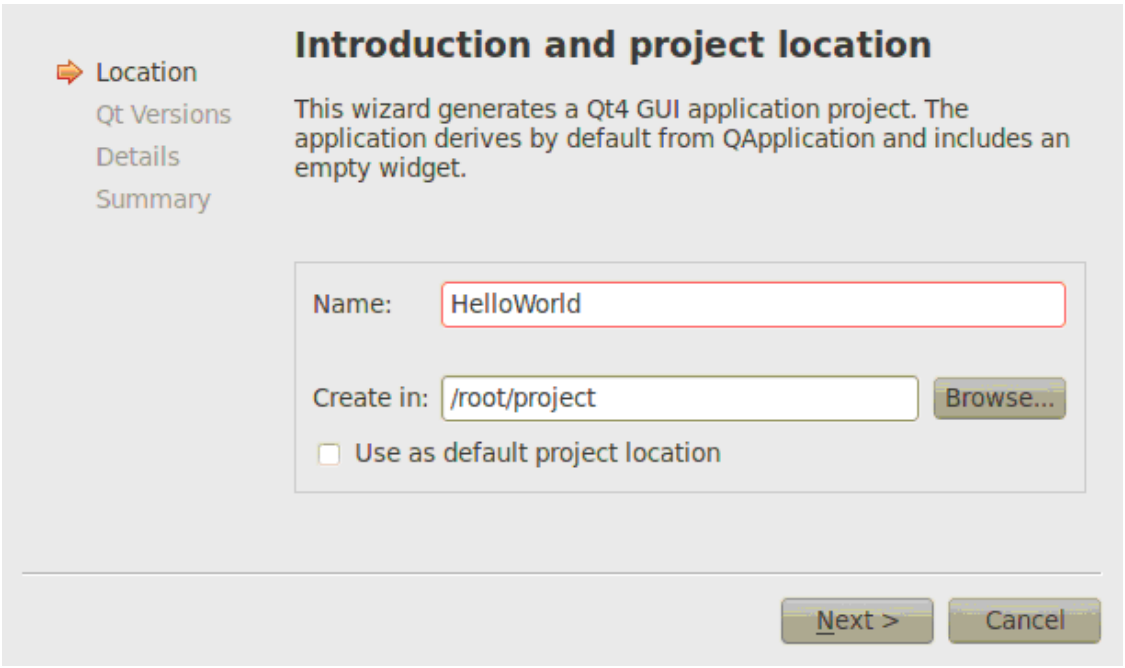


图2-79 Location

步骤 4：如图 2-80 所示：

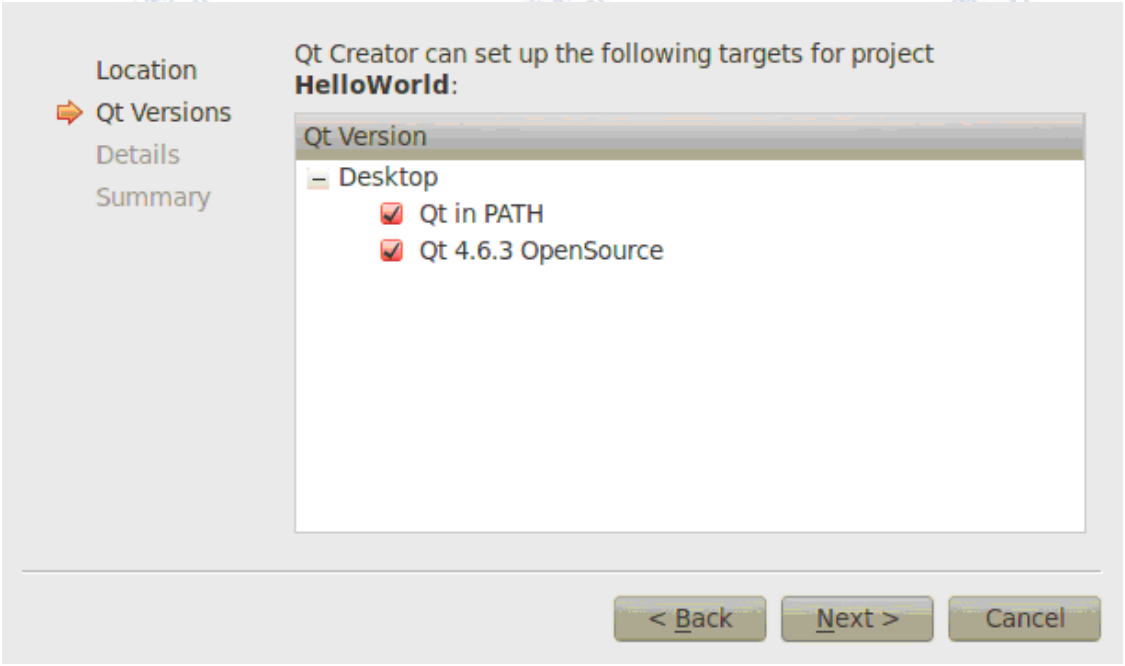


图2-80 QT Versions

步骤 5：在“Base class”中选择“QDialog”，其余可默认，并点击“Next”。如图 2-81 所示：

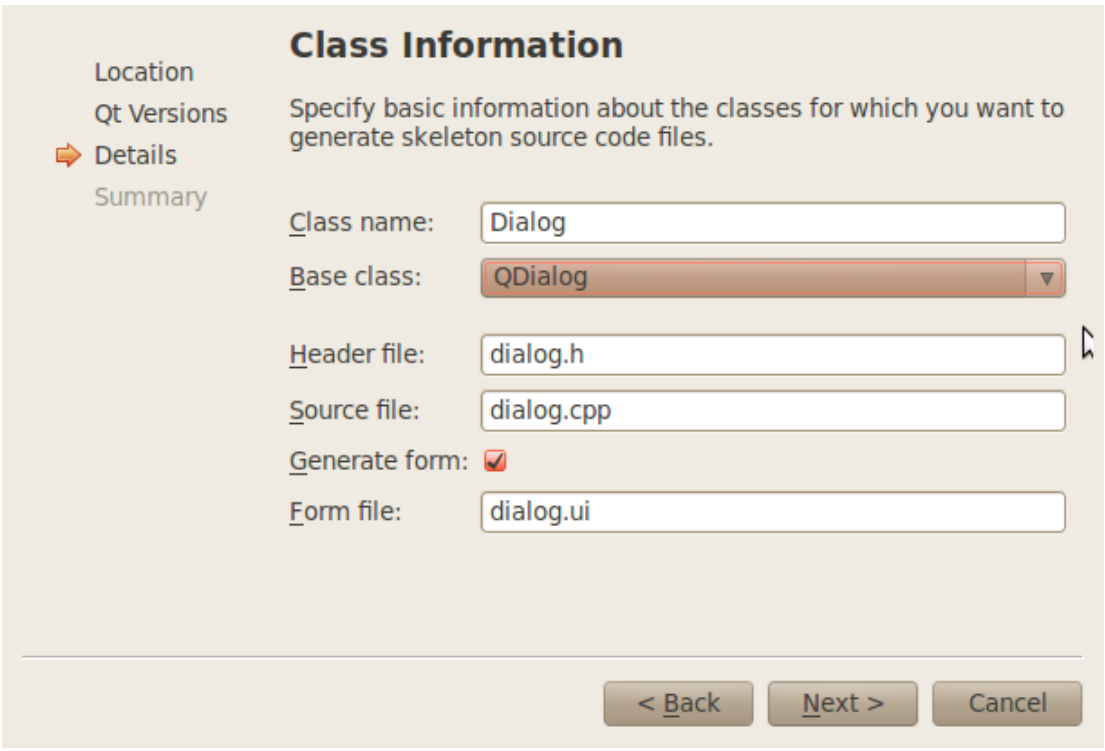


图2-81　Details

步骤 6：选择默认，点击“Finish”，即工程创建完成。如图 2-82 所示：

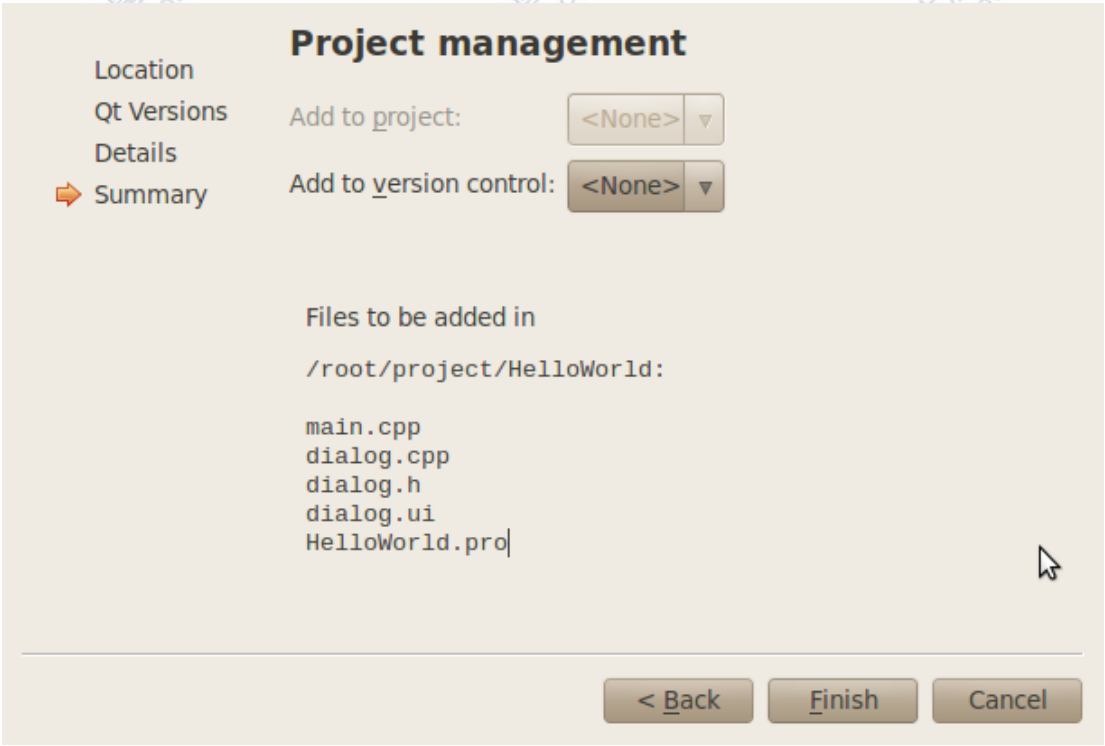


图2-82　Summary

步骤 7：　工程创建完成后，会出现如图 2-83 所示界面。

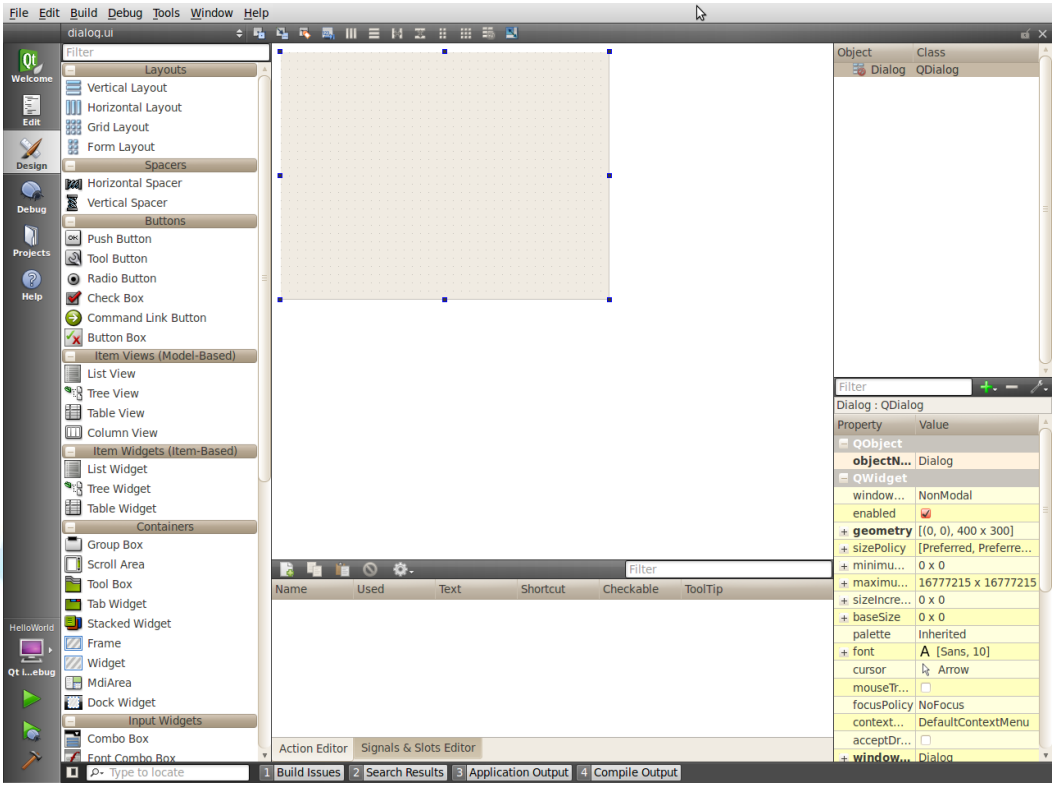


图2-83　　新工程界面

步骤 8：在窗体界面上创建一个 Lable，修改文本内容为“HelloWorld!!!”并保持。如图 2-84 所示：

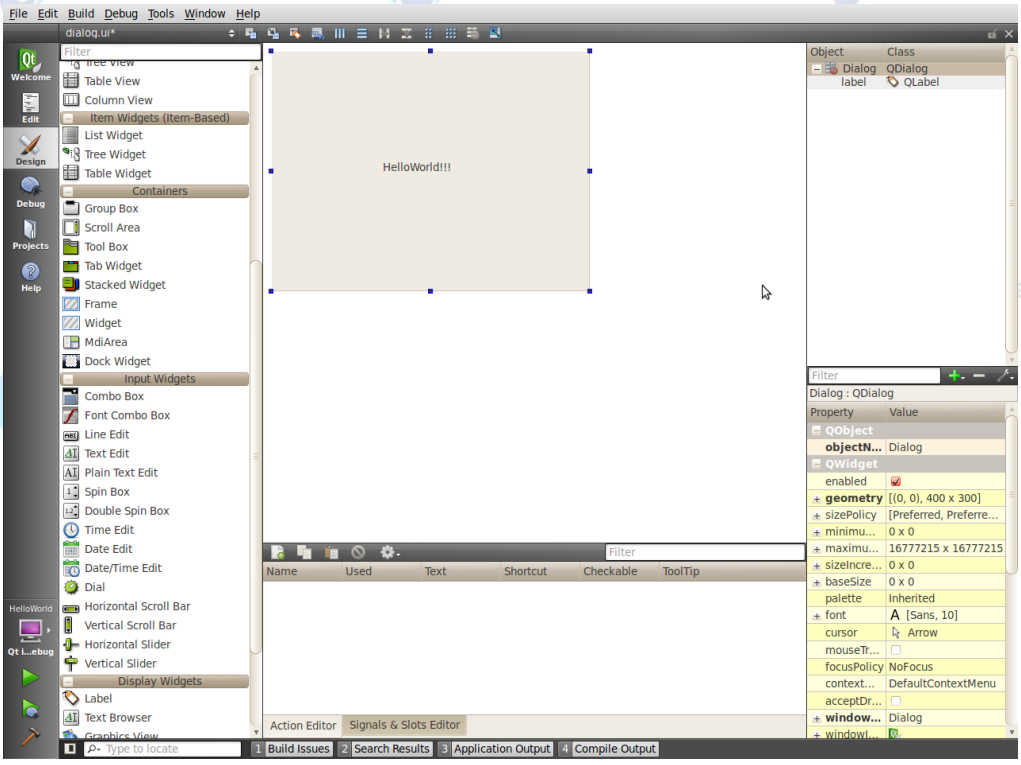


图2-84　　添加 label 标签

步骤 9： 编译运行该工程，编译成功后则出现如图 2-85 所示窗口。

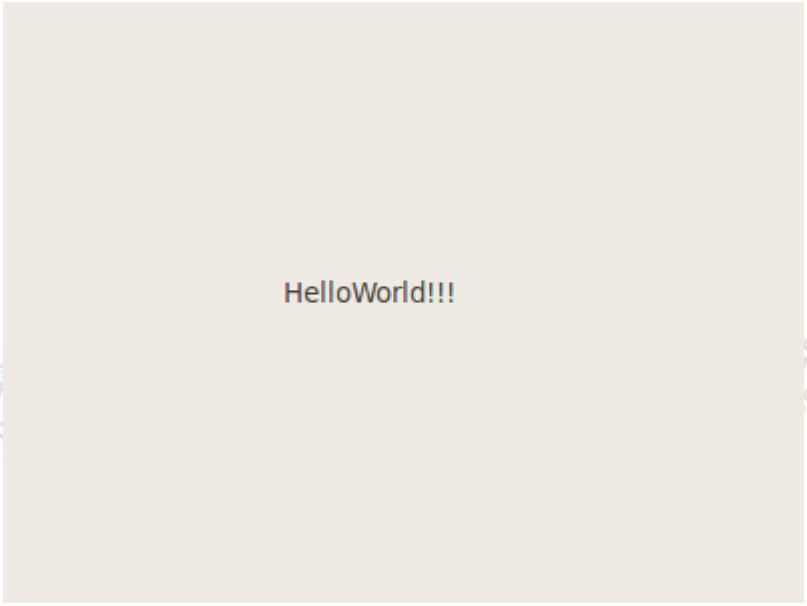


图2-85 编译运行

2.5 在 ARM 网关上搭建 LINUX 开发环境

我们采用一键烧写Linux的方法来搭建ARM网关的开发环境。所谓一键烧写就是借助SD卡、大唐提供的程序和系统映像，通过一系列操作，非常迅速的烧写Linux 到开发板的NandFlash中。当您由于各种原因造成的开发板无法启动时，或者是需要更新uboot、内核（zImage）、文件系统（Cramfs 文件系统）中的一个或者多个时，就可以使用一键烧写的方法解决。一键烧写的优点在于烧写速度快，一次烧写必须烧写所有文件，并可以通过串口查看烧写的状态。

下面开始介绍一键烧写的具体方法。烧写之前，请先准备好一个SD卡和一个SD读卡器。本节介绍的环境以及提供的文件经过大唐公司的测试，可以按照本书说明的方法来搭建开发环境。本章所需文件路径：E-Box300\02-开发环境搭建\01-ARM嵌入式网关\linux系统文件

2.5.1 制作用于一键烧写 LINUX 的 SD 卡

步骤1： 将SD卡格式化为FAT32格式，将SD卡接入SD读卡器中，把SD读卡器插在PC机的USB 口中。如图2-86所示：

图2-86　SD 卡

等到PC机能够正常识别出SD卡，把SD卡格式化为FAT32格式。如图2-87所示。

图2-87　格式化 SD 卡

步骤2.：通过DTmobile SD_Writer.exe将mmc.bin烧写到SD卡中。打开DTmobile SD_Writer.exe，下图2-88是xp系统中DTmobile SD_Writer.exe界面截图。

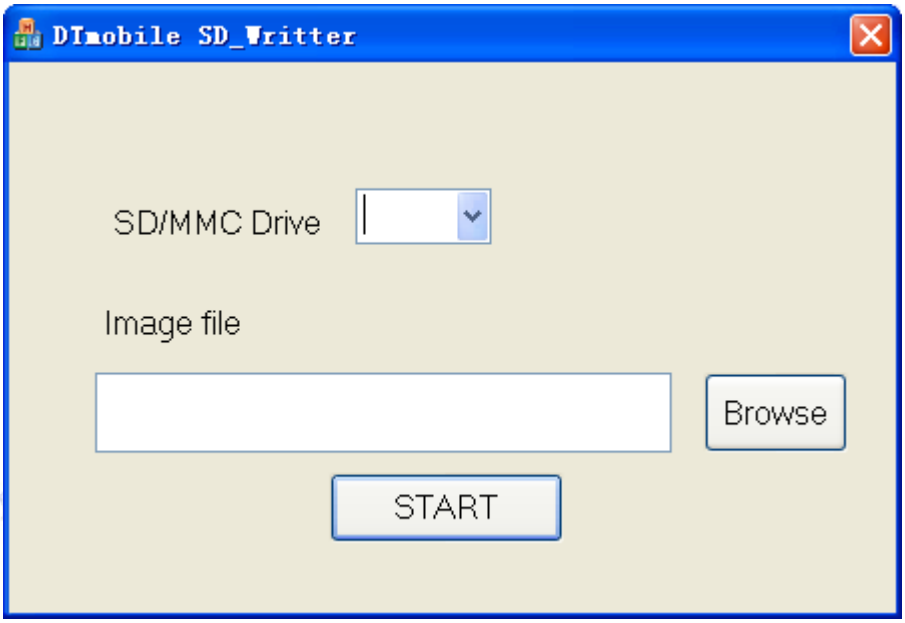


图2-88　DTmobile SD_Writer 软件界面

步骤3：从“SD/MMC Drive”选项的下拉列表中，选择SD卡在本机中所在的盘符。

步骤4：点击 ”Browse”，选择mmc.bin。如图2-89所示。

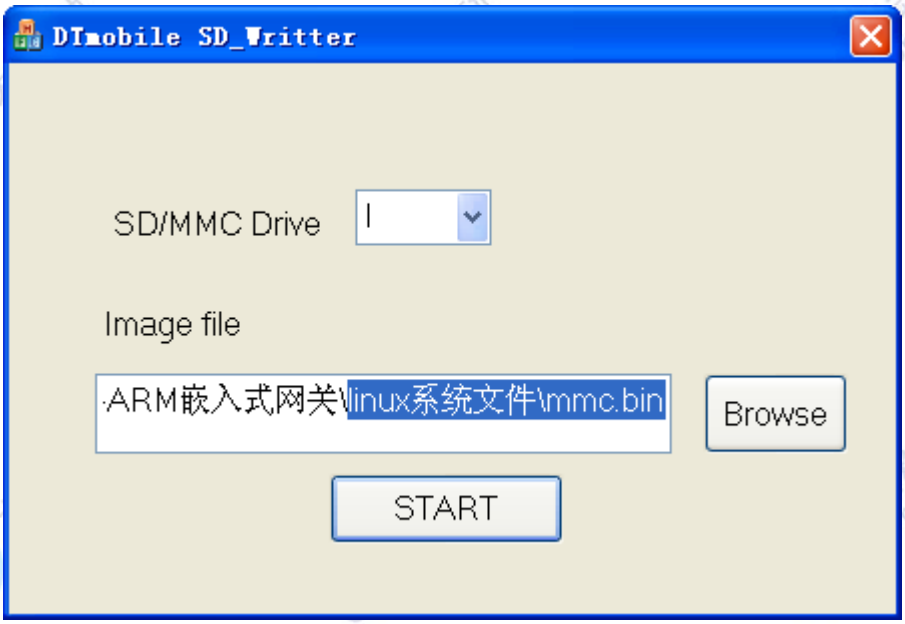


图2-89　DTmobile SD_Writer

步骤5：点击”START”，弹出对话框提示操作成功，然后退出DT mobile SD_Writer.exe。

步骤6：将u-boot.bin， zImage和cramfs拷贝到SD卡中。

经过上述步骤操作后，正确的文件和文件名如图2-90所示：

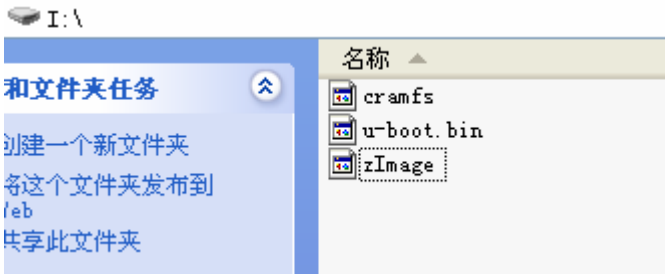


图2-90　SD 卡内的文件

2.5.2　烧写 LINUX 到开发板的 FALSH 中

步骤1：将制作好的SD卡插入开发板SD的插槽。如图2-91所示：

图2-91　插入 SD 卡

步骤2：接好5V直流电源（大唐提供此电源，请使用大唐提供的电源）。

开发板电源（开发板左下角）连接如图2-92所示：

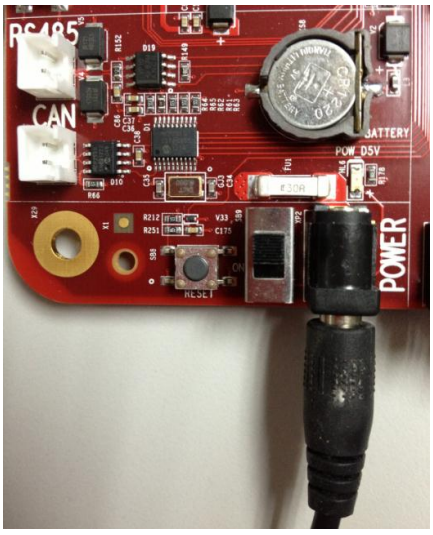


图2-92　开发板电源

步骤3：拨码开关设置为SD卡启动。开发板共有两个拨码开关，包括系统控制模块的SB1

和wifi模块的SB11，需要设置的是前者。

拨码开关在底板SD卡启动的拨码开关设置如下表2-1：

引脚号	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
引脚定义	SELNAND	OM4	OM3	OM2	OM1	GPN15	GPN14	GPN13
SD卡启动	1	1	1	1	1	0	0	0

表2-1

注：上表中。1表示拨码需要调整到On；0表示拨码需要调整到Off。

在拨动开关时，务必把开关拨到底。如果没有拨到底，发生接触不良，会导致烧写失败。拨码开关设置SD卡启动如图2-93所示：

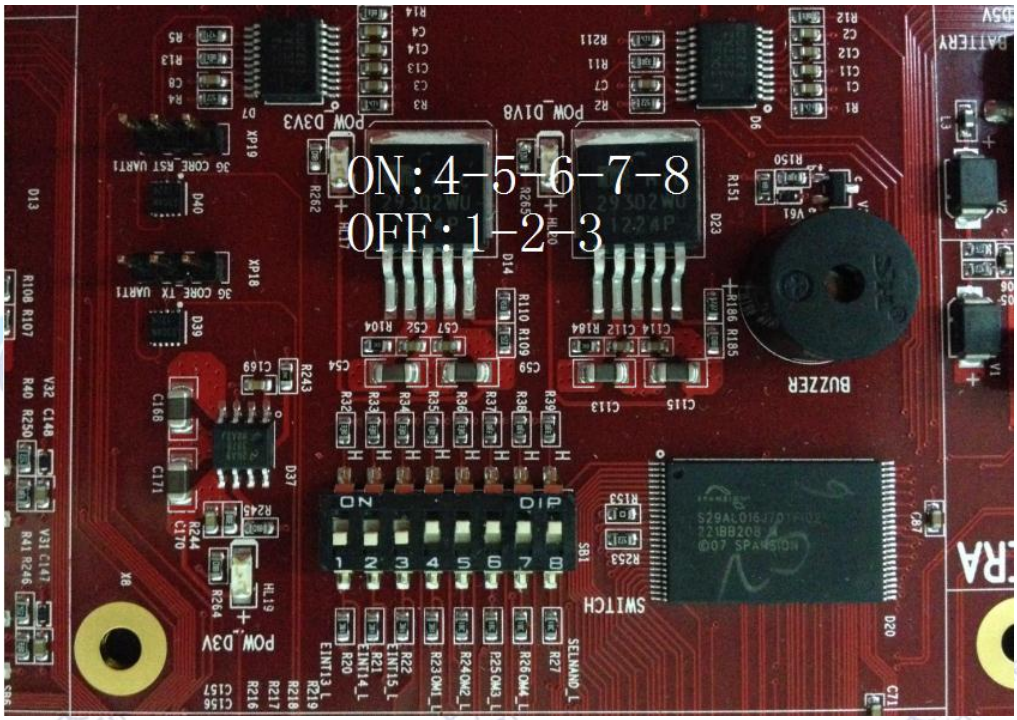


图2-93　拨码开关设置

步骤4：将大唐提供的直连串口延长线连接开发板的COM0和PC机的串口（大唐提供的直连串口线仅限于开发板和PC的连接，连接其他设备请先确定串口的线序），如图2-94所示：

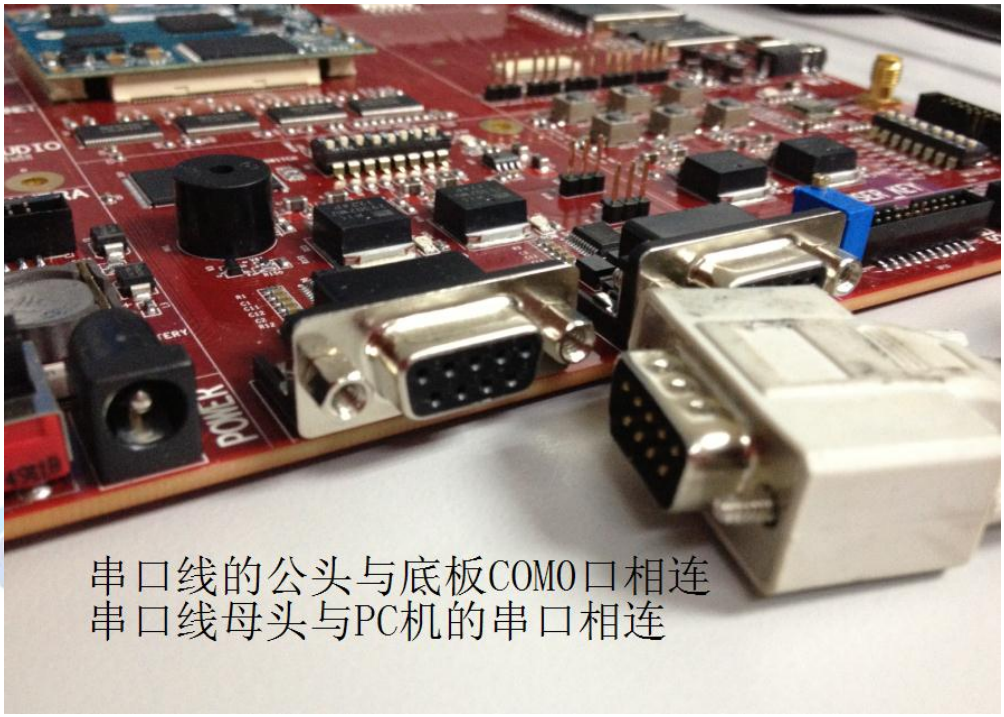


图2-94　连接串口线

步骤5：打开PC机自带的超级终端软件（开始菜单→附件→通讯→超级终端），如图2-95所示。

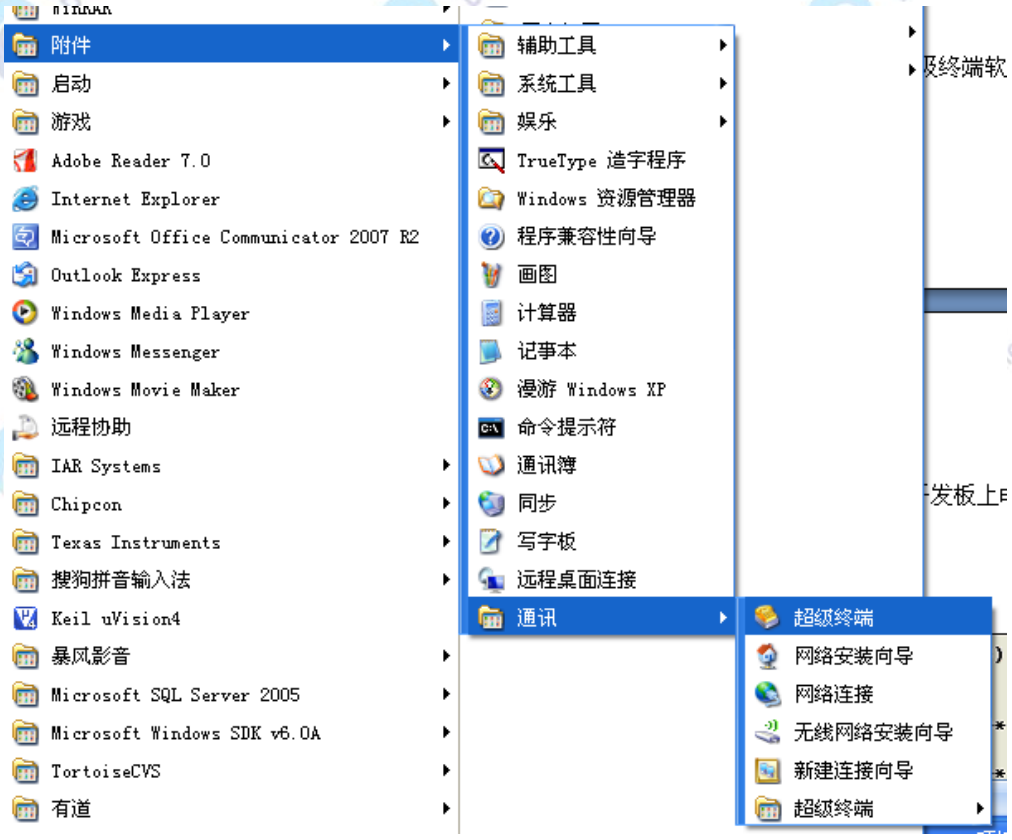


图2-95　超级终端

设置超级终端的连接名称以及图标如图2-96所示。

图2-96　连接描述

连接时使用COM1如图2-97所示。

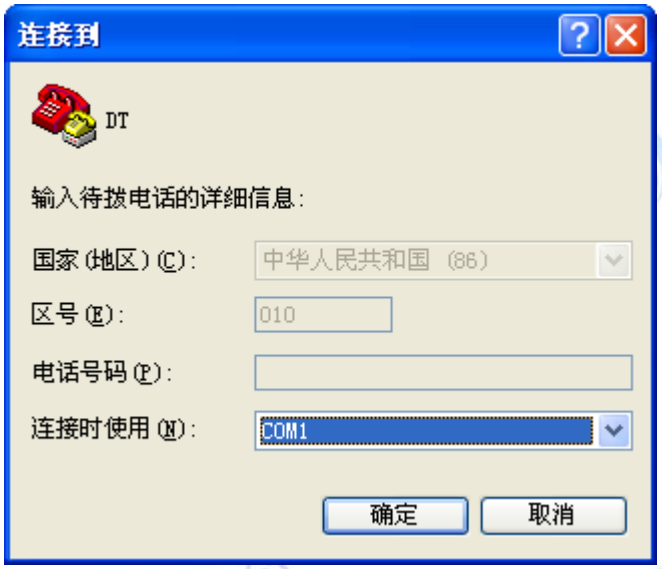


图2-97　连接设置

设置COM1的属性如图2-98所示，每秒位数设置为115200，数据流控制为无，点击确定，进入超级终端界面。

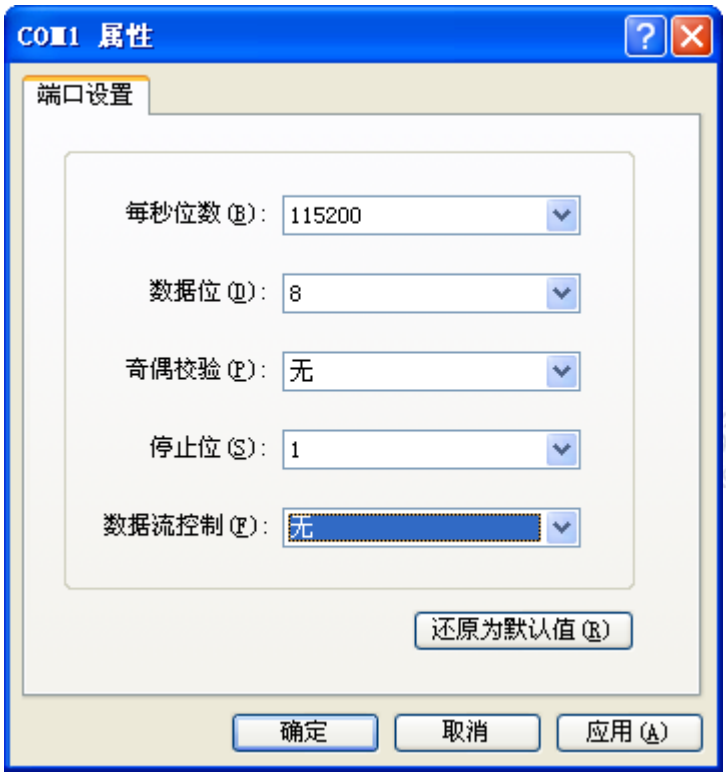


图2-98　COM1 属性

步骤6：拨动电源开关，给开发板上电。会出现如下的串口信息，因为信息较多，贴出开始时和结束后的串口信息：

开始时（如图2-99）：

```
U-Boot 1.1.6 (Jan 1811 - 17:03:38) for SMDK60

**********************************
**    ot 1.1.6                  **
**    Updated DT_EBox.6410 Board        **
*    Version 1.0 (10-01-1)          **
**    OEMlinx Embedded
**    Web: http://wwwtech.com.cn    **
**********************************

CPU:     S3C6410 32MHz
         Fclk = 5MHz, Hclk = 133MHz, Pclk 66MHz, Serial = CLKUART(SYNC Mode)
Board:   SMK6410
DRAM:    128 MB
Fh:   0 kB
NAND:    tmp =
select s3c_nand_oob_mlc_4
```

图2-99　开始时信息截图

结束时（如图2-100）：

```
OK
reading u-boot.bin

196608 bytes read

NAND write: device 0 offset 0x0, size 0x100000
 1032192 bytes written: OK
reading zImage

3577044 bytes read

NAND write: device 0 offset 0x100000, size 0x500000

Writing data at 0x5ff000 -- 100% complete.
 5242880 bytes written: OK
reading cramfs

106545152 bytes read

NAND write: device 0 offset 0x600000, size 0x8000000

Writing data at 0x85ff000 -- 100% complete.
 134217728 bytes written: OK
SMDK6410 #
```

图2-100　结束时信息截图

步骤7：拨动电源开关，开发板断电，将拨码开关SB1设置为nand flash启动。设置如下表2-2所示：

引脚号	Pin 8	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
引脚定义	SELNAND	OM4	OM3	OM2	OM1	GPN15	GPN14	GPN13
NandFlash卡启动	1	0	0	1	1	0	0	0

注：上图中。1表示拨码需要调整到On；0表示拨码需要调整到Off。

在拨动开关时，务必把开关拨到底。如果没有拨到底，发生接触不良，会导致烧写失败。拨码开关设置NandFlash启动如图2-101所示：

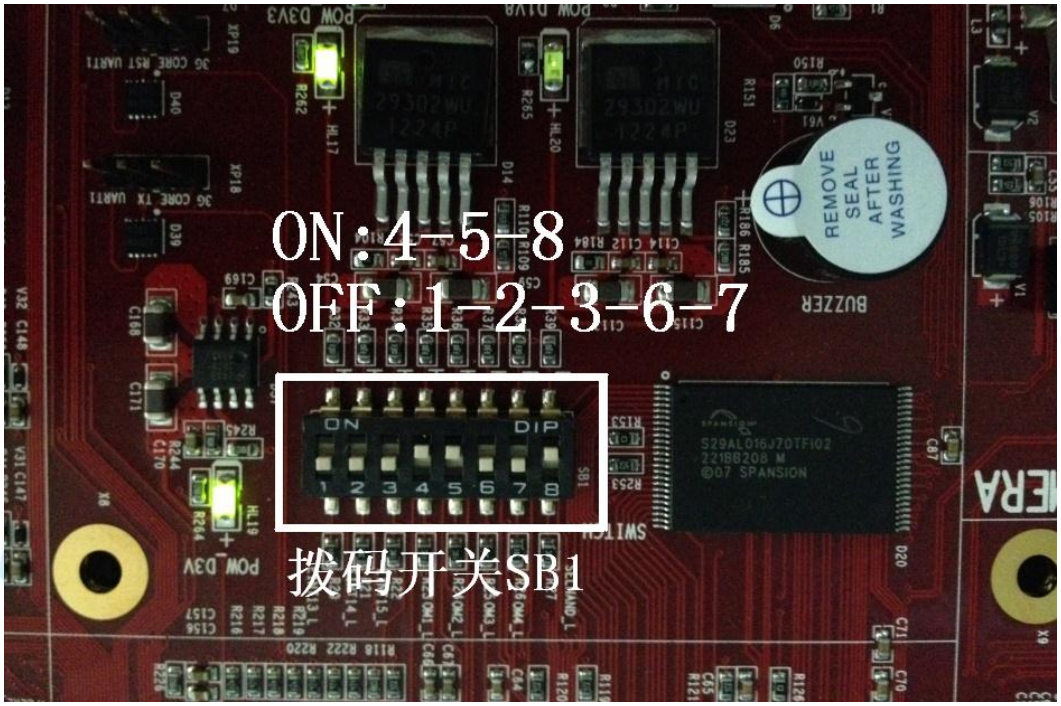


图2-101　拨码开关设置

步骤2：重新开启电源，Linux系统可以正常启动了。重启时，超级终端打印的信息结束时如图2-102所示。

```
can: broadcast manager protocol (rev 20090105 t)
lib80211: common routines for IEEE802.11 drivers
Registering the dns_resolver key type
s3c-rtc s3c64xx-rtc: setting system clock to 2000-01-01 00:00:02 UTC (946684802)

VFS: Mounted root (cramfs filesystem) readonly on device 31:2.
devtmpfs: mounted
Freeing init memory: 172K
in sysfs mount
in sysfs mount 1
FAT-fs (mmcblk0p1): utf8 is not a recommended IO charset for FAT filesystems, fi
lesystem will be case sensitive!

/etc/rc.d/init.d/netd: line 16: /usr/sbin/inetd: not found
mkdir: cannot create directory '/mnt/disk': File exists
yaffs: dev is 32505859 name is "mtdblock3" rw
yaffs: passed flags ""
Try to bring eth0 interface up......eth0: link down
Done

Starting Qtopia2, please waiting...

Please press Enter to activate this console. touch...
-
```

图2-102　linux 正常启动后结束时信息截图

2.5.3　烧写 YAFFS 文件系统

这一步最关键，主要的目的是：将yaffs.tar.gz 内的文件解压缩到开发板的/mnt/disk 路径下。

步骤1：按回车键，在超级终端软件[root@DTmobile]#提示符下输入如下命令：

```
#tar  zxf  /sdcard/yaffs.tar.gz  -C  /mnt/disk
```

这个步骤需要一定的时间来解压文件，请等待解压完成。

看到显示[root@DTmobile]#表示已压缩完成，可以继续操作。

步骤2：重启开发板，等到出现延时1 秒启动系统时，在DNW软件中按PC键盘的空格键使开发板停留在uboot状态。因为停留时间只有1秒，所以需要很快的按下空格键。如图2-103：

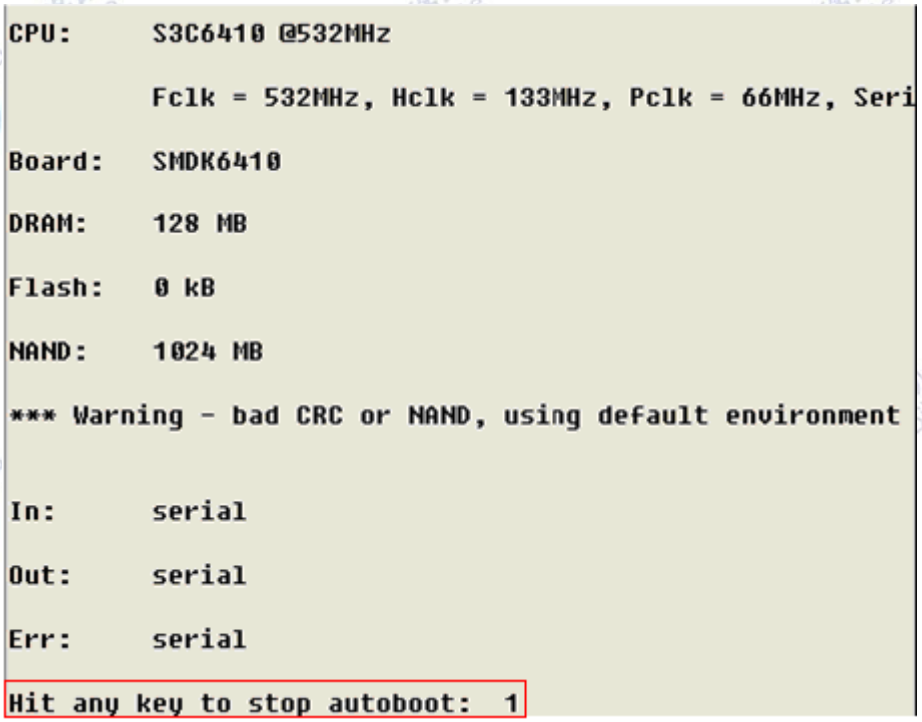


图2-103　开发板打印信息

步骤3：等到停在uboot状态时，修改boot启动参数。在SMDK6410 # _提示符下运行命令：

```
#setenv  bootargs  “root=/dev/mtdblock3  rootfstype=yaffs2  console=ttySAC0,
115200  lcdsize=70”
#saveenv （保存上面uboot 的启动参数）
#reset （重启开发板）
```

步骤4：等待完全启动Linux3.0.1的yaffs根文件系统后，就完成了整个烧写yaffs根文件系统的操作。

2.5.4 切换鼠标模式操作

按照以上操作烧写的系统，液晶屏默认为触屏模式，如果要用鼠标模式进行操作，可按照以下两种方法进行。

方法一：在 ARM 网关终端编辑脚本内容。

1) 打开 ARM 网关进入终端，键入 vi /etc/init.d/rcS 命令。

2) 在脚本编辑模式下，文件追加 sleep 2/sbin/mouseinput。

3) 保存脚本并退出 vi。

4) 插上 U 口鼠标。

5) 关闭 ARM 板并重新启动 ARM 网关。

即可完成鼠标模式的切换。

方法二：通过超级终端软件修改脚本内容。

1）将大唐提供的直连串口线连接开发板的COM0和PC机的串口。

2）打开PC机的超级终端软件（位置：开始→附件→通讯→超级终端），设置超级终端连接名称，并将波特率设置为115200，其他选择默认即可。

3）进入超级终端软件，按回车键后出现 [root@DTmobile]# 提示符，然后输入以下命令：

```
#mouseinput
```

按回车键，出现如图2-104所示提示符后。重启ARM网关即可。

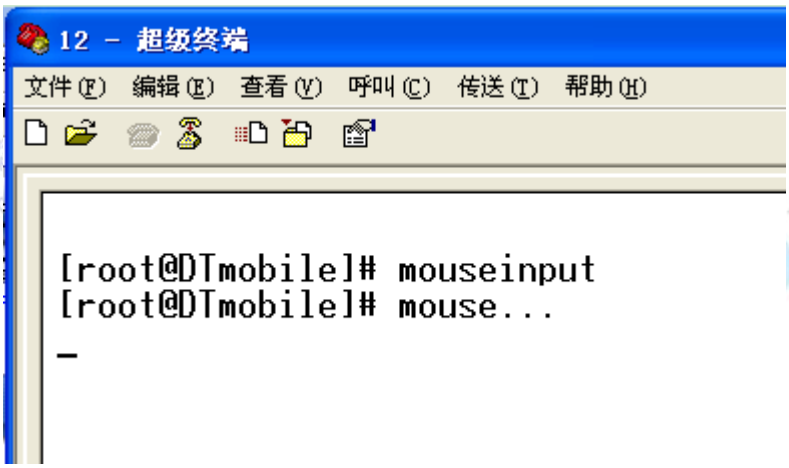


图2-104 超级终端

对于键盘来说，无论是触屏模式还是鼠标模式，插入 USB 键盘都可以使用。

2.5.5 出现坏块的应对措施

对于坏块的处理：使用NandFlash，免不了出现个别的坏块。没有关系，一般的坏块就交给大唐开发板自己处理吧。大唐开发板对坏块有一套完善的处理机制。

如果出现因为坏块无法启动Linux 操作系统，那就需要一个方法来处理这些逻辑上的坏块（实际上坏块不一定是真的坏了）。逻辑坏块引起的系统无法启动，可以使用下面这种方法：

步骤1：先用串口线连接好开发板COM0与PC机的串口，打开超级终端软件。

步骤2：然后给开发板上电，等到出现延时1秒启动系统时，在超级终端软件中按PC键盘的空格键使开发板停留在uboot状态。因为停留时间只有1秒，所以需要很快的按下空格键。如图2-105：

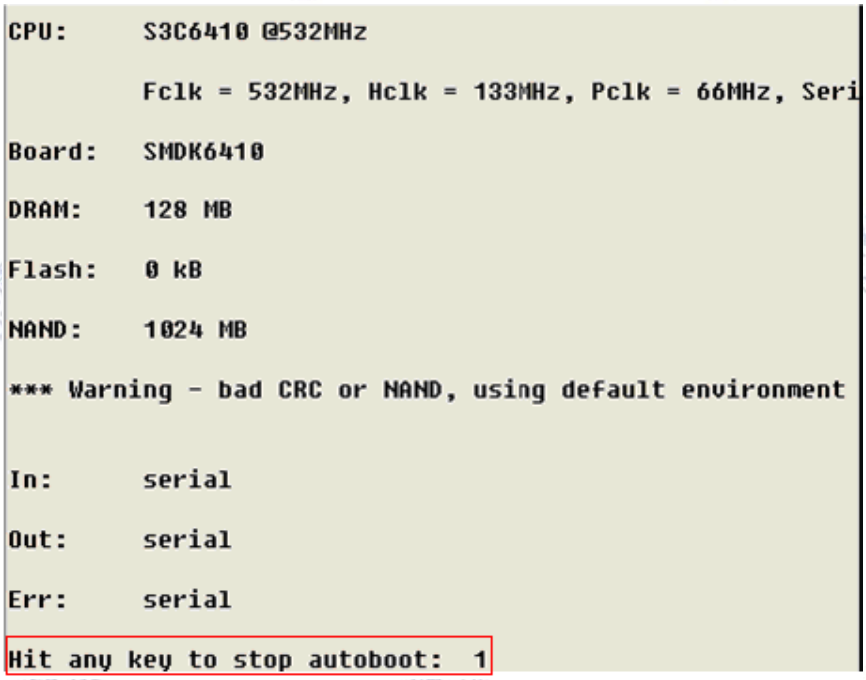


图2-105 uboot 信息

步骤3：在uboot 状态下键入#nand scrub，如图2-106所示：

```
SMDK6410 # nand scrub

NAND scrub: device 0 whole chip

Warning: scrub option will erase all factory set bad blocks!
         There is no reliable way to recover them.
         Use this command only for testing purposes if you
         are sure of what you are doing!

Really scrub this NAND flash? <y/N>
```

图2-106　输入信息

步骤4：接着键入‘y’，键入的‘y’在超级终端中并没有显示出来，接着回车。稍等几秒，会完成擦除坏块的工作，如下图2-107：

```
Erasing at 0x3f5c0000 --  99omplete.
Erasing at 0x3ffc0000 -- 100omplete.

Scanning device for bad blocks

OK

SMDK6410 #
```

图2-107　擦除坏块

这时候，整个开发板的NandFlash会被清空，坏块也会被处理。您可以使用一键烧写Linux的方法把linux重新烧写一遍。

注：在开发和学习的过程免不了产生一些逻辑上的坏块，用上面的方法可以处理。但是nand scrub命令本身不宜经常使用，可能会对nand进行错误的校正。这些坏块是nand的一个特征。如果您有兴趣，可以进一步了解nand的特性以及处理机制。

2.6　物联网实践实验平台

物联网实践实验平台具有形式新颖的实验内容，良好的操作界面实现数据的显示和控制功能，便于老师和学生理解，使教学具有更强的直观性。并提供多种互动体验形式，通过简单的操作，如改变配置参数或系统反馈控制等，就可实现不同的实验过程及效果。实验内容从简单至复杂、从基础到高端，多层次覆盖物联网体系架构的各个组成部分，通过由浅入深的实验内容，从基础实验到实训项目，帮助师生逐渐理解物联网的技术内涵和结构，树立并提升对物联网概念的理解，并平滑过渡到物联网的行业应用中。

在实训项目中，通过多个具有典型的应用领域为模型，通过人性化设计和智能化管理，

生动、形象、快速、准确地展示了物联网在众多领域的实际应用情况，有助于提高学生对物联网应用的理解和学习。

实验平台安装软件位置：E-Box300\02-开发环境搭建\03-上位机实验平台，具体安装方法可参见与安装软件同目录下的《上位机环境搭建用户手册》。

图2-108　物联网实践实验平台

2.6.1 PC 机环境配置

步骤 1：设置 PC 机 IP 地址为：192.168.1.120（ARM 网关默认的 IP 地址），子网掩码为：255.255.255.0。如图 2-109 所示：

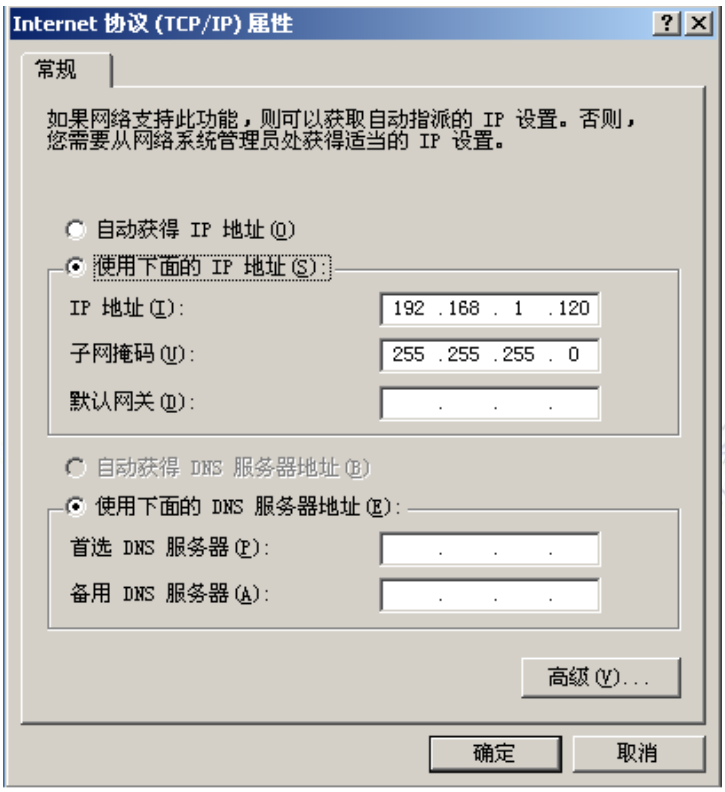


图2-109　设置 IP

ARM 网关网口灯亮，PC 机右下角如图 2-110 所示，ARM 网关与 PC 机连接成功。

图2-110　网络连接

2.6.2 软件平台登录

双击物联网智能家居实践实验平台快捷图标，出现登录界面。类型选择默认有两种，分别为管理员和普通用户。管理员默认的登陆用户名为：Admin，密码：admin。管理员的权限比较高。登录之后，可以进行密码修改（包括普通用户），普通用户则没有修改密码的权限。若用普通用户登录，用户名可以随意取名，密码全部默认为 admin，软件平台会记录登录信息。登录后，会在安装文件目录下 C:\Program Files\SmartHome\Experiment 自动新建一个同名称的文件夹。在后续进行 python 综合实验时会将生成的 python 文件保存在其中，方便不同班级学生做实验时进行管理和区分。

点击登录，进入物联网智能家居实践实验平台，软件界面包括菜单栏、工具栏、系统导航和显示界面。选择系统配置，对软件平台进行一些基本的配置。

（1）预警服务设置。配置手机号码可以在环境监控出现状况的时候以短信的方式提示报警信息

（2）接入参数配置。自动请求 Rssi（Received Signal Strength Indication，即接收的信号强度指示）是指上位机是否向 zigbee 协调器发送广播请求；Rssi 刷新间隔是指每隔多长时间 zigbee 协调器向上位机发送一次数据；能接受未收到次数是指如果上位机多次接收不到 zigbee 协调器发送的信息则默认某个传感器不在网络中。（建议将自动请求 Rssi 设置为否）。

（3）端口配置。服务器 IP 地址已经设置好，不可修改，与 PC 机的 IP 地址一致。

（4）串口配置。串口配置方便扩展其他各种功能的测试，这里选择默认的即可，无需修改。

（5）管理员密码修改。修改管理员登陆密码。管理员登陆有权限修改，普通用户登录无权限修改。

（6）普通用户密码修改。对于普通用户的密码修改，指的是对当前软件平台已经设置过的所有普通用户的密码修改。管理员登陆有权限修改，普通用户登录无权限修改。

第三章 物联网基础实验

3.1物联网 ZIGBEE 模块单片机基础实验

3.1.1 实验一 自动闪烁

一、 实验目的

1. 掌握CC2531的I/0来控制外设的方法。
2. 了解并学习CC2531的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

本实验以LED灯为外设，用CC2531控制LED灯的闪烁。

四、 实验原理

1. LED 灯控制电路：（如图 3-1 所示）

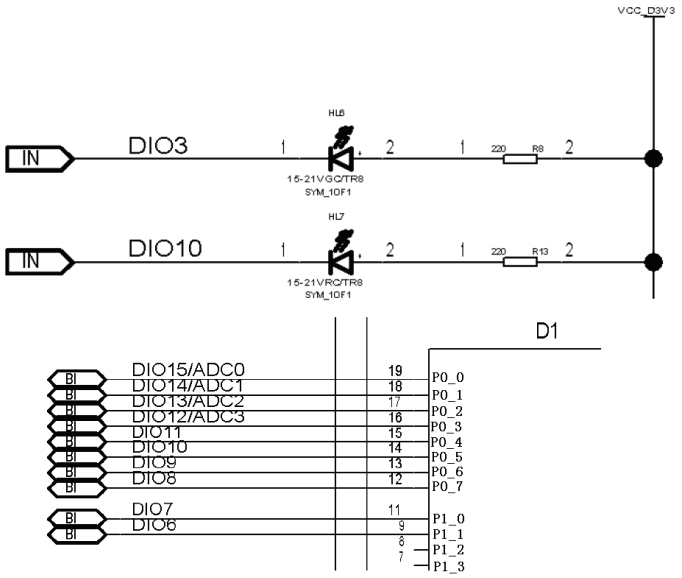


图3-1 LED 灯控制电路

图 3-1 是 ZigBee 模块中 LED 灯的控制电路。从图中可以看出，当 P0_5 和 P2_0 口为

高电平时，LED 灯两端电压相同，不会产生电流，两灯都是灭的；当 P0_5 和 P2_0 口为低电平时，两端相差 3.3V 电压，可以产生电流，两灯都是亮的。

2. 实验相关寄存器：

实验中操作的寄存器有P0，P2口寄存器和P0DIR，P2DIR寄存器，其他寄存器取默认值。

2.1 P0（P0口寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	P0[7:0]	0x00	可读/写	P0端口普通功能寄存器，可位寻址

2.2 P0DIR（P0口方向寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	DIRP0-[7:0]	0x00	可读/写	P0-[7:0]方向 0：输入；1：输出

P0DIR寄存器设置P0口对应位的输入或输出方向，向对应位写入0，则设置其为输入方向，向对应位写入1，则设置其为输出方向。复位值为0x00，即默认为输入方向。

注：CC2531的21个可编程I/O引脚特性一致，可设置为通常的I/O口，也可设置为外围I/O口使用，在输入时有上拉和下拉能力，并且也都具有相应外部中断的能力。故与三组I/0相关的寄存器配置参考其一即可，P2口寄存器配置不再重复（下同）。

有关I/O口配置的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第7.11节。

3. 实验相关函数：

写在程序中子函数及功能列写如下：

3.1 void Delay(uint n)；

函数原型是：

```
void Delay(uint n)
{
    uint tt;
    for(tt = 0;tt < n;tt++);
    for(tt = 0;tt < n;tt++);
    for(tt = 0;tt < n;tt++);
    for(tt = 0;tt < n;tt++);
    for(tt = 0;tt < n;tt++);
```

```
}
```

函数功能是软件延时。执行5次从0到n的空循环来实现软件延时，延时时间约为5*n/32μs。

3.2 void Initial(void)；

函数原型是：

```
void Initial(void)
{
    P0DIR |= 0x20; //P0_5定义为输出
    P2DIR |= 0x01; //P2_0定义为输出
    RLED = 1;
    YLED = 1;    //LED 灯灭
}
```

函数功能是把连接LED的两个I/0设置为输出，同时将它们设为高电平（此时LED灭）。

4． main（）函数：

```
void main(void)
{
    Initial();        //调用初始化函数
    while(1)
    {
            YLED = !YLED;
            RLED = !RLED;
            Delay(10000);
    }
}
```

main函数功能是实现了对两个LED灯的循环闪烁的控制。

五、 实验步骤

1. 连接硬件：将数传模块（EBM-TM）插入ZigBee模块，注意板卡上箭头指示方向和防反插指针位置（后续实验将这两个模块统称为ZigBee模块）。连接ZigBee模块和仿真器，连接ZigBee模块与电源适配器（5V-1A），打开ZigBee模块上的电源开关拨向A端（表示XP4供电）。如下图3-2所示：

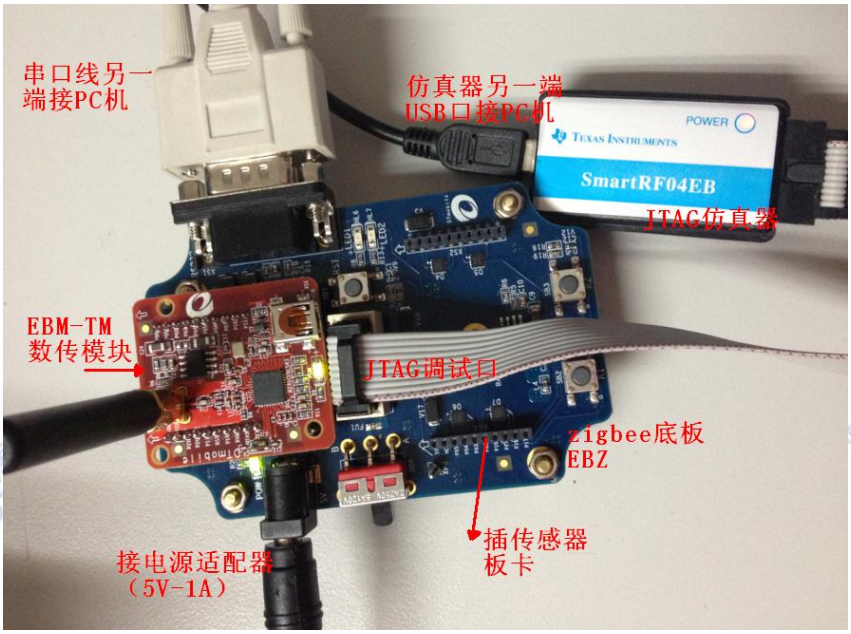


图3-2　　连接图

2. 打开IAR集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\1_Auto glint）。

2.1. 编译，如图3-3所示：

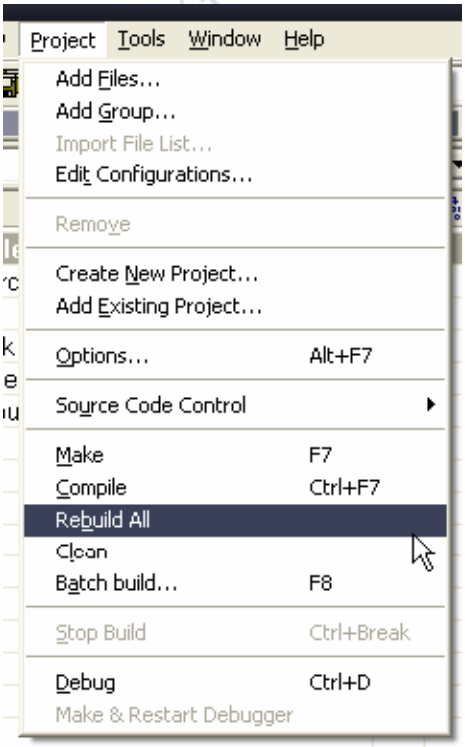


图3-3　　编译

2.2. 下载程序，如图3-4、3-5两种方式：

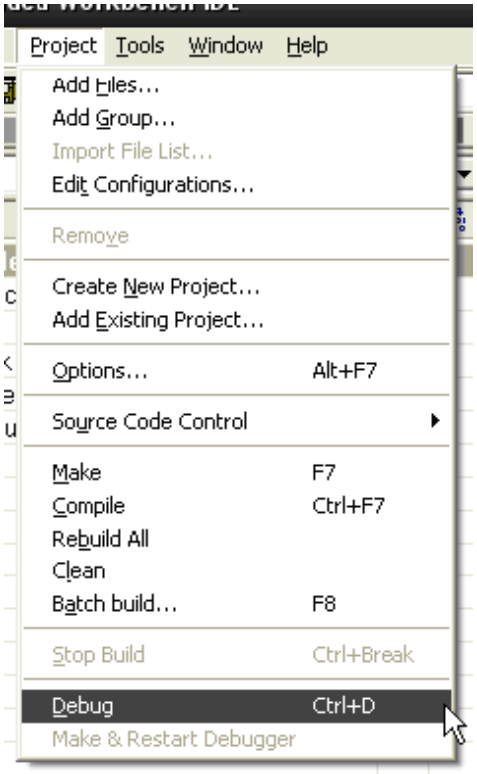


图3-4　　下载程序

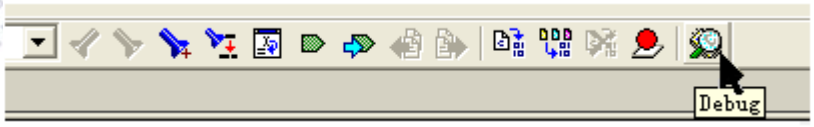


图3-5　　下载程序

3.观察实验现象。观察是ZigBee模块的两个LED灯是否不停闪烁。

六、　　拓展思考

CC2531一共有多少个IO管脚供用户使用？

七、　　参考阅读

请参考阅读　“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.2　实验二 按键控制 LED 灯开关实验

一、　　实验目的

1. 掌握 CC2531 的按键使用。
2. 通过对 CC2531 相关寄存器的修改实现对按键功能的应用。

二、　　实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

本实验是用两个按键分别控制两个 LED 灯的开和关。

四、 实验原理

1. 按键控制电路：(如图3-6所示)

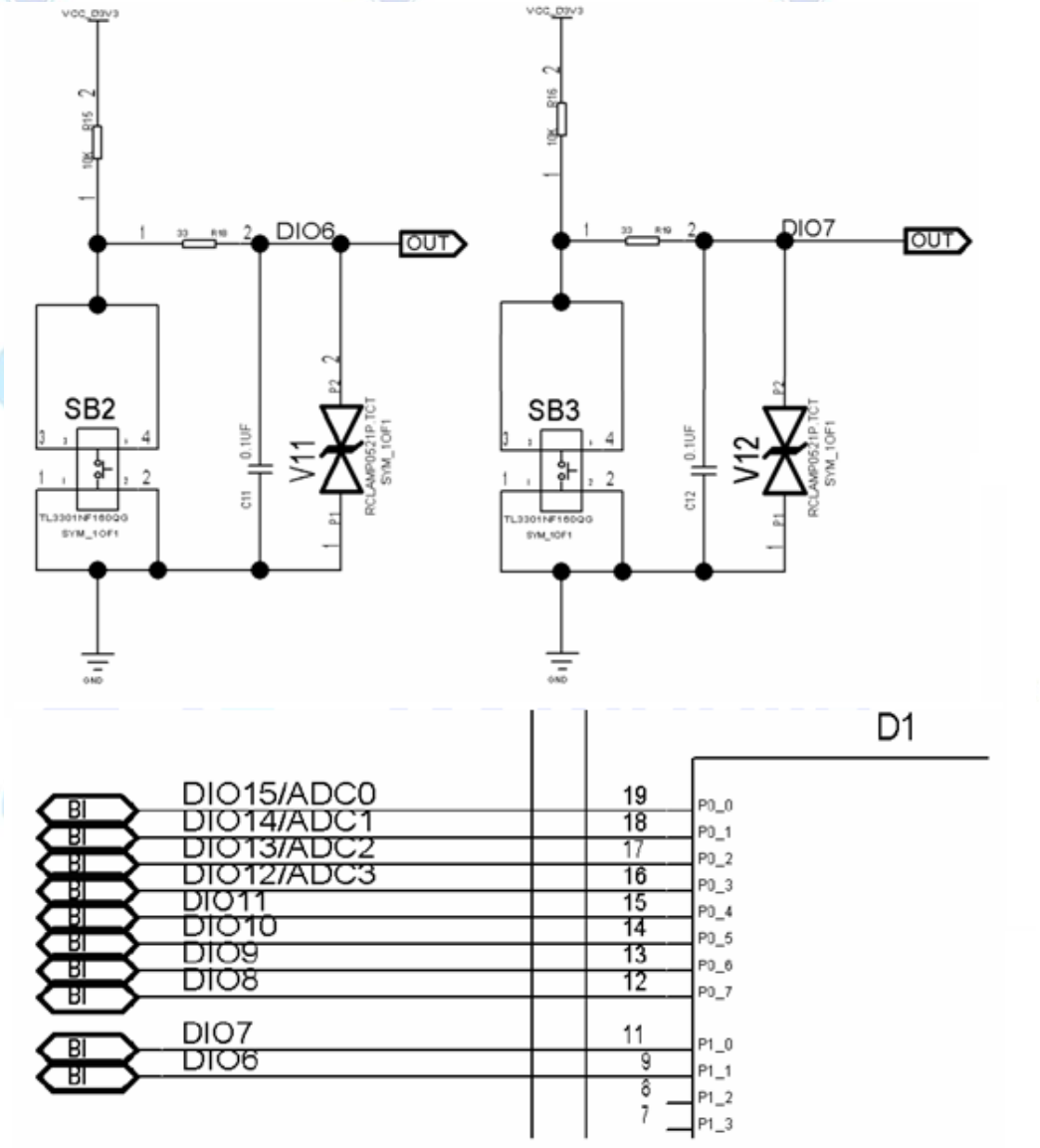


图3-6 按键控制电路图

上图3-6为ZigBee模块的按键控制电路。当按键按下之前，输出的是电源电压3.3V；当按键按下之后，电源与地导通，输出的电压为0V。

2. 实验相关寄存器：

实验中操作了的寄存器有P0，P1，P2（实验一），P0DIR，P2DIR（实验一），P1SEL，P1DIR，P1INP。

2.1 P1SEL(P1 功能选择寄存器) ：

位号	位名	复位值	操作性	功能描述
7:0	SELP1-[7:0]	0x00	可读/写	P1-[7:0]功能 0：普通I/O口；1：外设功能

P1SEL寄存器设置P0口对应位为通用I/O功能或者外围设备功能，向对应位写入0，则设置其为通用I/O功能，向对应位写入1，则设置其为外围设备功能。复位值为0x00，即默认为通用I/O功能。

2.2 P1DIR（P1 方向寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	DIRP1-[7:0]	0x00	可读/写	P1-[7:0]方向 0：输入；1：输出

P1DIR寄存器设置P0口对应位的输入或输出方向，向对应位写入0，则设置其为输入方向，向对应位写入1，则设置其为输出方向。复位值为0x00，即默认为输入方向。

2.3 P1INP (P1 口输入模式寄存器)：

位号	位名	复位值	可操作性	功能描述
7:0	MDP1_ [7:2]	0x00	R/W	P1_[7:2] ：输入模式。 0：上拉/下拉；1：三态

P1INP寄存器设置P0口对应位的输入模式为内部上拉/下拉模式或者三态模式，向对应位写入0，则设置其为内部上拉/下拉模式，向对应位写入1，则设置其为三态模式。复位值为0x00，即默认为内部上拉/下拉模式。其中，第1位和第0位未使用。

3. 实验相关函数：

写在程序中子函数及功能列写如下：

3.1 void InitKey(void)；

函数原型：

```
void InitKey(void)
{
```

```
    P1SEL &= ~0X03;
    P1DIR &= ~0X03; //按键在P1_0，P1_1
    P1INP |= 0x03;
}
```

函数功能是将 I/O P1_0、P1_1设为输入，以读取按键的状态。

3.2 unsigned char KeyScan(void)；

函数原型：

```
uchar KeyScan(void)
{
    if(K1 == 0)          //低电平有效
    {
        Delay(100);       //检测到按键
        if(K1 == 0)
        {
            while(!K1);    //直到松开按键
            return(1);
        }
    }
    if(K2 == 0)
    {
        Delay(100);
        if(K2 == 0)
        {
            while(!K2);
            return(2);
        }
    }
    return(0);
}
```

函数功能是检测按键是否按下，若有键按下，则返回相应的值，如P1_0对应的按键按下则返回1，P1_1对应的按键按下返回2。

4. main（）函数：

```
void main(void)
{
    Initial();        //调用初始化函数
    InitKey();
    YLED = ON;        //LED1黄灯亮表示开始工作
    RLED = OFF;             //LED2
    while(1)
    {
        Keyvalue = KeyScan();
        if(Keyvalue == 1)
        {
          RLED = !RLED;          //red
          Keyvalue = 0;          //清除键值
        }
        if(Keyvalue == 2)
        {
           YLED = !YLED;        //yellow
           Keyvalue = 0;
        }
    }
}
```

main（）函数功能是实现了按键K1控制黄色LED灯开和关，按键K2控制红色LED灯开和关。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。
2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整

程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\2_Key control）。

3. 观察实验现象。观察按键是否能够分别控制两个 LED 灯的开关状态。

六、拓展思考

CC2531与80C51的IO读写方式有何差别？

七、参考阅读

请参考阅读 “\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.3 实验三 按键控制 LED 灯闪烁实验

一、 实验目的

1. 掌握 CC2531 的按键应用这一常用人机交互方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

这个实验中使用两个按键分别控制LED灯或闪烁，或熄灭。

四、 实验原理

1. 实验相关寄存器：

实验中操作的寄存器有P0，P1，P2（实验一），P0DIR，P2DIR（实验一），P1DIR（实验二），P1SEL（实验二），P1INP（实验二）。

2. main ( ) 函数：

```
void main(void)
{
    Initial();              //调用初始化函数
```

```
    InitKey();
    YLED = ON;      //黄灯亮一下表示开始工作
    Delay(4000);
    Delay(4000);
    while(1)
    {
        Keyvalue = KeyScan(); //扫键
        if(Keyvalue>0)
        {
          if(Keyvalue == 1)
            GlintFlag[0] = !GlintFlag[0];
          if(Keyvalue == 2)
            GlintFlag[1] = !GlintFlag[1];
        };
      if(GlintFlag[0]==1)
      {
        RLED = !RLED;          //闪灯
        Delay(4000);
      }
      else
        RLED = OFF;            //关灯
      if(GlintFlag[1]==1)
      {
        YLED = !YLED;
        Delay(4000);
      }
      else
        YLED = OFF;
    }
}
```

main（）函数功能是实现了按键分别控制两个LED灯或闪烁，或熄灭。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\3_Key glint）。

3. 观察实验现象。观察两个按键是否能够分别控制两个 LED 灯的闪烁或者熄灭。

六、拓展思考

CC2531与80C51的IO读写方式有何差别？

七、参考阅读

请参考阅读 “\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.4 实验四 定时器 T1 的使用

一、 实验目的

1. 掌握 T1 的使用方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

用定时器1来改变小灯的状态，T1每溢出两次，红色LED灯闪烁一次。

四、 实验原理

1. 定时器1

定时器1是一个独立的16位定时器，支持典型的定时/计数功能，比如输入捕获，输出比

较和PWM 功能。定时器有五个独立的捕获/比较通道。每个通道定时器使用一个I/O 引脚。定时器用于范围广泛的控制和测量应用，可用的五个通道的正计数/倒计数模式将允许诸如电机控制应用的实现。。定时器有一个很重要的概念：操作模式。操作模式包含：自由运行模式（free-running）、模模式（modulo）和计数/倒计数模式（up-down）。自由运行模式的溢出值位0xFFFF不可变；而其他两种模式则可通过对T1CC0赋值，以精确控制定时器的溢出值。本实验正式利用此特性，通过特定的T1CC0，使定时器每个1s触发一次中断，从而精确控制LED灯的闪烁间隔为1s。

有关定时器1的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第9节。

2. 实验相关寄存器：

实验中操作了的寄存器有 P0，P2（实验一），P0DIR，P2DIR（实验一），T1CTL。

1.1 T1CTL(T1控制＆状态寄存器)：

位号	位名	复位值	操作性	功能描述
7:4	----	0000	可读/写	保留
3:2	DIV[1:0]	00	可读/写	定时器1计数时钟分步选择 00：不分频 01：8 分频 10：32分频 11：128分频
1:0	MODE[1:0]	00	可读/写	定时器模式选择 00：暂停；01：自动重装0x0000-0xffff 10：比较计数0x0000-T1CC0 11：PWM方式0x0000-TICC0-0X0000

本实验使用定时器1为128分频和自动重装模式，因此只需设置T1CTL的第3-0位，即T1CTL = 0x0d。自动重装是指当定时器的计数器从初值开始计数直到溢出产生中断后，会自动将其重新初始化位初值。

3. 实验相关函数：

写在程序中的子函数及功能列写如下：

3.1 void Initial(void)；

函数原型：

```
void Initial(void)
```

```
{
    P2DIR |= 0x01;
    P0DIR |= 0x20;
    YLED = 1;
    RLED = 1;    //LED
    T1CTL = 0x0d;    //128分频;自动重装模式(0x0000->0xffff);
}
```

函数功能是将P2_0，P0_5设为输出，并将定时器1设为自动重装模式，计数时钟为0.25M。

4. mian ( ) 函数：

```
void main()
{
    Initial();    //调用初始化函数
    YLED = 0;     //点亮黄色LED
    while(1)      //查询溢出
    {
      counter++;
      if(counter>1)
      {
        if(IRCON > 0)
        {
          IRCON = 0;                    //清溢出标志
          RLED = !RLED;
        }
      }
    }
}
```

main（）函数功能是实现了T1每溢出两次，红色LED灯闪烁一次。IRCON为中断标志位。当T1计时的时候，IRCON=0，不进入中断，当T1溢出时，IRCON=1，进入中断。再把IRCON置0，以便下一次溢出在再入中断。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\4_Timer1）。

3. 观察实验现象。观察红色 LED 灯改变的状态。

六、拓展思考

在本实验中，定时器1经过524.28毫秒溢出一次，怎么计算得到这个定时时间？

七、参考阅读

请参考阅读 “\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.5 实验五 定时器 T2 的使用

一、 实验目的

1. 掌握 T2 的使用方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

用定时器 2 来改变小灯的状态，T2 每发生 200 次中断小灯改变状态一次。

四、 实验原理

1. 定时器2

定时器2主要用于为802.15.4 CSMA-CA算法提供定时，以及为802.15.4 MAC层提供一般的计时功能。定时器2的主要特性如下：16位定时器正计数提供的符号/帧周期；可变周期

可精确到31.25ns；2×16位定时器比较功能；24位溢出计数；2×24位溢出计数比较功能；帧首定界符捕捉功能；定时器启动/停止同步于外部32kHz时钟以及由睡眼定时器提供定时；比较和溢出产生中断；具有DMA触发功能；通过引入延迟可调整定时器值。

有关定时器2的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第18节。

2. 实验相关寄存器：

实验中操作了的寄存器有 P0，P2（实验一），P0DIR,，P2DIR（实验一），P0SEL（实验二），T2MSEL ，T2M0，T2M1，T2MOVF0，T2MOVF1，T2MOVF2，T2CTRL，IEN0等寄存器。

2.1 T2MSEL（定时器2复用选择）

位号	位名	复位	R/W	功能描述
7	-	0	R0	保留。读作0
6::4	T2MOVFSEL	0	R/W	该寄存器的值选择当访问T2MOVF0、T2MOVF1 和T2MOVF2 时修改或读的内部寄存器。 000：t2ovf（溢出计数器） 001：t2ovf_cap（溢出捕获） 010：t2ovf_per（溢出周期） 011：t2ovf_cmp1（溢出捕获1） 100：t2ovf_cmp2（溢出捕获2） 101 to 111：保留
3	-	0	R0	保留。读作0
2:0	T2MSEL	0	R/W	该寄存器的值选择当访问T2M0 和T2M1 时修改或读的内部寄存器。 000：t2tim（定时器计数值） 001：t2_cap（定时器捕获） 010：t2_per（定时器周期） 011：t2_cmp1（定时器比较1） 100：t2_cmp2（定时器比较2）

				101 to 111：保留

2.2 T2M0（定时器2复用寄存器0）

位号	位名	复位值	操作性	功能描述
7:0	T2M0	0	R/W	根据T2MSEL.T2MSEL 的值，直接返回/修改一个内部寄存器位[7:0]。 当读T2M0 寄存器，T2MSEL.T2MSEL 设置为000，且T2CTRL.LATCH_MODE设置为0，定时器（t2tim）值被锁定。 当读T2M0 寄存器，T2MSEL.T2MSEL 设置为000，且T2CTRL.LATCH_MODE设置为1，定时器（t2tim）和溢出计数器（t2ovf）值被锁定。

2.3 T2M1（定时器2复用寄存器1）

位号	位名	复位值	操作性	功能描述
7:0	T2M1	0	R/W	根据T2MSEL.T2MSEL 的值，直接返回/修改一个内部寄存器位[15:8]。 当读T2M0 寄存器，T2MSEL.T2MSEL 设置为000，定时器（t2tim）值被锁定。 当读该寄存器T2MSEL.T2MSEL 设置为000，返回t2tim[15:8]锁定的值。

2.4 T2MOVF0（定时器2复用溢出寄存器0）

位号	位名	复位值	操作性	功能描述
7:0	T2MOVF0	0	R/W	根据T2MSEL.T2MOVFSEL 的值，直接返回/修改一个内部寄存器位[7:0]。 当读T2MOVF0寄存器T2MSEL.T2MOVFSEL 设置为000 且T2CTRL.LATCH_MODE 设置为0，溢出计数器值（t2ovf）被锁定。 当读T2M0 寄存器， T2MSEL.T2MOVFSEL

				设置为000 且T2CTRL.LATCH_MODE 设置为0，溢出计数器值（t2ovf）被锁定。

2.5 T2MOVF1（定时器2复用溢出寄存器1）

位号	位名	复位值	操作性	功能描述
7:0	T2MOVF1	0	R/W	根据T2MSEL.T2MOVFSEL 的值，直接返回/修改一个内部寄存器位[15:8]。 当读该寄存器，T2MSEL.T2MOVFSEL 设置为000，返回t2ovf[15:8]被锁定的值。

2.6 T2MOVF2（定时器2复用溢出寄存器2）

位号	位名	复位值	操作性	功能描述
7:0	T2MOVF2	0	R/W	根据T2MSEL.T2MOVFSEL 的值，直接返回/修改一个内部寄存器位[23:16]。 当读该寄存器，T2MSEL.T2MOVFSEL 设置为000，返回t2ovf[23:16]被锁定的值。

2.7 T2CTRL（定时器2控制寄存器）

位号	位名	复位	R/W	功能描述
7:4	----	0000	R0	保留。读作0
3	LATCH_MODE	0	R/W	0：读T2M0，T2MSEL.T2MSEL = 000 锁定定时器的高字节，使它准备好从T2M1读。读T2MOVF0，T2MSEL.T2MOVFSEL= 000锁定溢出计数器的两个最高字节，使可以从T2MOVF1 和T2MOVF2 读它们。 1：读T2M0，T2MSEL.T2MSEL = 000一次锁定定时器和整个溢出计数器，使可以从读T2M1、T2MOVF0、T2MOVF1 和T2MOVF2 值。
2	STATE	0	R	定时器2 的状态 0：定时器空闲

				1：定时器运行
1	SYNC	1	R/W	0：启动和停止定时器是立即的，即和 clk_rf_32m 同步。 1：启动和停止定时器在第一个正32 kHz 时钟边沿发生。
0	RUN	0	R/W	写1 启动定时器，写0 停止定时器。读时，返回最后写入的值。

2.8 IEN0(中断使能控制寄存器0)：

位号	位名	复位值	操作性	功能描述
7	EAL	0	可读/写	总中断使能 0：禁止所有中断；1：允许中断
6	——	0	读0	未用，读出为0
5	STIE	0	可读/写	睡眠定时器中断使能 0：关中断；1：开中断
4	ENCIE	0	可读/写	AES加解密中断使能 0：关中断；1：开中断
3	URX1IE	0	可读/写	串口1接收中断使能 0：关中断；1：开中断
2	URX01E	0	可读/写	串口0接收中断使能 0：关中断；1：开中断
1	ADCIE	0	可读/写	ADC中断使能 0：关中断；1：开中断
0	RFERRIE	0	可读/写	射频TX/RX FIFO中断 0：关中断；1：开中断

IEN0的第7位EAL为总中断使能位，如果此位为0，所有中断都不会被应答。

3. 实验相关函数：

写在程序中的子函数及功能列写如下：

3.1 void Initial(void)；

函数原型：

```
void Initial(void)
{
    LED_ENALBLE();//启用 LED
    //用T2来做实验
    SET_TIMER2_CAP_INT();                    //开比较中断
    SET_TIMER2_CAP_COUNTER(0X00ff);          //设定溢出值
}
```

函数功能是启用LED，使用LED可控，开T2比较中断。

3.2 void T2_ISR(void)；

函数原型：

```
#pragma vector = T2_VECTOR
__interrupt void T2_ISR(void)
 {
        SET_TIMER2_CAP_COUNTER(0X00ff);
        CLEAR_TIMER2_INT_FLAG();        //清T2中断标志
         if(counter<200)counter++;          //200次中断LED闪烁一轮
         else
         {
            counter = 0;                    //计数清零
            TempFlag = 1;                   //改变闪烁标志
         }
 }
```

函数功能是T2中断服务程序，每200次中断LED灯闪烁一轮，200次以后计数清零，改变闪烁标志。在IAR编译器里，用关键字_interrupt来定义一个中断函数，用#pragma来提供中断函数的入口地址。每种中断的入口地址在头文件里都有描述。函数名T2_ISR可以为任意名称。中断函数会自动保护局部变量，但不会保护全局变量。

3. 重要的宏定义：

3.1 开比较中断

```
#define SET_TIMER2_CMP_INT()        \
  do{                               \
    EA = 1;                         \
    T2IE = 1;                       \
    T2PEROF2 |= 0x40;               \
    }while(0)
```

开定时器中断需要三个操作：全局中断允许，定时器本身允许中断，定时器的溢出屏蔽中断。在本实验中，EA为IEN0（中断使能控制寄存0）的第7位（从0开始），置1时位允许总中断使能。T2IE为Timer2的中断使能，IEN1的第2位（从0开始），置1为开定时器2中断使能。T2PEROF2|= 0x40，相当于T2溢出计数器2寄存器的第6位（从0开始）置1，即开溢出中断。..

4． main ( ) 函数：

```
void main()
{
    Initial();   //调用初始化函数
    YLED = 0;    //点亮黄色LED表示系统工作
    RLED = 1;
    TIMER2_RUN();
    while(1)        //等待中断
    {
        if(TempFlag)
        {
            RLED = !RLED;
            TempFlag = 0;
         }
    }
}
```

main（）函数功能是实现了T2每发生200次中断小灯改变状态一次。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\5_Timer2）。本实验程序代码中有自定义的头文件，注意在建立工程的时候在 Project→Option→Preprocessor 中添加相应包含文件的路径。

3. 观察实验现象。观察实验现象是否为每间隔一定的时间，红色 LED 灯改变状态一次。

六、 拓展思考

改变相关寄存器的设置，使得每次闪烁的间隔时间为 2 秒。

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.6 实验六 定时器 T3 的使用

一、 实验目的

1. 掌握 T3 的使用方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

用定时器3来改变小灯的状态，T3每发生200次中断小灯改变状态一次。

四、 实验原理

1. 定时器3和4是两个8位的定时器。每个定时器有两个独立的比较通道，每个通道上使用一个I/O引脚。

定时器3/4的特性如下：

- 两个捕获/比较通道
- 设置、清除或切换输出比较
- 时钟分频器，可以被1，2，4，8，16，32，64，128 整除
- 在每次捕获/比较和最终计数事件发生时产生中断请求
- DMA触发功能

有关定时器3和定时器4的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第10节。

2. 实验相关寄存器：

实验中操作了的寄存器有P0，P2（实验一），P0DIR,，P2DIR（实验一），T3CNT，T3CTL，T3CCTL0，T3CC0，T3CCTL1，T3CC1等寄存器。

2.1 T3CNT（定时器3计数器）：

位号	名称	复位	操作性	功能描述
7:0	CNT[7:0]	0x00	R	定时器计数字节。包含8位计数器当前值

2.2 T3CTL（T3控制寄存器）：

位号	位名	复位值	操作性	功能描述
7:5	DIV[2:0]	000	可读/写	定时器时钟再分频数（对CLKCON.TICKSPD 分频后再次分频） 000：不再分频； 001：2 分频 010：4 分频； 011：8 分频 100：16 分频； 101：32 分频 110：64 分频； 111：128 分频
4	START	0	可读/写	T3 起停位 0：暂停计数；1：正常运行
3	OVFIM	1	可读/写 0	溢出中断掩码 0：关溢出中断；1：开溢出中断
2	CLR	0	可读 0/写 1	清计数值，写 1 使 T3CNT=0x00
1:0	MODE[1:0]	00	可读/写	T3 模式选择 00：自动重装

				01：DOWN（从 T3CC0 到 0x00 计数一次） 10：模拟数（反复从 0x00 到 T3CC0 计数） 11：UP/DOWN（反复从 0x00 到 T3CC0 再到 0x00）

2.4 T3CCTL0（T3 通道 0 捕获/比较控制寄存器）：

位号	位名	复位值	操作性	功能描述
7	—	0	可读 0	保留
6	IM	1	可读/写	通道 0 中断掩码 0：关中断；1：开中断
5:3	CMP[7:0]	000	可读/写	通道 0 比较输出模式选择，指定计数值过 T3CC0 时的发生事件 000：输出置 1（发生比较时） 001：输出清 0（发生比较时） 010：输出翻转 011：输出置 1（发生上比较时）输出清 0（计数值为 0 或 UP/DOWN 模式下 发生下比较） 100：输出清 0（发生上比较时）输出置 1（计数值为 0 或 UP/DOWN 模式下 发生下比较） 101：输出置 1（发生比较时）输出清 0（计数值为 0xff 时） 110：输出清 0（发生比较时）输出置 1（计数值为 0x00 时） 111：没用
2	MODE-	0	可读/写	T3 通道 0 模式选择 0：捕获；1：比较
1:0	CAP	00	可读/写	T3 通道 0 捕获模式选择 00：没有捕获； 01：上升沿捕获

				10：下降沿捕获； 11：边沿捕获

2.5 T3CC0（T3 通道 0 捕获/比较值寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	VAL[7:0]	0x00	可读/写	T3 通道 0 比较/捕获值

2.6 T3CCTL1(T3 通道 1 捕获/比较控制寄存器)：

位号	位名	复位值	操作性	功能描述
7	—	0	可读 0	保留
6	IM	1	可读/写	通道 1 中断掩码 0：关中断；1：开中断
5:3	CMP[7:0]	000	可读/写	通道 1 比较输出模式选择，指定计数值过 T3CC0 时的发生事件 000：输出置 1（发生比较时） 001：输出清 0（发生比较时） 010：输出翻转 011：输出置 1（发生上比较时）输出清 0（计数值为 0 或 UP/DOWN 模式下 发生下比较） 100：输出清 0（发生上比较时）输出置 1（计数值为 0 或 UP/DOWN 模式下 发生下比较） 101：输出置 1（发生比较时）输出清 0（计数值为 0xff 时） 110：输出清 0（发生比较时）输出置 1（计数值为 0x00 时） 111：没用
2	MODE-	0	可读/写	T3 通道 1 模式选择 0：捕获；1：比较
1:0	CAP	00	可读/写	T3 通道 1 捕获模式选择 00：没有捕获； 01：上升沿捕获 10：下降沿捕获；11：边沿捕获

2.7　T3CC1（T3 通道 1 捕获/比较值寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	VAL[7:0]	0x00	可读/写	T3 通道 1 比较/捕获值

3.　实验相关函数：

写在程序中的子函数及功能列写如下：

3.1 void Init_T3_AND_LED(void)；

函数原型：

```
void Init_T3_AND_LED(void)
{
    P2DIR |= 0x01;
    P0DIR |= 0x20;
    RLED = 1;
    YLED = 1;
    TIMER34_INIT(3);                          //初始化T3
    TIMER34_ENABLE_OVERFLOW_INT(3,1);   //开T3中断
    TIMER3_SET_CLOCK_DIVIDE(16);          //定时器时钟16分频
    TIMER3_SET_MODE(T3_MODE_FREE);
    TIMER3_START(1);                          //启动
};
```

函数功能：将I/0 P2_0,P0_5设置为输出去控制LED，将T3设置为自动重装模式，定时器时钟16分频，并启动T3。

3.2 void T3_ISR(void)；

函数原型：

```
#pragma vector = T3_VECTOR
 __interrupt void T3_ISR(void)
 {
        if(counter<200)counter++;          //200次中断LED闪烁一轮
        else
        {
```

```
            counter = 0;                    //计数清零
            RLED = !RLED;                   //改变小灯的状态
          }
    }
```

函数功能：这是一个中断服务程序，每200次中断改变一次LED灯的状态。

4. 重要的宏定义：

4.1 复位 T3 相关寄存器：

```
#define TIMER34_INIT(timer)     \
    do {                        \
        T##timer##CTL     = 0x06; \
        T##timer##CCTL0 = 0x00; \
        T##timer##CC0     = 0x00; \
        T##timer##CCTL1 = 0x00; \
        T##timer##CC1     = 0x00; \
    } while (0)
```

功能：将T3相关的寄存器复位到0。

4.3 打开T3溢出中断：

```
#define TIMER34_ENABLE_OVERFLOW_INT(timer,val) \
    (T##timer##CTL =    (val) ? T##timer##CTL | 0x08 : T##timer##CTL & ~0x08)
```

4.2 控制 T3 起停：

```
#define TIMER3_START(val)                       \
    (T3CTL = (val) ? T3CTL | 0X10 : T3CTL&~0X10)
```

功能：val为1，T3正常运行，val为0，T3停止计数。

4.3 设置T3工作方式：

```
#define TIMER3_SET_MODE(val)                    \
do{                                             \
    T3CTL &= ~0X03;                             \
    (val==1)?(T3CTL|=0X01):   /*DOWN          */ \
    (val==2)?(T3CTL|=0X02):   /*Modulo        */ \
    (val==3)?(T3CTL|=0X03):   /*UP / DOWN     */ \
```

```
    (T3CTL|=0X00);              /*free runing */          \
}while(0)
```

功能：根据 val 的值将 T3 设置为不同模式，一共 4 种模式。

4.4　设置 T3 的时钟分步选择：

```
#define TIMER3_SET_CLOCK_DIVIDE(val)          \
do{                                           \
    T3CTL &= ~0XE0;                           \
    (val==2) ? (T3CTL|=0X20):                 \
    (val==4) ? (T3CTL|=0x40):                 \
    (val==8) ? (T3CTL|=0X60):                 \
    (val==16)? (T3CTL|=0x80):                 \
    (val==32)? (T3CTL|=0xa0):                 \
    (val==64) ? (T3CTL|=0xc0):                \
    (val==128) ? (T3CTL|=0XE0):               \
    (T3CTL|=0X00);              /* 1 */       \
}while(0)
```

功能：根据 val 的值将 T3 设置为不同时钟分步，一共 8 种模式。本实验设置 val=16，即定时器时钟 16 分频。

5．main ( ) 函数：

```
void main(void)
{
    Init_T3_AND_LED();
    YLED = 0;
    while(1);                   //等待中断
}
```

main（）函数功能是实现了 T3 每发生 200 次中断小灯改变状态一次。

五、　实验步骤

1. 按要求连接好硬件，如实验一所示。
2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整

程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\6_Timer3）。

3. 观察实验现象。观察红色 LED 灯是否自动不停的改变状态。

六、 拓展思考

1. 本实验中，定时器3每240uS进入一次中断程序，即定时器3的定时时间为240uS，怎么计算得到这个定时时间？

2. 如果要求定时器3每480uS进入一次中断程序，即定时器3的定时时间为480uS，应做怎样的修改？

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.7 实验七 定时器 T4 的使用

一、 实验目的

1. 掌握 T4 的使用方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

用定时器4来改变小灯的状态，T4每发生200次中断小灯改变状态一次。

四、 实验原理

1. CC2531 定时器 T4 与定时器 T3 使用方法一样，在此不再做详细介绍。

2. 实验相关寄存器：

实验中操作了的寄存器有P0，P2（实验一），P0DIR，P2DIR（实验一），T4CTL，T4CCTL0，T4CC0，T4CCTL1，T4CC1等寄存器。

2.1 T4CNT（定时器4计数器）

位号	名称	复位	操作性	功能描述
7:0	CNT[7:0]	0x00	R	定时器计数字节。包含8位计数器当前值

2.2 T4CTL(T4 控制寄存器)：

位号	位名	复位值	操作性	功能描述
7:5	DIV[2:0]	000	可读/写	定时器时钟再分频数（对 CLKCON.TICKSPD 分频后再次分频） 000：不再分频；001：2 分频 010：4 分频； 011：8 分频 100：16 分频； 101：32 分频 110：64 分频； 111：128 分频
4	START	0	可读/写	T4 起停位 0：暂停计数；1：正常运行
3	OVFIM	1	可读/写 0	溢出中断掩码 0：关溢出中断；1：开溢出中断
2	CLR	0	可读 0/写 1	清计数值，写 1
1:0	MODE[1:0]	00	可读/写	T4 模式选择 00：自动重装 01：DOWN（从 T4CC0 到 0x00 计数一次） 10：模拟数（反复从 0x00 到 T4CC0 计数） 11：UP/DOWN（反复从 0x00 到 T4CC0 再到 0x00）

2.3 T4CCTL0（T4 通道 0 捕获/比较控制寄存器）：

位号	位名	复位值	操作性	功能描述
7	—	0	可读 0	保留
6	IM	1	可读/写	通道 0 中断掩码 0：关中断；1：开中断

5:3	CMP[7:0]	000	可读/写	通道 0 比较输出模式选择，指定计数值过 T4CC0 时的发生事件 000：输出置 1（发生比较时） 001：输出清 0（发生比较时） 010：输出翻转 011：输出置 1（发生上比较时）输出清 0（计数值为 0 或 UP/DOWN 模式下 发生下比较） 100：输出清 0（发生上比较时）输出置 1（计数值为 0 或 UP/DOWN 模式下 发生下比较） 101：输出置 1（发生比较时）输出清 0（计数值为 0xff 时） 110：输出清 0（发生比较时）输出置 1（计数值为 0x00 时） 111：没用
2	MODE-	0	可读/写	T4 通道 0 模式选择 0：捕获；1：比较
1:0	CAP	00	可读/写	T4 通道 0 捕获模式选择 00：没有捕获；01：上升沿捕获 10：下降沿捕获；11：边沿捕获

2.4 T4CC0（T4 通道 0 捕获/比较值寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	VAL[7:0]	0x00	可读/写	T4 通道 0 比较/捕获值

2.5 T4CCTL1(T4 通道 1 捕获/比较控制寄存器)：

位号	位名	复位值	操作性	功能描述
7	—	0	可读 0	保留
6	IM	1	可读/写	通道 1 中断掩码

				0：关中断；1：开中断
5:3	CMP[7:0]	0	可读/写	通道 1 比较输出模式选择，指定计数值过 T4CC0 时的发生事件 000：输出置 1（发生比较时） 001：输出清 0（发生比较时） 010：输出翻转 011：输出置 1（发生上比较时）输出清 0（计数值为 0 或 UP/DOWN 模式下 发生下比较） 100：输出清 0（发生上比较时）输出置 1（计数值为 0 或 UP/DOWN 模式下 发生下比较） 101：输出置 1（发生比较时）输出清 0（计数值为 0xff 时） 110 输出清 0（发生比较时）输出置 1（计数值为 0x00 时） 111 没用
2	MODE-	0	可读/写	T4 通道 1 模式选择 0：捕获；1：比较
1:0	CAP	0000	可读/写	T4 通道 1 捕获模式选择 00：没有捕获；01：上升沿捕获 10：下降沿捕获；11：边沿捕获

2.6 T4CC1（T4 通道 1 捕获/比较值寄存器）：

位号	位名	复位值	操作性	功能描述
7:0	VAL[7:0]	0x00	可读/写	T4 通道 1 比较/捕获值

3. 实验相关函数：

定时器T4的程序与T3相似，不再一一列出，参考实验六的相关函数以及完整程序代码即可。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\7_Timer4）。

3. 观察实验现象。观察红色 LED 灯是否自动不停的改变状态。

六、 拓展思考

1. 本实验中，定时器4每2048uS进入一次中断程序，即定时器定时时间为2048uS，怎么计算得到这个定时时间？

2. 修改寄存器值，使定时器4定时时间为240uS。

六、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.8 实验八 外部中断实验

一、 实验目的

1. 掌握 CC2531 外部中断的使用方法。
2. 了解 CC2531 中断寄存器的配置。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

使用两个按键K1和K2产生中断信号，翻转LED灯的状态。这里两个按键不是做键盘用，而是产生中断触发信号。

四、 实验原理

1. 中断：

CPU 有 18 个中断源。每个中断源都有它自己的位于一系列 SFR 寄存器中的中断请求

标志。相应标志位请求的每个中断可以分别使能或禁用。中断分别组合为不同的、可以选择的优先级别。

有关中断的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第2.5节。

2. 硬件原理：（如图 3-7 所示）

按键 SB2 连接 CC2531 的管脚 P1_1。不按下按键 SB2 时，管脚 P1_1 为高电平 3.3V，即逻辑 1。按下按键 SB2 时，管脚 P1_1 由高电平 3.3V 变为低电平 0V，即由逻辑 1 变为逻辑 0。

CC2531 有 21 个可编程的 I/O 口引脚，P0、P1 口是完全的 8 位口，P2 口只有 5 个可使用的位（P2_4，P2_3，P2_2，P2_1，P2_0），全部 21 个数字 I/O 口引脚都具有响应外部中断的能力。如果需要响应外部设备，可对 I/O 口引脚产生中断，同时外部中断事件也能被用来唤醒休眠模式。

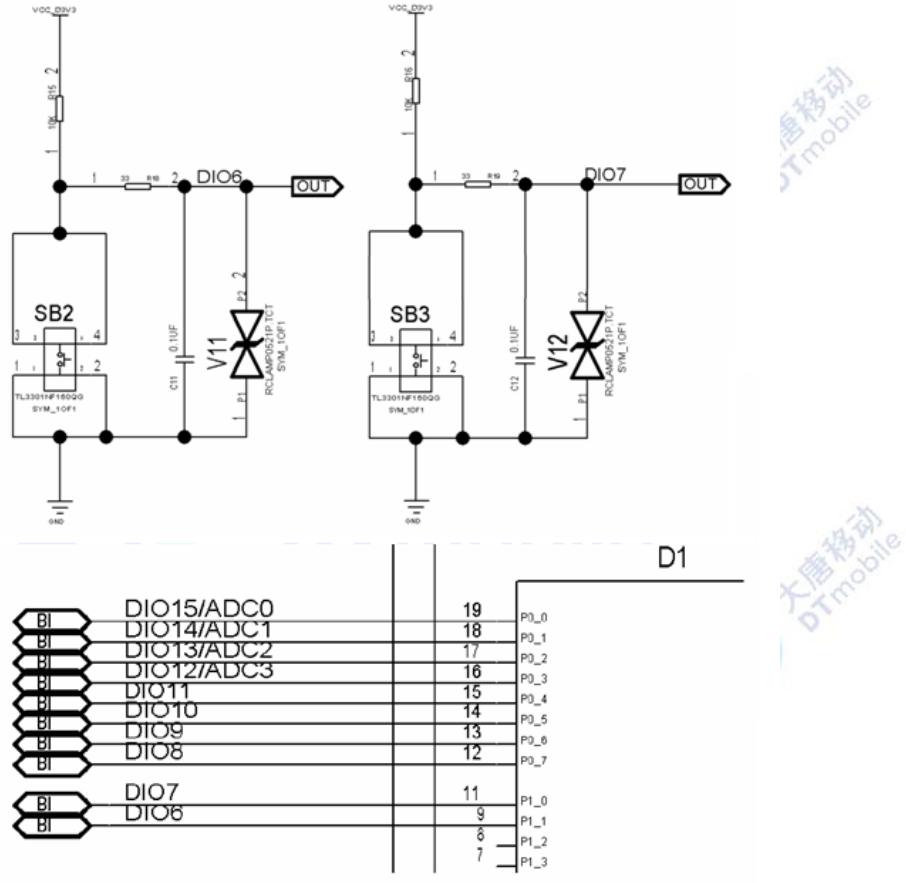


图3-7　　按键原理图

把 P1_1 配置为输入，并且使能 P1_1 的外部中断功能，将中断设置为下降沿触发，则按下按键时触发 P1_1 的外部中断。

SB3 的工作原理同 SB2。

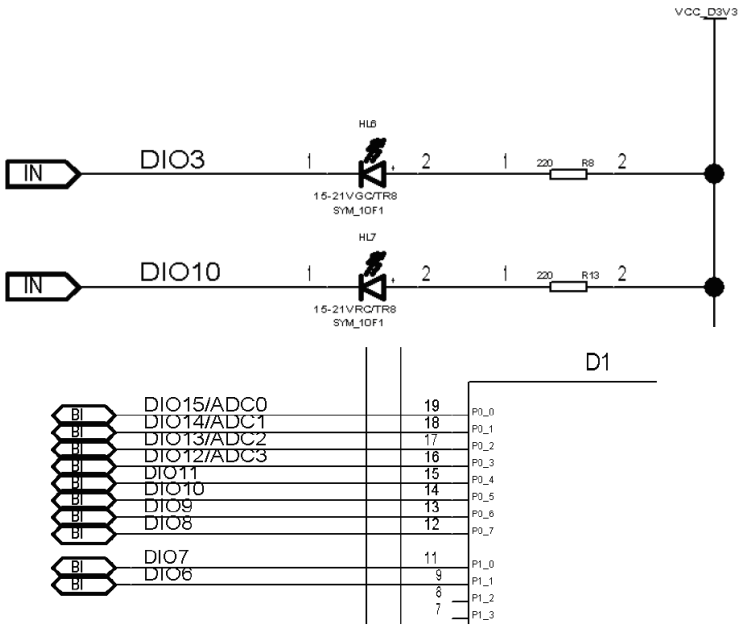


图3-8　　LED 原理图

图 3-8 为 LED 原理图，LED 灯 HL7 的阴极连接管脚 P0_5，当 P0_5 为 0 时，LED 灯 HL7 亮，当 P0_5 为 1 时，LED 灯 HL7 灭。

3. 实验相关寄存器：

实验中操作了的寄存器有P0，P2（实验一），P1SEL（实验二），P0DIR（实验一），P1DIR（实验三），P1INP（实验二），P1IEN，PICTL，P1IFG，IEN0，IEN2，等寄存器。

3.1　P1IEN (P1 口中断使能寄存器)：

位号	位名	复位值	可操作性	功能描述
7:0	P1_[7:0]IEN	0x00	R/W	P1_[7:0] ：中断使能 0：禁止中断；1：使能中断

P1IEN寄存器使能或禁止P1口对应位的中断功能。向对应位写入0，则禁止其中断功能，向对应位写入1，则使能其中断功能。复位值为0x00，即默认为禁止所有位的中断功能。

3.2　PICTL(中断控制寄存器)：

位号	位名	复位值	可操作性	功能描述
7	—	0	R0	未使用
6	PADSC	0	R/W	输出驱动能力选择： 0：最小驱动能力；1：最大驱动能力

5	P2IEN	0	R/W	P2（4-0）：中断使能位 0：关中断；1：开中断
4	P0IENH	0	R/W	P0（7-4）：中断使能位 0：关中断；1：开中断
3	P0IENL	0	R/W	P0（3-0）中断使能位 0：关中断；1：开中断
2	P2ICON	0	R/W	P2（4-0）中断配置 0：上升沿触发；1：下降沿触发
1	P1ICON	0	R/W	P1（7-0）中断配置 0：上升沿触发；1：下降沿触发
0	P0ICON	0	R/W	P0（7-0）中断配置 0：上升沿触发；1：下降沿触发

本实验使用的端口为 P1，因此只需设置 PICTL 的第 1 位 P1ICON 为 1，即下降沿触发。

3.3 P1IFG(P1 口中断标志寄存器)：

位号	位名	复位值	可操作性	功能描述
7:0	P1IF[7:0]	0x00	R/W0	P1[7:0]中断标志位，在中断发生情况下，相应位自动被置为 1。

3.4 IEN2（中断使能寄存器 2）：

位号	位名	复位值	可操作性	功能描述
7:6	——	00	R0	未使用，读出为 0
5	WDTIE	0	R/W	看门狗定时器中断使能 0：关中断；1：开中断
4	P1IE	0	R/W	P1 中断使能 0：关中断；1：开中断
3	UTX1IE	0	R/W	串口 1 发送中断使能 0：关中断；1：开中断
2	UTX0IE	0	R/W	串口 0 发送中断使能

				0：关中断；1：开中断
1	P2IE	0	R/W	P2 口中断使能 0：关中断；1：开中断
0	RFIE	0	R/W	射频中断使能 0：关中断；1：开中断

本实验使用的端口为 P1，因此只需设置 IEN2 的第 4 位 P1IE 为 1，即使能 P1 口中断。

4. 中断的使用方法

为了使能任一中断功能，应当采取下列步骤：

- 清除中断标志。
- 如果有，则设置 SFR 寄存器中对应的各中断使能位为 1。
- 设置寄存器 IEN0、IEN1 和 IEN2 中对应的中断使能位为 1。
- 设置 IEN0 中的 EA 位为 1 使能全局中断。
- 在该中断对应的向量地址上，运行该中断的服务程序。

5. 实验相关函数：

写在程序中的子函数及功能列写如下：

5.1 void Init_IO_AND_LED(void)；

函数原型：

```
void Init_IO_AND_LED(void)
{
    P2DIR |= 0x01;
    P0DIR |= 0x20;
    RLED = 1;
    YLED = 0;            //黄灯亮表示系统开始工作
    P1IEN = 0X03;    //开P1_0 ，P1_1中断掩码
    PICTL = 0X02;    //P1下降沿触发
    EA = 1;             //开总中断
    IEN2 |= 0X10;    // P1IE = 1;
    P1IFG |= 0x00;    //中断标志初始化位0
}
```

函数功能：将I/0 P0_5设置为输出，去控制红色LED灯，使能P1_0和P1_1中断，且配置为下降沿触发。

5.2 void P1_ISR(void)；

函数原型：

```
#pragma vector = P1INT_VECTOR
interrupt void P1_ISR(void)
{
    if(P1IFG>0) //按键中断
    {
        P1IFG = 0;
        RLED = !RLED;
    }
    P1IF = 0; //清中断标志
}
```

函数功能：在P1_0，P1_1触发中断的时候将红色LED灯的状态翻转。

6. main（）函数：

```
void main(void)
{
    Init_IO_AND_LED();
    while(1)
    {
    };
}
```

main（）函数功能是实现了使用两个按键来翻转红色LED灯的状态。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\ 9_External Interrupt）。

3. 观察实验现象。按下按键 K1 和 K2，观察是否能够翻转 LED 灯 HL6 的亮灭状态。

六、 拓展思考

1. CC2531有哪些中断源？
2. 本实验中的中断属于哪一类中断源？

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.9 实验九 定时器中断

一、 实验目的

1. 掌握定时器内部中断的使用方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

用定时器4来改变小灯的状态，T4每2000次中断小灯闪烁一轮，闪烁的时间长度为1000次中断所耗时间。

四、 实验原理

1.本实验是利用定时器 4 的中断功能完成内部中断来控制 LED 灯闪烁的实验。相关内容可参考实验六实验七。

2.实验相关寄存器：

实验中操作了的寄存器有P1（实验一），P1SEL（实验一），P0DIR（实验一），P1DIR（实验一），T4CTL（实验七），T4CCTL0（实验七），T4CC0（实验七），T4CCTL1（实

验七），T4CC1（实验七），IEN0（实验五），IEN1等寄存器。

2.1 IEN1(中断使能控制寄存器1)：

位号	位名	复位值	操作性	功能描述
7：6	——	00	可读0	保留
5	P0IE	0	可读/写	P0口中断使能 0：关中断；1：开中断
4	T4IE	0	可读/写	定时器4中断使能 0：关中断；1：开中断
3	T3IE	0	可读/写	定时器3中断使能 0：关中断；1：开中断
2	T2IE	0	可读/写	定时器2中断使能 0：关中断；1：开中断
1	T1IE	0	可读/写	定时器1中断使能 0：关中断；1：开中断
0	DMAIE	0	可读/写	DMA传输中断使能 0：关中断；1：开中断

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\8_Timer Interrupt）。

3. 观察实验现象。观察 LED 灯是否自动不停的改变状态。

六、 拓展思考

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.10 实验十 单片机串口通信实验

一、 实验目的

1. 掌握单片机串口发数的方法，测试串口功能。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 直连串口线

三、 实验内容

从CC2531上通过串口不断地发送字串“DTMobile”。实验使用CC2531的串口1，波特率为57600。

四、 实验原理

1. 串口部分电路：

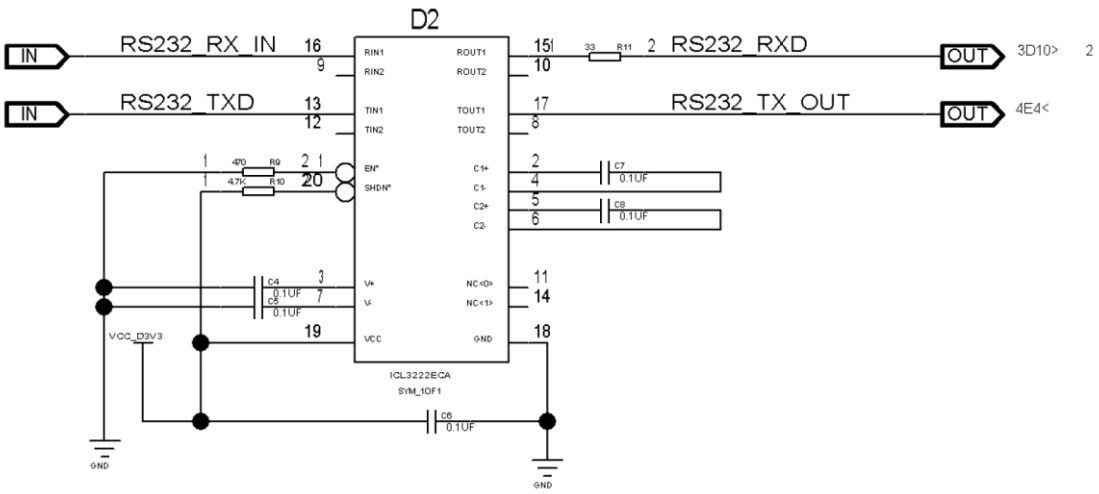


图3-9 串口所连处理器电路图

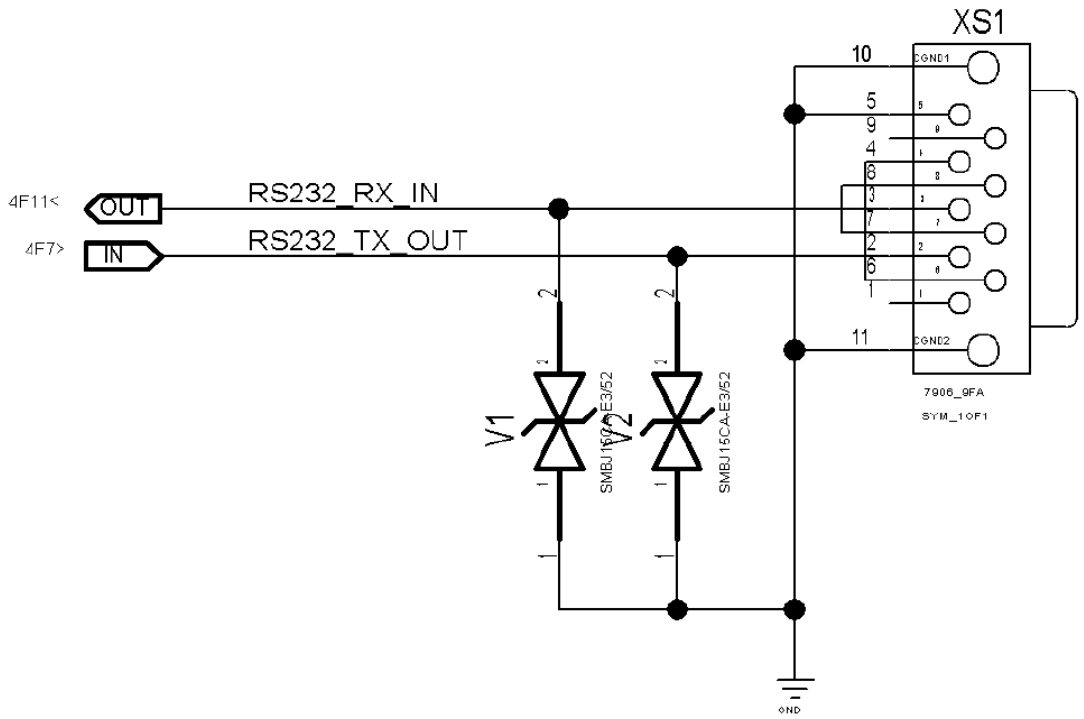


图3-10　串口部分电路图

上图3-9、3-10分别是串口所连处理器和串口部分电路图。处理器的RS232_TX_IN和RS232_TX_OUT两个管脚与串口相连，RS232_TXD和RS232_RXD两个管脚与CC2531单片机相连。CC2531单片机通过RS232_TXD和RS232_RXD两个管脚与处理器进行通讯，处理器接收到数据后，根据内部电路进行处理，再通过RS232_TX_IN和RS232_TX_OUT两个管脚发送到串口。串口通过网线与PC机相连，这样，数据就能直接发送到电脑。

2．CC2531串口使用方法：

CC2531包括2个串行通信接口，USART0与USART1，每个串口包括两个模式：UART（异步）模式，SPI（同步）模式，模式的选择由串口控制/状态寄存器的U0CSR.MODE决定。UART模式的操作具有下列特点：8位或者9位数据；奇校验、偶校验或者无奇偶校验；配置起始位和停止位电平；配置LSB或者MSB首先传送；独立收发中断；独立收发DMA触发；秋季哦啊眼和帧校验出错状态。本实验仅涉及UART模式。

有关UART的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第16节。

在使用串口时需要明确串口的任务是接收数据还是发送数据，或者两者都要使用。串口接收数据有两种方法：查询法和中断法。查询法就是要串口一直处于等待的状态，看串口上是不是有数据（主要是看URX0IF的值，一旦是1，表示串口上有数据并且串口上的数据已经接收完毕可以进行下一步的操作了），若数据接收完毕，就开始对接收的数据进行相应的操作。这种方法稳定性比较高。还有一种方法是中断法，这种方法是运用串口的中断服务子程序（ISR）来完成的。如果串口上有值的话，那么会调用那个中断向量，中断向量则把程序

指针指到相应的ISR中去。对接收到的数据的操作在ISR中进行，ISR完成之后程序指针会跳回中断前的地方继续进行刚才被中断的事情。串口发送数据只有查询法一种方法。

3. 实验相关寄存器：

实验中操作了的寄存器有P0，P2（实验一），P0DIR，P2DIR（实验一），P1DIR（实验一），CLKCONCMD，SLEEPSTA，SLEEPCMD，PERCFG，U0CSR，U0GCR，U0BAUD，U0BUF，IEN0（实验五）等寄存器。

3.1 CLKCONCMD（时钟控制寄存器）：

位号	位名	复位值	可操作性	功能描述
7	OSC32K	1	R/W	32kHz时钟源选择 0：32K 晶振；1：32K RC振荡
6	OSC	1	R/W	系统时钟源选择 0：32M晶振；1：16M RC振荡
5:3	TICKSPD[2:0]	001	R/W	定时器计数时钟分频（该时钟频不大于OSC决定频率） 000：32M； 001：16M； 010：8M 011：4M； 100 ：2M； 101：1M 110：0.5M； 111：0.25M
2:0	CLKSPD	001	R/W	时钟速度（不能高于通过OSC位设置的系统时钟设置，表示当前系统时钟频率） 000：32M； 001：16M； 010：8M 011：4M； 100 ：2M； 101：1M 110：0.5M； 111：0.25M

3.2 SLEEPSTA（睡眠模式控制状态寄存器）：

位号	位名	复位值	可操作性	功能描述
7	OSC32K_CALDIS	0	R	32 kHz RC 振荡器校准状态 SLEEPSTA.OSC32K_CALDIS 显示禁用 32 kHz RC 校准的当前状态。在芯片运行在32 kHz RC 振荡器之前，该位设置的值

				不等于SLEEPCMD.OSC32K_CALDIS。 这一设置可以在任何时间写入，但是在芯片运行在16MHz 高频RC 振荡器之前不起作用。
6:5	-	00	R	保留
4:3	RST[1:0]	XX	R	状态位，表示上一次复位的原因。如果有多个复位，寄存器只包括最新的事件。 00：上电复位；01：外部复位 10：看门狗复位；11 时钟丢失复位
2:1	--	00	R	保留
0	CLK32K	0	R	32kHz时钟信号（与系统时钟同步）

3.3 SLEEPCMD（睡眠模式控制寄存器）：

位号	名称	复位	R/W	描述
7	OSC32K_CALDIS	0	R/W	禁用32 kHz RC振荡器校准 0：使能32 kHz RC振荡器校准 1：禁用32 kHz RC振荡器校准 这个设置可以在任何时间写入，但是在芯片运行在16MHz高频RC振荡器之前不起作用。
6:3	----	0000	R0	保留
2	-	1	R/W	保留。总是写作1
1:0	MODE[1:0]	00	R/W	供电模式设置 00：主动/空闲模式 01：供电模式1 10：供电模式2 11：供电模式3

3.4 PERCFG（外设控制寄存器）：

位号	位名	复位值	可操作性	功能描述
7	—	0	R0	未用
6	T1CFG	0	R/W	T1 I/O位置选择 0：位置1；1：位置2
5	T3CFG	0	R/W	T3 I/O位置选择 0：位置1；1：位置2
4	T4CFG	0	R/W	T4 I/O位置选择 0：位置1；1：位置2
3：2	——	00	R0	未用
1	U1CFG	0	R/W	串口1位置选择 0：位置1；1：位置2
0	U0CFG	0	R/W	串口0位置选择 0：位置1；1：位置2

3.5 U0CSR（串口0控制＆状态寄存器）：

位号	位名	复位值	可操作性	功能描述
7	MODE	0	R/W	串口模式选择 0：SPI模式；1：UART模式
6	RE	0	R/W	接收使能 0：关闭接收；1：允许接收
5	SLAVE	0	R/W	SPI主从选择 0：SPI主；1：SPI从
4	FE	0	R/W0	串口帧错误状态 0：没有帧错误；1：出现帧错误
3	ERR	0	R/W0	串口校验结果 0：没有校验错误；1：字节校验出错
2	RX_BYTE	0	R/W0	接收状态

				0：没有接收到数据；1：接收到一字节数据
1	TX_BYTE	0	R/W	发送状态 0：没有发送 1：最后一次写入U0BUF的数据已经发送
0	ACTIVE	0	R	串口忙标志 0：串口闲；1：串口忙

3.6 U0GCR（串口0常规控制寄存器）：

位号	位名	复位值	可操作性	功能描述
7	CPOL	0	R/W	SPI时钟极性 0：低电平空闲；1：高电平空闲
6	CPHA	0	R/W	SPI时钟相位 0：由CPOL跳向非CPOL时采样，由非CPOL跳向CPOL时输出 1：由非CPOL跳向CPOL时采样，由CPOL跳向非CPOL时输出
5	ORDER	0	R/W	传输位序 0：低位在先；1：高位在先
4:0	BAUD_E[4:0]	0x00	R/W	波特率指数值，与BUAD_F决定比特率

3.7 U0BAUD(串口0波特率控制寄存器)：

位号	位名	复位值	可操作性	功能描述
7:0	BAUD_M[7:0]	0X00	R/W	波特率位数，与BAUD_E决定波特率

3.8 U0BUF（串口0收发缓冲器）：

位号	位名	复位值	可操作性	功能描述
7:0	DATA[7:0]	0X00	R/W	UARTO收发寄存器

4. 实验相关函数：

写在程序中的子函数及功能列写如下：

4.1 void initUARTtest(void)；

函数原型：

```
void initUARTtest(void)
{
    CLKCONCMD &= ~0x40;              //晶振
    while(!(SLEEPSTA & 0x40));       //等待晶振稳定
    CLKCONCMD &= ~0x47;              //TICHSPD128分频，CLKSPD不分频
    SLEEPCMD |= 0x04;                //关闭不用的RC振荡器

    PERCFG = 0x01;                   //串口0 位置2
    P1SEL = 0x30;          //P1用作串口(串口RX,TX硬件分别与连接P1_4，P1_5)

    U0CSR |= 0x80;                   //选择UART方式
    U0GCR |= 10;                     //baud_e
    U0BAUD |= 216;                   //波特率设为57600
    }
```

函数功能：初始化串口0，将I/O映射到P1口，P0优先作为串口0使用，UART工作方式，波特率为57600（波特率的数值设置请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第16.4）。使用晶振作为系统时钟源。

4.2 void UartTX_Send_String(char *Data,int len)；

函数原型：

```
void UartTX_Send_String(char *Data,int len)
{
  int j;
  for(j=0;j<len;j++)
  {
    U0DBUF = *Data++;
    while(UTX0IF == 0);
    UTX0IF = 0;
  }
```

```
}
```

函数功能：串口发字串，*Data 为发送缓存指针，len为发送字串的长度，只能是在初始化函数void initUARTtest(void)之后调用才有效。发送完毕后返回，无返回值。

5. main ( ) 函数：

```
void main(void)
{
    P2DIR |= 0x01;
    P0DIR |= 0x20;
    YLED = 0;          //黄灯亮表示系统开始工作
    RLED = 1;          //LED
    initUARTtest();
    while(1)
    {
        UartTX_Send_String(Txdata,strlen(Txdata)); //串口发送数据
        Delay(50000);                              //延时
        Delay(50000);                              //延时
        RLED = !RLED;
    }
}
```

main（）函数功能是实现了从CC2531上通过串口不断地发送字串“DTMobile”。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，并将直连串口线两端分别连接ZigBee模块和PC机。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\10_UART-Test）。打开串口调试软件，选择相应的串口，并设置好波特率。

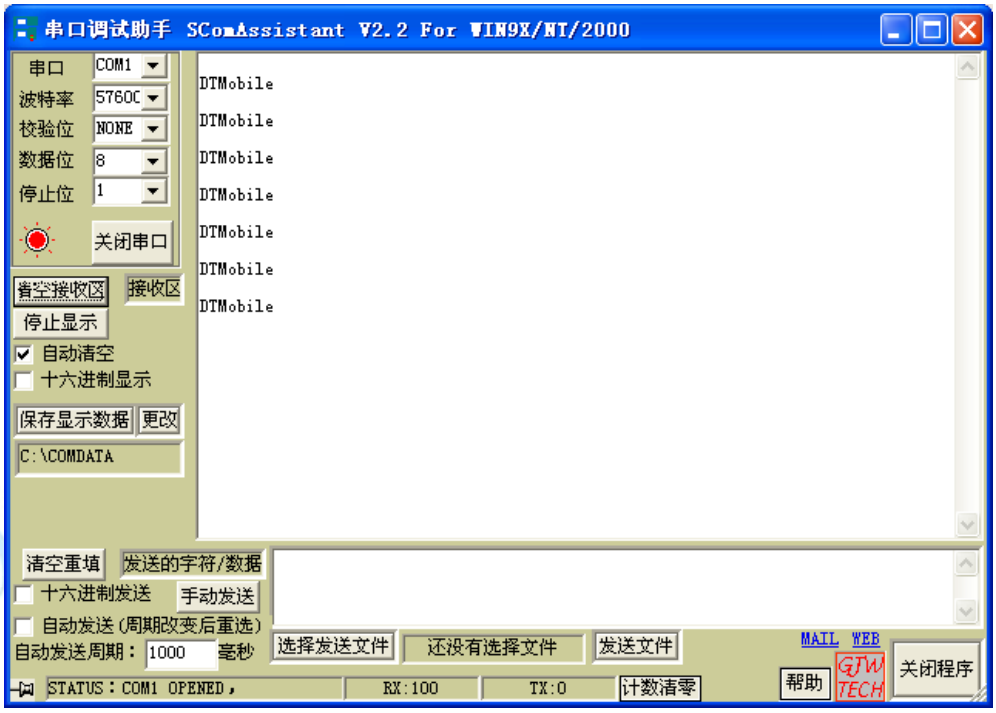


图3-11　　串口调试助手界面

3.　观察实验现象。观察 LED 灯的指示情况，以及串口调试软件是否逐行不停的显示“DTMobile”。

六、　拓展思考

串口线有哪些类型，为什么本实验使用直连串口线？请根据本实验原理图，结合PC机标准串口的线序，得出结论。

七、　参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.11　实验十一 在 PC 上通过串口控制 CC2531 的 LED 灯实验

一、　实验目的

1.　掌握在 PC 用串口控制 LED 的方法。

2.　了解并学习 CC2531 的相关寄存器的使用。

二、　实验设备

- ZigBee模块
- JTAG仿真器

- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 直连串口线

三、 实验内容

在PC上从串口向CC2531发数，即可控制LED灯的亮灭，控制数据的格式为“灯编号开丨关#”，红色LED编号为1，黄色LED编号为2，0是关灯，1是开灯。如打开红色LED的命令是“11#”。

四、 实验原理

1. 实验相关寄存器：

实验中操作了的寄存器有：P0，P2（实验一），P0DIR，P2DIR（实验一），P1DIR（实验二），P1SEL（实验二），CLKCONCMD（实验十），SLEEPSTA，SLEEPCMD（实验十），PERCFG（实验十），U0CSR（实验十），U0GCR（实验十），U0BAUD（实验十），U0BUF（实验十），IEN0（实验五）等寄存器。

2. 实验相关函数：

写在程序中的子函数及功能列写如下：

2.1 void UART0_ISR(void)；

函数原型：

```
#pragma vector = URX0_VECTOR
 __interrupt void UART0_ISR(void)
 {
     URX0IF = 0;                    //清中断标志
     temp = U0DBUF;
 }
```

函数功能：串口中断函数用来接收数据，一旦有数据从串口送到CC2531，则立即进入中断，进入中断后将接收的数据先存放到temp变量，然后在主程序中去处理接收到的数据。

3.main ( ) 函数：

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

```
    uchar ii;
    Init_LED_IO();
    initUARTtest();
    while(1)
    {
        if(RTflag == 1)                 //接收
        {
            if( temp != 0)
            {
                if((temp!='#')&&(datanumber<3))
                {                                       //‘#’被定义为结束字符
                                                        //最多能接收3个字符
                    Recdata[datanumber++] = temp;
                }
                else
                {
                    RTflag = 3;                         //进入改变小灯的程序
                }
                if(datanumber == 3)RTflag = 3;
                temp    = 0;
            }
        }
        if(RTflag == 3)
        {
            if(Recdata[0]=='1')
            {
                if(Recdata[1]=='0')       // 10#  关红色LED
                {
                    led1 = 1;
                    UartTX_Send_String("\nLed1 has been turn off!\n",25);
```

```
                    }
                    else
                    {
                        led1 = 0;                                   // 11#  开红色LED
                        UartTX_Send_String("\nLed1 has been turn on!\n",25);
                    }
                }
                if(Recdata[0]=='2')
                {
                    if(Recdata[1]=='0')         // 20#  关绿色LED
                    {
                        led2 = 1;
                        UartTX_Send_String("\nLed2 has been turn off!\n",25);
                    }
                    else
                    {
                        led2 = 0;                                   // 21#  开绿色LED
                        UartTX_Send_String("\nLed2 has been turn on!\n",25);
                    }
                }
                RTflag = 1;
                for(ii=0;ii<3;ii++)Recdata[ii]=' '; //清除刚才的命令
                datanumber = 0;                                     //指针归位
            }
        }//while
    }
```

main（）函数功能是实现了在PC上从串口向CC2531发数，即可控制LED灯的亮灭。

4. 实验流程图（如图3-12）：

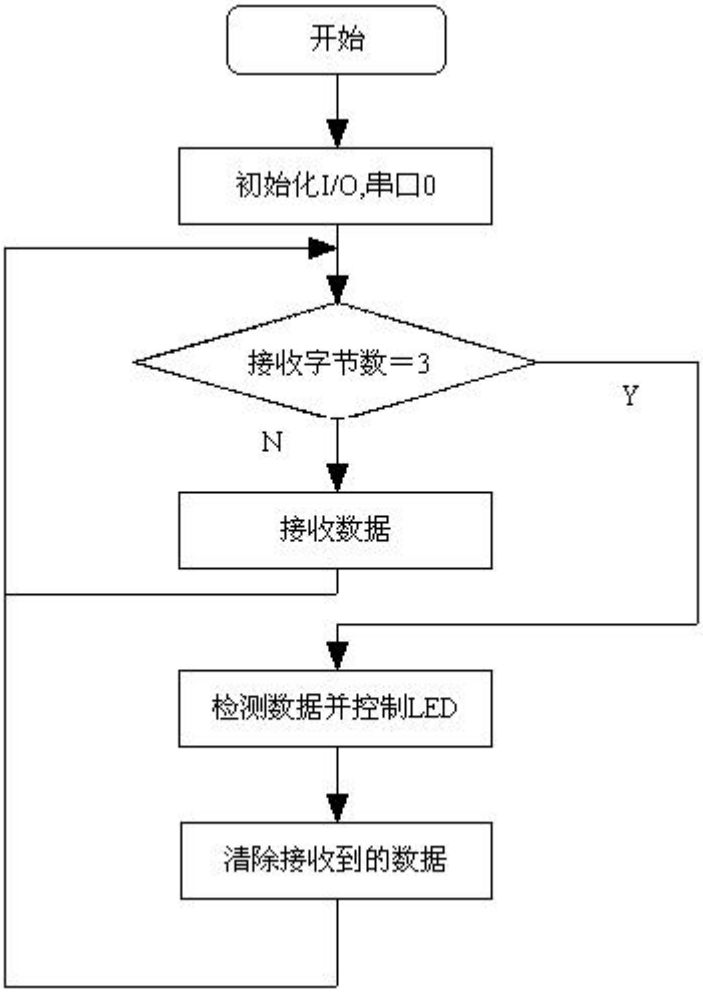


图3-12　实验十一流程图

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，并将直连串口线两端分别连接ZigBee模块和PC机。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\11_URAT-ctrl-led）。打开串口调试软件，选择相应的串口，并设置好波特率。在 PC 串口调试软件中输入实验数据（10#，11#，20#，21#），选择手动发送。

3. 观察实验现象。观察 ZigBee 模块小灯的变化情况，并与串口界面所返回的提示信息作对比，看是否吻合。

六、 拓展思考

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.12 实验十二 在 PC 上用串口收发数据实验

一、 实验目的

1. 掌握在 PC 用串口收发数的方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 直连串口线

三、 实验内容

在PC上从串口向CC2531发任意长度为30字节的字串，若长度不足30字节，则以“#“为字串末字节，CC2531在收到字节后会将这一字串从串口反向发向PC，用串口助手可以显示出来。

四、 实验原理

1. 实验相关寄存器：

实验中操作了的寄存器有：P0（实验一），P0DIR（实验一），P1DIR（实验二），P1SEL（实验二），CLKCON（实验十），SLEEP（实验十），PERCFG（实验十），U0CSR（实验十），U0GCR（实验十），U0BAUD（实验十），U0BUF（实验十），IEN0（实验五）等寄存器。

2. main（ ）函数：

```
void main(void)
{
    P2DIR |= 0x01;
    P0DIR |= 0x20;
```

```
    YLED = 1;
    RLED = 1;    //LED
    initUARTtest();
    stringlen = strlen((char *)Recdata);
    UartTX_Send_String(Recdata,stringlen);          //DTmobile
    while(1)
    {
        if(RTflag == 1)             //接收
        {
          led2=0;                   //接收状态指示
          if( temp != 0)
          {
              if((temp!='#')&&(datanumber<30))
              {                                     //’＃‘被定义为结束字符
                                                    //最多能接收30个字符
                Recdata[datanumber++] = temp;
              }
              else
              {
                RTflag = 3;                         //进入发送状态
              }
              if(datanumber == 30)RTflag = 3;
            temp    = 0;
          }
        }
        if(RTflag == 3)               //发送
        {
          led2 = 1;                    //关绿色LED
          led1 = 0;                    //发送状态指示
```

```
                U0CSR &= ~0x40;          //不能收数
                UartTX_Send_String(Recdata,datanumber);
                U0CSR |= 0x40;           //允许收数
                RTflag = 1;              //恢复到接收状态
                datanumber = 0;          //指针归0
                led1 = 1;                //关发送指示
            }
        }
    }
```

main（）函数功能是实现了在PC上用串口收发数据。

2. 实验流程图（如图3-13）：

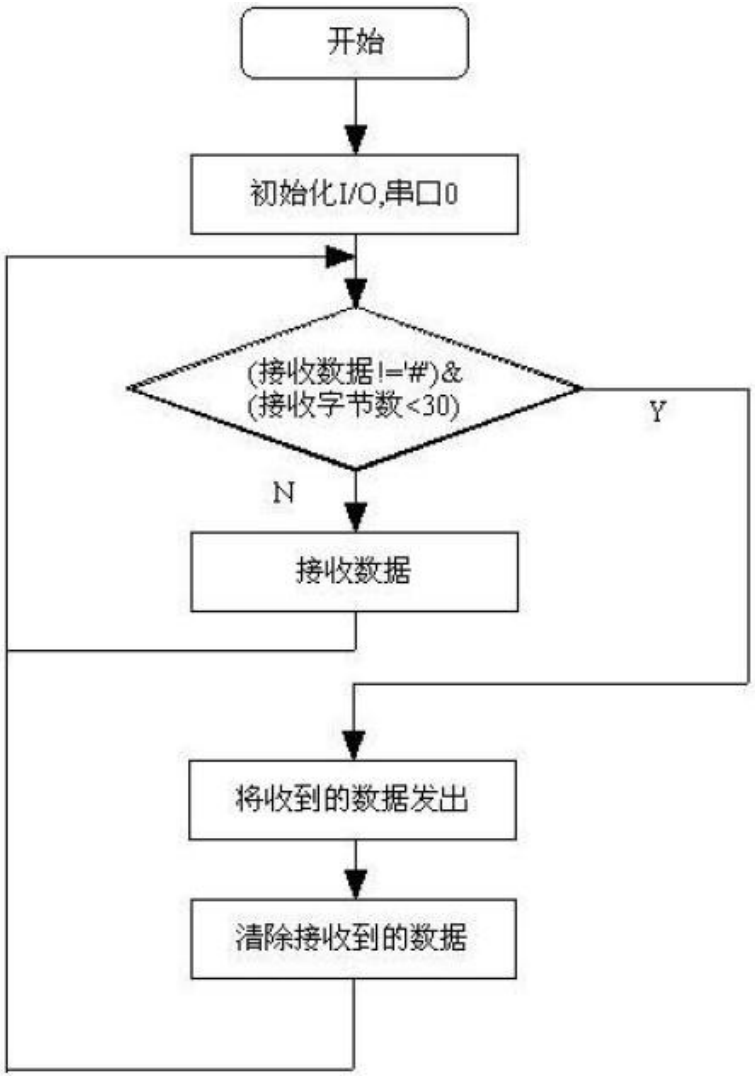


图3-13　实验十二流程图

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，并将直连串口线两端分别连接ZigBee模块和PC机。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\12_UART-R-S）。打开串口调试软件，选择相应的串口，并设置好波特率。在发送数据栏中输入字符串（字符串长度如果小于 30 则以'#'结束），选择手动发送，或者自动发送。

3. 观察实验现象。观察串口接收栏中收到的数据是否与发送的一致，以及 ZigBee 模块指示灯的亮灭情况

六、 拓展思考

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.13 实验十三 串口时钟 PC 显示实验

一、 实验目的

1. 掌握在 PC 显示串口时钟的方法
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 直连串口线

三、 实验内容

利用CC2531的定时器1产生1秒信号，通过串口通信在PC上每隔一秒显示时间。

四、 实验原理

1. 本实验是T1计数器与串口通信相结合的一个实验，用计数器的中断来模拟时钟，通过一个数组上各个元素的一系列的变化，生成一个所谓的时间，然后通过串口向PC机上发送。

2. 实验相关寄存器：

实验中操作了的寄存器有：P1（实验一），P0DIR（实验一），P1DIR（实验二），P1SEL（实验二），T1CTL（实验四），T1CCTL0，T1CC0H，T1CC0L，IEN0（实验五），IEN1（实验八），CLKCON（实验十），SLEEP（实验十），PERCFG（实验十），U0CSR（实验十），U0GCR（实验十），U0BAUD（实验十），U0BUF（实验十）等寄存器。

2.1 T1CCTL0(T1通道0捕获/比较寄存器)：

位号	位名	复位值	操作性	功能描述
7	CPSEL	0	R/W	T1 通道 0 捕捉设定 0：捕捉引脚输入；1：捕捉 RF 中断
6	IM	1	R/W	T1 通道 0 中断掩码 0：关中断；1：开中断
5:3	CMP[2:0]	000	R/W	T1 通道 0 比较输出模式选择，指定计数值过 T3CC0 时的发生事件 000：输出置 1（发生比较时） 001：输出清 0（发生比较时） 010：输出翻转 011：输出置 1（发生上比较时）输出清 0（计数值为 0 或 UP/DOWN 模式下 发生下比较） 100：输出清 0（发生上比较时）输出置 1（计数值为 0 或 UP/DOWN 模式下 发生下比较） 101：没用；110：没用；111：没用
2	MODE-	0	R/W	T1 通道 0 模式选择 0：捕获；1：比较
1:0	CAP[1:0]	00	R/W	T1 通道 0 捕获模式选择 00：没有捕获；01：上升沿捕获 10：下降沿捕获；11：边沿捕获

2.2 T1CC0H(T1通道0捕获值/比较值高字节寄存器)：

位号	位名	复位值	操作性	功能描述
7	T1CC0[15:8]	0X00	R/W	T1通道0捕获值/比较值高字节

2.3 T1CC0L(T1通道0捕获值/比较值低字节寄存器)：

位号	位名	复位值	操作性	功能描述
7	T1CC0[7:0]	0X00	R/W	T1通道0捕获值/比较值低字节

3. 实验相关函数：

写在程序中的子函数及功能列写如下：

3.1 void InitT1(void)；

函数原型：

```
void InitT1(void)
{
   T1CCTL0 = 0X44;
   T1CC0H = 0x04;
   T1CC0L = 0x00;
   T1CTL |= 0X02;
   IEN1 |= 0X02;
   IEN0 |= 0X80;      //开T1中断
}
```

函数功能：开T1中断，T1为比较计数模式。

3.2 void InitClock(void)；

函数原型：

```
void InitClock(void)
{
   CLKCON = 0X38;
   while(!(SLEEP&0X40));       //等晶振稳定
}
```

函数功能：设置系统时钟为晶振，同时将计数器时钟设为0.25M。晶振振荡稳定后退出函数。

3.3　void T1_ISR(void)；

函数原型：

```
#pragma vector = T1_VECTOR
 __interrupt void T1_ISR(void)
 {
        IRCON &= ~0x02;//清中断标志
        counter++;
        if(counter == 250)
        {
          counter = 0;
          timetemp = 1;          //一秒到
          led1 = ~led1;          // 调试指示用
        }
 }
```

函数功能：T1中断服务程序，每250次中断将timetemp置1，表示1秒时间到，同时改变LED的状态。

4. main（ ）函数：

```
void main(void)
 {
   InitClock();
   InitT1();
   InitUART0();
   InitIO();
   UartTX_Send_String(SendData,8);
   do{
     if(timetemp == 1)     //if0
     {
       if(time[2]<59)     //second    //if1
       {
         time[2]++;
```

```
            }
            else
            {
               time[2] = 0;
               if(time[1]<59) //minute    //if2
               {
                  time[1]++;
               }
               else
               {
                  time[1] = 0;
                  if(time[0]<23) //hour    //if3
                  {
                     time[0]++;
                  }
                  else
                  {
                     time[0] = 0;
                  }                        //end if3
               }                           //end if2
            }                              //end if1
         timetemp = 0;
         }//end if0
         if(temp != 0)
         {
            Recdata[number++]=temp;
            temp = 0;
         }
   /********上程序段用来计时，最小精度为秒**********************/
```

```
        if((Recdata[0] == '#')&&(number == 9))        //F0  为设时间的数据段首字节
        {
            time[2] = (Recdata[7]-48)*10+(Recdata[8]-48);
            if(time[2]>59)time[2]=0;
            time[1] = (Recdata[4]-48)*10+(Recdata[5]-48);
            if(time[1]>59)time[1]=0;
            time[0] = (Recdata[1]-48)*10+(Recdata[2]-48);
            if(time[0]>59)time[0]=0;
            YLED = !YLED;                                    //for test
            Recdata[0] = 0;
            number = 0;
        }
/************************以上程序段用来处理PC命令****************************/
        if(follow_second != time[2])
        {
                SendData[7] = (char)(time[2])%10+48;
                SendData[6] = (char)(time[2])/10+48;
                SendData[4] = (char)(time[1])%10+48;
                SendData[3] = (char)(time[1])/10+48;
                SendData[1] = (char)(time[0])%10+48;
                SendData[0] = (char)(time[0])/10+48;
/*************************以上程序将时间数据打包*****************************/
                UartTX_Send_String(SendData,10);
                follow_second = time[2];
        }//end if
    }while(1);//end
 }//end main()
```

main（）函数功能是实现了利用CC2531的T1定时器产生秒信号。

5. 实验流程图（如图3-14）：

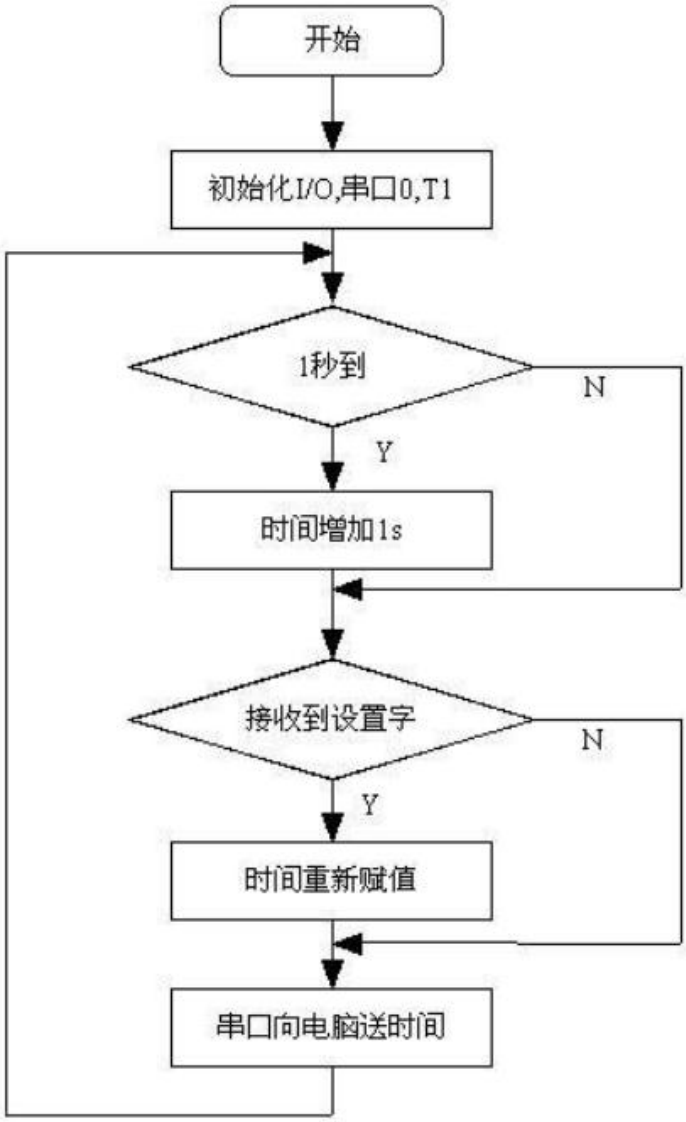


图3-14　　实验十二程序图

五、　实验步骤

1.　按要求连接好硬件，如实验一所示，并将串口线两端分别连接ZigBee模块和PC机。

2.　打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\13_UART-Clock）。打开串口调试软件，选择相应的串口，并设置好波特率。

3.　观察实验现象。观察串口调试软件接收到的数据，是否每隔一秒安装指定格式显示时间，起始时间为 00:00:00。

六、　拓展思考

七、　参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.14 实验十四 1/3AVDD 实验

一、 实验目的

1. 掌握将 1/3 电源电压转换为 AD 源的方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR集成开发环境
- 串口调试软件
- 直连串口线

三、 实验内容

将AD的转换源设为1/3电源电压，并将计算得到的电压值通过串口送至电脑。

四、 实验原理

1. 14位模/数转换器（ADC）：

CC2531的ADC支持多达14位的模拟数字转换，具有多达12位的ENOB（有效数字位）。它包括一个模拟多路转换器，具有多达8个各自可配置的通道；以及一个参考电压发生器。转换结果通过DMA写入存储器。还具有若干运行模式。

CC2531的ADC具有以下特征：

- ADC转换位可选，8~14位
- 8个独立可配置输入通道
- 参考电压发生器可作为内/外部单一参考电路，外部差分电路或AVDD_Soc
- 产生中断
- 转换完成触发DMA
- 温度传感输入

- 电池电压检测

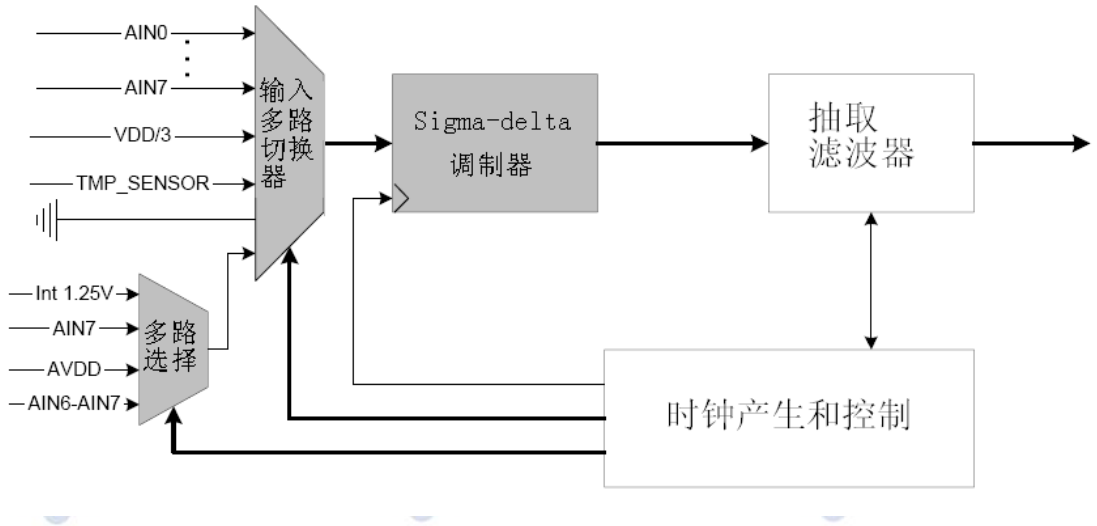


图3-15　ADC 框图

当使用ADC时，P0口必须配置成ADC输入作为8位ADC输入；把P0相应的引脚当做ADC输入时，寄存器ADCCFG相应的位设置为1，否则寄存器ADCCFG的各位初始值是0。

ADC完成顺序模/数转换以及把记过送至内存（使用DMA模式）而不需要CPU的干涉。

ADC寄存器包括ADCCFG（ADC输入配置寄存器），ADCL（ADCL数据低位），ADCH（ADCL数据高位），ADCCON1（ADCL控制寄存器1），ADCCON2（ADCL控制寄存器2），ADCCON3（ADCL控制寄存器3）。

有关ADC的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第12节。

2. ADC一般涉及到6个SFR：

ADCCON1	用于ADC通用控制，包括转换结束标志、ADC触发方式、随机数发生器
ADCCON2	用于连续ADC转换的配置（本实验不涉及连续ADC转换，故不使用此SFR）
ADCCON3	用于单次ADC转换的配置，包括选择参考电压、分辨率、转换源
ADCH[7:0]	ADC转换结果的高位，即ADC[13:6]
ADCL[7:2]	ADC转换结果的低位，即ADC[5:0]
ADCCFG	选择P0.0~P0.7作为ADC输入的AIN0~AIN7(本实验选择片内温度传感器作为转换源，不涉及AIN0~AIN7)

3. 本实验ADCCON3.ECH=1111，即选择1/3电源电压为AD的输入源。

电压计算公式:$V = \frac{ADC\ 值}{2^{13}} * 参考电压$ 。

ADC值：将AD转换后得到的ADCL，ADCH做处理，将ADCL（低6位）放在低字节，ADCH（高8位）放在高字节。将16位二进制位右移两位（最低两位没有用），即得到14位ADC值。其中，14位ADC数据结果中，第13位为符号位。

参考电压：可选内部参考电压或者外部参考电压。例如本实验选择管脚AVDD5电压作为参考电压，为3.3V。

4. 实验相关寄存器：

实验中操作了的寄存器有CLKCONCMD（实验十），SLEEPSTA（实验十），PERCFG（实验十），U0CSR（实验十），U0GCR（实验十），U0BAUD（实验十），IEN0（实验五），U0DUB（实验十），ADCCON1（实验十），ADCCON3（实验十），ADCH，ADCL等寄存器。

5. 实验相关参数：

写在程序中的子函数及功能列写如下：

5.1 void InitialAD(void)；

函数原型：

```
void InitialAD(void)
{
    ADCH &= 0X00;  //清EOC标志
    ADCCON3=0xbf; //单次转换，参考电压为电源电压，对1/3 AVDD进行A/D转换
                  //14位分辨率
    ADCCON1 = 0X30;    //停止A/D
    ADCCON1 |= 0X40;   //启动A/D
}
```

函数功能：将AD转换源设为电源电压，ADC结果分辨率设为14 位，AD模式为单次转换，启动ADC转换。

6. main ( ) 函数 ：

```
void main(void)
{
    char temp[2];
    float num;
```

```
        initUARTtest();                                          //初始化串口
        InitialAD();                                             //初始化ADC
        YLED = 0;
        RLED = 1;
        while(1)
        {
            if(ADCCON1>=0x80)
            {
                RLED = 1;               //转换完毕指示
                temp[1] = ADCL;
                temp[0] = ADCH;
                ADCCON1 |= 0x40;        //开始下一转换
                temp[1] = temp[1]>>2;
                temp[1] |= temp[0]<<6;
                temp[0] = temp[0]>>2; //数据处理
                temp[0] &= 0x3f;
                num = (temp[0]*256+temp[1])*3.3/8192;   //有一位符号位,取2^13;
                adcdata[1] = (char)(num)%10+48;
                adcdata[3] = (char)(num*10)%10+48;
                UartTX_Send_String(adcdata,6);    //串口送数
                Delay(30000);
                RLED = 0;                                //完成数据处理
                Delay(30000);
                }
            }
    }
```

本实验的main ( ) 函数对于转换得到的ADC值代入公式计算得到对应的电压值。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，并将串口线两端分别连接ZigBee模块和PC机。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\14_1-3AVDD）。打开串口调试软件，选择相应的串口，并设置好波特率。

3. 观察实验现象。观察串口调试软件接收到的电压值，如图 3-16 所示。

图3-16　串口调试助手界面

六、　拓展思考

CC2531的ADC输入源有哪几种？本实验中ADC输入源属于什么类型？

七、　参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.15 实验十五 AVDD 实验

一、　实验目的

1. 掌握将参考电压设为 AVDD 的方法。

2. 了解并学习 CC2531 的相关寄存器的使用。

二、　实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 直连串口线

三、 实验内容

将ADC的输入源设为AVDD，并将转换得到电压值通过串口发送至电脑。

四、 实验原理

1. 本实验与实验十四的区别仅在于对AD转换初始化的设置。本实验设置ADCCON3=0xbd。即参考电压选择模拟电源电压（3.3V），转化精度仍是14位不变。代入公式计算得到转换电压值。

2. 实验相关寄存器：

实验中操作了的寄存器有CLKCONCMD（实验十），SLEEP（实验十），PERCFG（实验十），U0CSR（实验十），U0GCR（实验十），U0BAUD（实验十），IEN0（实验五），U0DUB（实验十），ADCCON1（实验十三），ADCCON3（实验十三），ADCH（实验十三，ADCL（实验十三）等寄存器。

3. 实验相关函数：

写在程序中的子函数及功能列写如下：

3.1 void InitialAD(void)； 函数原型：

```
void InitialAD(void)
{
    ADCH &= 0X00;    //清EOC标志
    ADCCFG |= 0X80;
    ADCCON3=0xbd;   //单次转换,参考电压为电源电压，14位分辨率
   ADCCON1 = 0X30;     //停止A/D
    ADCCON1 |= 0X40;     //启动A/D
}
```

函数功能：将AD转换源设为电源电压， ADC结果分辨率设为14位，AD模式为单次转换，

启动ADC转换。

4. main ( ) 函数：

可参考实验十四的main()函数。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，并将串口线两端分别连接ZigBee模块和PC机。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\15_AVDD）。打开串口调试软件，选择相应的串口，并设置好波特率。

3. 观察实验现象。观察串口调试软件接收到的电压值。

六、 拓展思考

CC2531的ADC分辨率是多少？即最小能分辨的电压是多少？

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.16 实验十六 系统睡眠工作状态实验

一、 实验目的

1. 掌握最低功耗的测定方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

在红色LED灯闪烁10次以后进入低功耗模式PM3。

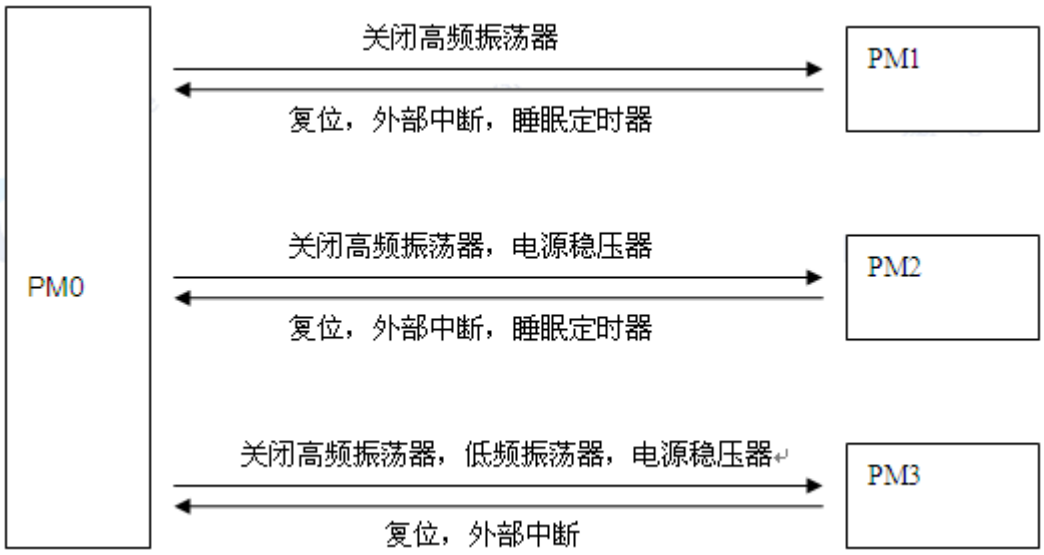
四、 实验原理

1. 4种功耗模式介绍：

CC2531一共有4种功耗模式，分别是PM0，PM1，PM2，PM3，越靠后被关闭的功能越多，功耗也越来越低，PM3功耗最低。

电源模式	高频振荡器	低频振荡器	电源稳压器（数字）
配置	A.无 B.32MHz晶体振荡器 C.16MHzRC振荡器	A.无 B.32.753kHzRC振荡器 C.32.768kHz晶体振荡器	
PM0	BC	B或C	开
PM1	A	B或C	开
PM2	A	B或C	关
PM3	A	A	关

它们之间的转换关系如下：



有关功耗模式的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC253x数据手册第4.1节。

2. 实验相关寄存器：

实验中操作了的寄存器有P0（实验一），P0DIR（实验一），P1DIR（实验二），P1SEL（实验二），CLKCONCMD（实验十），SLEEP（实验十），PCON等寄存器。

2.1 PCON（电源模式控制寄存器）：

位号	位名	复位值	可操作性	功能描述

7:1	——	0000 000	R/W	未使用，总是写0000 000。
0	IDLE	0	R0/W H0	供电模式控制。写1 到该位强制设备进入SLEEP.MODE设置的供电模式，这位读出来一直是0。

3. 重要的宏定义：

3.1 #define SET_POWER_MODE(mode)；

```
#define SET_POWER_MODE(mode)                                    \
    do {                                                        \
        SLEEPCMD &= ~0X03;                                      \
        if(mode == 0)           { SLEEPCMD &= ~0x03; }          \
        else if (mode == 3)   { SLEEPCMD |= 0x03;   }           \
        else { SLEEPCMD &= ~0x03; SLEEPCMD |= mode;   } \
        PCON |= 0x01;                                           \
        asm("NOP");                                             \
    }while (0)
```

函数功能：由SLEEPCMD设置CC2531功耗模式，选定后PCON写1立刻进入相应功耗模式。

4. main ( ) 函数：

```
void main()
{
    uchar count = 0;
    Initial();
    YLED = 0;          //开红色LED，系统工作指示
    Delay();           //延时
    while(1)
    {
        RLED = !RLED;
        if(count > 20)
        {
```

```
                SET_POWER_MODE(3);
            }//10次闪烁（一亮一灭循环20次）后进入睡眠状态
            count++;
            Delay();//延时函数无形参，只能通过改变系统时钟频率
                    //来改变小灯的闪烁频率
        }
    }
```

main（）函数功能是实现了红色LED灯闪烁10次后进入睡眠状态。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\ 16_S-sleep）。

3. 观察实验现象。按下复位键后，观察红色 LED 灯的变化情况。

六、拓展思考

怎样唤醒处于低功耗模式下的处理器？

七、参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.17 实验十七 系统唤醒实验

一、 实验目的

1. 掌握系统唤醒的方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

本次实验使能外部I/O中断唤醒CC2531，每次唤醒（按键控制）红色LED闪烁10次，然后进入低功耗模式，在进入PM3之前程序会将两个LED灯关闭。

四、 实验原理

1. 实验相关寄存器：

实验中操作了的寄存器有P0（实验一），P0DIR（实验一），P1DIR（实验二），P1INP（实验二），P1SEL（实验二），IEN0（实验五），P1IEN（实验九），P1CTL（实验九），IEN2（实验九），P1IFG（实验九），CLKCONCMD（实验十），SLEEP（实验十），P2INP等寄存器。

1.1 P2INP(P2输入模式寄存器)：

位号	位名	复位值	操作性	功能描述
7	PDUP2	0	可读/写	P2口上/下拉选择 0：上拉；1：下拉
6	PDUP1	0	可读/写	P1口上/下拉选择 0：上拉；1：下拉
5	PDUP0	0	可读/写	P0口上/下拉选择 0：上拉；1：下拉
4	MDP2-4	0	可读/写	P2-4口输入模式 0：上拉；1：下拉
3	MDP2-3	0	可读/写	P2-3口输入模式 0：上拉；1：下拉
2	MDP2-2	0	可读/写	P2-2口输入模式 0：上拉；1：下拉
1	MDP2-1	0	可读/写	P2-1口输入模式 0：上拉；1：下拉

2. 实验相关函数：

写在程序中的子函数及功能列写如下：

void Init_IO_AND_LED(void)；

函数原型：

```
void Init_IO_AND_LED(void)
{
    P0DIR = 0x03;
    RLED = 1;
    YLED = 1;
    P1INP &=~ 0x0C;   //有上拉、下拉
    P2INP &=~ 0x40;   //选择上拉
    P1IEN |= 0x0C;    //P1_1、P1_0
    PICTL |= 0x02;    //下降沿
    EA = 1;
    IEN2 |= 0x10;     //P1IE = 1;
    P1IFG |= 0x00;    //P1_1、P1_0
};
```

函数功能：置P1_0,P1_1为输出，打开P1口的中断，P1口下降沿触发中断。

void Set_PowerMode(uchar sel);

函数原型：

```
void Set_PowerMode(uchar sel)
{
    uchar i,j;
    i = sel;
    if(sel<4)
    {
        SLEEP &= 0xfc;
        SLEEP |= i;
        for(j=0;j<4;j++);
        PCON = 0x01;
    }
    else
```

```
    {
        PCON = 0x00;
    }
}
```

函数功能：改变系统的电源功耗模式，使系统进入sel指定的电源模式下。

2.3 void P1_ISR(void)；函数原型：

```
#pragma vector = P1INT_VECTOR
 __interrupt void P1_ISR(void)
 {
        if(P1IFG>0)
        {
          P1IFG = 0;
        }
        P1IF = 0;
        PowerMode(7);
 }
```

函数功能：外部I/O中断唤醒CC2531。

3．main ( ) 函数：

```
void main()
{
    uchar count = 0;
    Init_IO_AND_LED();
    YLED = 0 ;          //开黄色LED，系统工作指示
    Delay();            //延时
    while(1)
    {
        YLED = 0;
        RLED = !RLED;
        count++;
        if(count >= 20)
```

```
            {
                YLED = 1;
                count = 0;
                PowerMode(3);//10次闪烁后进入睡眠状态
                RLED = 1;
            }
            Delay();//延时函数无形参，只能通过改变系统时钟频率
                    //来改变小灯的闪烁频率
        }
    }
```

main（）函数功能是使能外部I/O中断唤醒CC2531，每次唤醒红色LED闪烁10次，然后进入低功耗模式。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。
2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\ 17_S-wake-up）。
3. 观察实验现象。按下按键 K1 或者 K2，观察 LED 灯的变化情况。

六、 拓展思考

在实际应用中，怎样在低功耗模式与激活模式之间切换？

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.18 实验十八 睡眠定时器的使用实验

一、 实验目的

1. 掌握睡眠定时器的使用方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

在红色LED灯快速闪烁5次后进入睡眠状态PM2，在PM2下睡眠定时器SLEEP TIMER（ST）仍然可以正常工作，从0x000000到0xffffff反复计数，当ST计数超过写入ST[2－0]的0x000f00时，系统由中断唤醒，小灯闪烁5次后进入PM2，这样周而复始的唤醒工作然后睡眠。

四、 实验原理

1. 睡眠定时器介绍：

CC2531的睡眠定时器（SLEEP TIMER）是运行于32.768kHz的24位定时器，当系统运行在除了PM3之外的所有的电源模式下，睡眠定时器都会不间断运行。睡眠定时器使用的寄存器有：ST2，ST1，ST0。下面会做简单介绍。使用睡眠定时器来唤醒系统的流程为：开睡眠定时器中断→设置睡眠定时器的定时间隔→设置电源模式。本实验是通过CC2531内部的睡眠定时器，设定睡眠时间（3s），在到达时间后自动唤醒。

2. 实验相关寄存器：

实验中操作了的寄存器有P0（实验一），P0DIR（实验一），P1DIR（实验二），P1SEL（如自动闪烁实验所示），IEN0（实验五），CLKCONCMD（实验十四），SLEEP（实验十四）ST2，ST1，ST0，等寄存器。

2.1 ST2（睡眠定时器2）：

位号	位名	复位值	操作性	功能描述
7:0	ST2[7:0]	0X00	R/W	睡眠定时器计数/比较值[23-16]位。读出为ST计数值，写入为比较值。读寄存器应先读ST0，写寄存器就后写STO。

2.2 ST1（睡眠定时器1）：

位号	位名	复位值	操作性	功能描述
7:0	ST1[7:0]	0X00	R/W	睡眠定时器计数/比较值[15-8]位。读出为ST计数值，写入为比较值。读寄存器应先读ST0，

				写寄存器就后写STO。

2.3 ST0（睡眠定时器0）：

位号	位名	复位值	操作性	功能描述
7:0	ST0[7:0]	0X00	R/W	睡眠定时器计数/比较值[7-0]位。读出为ST计数值，写入为比较值。读寄存器应先读ST0，写寄存器就后写STO。

它们的功能包括读、写两方面。读：用于读取当前定时器的计数值，读的顺序必须遵循：读ST0→读ST1→读ST2。写：用于设置定时器的比较值（当定时器的计数值=比较值时，产生中断），写的顺序必须遵循：写ST2→写ST1→写ST0。

有关Sleep Timer配置的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC2531数据手册第11节。

3. 实验相关函数：

写在程序中的子函数及功能列写如下：

void Init_SLEEP_TIMER(void)；

函数原型：

```
void Init_SLEEP_TIMER(void)
{
   ST2 = 0X00;
   ST1 = 0X0f;
   ST0 = 0X0F;
   EA = 1;  //开中断
   STIE = 1;
   STIF = 0;
}
```

函数功能：打开睡眠定时器SLEEP TIMER(ST)中断，设置ST的中断发生时间为计数值达到0x000f00时。

void LedGlint(void);

函数原型：

```
void LedGlint(void)
{
```

```
    uchar jj=10;
    while(jj--)
    {
      RLED = !RLED;
      Delay(10000);
    }
}
```

函数功能：让LED闪烁5次，无返回值。

3.3 void ST_ISR(void)；函数原型：

```
#pragma vector = ST_VECTOR
 __interrupt void ST_ISR(void)
 {
    STIF = 0;
    LEDBLINK = 1;
    return;
 }
```

函数功能：睡眠定时器中断服务程序，清中断标志，无其他操作。

void Set_ST_Period(uint sec)；

函数原型：

```
void Set_ST_Period(uint sec)
{
    UINT32 sleepTimer = 0;   //用于接收睡眠定时器的当前计数值
    sleepTimer |= ST0;
    sleepTimer |= (UINT32)ST1 <<   8;
    sleepTimer |= (UINT32)ST2 << 16;
    sleepTimer += ((UINT32)sec * (UINT32)32768); //加上所需要的定时时长
    ST2 = (UINT8)(sleepTimer >> 16); //设置睡眠定时器的比较值
    ST1 = (UINT8)(sleepTimer >> 8);
    ST0 = (UINT8) sleepTimer;
}
```

函数功能：设置睡眠时间。在sec秒以后由ST唤醒CC2531，在调用这个函数之前必须先调用Init_SLEEP_TIMER（），否则不能唤醒CC2531。通常在这个函数以后会出现SET_POWER_MODE(2)语句。因为定时器是工作在32.768kHz之下，所以定时器每加1，需耗时1/32768s，所以加32768，就需要1s。

4. 重要的宏定义：

4.2 选择系统工作时钟源并关闭不用的时钟源：

```
#define SET_MAIN_CLOCK_SOURCE(source) \
do {                                                    \
        if(source) {                                    \
          CLKCONCMD |= 0x40;   /*RC*/                   \
          while(!(SLEEPSTA&0X20));   /*待稳*/           \
          SLEEPCMD |= 0x04;            /*关掉不用的*/    \
        }                                               \
        else {                                          \
          SLEEPCMD &= ~0x04;   /*全开*/                 \
          while(!(SLEEPSTA&0X40));/*待稳*/              \
          asm("NOP");                                   \
          CLKCONCMD &= ~0x47; /*晶振*/                  \
          SLEEPCMD |= 0x04;    /*关掉不用的*/           \
        }                                               \
    }while (0)
```

函数功能：选择主时钟，关闭不用的时钟。

4.3 选择系统低速时钟源：

```
#define SET_LOW_CLOCK(source)                                   \
do{                                                             \
     (source==RC)?(CLKCONCMD |= 0X80):(CLKCONCMD &= ~0X80); \
    }while(0)
```

函数功能：选择低速时钟。

5. main ( ) 函数：

```
void main(void)
```

```
{
  Initial();
      SET_MAIN_CLOCK_SOURCE(CRYSTAL);
      SET_LOW_CLOCK(CRYSTAL);
      Init_SLEEP_TIMER();
      LedGlint();
      Set_ST_Period(3);
      while(1)
      {
          if(LEDBLINK)
          {
                LedGlint();
                Set_ST_Period(3);
                YLED = !YLED;
                LEDBLINK = 0;
          }
        Delay(100);
      }
}
```

main（）函数功能是实现了在小灯快速闪烁5次后进入睡眠状态PM2，在PM2下睡眠定时器SLEEP TIMER（ST）仍然可以正常工作，从0x000000到0xffffff反复计数，当ST计数超过写入ST[2－0]的0x000f00时，系统由中断唤醒，小灯闪烁5次后进入PM2，这样周而复始的唤醒工作然后睡眠。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\ 18_Sleep-Timer）。

3. 观察实验现象。观察 LED 灯闪烁情况。

六、 拓展思考

睡眠定时器在什么情况下使用？

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.19 实验十九 看门狗定时器实验

一、 实验目的

1. 学会初始化看门狗，了解看门狗的作用。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

学习使用看门狗。

四、 实验原理

1. 看门狗（Watch Dog）

看门狗，是专门用来监测单片机程序运行状态的电路结构，其基本原理是：启动看门狗定时器后，它就会从0开始计数，若程序在规定的时间间隔内没有及时对其清零，看门狗定时器就会复位系统。如图3-17所示：

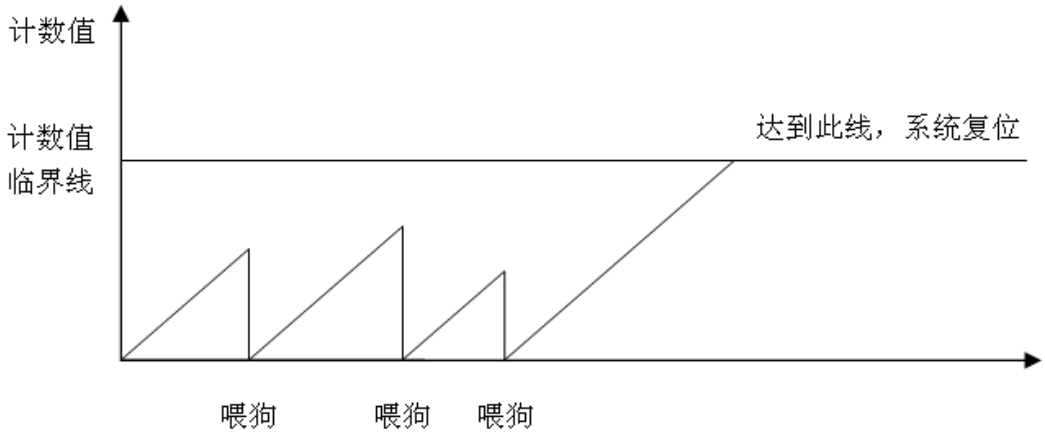


图3-17　看门狗工作原理图

看门狗的使用可以总结为：选择模式→选择定时器间隔→放狗→喂狗。看门狗定时器有两种模式，即“看门狗模式”和“定时器模式”。在定时器模式下，它就相当于普通的定时器，达到定时劲歌会产生中断；在看门狗模式下，当达到定时间隔时，不会产生这个那段，取而代之的是向系统发送一个复位信号。本实验中，通过WDCTL.MODE=0来选择为看门狗模式。

有关看门狗定时器的详细内容，请参考阅读\E-Box300\05-芯片数据手册\CC2531下的CC2531数据手册第15节。

2.　实验相关寄存器：

实验中操作了的寄存器有P0，P2（实验一），P0DIR，P2DIR（实验一），P1DIR（实验二），P1SEL（实验二），　WDTCL，CLKCONCMD（实验十），SLEEP（实验十）等寄存器。

2.1 WDCTL（看门狗定时器控制寄存器）：

位号	位名	复位值	操作性	功能描述
7:4	CLR[3:0]	0000	R/W	看门狗复位，先写0xa再写0x5复位看门狗，两次写入不超过0.5个看门狗周期，读出为0000
3	EN	0	R/W	看门狗定时器使能位，在定时器模式下写0停止计数，在看门狗模式下写0无效 0：停止计数；1：启动看门狗/开始计数
2	MODE	0	R/W	看门狗定时器模式 0：看门狗模式；1：定时器模式

1:0	INT[1:0]	00	R/W	看门狗时间间隔选择 00：1秒；01：0.25秒；10：15.625毫秒 11：1.9毫秒（以32.768K时钟计算）

3. 实验相关函数：

写在程序中的子函数及功能列写如下：

void Init_Watchdog(void);看门狗初始化函数

函数原型：

```
void Init_Watchdog(void)
{
    WDCTL = 0x00; //时间间隔一秒，看门狗模式
    WDCTL |= 0x08; //启动看门狗
}
```

函数功能：以看门狗模式启动看门狗定时器，看门狗复位时隔1秒。

3.2 void Init_Clock(void);函数原型：

```
void Init_Clock(void)
{
    CLKCONCMD = 0X00;
}
```

函数功能：将系统时钟设为晶振，低速时钟设为晶振，程序对时钟要求不高，不用等待晶振稳定。

4. main ( ) 函数：

```
void main(void)
{
    Init_Clock();
    Init_IO();
    Init_Watchdog();
    YLED=0;
    Delay();
    RLED=0;
```

```
    while(1)
        {
        }
}
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\19_Watch-Dog）。

3. 实验现象。观察 LED 灯闪烁情况。

六、 拓展思考

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

3.1.20 实验二十 喂狗实验

一、 实验目的

1. 掌握看门狗复位的方法。
2. 了解并学习 CC2531 的相关寄存器的使用。

二、 实验设备

- ZigBee模块
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境

三、 实验内容

本实验着重学习复位看门狗，复位看门狗后小灯不会闪烁。

四、 实验原理

1. 实验相关寄存器：

实验中操作了的寄存器有P0，P2（实验一），P0DIR，P2DIR（实验一），P1DIR（实验二），P1SEL（实验二）， WDTCL，CLKCONCMD（实验十），SLEEP（实验十）等寄存器。

2. 实验相关函数：

写在程序中的子函数及功能列写如下：

void FeedDog(void)；喂狗函数

函数原型：

```
void FeedDog(void)
{
    WDCTL = 0xa0;
    WDCTL = 0x50;
}
```

函数功能：复位看门狗，必须在看门狗时间间隔内调用本函数复位看门狗，否则系统会被强制复位，此时调用本函数已无意义。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示。

2. 打开 IAR 集成开发环境，建立工程和文件，写好代码后，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\01-WSN\01-基础实验代码\ 21_Feet-Dog）。

3. 观察实验现象。观察 LED 灯是否一直亮着。

六、 拓展思考

在实际应用中应该怎样喂狗以保证系统在正常情况下不复位？

七、 参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的 CC253x 数据手册。

3.2 物联网网关基础实验

3.2.1 实验二十一 shell 编程

一、 实验目的

1. 了解什么是shell。

2. 掌握shell 编程。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编程实现将输入的字符串输出。

四、 实验原理

1. Shell简介

操作系统与外部最主要的接口就叫做shell。shell是操作系统最外面的一层。shell管理你与操作系统之间的交互：等待你输入，向操作系统解释你的输入，并且处理各种各样的操作系统的输出结果。

用户登录或运行终端类比程序时，实际进入了Shell。那么，Shell是什么呢？确切一点说，Shell就是一个命令行解释器，它的作用就是遵循一定的语法将输入的命令加以解释并传给系统。它为用户提供了一个向Linux发送请求以便运行程序的接口系统级程序，用户可以用Shell来启动、挂起、停止甚至是编写一些程序。

Shell本身是一个用C语言编写的程序，它是用户使用Linux的桥梁。Shell既是一种命令语言，又是一种程序设计语言。作为命令语言，它互动式地解释和执行用户输入的命令；作为程序设计语言，它定义了各种变量和参数，并提供了许多在高阶语言中才具有的控制结构，包括循环和分支。它虽然不是Linux系统内核的一部分，但它调用了系统内核的大部分功能来执行程序、创建文档并以并行的方式协调各个程序的运行。因此，对于用户来说，Shell是最重要的实用程序，深入了解和熟练掌握Shell的特性极其使用方法，是用好Linux系统的关键。可以说，Shell使用的熟练程度反映了用户对Linux使用的熟练程度。

当用户使用Linux时是通过命令来完成所需工作的。一个命令就是用户和Shell之间对话的一个基本单位，它是由多个字符组成并以换行结束的字串。

2. 交互与非交互式shell

为shell与用户进行交互。这种模式也是大多数用户非常熟悉的：登录、执行一些命令、签退。当你签退后，shell 也终止了。

shell也可以运行在另外一种模式：非交互式模式。在这种模式下，shell不与你进行交互，而是读取存放在文件中的命令,并且执行它们。当它读到文件的结尾，shell也就终止了。

3. Shell类型

在UNIX中主要有两大类shell：

3.1 Bourne shell (包括sh, ksh, and bash)

Bourne shell (sh)

Korn shell (ksh)

Bourne Again shell (bash)

POSIX shell (sh)

3.2 C shell (包括 csh and tcsh)

C shell (csh)

TENEX/TOPS C shell (tcsh)

Bourne Shell

最初的UNIX shell是由Stephen R. Bourne 于20世纪70年代中期在新泽西的AT&T贝尔实验室编写的，这就是Bourne shell。Bourne shell是一个交换式的命令解释器和命令编程语言。Bourne shell可以运行为login shell或者login shell的子shell(subshell)。只有login命令可以调用Bourne shell作为一个login shell。此时，shell先读取/etc/profile文件和$HOME/.profile文件。/etc/profile文件为所有的用户定制环境,$HOME/.profile文件为本用户定制环境。最后，shell会等待读取你的输入。

C Shell

Bill Joy 于20世纪80年代早期，在Berkeley的加利福尼亚大学开发了C shell。它主要是为了让用户更容易的使用交互式功能，并把ALGOL 风格的语法结构变成了C 语言风格。它新增了命令历史、别名、文件名替换、作业控制等功能。

Korn Shell

有很长一段时间，只有两类shell供人们选择，Bourne shell用来编程，C shell用来交互。为了改变这种状况，AT&T的bell实验室David Korn开发了Korn shell。ksh结合了所有的C shell的交互式特性，并融入了Bourne shell的语法。因此，Korn shell广受用户的欢迎。它还新增了数学计算、进程协作(coprocess)、行内编辑(inline editing)等功能。Korn Shell是一个交互式的命令解释器和命令编程语言。它符合POSIX——一个操作系统的国际标准。POSIX不是一个操作系统，而是一个目标在于应用程序的移植性的标准——在源程序一级跨越多种平台。

Bourne Again Shell (bash)

bash是GNU计划的一部分，用来替代Bourne shell。它用于基于GNU的系统如Linux。大多数的Linux(Red Hat, Slackware, Caldera)都以bash作为缺省的shell，并且运行shell时，其实调用的是bash。

POSIX Shell

POSIX shell是Korn shell的一个变种。当前提供POSIX shell的最大卖主是

Hewlett-Packard。在HP-UX 11.0 ， POSIX shell就是/bin/sh，而bsh是/usr/old/bin/sh。

各主要操作系统下缺省的shell:

AIX下是Korn Shell.

Solaris和FreeBSD 缺省的是Bourne shell.

HP-UX缺省的是POSIX shell.

Linux是Bourne Again shell.

4. 什么是脚本

shell也可以运行在另外一种模式：非交互式模式。在这种模式下，shell不与你进行交互，而是读取存放在文件中的命令，并且执行它们。当它读到文件的结尾，shell也就终止了。脚本是批处理文件的延伸，是一种纯文本保存的程序，一般来说的计算机脚本程序是确定的一系列控制计算机进行运算操作动作的组合，在其中可以实现一定的逻辑分支等。

脚本程序相对一般程序开发来说比较接近自然语言，可以不经编译而是解释执行，利于快速开发或一些轻量的控制。

本质上，shell script是命令行命令简单的组合到一个文件里面。

5. Shell的特点

- 用户与Linux 的接口
- 命令解释器
- 支持多用户
- 支持复杂的编程语言
- Shell 有很多种，如：csh,tcsh,pdksh,ash,sash,zsh,bash 等
- Linux 的缺省Shell 为bash(Bourne Again Shell)

6. 创建脚本文件遵循的步骤

- 使用编辑器加载文件
- 确认脚本文件的第一行是：#! /bin/bash
- 保存脚本文件并，退出编辑器
- 使用“chmod u+x 脚本文件名”，标注脚本文件的可执行属性
- 使用“./脚本文件”，执行脚本

编写代码

```
#! /bin/sh
    echo –n "please input your name:“
    read name
    echo "thanks,“
    echo $name
```

五、 实验步骤

运行：

Shell 文件不用编译可以直接运行

1. PC机上运行

```
root@DT_EBox6410:/usr/test/shell# ./dircmp
please input your name:DT_EBox6410
thanks,
Hello DT_EBox6410
root@DT_EBox6410:/usr/test/shell#
```

2. 开发板上运行

```
[root@DT_EBox6410 /tmp]# chmod +x dircmp
[root@DT_EBox6410 /tmp]# ./dircmp
please input your name: DT_EBox6410
thanks.
Hello DT_EBox6410
[root@DT_EBox6410 /tmp]# _
```

六、 拓展思考

七、 参考阅读

3.2.2 实验二十二 文件 IO 实验

一、 实验目的

1. 熟悉linux下的文件I/O相关的操作。
2. 熟练运用open、close、write、read函数的操作。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编写实验代码，要求从系统文件/etc/initab文件中读取全部数据，然后写入本地某文件中。其中，本地某文件由程序参数指定。

四、 实验原理

数据库系统是基于文件系统的，其性能和设备读写的机制有密切的关系。和数据库性能密切相关的文件 I/O 操作的三个操作：

- open 打开文件
- write 写文件
- fdatasync flush 操作（将文件缓存刷到磁盘上）。

1. Open 操作

```
open("test.file",O_WRONLY|O_APPDENT|O_SYNC))
```

系统调用 Open 会为该进程一个文件描述符 fd。这里使用了 O_WRONLY|O_APPDENT|O_SYNC 打开文件:

1.1 O_WRONLY 表示我们以"写"的方式打开，告诉内核我们需要向文件中写入数据；

1.2 O_APPDENT 告诉内核以"追加"的方式写文件；

1.3 O_DSYNC 告诉内核，当向文件写入数据的时候，只有当数据写到了磁盘时，写入操作才算完成（write 才返回成功）。

1.4 和 O_DSYNC 同类的文件标志，还有 O_SYNC,O_RSYNC，O_DIRECT。

- O_SYNC 比 O_DSYNC 更严格，不仅要求数据已经写到了磁盘，而且对应的数据文件的属性（例如文件长度等）也需要更新完成才算 write 操作成功。可见 O_SYNC 较之 O_DSYNC 要多做一些操作。
- O_RSYNC 表示文件读取时，该文件的 OS cache 必须已经全部 flush 到磁盘了；
- 如果使用 O_DIRECT 打开文件，则读/写操作都会跳过 OS cache，直接在 device（disk）上读/写。因为没有了 OS cache，所以会 O_DIRECT 降低文件的顺序读写的效率。

2. Write 操作

```
write(fd,buf,num)
```

在使用 open 打开文件获得文件描述符之后，我们就可以调用 write 函数来写入数据了，write 会根据前面的 open 参数不同，而表现不同。

3. Flush 阶段

```
fdatasync(fd) == -1
```

write 操作后，我们还调用了 fdatasync 来确保文件数据 flush 到了 disk 上。fdatasync 返回成功后，那么可以认为数据已经写到了磁盘上。像这样的 flush 的函数还有 fsync、sync。

- Fsync 和 fdatasync 的区别等同于 O_SYNC 和 O_DSYNC 的区别。
- Sync 函数表示将文件在 OS cache 中的数据排入写队列，并不确认是否真的写磁盘了，所以 sync 并不可以靠。

编写代码：

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main(int argc ,char **argv)
{
    printf("%s\n",argv[1]);
    int fdt,fds;
    char buf[20];
    int num=0;
    if((fds=open("/etc/inittab",O_RDONLY))<0)
    {
        printf("open fail\n");
```

```
        }
        if((fdt=open(argv[1],O_CREAT|O_TRUNC|O_RDWR))<0)
        {
            printf("open fail\n");
            return 1;
        }
        while(1)
        {
            if((num=read(fds,buf,20))<0)
            {
                printf("read fail\n");
                return 1;
            }
            if(write(fdt,buf,num)<0)
            {
                printf("write fail\n");
                return 1;
            }
            if(num!=20)
            break;
        }
        close(fds);
        close(fdt);
        return 0;
    }
```

五、 实验步骤

1. 编译运行

在pc机上使用 arm-linux-gcc 工具对源文件进行交叉编译。$arm-linux-gcc –o hello hello.c

在实验板运行（备注：需要将PC机上源文件目录挂载到开发板上）$./hello

2. 运行结果

```
[root@(none) /]./hello
Hello World
```

六、 拓展思考

七、 参考阅读

3.2.3 实验二十三 多线程实验

一、 实验目的

1. 掌握 Linux 中线程的创建及使用。
2. 熟悉 Linux 中的多线程编程的相关函数。
3. 掌握 Linux 中线程同步互斥的方法。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编写实验代码，要求创建两个子线程，线程pthread1不断地改变一个全局变量中的数据，线程pthread2从这个全局变量读取数据并输出到终端。两线程之间使用互斥锁。

四、 实验原理

1. Linux 线程的定义

线程是在共享内存空间中并发的多道执行路径，它们共享一个进程的资源，如文件描述和信号处理。在两个普通进程(非线程)间进行切换时，内核准备从一个进程的上下文切换到另一个进程的上下文要花费很大的开销。

这里上下文切换的主要任务是保存老进程CPU 状态，并加载新进程的保存状态，用新进程的内存映像替换老进程的内存映像。线程允许进程在几个正在运行的任务之间进行切换，而不必执行前面提到的完整的上下文。

2. Linux 线程实现方法

目前线程有用户态线程和核心态线程两种方法实现。

2.1 用户态线程

用户态线程是一个精细的软件工具，允许多线程的程序运行时不需要特定的内核支持。如果一个进程中的某一个线程调用了一个阻塞的系统调用，则该进程就会被阻塞，该进程中的其它所有线程也同时被阻塞。因此，Unix 使用了异步I/O 机制。这种机制主要的缺点在于，在一个进程中的多个线程调度中无法发挥多处理器的优势（如上述的阻塞情况）。

用户态线程优点如下：

- 某些线程操作的系统消耗大大减少。比如，对属于同一个进程的线程之间进行调度切换时，不需要调用系统调用，因此将减少额外的消耗，一个进程往往可以启动上千个线程。
- 用户态线程的实现方式可以被定制或修改，以适应特殊应用的要求。它对于多媒体实时过程等尤其有用。另外，用户态线程可以比核心态线程实现方法默认情况支持更多的线程。

2.2 核心态线程

核心态线程的实现方法允许不同进程中的线程按照同一相对优先调度方法进行调度，这样有利于发挥多处理器的并发优势。

目前，线程主要的实现方法是用户态线程。有几个研究项目已经实现了一些核心态线程的形式，其中比较著名的是MACH 分布式操作系统。

通过允许用户代码对内核线程调度的参与，该系统将用户态和核心态两种线程实现方法的优点结合了起来。通过提供这样一个两级调度机制，内核在保留了对处理器时间分配控制的同时，也使一个进程可以充分利用多处理器的优势。

编写代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<errno.h>
pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER;
int var=0;
void pthread1(void *arg);
void pthread2(void *arg);
int main()
{
    pthread_t id1,id2;
```

```
    int ret;
    pthread_mutex_init(&mutex,NULL);
    ret=pthread_create(&id1,NULL,(void *)pthread1,NULL);
    if(ret!=0)
        perror("Failed to create a thread0\n");
    ret=pthread_create(&id2,NULL,(void *)pthread2,NULL);
    if(ret!=0)
        perror("Failed to create a thread1\n");
    pthread_join(id1,NULL);
    pthread_join(id2,NULL);
    exit(0);
}
void pthread1(void *arg)
{
    while(var<5)
    {
        if(pthread_mutex_lock(&mutex)!=0)
        {
            perror("Thread1 lock failure\n");
        }
        else
            printf("Thread1 lock the variable\n");
        sleep(1);
        var=var+1;
        if(pthread_mutex_unlock(&mutex)!=0)
        {
            perror("Thread1 unlock failure\n");
        }
        else
            printf("Thread1 unlock the variable\n");
```

```
            sleep(1);
        }
    }
    void pthread2(void *arg)
    {
        int ret;
        while(var<5)
        {
            ret=pthread_mutex_trylock(&mutex);
            if(ret==EBUSY)
                printf("The variable is locked by pthread1\n");
            else
            {
                if(ret!=0)
                {
                    perror("Thread2 lock failure\n");
                    exit(1);
                }
                else
                printf("Thread2 got lock\n");
                printf("vat = %d \n",var);
                if(pthread_mutex_unlock(&mutex)!=0)
                    perror("Thread2 unlock failure\n");
                else
                    printf("pthred2 unlock the variable\n");
            }
            sleep(1);
        }
    }
```

五、 实验步骤

1. 编译运行

在 pc 机上使用 arm-linux-gcc 工具对源文件进行交叉编译。$ arm-linux-gcc –o pthread pthread.c -lpthrread

在实验板运行（备注：需要将PC机上源文件目录挂载到开发板上）$./pthread

2. 运行结果：

```
[root@(none) /] ./pthread
Thread1 lock the variable
The variable is locked by pthread1
Thread1 unlock the variable
Thread2 got lock
vat = 1
pthred2 unlock the variable
Thread1 lock the variable
The variable is locked by pthread1
Thread1 unlock the variable
Thread2 got lock
vat = 2
pthred2 unlock the variable
Thread1 lock the variable
The variable is locked by pthread1
Thread1 unlock the variable
Thread2 got lock
vat = 3
……
```

六、 拓展思考

七、 参考阅读

3.2.4 实验二十四 进程间通讯

一、 实验目的

1. 了解进程间通信原理。
2. 掌握进程间通信方式。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编程实现信号量和共享内存。

四、 实验原理

1. 基本介绍

Linux 下的进程通信手段基本上是从Unix 平台上的进程通信手段继承而来的。而对Unix 发展做出重大贡献的两大主力AT&T 的贝尔实验室及BSD（加州大学伯克利分校的伯克利软件发布中心）在进程间通信方面的侧重点有所不同。前者对Unix 早期的进程间通信手段进行了系统的改进和扩充，形成了“system V IPC”，通信进程局限在单个计算机内；后者则跳过了该限制，形成了基于套接口（Socket）的进程间通信机制。Linux 则把两者继承了下来，如图3-18示：

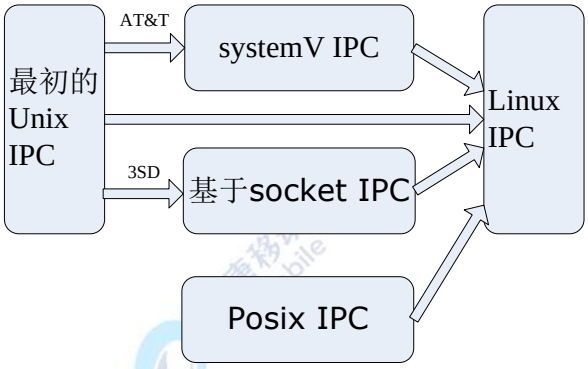


图3-18 linux 所继承进程间通信

其中，最初 Unix IPC 包括：管道、FIFO、信号；System V IPC 包括：System V 消息队列、System V 信号灯、System V 共享内存区；Posix IPC 包括： Posix 消息队列、Posix 信号灯、Posix 共享内存区。有两点需要简单说明一下：

- 由于 Unix 版本的多样性，电子电气工程协会（IEEE）开发了一个独立的 Unix 标准，这个新的 ANSI Unix 标准被称为计算机环境的可移植性操作系统界面（PSOIX）。现有大部分 Unix 和流行版本都是遵循 POSIX 标准的，而Linux 从一开始就遵循 POSIX 标准；

- BSD 并不是没有涉足单机内的进程间通信（Socket本身就可以用于单机内的进程间通信）。事实上，很多Unix版本的单机IPC留有BSD的痕迹，如4.4BSD 支持的匿名内存映射、4.3+BSD 对可靠信号语义的实现等等。

图3-16给出了Linux 所支持的各种IPC 手段，在本文接下来的讨论中，为了避免概念上的混淆，在尽可能少提及Unix的各个版本的情况下，所有问题的讨论最终都会归结到Linux 环境下的进程间通信上来。并且，对于Linux所支持通信手段的不同实现版本（如对于共享内存来说，有Posix共享内存区以及System V共享内存区两个实现版本），将主要介绍Posix API。

2. Linux下进程间通信的几种主要手段简介：

- 管道（Pipe）及有名管道（named pipe）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

- 信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；Linux 除了支持Unix 早期信号语义函数sigal 外，还支持语义符合Posix.1 标准的信号函数sigaction（实际上，该函数是基于BSD 的，BSD 为了实现可靠信号机制，又能够统一对外接口，用sigaction函数重新实现了signal 函数）；

- 报文（Message）队列（消息队列）：消息队列是消息的链接表，包括Posix 消息队列system V 消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

- 共享内存：使得多个进程可以访问同一块内存空间，是最快的可用IPC 形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

- 信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。

- 套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix 系统的BSD 分支开发出来的，但现在一般可以移植到其它类Unix 系统上：Linux 和System V 的变种都支持套接字。

基于上述几种方式，我们主要讨论共享内存和信号量。

3. 共享内存

共享内存是Linux 系统中最底层的通讯机制，也是最快速的通信机制。两个不同进程A、B共享内存的意思是，同一块物理内存被映射到进程A、B 各自的进程地址空间。进程A 可以即时看到进程B 对共享内存中数据的更新，反之亦然。由于多个进程共享同一块内存区域，必然需要某种同步机制，互斥锁和信号量都可以。

使用共享内存有两种方法：映射/dev/mem 设备和内存映像文件。比较而言，内存映像文件比较安全，使用映射/dev/mem 设备可能引起系统崩溃，但内存映像文件会带来文件系统的额外开销。我们主要讨论映射/dev/mem 设备实现共享内现存的方法。

每个共享内存都有一个与其相对应的结构shmid_ds，该结构定义如struct shmid_ds

```
struct shmid_ds
{
    struct ipc_perm shm_perm; //对应于该共享内存的ipc_perm 结构指针
    int shm_segsz; //以字节表示的共享内存区域的大小
    ushort shm_lkcnt; //共享内存区域被锁定的时间数
    pid_t shm_cpid; // 创建该共享内存的进程ID
    pid_t shm_lpid; //最后一次调用shmop 函数的进程ID
    ulong shm_nattch; //当前使用该共享内存区域的进程数
    time_t shm_atime; //最近一次附加操作的时间
    time_t shm_dtime; //最近一次分离操作的时间
    time_t shm_ctime; //最近一次改变的时间
};
```

3.1 共享内存的创建与打开

```
#include <sys/type .h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key , int size ,int flag);
```

参数说明：

key 表示所创建或打开的共享内存的键

size 表示共享内存区域的大小，指再创建一个新的共享内存时生效。

flag 表示调用函数的操作类型，也可用于设置共享内存的访问权限，两者通过逻辑或表示。

shmget()不仅可以创建一个新的共享内存，也可以用于打开一个已存在的共享内存。

调用该函数的作用由参数key 和flag 决定，相应约定如下：

- 当key 为IPC_PRIVATE 时，创建一个新的共享内存，此时参数flag 的取值对函数的操作不起任何作用。

- 当key 不为IPC_PRIVATE 时，且flag 设置了IPC_CREAT 位，而没有设置IPC_EXCL位，则执行操作有key 取值决定。如果key 为内核中某个已存在的共享内存的键，则执行打开这个键的操作；反之则执行创建共享内存的操作。

- 当key 不为IPC_PRIVATE 时，且flag 同时设置了IPC_CREAT 位和IPC_EXCL 位，则只执行创建共享内存的操作。参数key 的取值应与内核中已存在的任何共享内存的键值都不相同，否则函数调用失败。

此函数调用成功时，返回值为共享内存的引用标示符；调用失败时，返回-1。当调用该函数创建一个共享内存时，此共享内存的shmid_ds 结构被初始化。Ipc_perm 中的各个域被设置为相应值，shm_lpid、shm_nattch、shm_atime 和shm_dtime 被设置为0，shm_ctime 设置为当前时间。

3.2 附加

当一个共享内存创建或被打开后，某个进程如果要使用该共享内存则必须将此内存区域附加到他的地址空间，相关函数如下：

```
#include <sys/type .h>
#include <sys/ipc.h>
#include <sys/shm.h>
void *shmat (int shmid , void * addr , int flag)
```

参数说明：

Shmid 表示要附加的共享内存段的引用表示符

Flag 表示shmat 函数的操作方式，如果flag 设置了SHM_RDONLY 位，该内存区域被设置为只读，否则设置为可读写。

Addr 和flag 共同决定共享内存区域要放附加到的地址值，相应约定如下：

如果addr 为0，系统将自动查找进程地址空间，将共享内存区域附加到第一块有效内存区域上，此时flag 无效。

如过addr 不为0，而flag 未设置SHM_RND 位，则共享内存附加到由addr 指定的地指处。

如过addr 不为0 ， 而flag 设置SHM_RND 位， 则共享内存附加到由addr_(addrmodSHMLBA)指定的地址处。

此函数调用成功时，返回共享内存区域的指针；调用失败时，返回值为-1.

3.3 分离

当一个进程对共享内存区域的访问完成后，可以调用shmdt 函数使共享内存区域与该进程的地址空间分离，相关函数如下：

```
#include <sys/type .h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt (void *addr);
```

此函数仅用于将共享内存区域与进程的地址空间分离，并不删除共享内存本身。参数addr为要分离的共享内存区域的指针，是调用shmat 函数的返回值。调用成功时，返回值为0；失败时返回-1.

3.4 共享内存的控制

对共享内存区域的具体控制操作是通过函数shmctl 来实现的，说明如下：

```
#include <sys/type .h>
#include <sys/ipc.h>
#include <sys/shm.h>
Int shmctl (int shmid , int cmd , shmid_ds *buf);
```

参数说明：

Shmid 为共享内存的引用标示符

Buf 是指向shmid_ds 结构体的指针。

Cmd 表示调用该函数希望执行的操作，其取值和对应操作如下：

SHM_LOCK：将共享内存区域上锁，只能由超级用户执行

IPC_RMID：用于删除共享内存。执行该命令时，共享内存的引用标示符立刻被删除，则该共享内存不能在被其他进程所附加。但共享内存的真正删除要到所有附加了该共享内存的进程结束或断开于该共享内存的连接时才执行。

IPC_SET：按参数buf 指向的结构中的值设置该共享内存对应的shmid_ds 结构。只有有效用户ID 和共享内存的所有者ID 或创建者ID 相同的用户进程，以及超级用户进程可移植性这一操作。

IPC_STAT：用于取得该共享内存的shmid_ds 结构，保存于buf 指向的缓冲区。

SHM_UNLOCK：将上锁的共享内存区域释放，只能由超级用户执行。

4. 信号量

信号量是一种用于多个进程访问共享资源进行控制的机制。共享资源通常可分为两大类：一类是互斥共享资源，即任一时刻只允许一个进程访问该资源；一类是同步共享资源，同一是可允许多个进程访问该资源。

信号量是一个计数器的值，它可以被几个进程作为一个集合义原子的方式执行。信号量

的计数器控制着对资源的访问控制，信号量提供了两个主要的操作来处理计数器的值：

资源的使用者在使用资源之前等待信号量。如果信号量的值为0，则继续等待，如果大于0，则将信号量值减1，使用者开始使用资源。

资源的使用者在资源使用完毕后通知信号量。使用者通知信号量不再使用资源了，信号量的值加1，检查等待信号量的使用者的序列，以确定是否有其他的使用者在等待之中。

实际上，SYSV 子系统提供的信号量机制要复杂的多。不能单独定义一个信号量，而只能定义一个信号量集，其中包括一组信号量，统一信号量集中的信号量使用同一引用ID，这样设置是为了多个资源或同步操作的需要。每个信号量集都有一个与其相对应的结构，其中记录了信号量集的各种信息，该结构定义如下：

```
struct semid_ds
{
    struct ipc_perm sem_perm; //对应于该信号量集的ipc_perm 结构指针
    struct sem *sem_base; //指向信号量集中第一个信号量的sem 结构
    ushort sem_nsems; //信号量集种信号量的个数
    time_t sem_otime; //最近一次调用semop 函数的时间
    time_t sem_ctime; //最近一次改变该信号量集的时间
}
```

sem 结构记录了一个信号量的信息，其定义如下：

```
struct sem
{
    ushort semval; //信号量的值
    pid_t smepid; //最近一次访问资源的进程ID
    ushort semncnt; //等待可利用资源出现的进程数
    ushort semzcnt; //等待全部资源可被独占的进程数
}
```

4.1 信号量集的创建与打开

在程序使用信号量之前，必须首先创建信号量。

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semget(key_t key,int nsems,int semflg);
```

函数semget 用于创建一个新的信号量集或打开一个已存在的信号量集。其作用由参数key 和flag 决定，相应约定与shmget 类似。

参数说明：

key：要创建或要打开的信号量集的键。

nsems：要创建或者要访问的信号量集中信号量的数目。此参数只在创建一个新的信号量集时有效。

semflg：指定不同的选项和权限位的标志。可以为IPC_CREATE,IPC_EXCL.

该函数调用成功时，返回值为信号量的引用标示符；调用失败时，返回-1，同时error被设置值。当调用该函数创建一个信号量集时，他相应的semid_ds 结构被初始化。ipc_perm 中各个量被设置为相应值，sem_nsems 被设置为nsems 所表示的值，sem_otime 被设置为0，sem_ctime 被设置为当前时间。

Error=:

EACCESS(没有权限)

EEXIST(信号量集已经存在，无法创建)

EIDRM(信号量集已经删除)

ENOENT(信号量集不存在，同时没有使用IPC_CREAT)

ENOMEM(没有足够的内存创建新的信号量集)

ENOSPC(超出限制)

4.2 对信号量的操作

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid,struct sembuf semoparray[ ],unsigned nsops);
```

参数说明：

- semid：信号量集的引用ID.
- semoparray：是一个sembuf 结构数组，sembuf 结构用于指定用于semop 函数所做操作，数组semoparray 元素的个数由参数nops 指出。

sembuf 结构的定义和设定的操作如下：

```
struct sembuf
{
```

```
    ushort sem_num;
    short sem_op;
    short sem_flag;
}
```

sem_num 指要操作的信号量

sem_flag 为操作标记，于此函数相关的有IPC_NOWAIT和SEM_UNDO。

sem_op用于表示所要执行的操作，相应取值和含义定义如下：

sem_op>0：表示进程对资源使用完毕，交回该资源，此时信号量集的semid_ds结构的sem_base.semval将加上sem_op的值。若此时设置了SEM_UNDO位，则信号量的调整值将减去sem_op的绝对值。

sem_op=0：表示进程要等待，直至sem_base.semval变为0.

sem_op<0：表示进程希望使用资源。此时将比较sem_base.semval和sem_op的绝对值的大小，如果sem_base.semval大于等于sem_op的绝对值，说明资源足够分配给此进程，则sem_base.semval减去sem_op的绝对值。若此时设置了SEM_UNDO 位， 则信号量的调整值将加上sem_op 的绝对值。如果sem_base.semval 小于sem_op 的绝对值， 表示资源不足。若设置了IPC_NOWAIT 位， 则函数出错返回， 否则semid_ds 结构中的sem_base.semncnt 加1 ，进程等待直至 sem_base.semval 大于等于sem_op的绝对值sem_base.semval大于等于sem_op的绝对值活该信号量被删除。

semoparray是一个数组，其中每个元素表示一个操作，由于此函数是一个原子操作，一旦执行就将执行数组中的所有操作。

4.3 信号量的控制

对信号量的控制是通过函数semctl 来实现的

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl( int semid, int semnum, int cmd, union semun arg )
```

参数说明：

semid 为信号量集的引用标示符

semnum 指明某个特定信号量

cmd 表示调用该函数希望执行的操作

arg 是semun的联合，该联合定义如下：

```
union semun
{
    int val ;
    struct semid_ds *buf ;
    ushort array ;
};
```

此联合中各个量的使用情况与参数cmd设置有关，各个量的具体含义将和参数cmd的各种取值一起介绍如下：

GETALL：获得semid所表示的信号量集中信号量的个数，并将该值存在无符号短整数arg.array。

GETNCNT：获得semid所表示的信号量集中等待给定信号量锁的进程数目，即semid_ds结构中sem.semncnt的值。

GETPID： 获得semid所表示的信号量集中最后一个使用semop函数的进程ID，即semid_ds结构中sem.sempid的值。

GETVAL： 获得semid所表示的信号量集中semnum所指定信号量的值。

GETZCNT：获得semid所表示的信号量集中的等待信号量成为0的进程数目，即semid_ds结构中sem.semncnt的值。

IPC_RMID：删除该信号量。

IPC_SET： 按参数arg.buf指向的结构中的值设置该信号量对应的semid_ds结构。只有有效用户ID和信号量的所有者ID或创建者ID相同的用户进程，以及超级用户进程可以执行这一操作。

IPC_STAT：获得该信号量的semid.ds结构，保存在arg.buf指向的缓冲区。

SETALL：以arg.array中的值设置semid所表示的信号量集中信号量的个数。

SETVAL：设置semid所表示的信号量集中semnum所指定的信号量的值。

编写代码（gxneicun.c）：

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/shm.h>
#include<stdlib.h>
```

```
#include<errno.h>
#include<string.h>
#include<signal.h>
#if
defined(_GNU_LIBRARY_) && ! defined(_SEM_SEMUN_UNDEFINED)
#else
union semun
{
        int val;
        struct semid_ds *buf;
        unsigned short int *array;
        struct seminfo *_buf;
    };
#endif
#define SHMDATASIZE 1000
#define BUFFERSIZE (SHMDATASIZE - sizeof(int))
#define SN_EMPTY 0
#define SN_FULL 1
int DeleteSemid=0;
void server(void);
void client(int shmid);
void delete(void);
void sigdelete(int sginum);
void locksem(int semid,int semnum);
void unlocksem(int semid,int semnum);
void clientwrite(int shmid,int semid,char *buffer);
int main(int argc, char *argv[])//命令本身就是参数
{
    if(argc<2)
        server();
```

```
        else
            client(atoi(argv[1]));
        return 0;
    }
    void server(void)
    {
        union semun sunion;
        int semid,shmid;
        void *shmdata;
        char *buffer;
        semid=semget(IPC_PRIVATE,2,SHM_R|SHM_W);
        DeleteSemid=semid;
        /*删除信号量*/
        atexit(&delete);
        signal(SIGINT,&sigdelete);
        /*初始化信号量*/
        sunion.val=1;//信号量可用
        semctl(semid,SN_EMPTY,SETVAL,sunion);
        sunion.val=0;
        semctl(semid,SN_FULL,SETVAL,sunion);
        printf("%d",SETVAL);
        /*分配共享内存段*/
        shmid=shmget(IPC_PRIVATE,SHMDATASIZE,IPC_CREAT|SHM_R|SHM_W);
        /*影射到内存空间*/
        shmdata=shmat(shmid,0,0);
        /*结束进程后自动删除共享内存*/
        shmctl(shmid,IPC_RMID,NULL);
        *(int *)shmdata=semid;//关键，把信号量放到共享内存中,在下面取出来，shmdata
是共享内存的首址
        buffer=shmdata+sizeof(int);//buffer 指向的是shmdata 的首地址加上semid 所占
```

```
的空间的下一位地址
        printf("server is running with id %d\n",shmid);
        while(1)
        {
            printf("waiting untill full...");
            fflush(stdout);//清空缓冲区
            locksem(semid,SN_FULL);//停下来，等待。
            printf("done.\n");
            printf("message received:%s\n",buffer);
            unlocksem(semid,SN_EMPTY);
        }
    }
    void client(int shmid)
    {
        int semid;
        char *buffer;
        void *shmdata;
        shmdata=shmat(shmid,0,0);
        semid=*(int *)shmdata;
        buffer=shmdata+sizeof(int);
        printf("client operational:shm id is %d\n",shmid);
        while(1)
        {
            char input[3];
            printf("\n\nmenu\n1.send a message\n");
            printf("2.exit\n");
            fgets(input,sizeof(input),stdin);
            switch(input[0])
            {
                case '1':clientwrite(shmid,semid,buffer);break;
```

```
                    case '2':exit(0);break;
                }
            }
        }
    void delete(void)
    {
        printf("master exiting:deleting semaphore.\n");
        if(semctl(DeleteSemid,0,IPC_RMID,0)==-1)
        printf("error releasing semaphore.\n");
    }
    void sigdelete(int signum)
    {
        exit(0);
    }
    void locksem(int semid,int semnum)
    {
        struct sembuf sb;
        sb.sem_num=semnum;
        sb.sem_op=-1;
        sb.sem_flg=SEM_UNDO;
        printf("%d",sb.sem_flg);
        semop(semid,&sb,1);
    }
    void unlocksem(int semid,int semnum)
    {
        struct sembuf sb;
        sb.sem_num=semnum;
        sb.sem_op=1;
        sb.sem_flg=SEM_UNDO;
        semop(semid,&sb,1);
```

```
}
void clientwrite(int shmid,int semid,char *buffer)
{
    printf("waiting until empty...");
    fflush(stdout);
    locksem(semid,SN_EMPTY);
    printf("done.\n");
    printf("enter message:");
    fgets(buffer,BUFFERSIZE,stdin);
    unlocksem(semid,SN_FULL);
}
```

五、 实验步骤

1. 编译运行

编译程序 # gcc gxneicun.c –o gx 本实验也可通过交叉编译在开发板上运行

运行服务器程序 # ./gx

在另一个终端界面运行客户端程序./ gx xxx (xxx 为服务器程序打印的shmid 值)

2. 运行结果

当运行./gx 时显示16server is running with id 131073

waiting untill full...

然后运行./gx 131073 后会显示client operational:shm id is 131073

```
menu
1.send a message
2.exit
```

如果选择1，则在客户机敲什么，服务器上就会显示什么，然后又回到菜单界面

```
menu
1.send a message
2.exit
1
waiting until empty...4096done.
```

```
enter message:fhgjhhjhkjk
menu
1.send a message
2.exit
```

看服务器上的结果：

```
waiting untill full...4096done.
message received:fhgjhhjhkjk
waiting untill full...
```

六、 拓展思考

参考阅读

3.2.5 实验二十五 CAN 总线编程实验

一、 实验目的

1. 了解什么是 CAN 总线。
2. 掌握 CAN 总线编程。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编程实现CAN总线字符串通讯。

四、 实验原理

1.CAN总线简介

CAN 最初出现在 80 年代末的汽车工业中，由德国 Bosch 公司最先提出。当时，由于消费者对于汽车功能的要求越来越多，而这些功能的实现大多是基于电子操作的，这就使得电子装置之间的通讯越来越复杂，同时意味着需要更多的连接信号线。提出 CAN 总线的最初动机就是为了解决现代汽车中庞大的电子控制装置之间的通讯，减少不断增加的信号线。于是，他们设计了一个单一的网络总线，所有的外围器件可以被挂接在该总线上。1993 年，CAN 已成为国际标准 ISO11898(高速应用)和 ISO11519（低速应用）。

CAN是一种多主方式的串行通讯总线，基本设计规范要求有高的位速率，高抗电磁干扰性，

而且能够检测出产生的任何错误。当信号传输距离达到10Km时，CAN 仍可提供高达50Kbit/s的数据传输速率。 由于CAN总线具有很高的实时性能，因此，CAN已经在汽车工业、航空工业、工业控制、安全防护等领域中得到了广泛应用。

2. CAN协议

CAN通信是一种一点对多点的传输协议，不是基于地址的传统的点对点传输协议。当一个点传输数据时，总线上的其它点都可以为接受方，它们可以通过ID来作出对总线上传送数据的处理（接收或者丢弃）。并且当数据被正确接收到以后，接收方便会作出应答响应。CAN协议还有一个很实用的功能，就是总线上的任一个节点可以请求其它节点向其发送数据，这被称作远程发送请求（RTR）。除此以外，CAN协议还有一个优点，当总线新加入一个节点进行通信时无需更改原有的程序，新节点只要通过ID就可以知道是接收还是丢弃数据。

CAN设计的三层结构模型为：物理层、数据链路层和应用层。物理层和数据链路层的功能由CAN接口器件完成，包括硬件电路和通讯协议两部分。CAN通讯协议规定了四种不同用处的网络通讯帧，即数据帧、远程帧、错误指示帧和超载帧。

CAN 通信卡硬件实现 CAN 定义的物理层和数据链路层功能，收发报文中数据长度为 0～8 个字节，有 2032 个报文标识符。工作时通过报文标识符确定总线访问优先权，高优先级报文具有低延迟时间，数据传送速率可编程（最高为 1Mbps）。发送期间若丢二氧化碳仲裁或由于出错而破坏的报文可自动重发。具有成组和广播报文功能。

当 CAN 通信卡接收到一个报文时，数据保存在 CAN 通信卡上的接收缓存器中，并产生一接收中断。当一个报文被成功发关垢，发送缓冲器可再次被访问，产生一个发送中断信号。CAN 通信卡发送报文，将数据送入 CAN 通信卡上的发送缓存器中，CAN 通信卡将数据串行化发到 CAN 总线上。

五、 实验步骤

运行：

socket通讯程序经过交叉编译后,在两块EBA版上进行通讯实验.

1.SERVER端EBA版打开一终端,进入到/SDCARD/目录中，执行程序./can_server

2.CLIENT端EBA版打开—终端,进入到/SDCARD/目录中，执行程序./can_client

六、 拓展思考

CAN总线技术的软件设计主要包括：CAN总线控制器的初始化、报文发送和报文接收。设置工作方式、设置接收滤波方式，设置接收屏蔽寄存器(AMR)和接收代码寄存器(ACR)、设置波特率参数和中断允许寄存器(IER)等如何实现的?

七、 参考阅读

3.2.6 实验二十六 RS485 总线编程实验

一、 实验目的

1. 了解 RS485 总线。
2. 学习掌握 RS485 总线编程。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编程实现CAN总线字符串通讯。

四、 实验原理

1.RS485总线简介

RS-485（EIA-485标准）是RS-422的改进，因为它增加了设备的个数，从10个增加到32个，同时定义了在最大设备个数情况下的电气特性，以保证足够的信号电压。有了多个设备的能力，你可以使用一个单个RS-422口建立设备网络。出色抗噪和多设备能力，在工业应用中建立连向PC机的分布式设备网络、其他数据收集控制器、HMI或者其他操作时，串行连接会选择RS-485。RS-485是RS-422的超集，因此所有的RS-422设备可以被RS-485控制。RS-485可以用超过4000英尺的线进行串行通行。

RS485 接口的最大传输距离标准值为4 000 英尺，实际上可达3 000 m。另外，RS232

接口在总线上只允许连接1个收发器，即单站能力；而RS485 接口在总线上是允许连接多达128 个收发器，即具有多站能力。这样用户可以利用单一的．RS485 接口方便地建立起设备网络。

在程序中，很容易配置串口的属性，这些属性定义在结构体struct termios中。为在程序中使用该结构体，需要包含文件<termbits.h>，该头文件定义了结构体struct termios。该结构体定义如下：

```
#define NCCS 19
struct termios {
        tcflag_t c_iflag;
        tcflag_t c_oflag;
        tcflag_t c_cflag;
        tcflag_t c_ispeed;
        tcflag_t c_ospeed;
```

```
        cc_t c_line;
        cc_t c_cc[NCCS];
};
```

在前面已经提到linux下的串口访问是以设备文件形式进行的，所以打开串口也即是打开文件的操作。函数原型可以如下所示：

```
int open（“DE_name”，int open_Status）
```

串口读操作（接收端）

用open函数打开设备文件，函数返回一个文件描述符(file descriptors,fd)，通过文件描述符来访问文件。读串口操作是通过read函数来完成的。函数原型如下：

```
int read(int fd, *buffer,length)；
```

串口写操作（发送端）

写串口操作是通过write函数来完成的。函数原型如下：

```
write(int fd, *buffer,length);
```

关闭串口

对设备文件的操作与对普通文件的操作一样，打开操作之后还需要关闭，关闭串口用函数close( )来操作，函数原型为：

```
int close(int fd);
```

五、 实验步骤

编译运行

在 pc 机上使用 arm-linux-gcc 工具对源文件进行交叉编译。通讯程序经过交叉编译后，在两块EBA版上进行通讯实验。

1.SERVER端EBA版打开一终端,进入到/SDCARD/目录中，执行程序./server

2.CLIENT端EBA版打开—终端,进入到/SDCARD/目录中，执行程序./client

六、 拓展思考

串口硬件本身有没有buffer的， tmp_buffer那个指针在哪个中？

七、 参考阅读

3.2.7 实验二十七 OV9650 摄像头显示实验

一、 实验目的

1. 熟悉linux下的OV9650相关的操作。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

编写实验代码，实现OV9650摄像头能在EBA版LCD中显示图象。

四、 实验原理

ARM网关仅仅提供了一个摄像接口，因此要实现其功能，还需要摄像头。在这里，我们使用OV9650。OV9650 内部有大量的寄存器需要配置，这就需要另外的数据接口。OV9650的数据接口称为SCCB（串行摄像控制总线），它由两条数据线组成：一个是用于传输时钟信号的SIO_C，另一个是用于传输数据信号的SIO_D。SCCB 的传输协议与IIC 的极其相似，只不过IIC在每传输完一个字节后，接收数据的一方要发送一位的确认数据，而SCCB 一次要传输9 位数据，前8 位为有用数据，而第9 位数据在写周期中是Don’t-Care 位（即不必关心位），在读周期中是NA 位。SCCB 定义数据传输的基本单元为相（phase），即一个相传输一个字节数据。SCCB只包括三种传输周期，即3 相写传输周期（三个相依次为设备从地址，内存地址，所写数据），2 相写传输周期（两个相依次为设备从地址，内存地址）和2 相读传输周期（两个相依次为设备从地址，所读数据）。当需要写操作时，应用3 相写传输周期，当需要读操作时，依次应用2相写传输周期和2 相读传输周期。因此SCCB 一次只能读或写一个字节。下面我们就用EBA版的IIC 总线接口分别与OV9650 的SIO_C 和SIO_D 相连接来实现SCCB 的功能。

五、 实验步骤

1.编译运行

在pc机上使用 arm-linux-gcc 工具对源文件进行交叉编译。在EBA板运行（备注：需要将PC机上源文件目录挂载到开发板上）$./testcamera0

2.运行结果

在EBA版LCD屏中显示有OV9650设到的图像。

六、 拓展思考

像素时钟、帧同步时钟和行同步时钟频率各是多少？

七、 参考阅读

3.2.8 实验二十八 Makefile 实验

一、 实验目的

1. 了解 make。
2. 了解 makefile。
3. 掌握 makefile 编程。

二、 实验设备

PC机、ARM网关（EBA）、已装有交叉编译工具（如 arm-linux-gcc）

三、 实验内容

通过 makefile 来编译程序。

四、 实验原理

1.Make

Make 是大型程序的维护工具，当你有一个c 文件时，你可以用gcc 直接编译，但当你的主程序依赖于很多其他c 文件时，你再用gcc 一条一条的编译便显得很吃力，这时候用make便会使一切变得轻松。那make 是如何维护的呢？这就需要makefile。

2.makefile

当运行make 命令时，make 命令会在当前目录下按顺序寻找文件名为“GUNmakefile”、“Makefile”、“makefile”的文件，找到后解释这些文件。所以说make 是一个解释makefile中指令的命令工具。

Makefile 或 makefile: 告诉make 维护一个大型程序，该做什么。Makefile 说明了组成程序的各模块间的相互关系及更新模块时必须进行的动作，make 按照这些说明自动地维护这些模块。

简单的说makefile 就像批处理的脚本文件，里边写好了一些命令的集合，当运行make命令时，便会按着makefile 提供的命令及顺序来完成编译。

Makefile 文件包含了五部分内容：显示规则、隐式规则、变量定义、文件指示和注释。

- 显示规则：显示规则说明了如何生成一个或多个目标文件。这要由makefile 文件的创作者指出，包括要生成的文件、文件的依赖文件、生成的命令。
- 隐式规则：由于makefile 有自动推导功能，所以隐式的规则可以比较粗糙地简略书写makefile 文件，这是由make 所支持的。
- 变量定义：在makefile 文件中要定义一系列的变量，变量一般都是字符串，像C 语言中的宏。当makefile 文件被执行时，其中的变量都会扩展到相应的引用位置上。

- 文件指示：包括3 个部分，一个是在一个makefile 文件中引用另一个makefile 文件，就像C 语言中的include 一样；另一个是指根据某些情况指定makefile 文件中有效部分，就像C 语言中的预编译#if 一样；还有就是定义一个多行的命令。

- 注释部分：makefile 文件中只有行注释，和 UNIX 的 shell 脚本一样，其注释用“#”字符，就像 C 语言中的“/* */”、“//”一样。如果要在 makefile 文件中使用“#”字符，可以用反斜杠进行转义如“#”。

3.Makefile中的变量

Makefile里的变量就像一个环境变量。事实上，环境变量在make 中也被解释成make的变量。这些变量对大小写敏感，一般使用大写字母。几乎可以从任何地方引用定义的变量。

Makefile中的变量是用一个文本串在Makefile中定义的，这个文本串就是变量的值。只要在一行的开始写下这个变量的名字，后面跟一个“＝”号，以及要设定这个变量的值即可定义变量，下面是定义变量的语法：

VARNAME=string

使用时，把变量用括号括起来，并在前面加上$符号，就可以引用变量的值：

${VARNAME}

其中GUN make 种主要预定义的变量：

$* 不包含扩展名的目标文件名称。

$+ 所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件。

$< 第一个依赖文件的名称。

$? 所有的依赖文件，以空格分开，这些依赖文件的修改日期比目标的创建日期晚。

$@ 目标的完整名称。

$^ 所有的依赖文件，以空格分开，不包含重复的依赖文件。

$% 如果目标是归档成员，则该变量表示目标的归档成员名称。例如，如果目标名称为mytarget.so(image.o)，则$@为mytarget.so，而$%为image.o。

Make工作时的执行步骤

- 读入所有的makefile文件
- 读入被include包括的其他的makefile文件
- 初始化文件中的变量
- 推到隐式规则，并分析所有规则
- 为所有的目标文件创建依赖关键链
- 根据依赖关系，决定哪些目标要重新生成

- 执行生成命令

前五步为第一个阶段，后两步为第二个阶段。第一个阶段中，如果定义的变量被使用了，make会在它使用的位置把它展开。但make并不会马上完全展开，make使用的是拖延战术。如果变量出现在依赖关系的规则中，仅当这条依赖关系决定要使用时，变量才会在其内部展开。

以下边的Makefile为例，解释一下make是怎么运行的：

Makefile文件中定义的第一个目标，make首先将其读入，然后从第一行开始执行，把第一个目标test 作为它的最终目标，所有后面的目标的更新都会影响到test 的更新。第一条规则说明只要文件test 的时间戳比文件prog.o 或code.o、main.o中的任何一个旧，下一行的编译命令将会被执行。在检查文件prog.o和code.o、main.o的时间戳之前，make会在下面的行中寻找以prog.o和code.o、main.o为目标的规则，在第五行中找到了关于prog.o的规则，该文件的依赖文件是prog.c、prog.h和code.h。同样，make会在后面的规则行中继续查找这些依赖文件的规则，如果找不到，则开始检查这些依赖文件的时间戳，如果这些文件中任何一个的时间戳比prog.o的新，make将执行“gcc －c prog.c －o prog.o”命令，更新prog.o文件。以同样的方法，接下来对文件code.o、main.o 以同样的方法，接下来对文件code.o、main.o做类似的检查。当make执行完所有这些套嵌的规则后，make将处理最顶层的test规则。如果关于prog.o和code.o、main.o的三个规则中的任何一个被执行，至少其中一个.o目标文件就会比test新，那么就要执行test规则中的命令，因此make去执行gcc命令将prog.o和code.o、main.o连接成目标文件test。

编写代码：

```
/************************code.c*****************************************/
#include "code.h"
#include <stdio.h>
void circle(float r)
{
    printf(format_circle,2*PI*r);
}
/*********************************prog.c*****************************************/
#include "prog.h"
#include "code.h"
#include <stdio.h>
area(float r)
```

```
{
	printf("r=%f\n",r);
	printf(format_area,PI*r*r);
	printf("thank you\n");
}
/**********************************test.c**************************************/
#include <stdio.h>
extern area(float);
extern circle(float);
main()
{
	printf("dgjdkfhgkjfdg\n");
	area(2.5);
	circle(2.5);
	return 0;
}
/*****************************code.h**************************************/
#define PI 3.1415926
#define format_circle "circle=%f\n"
/*******************************prog.h**************************************/
#define format_area "area=%f\n"
/************************Makefile**************************************/
CC=/usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-gcc
#CC=gcc
test: prog.o code.o main.o
$(CC) -o test prog.o code.o main.o
main.o: test.c
$(CC) -o main.o -c test.c
prog.o: prog.c prog.h code.h
$(CC) -c prog.c -o prog.o
```

```
code.o: code.c code.h
$(CC) -c code.c -o code.o
clean:
rm -f *.o
```

五、 实验步骤

1.编译运行

在 pc 机上使用 arm-linux-gcc 工具对源文件进行交叉编译。$make

在实验板运行（备注：需要将PC机上源文件目录挂载到开发板上）$./test

2.运行结果

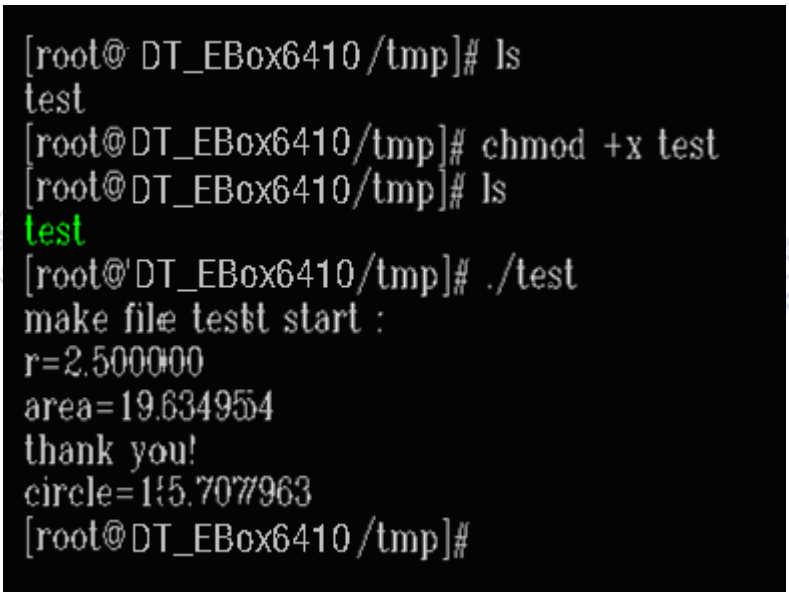


图3-19 运行结果

六、 拓展思考

七、 参考阅读

3.2.9 实验二十九 QT Hello World

一、 实验目的

1. 学习并了解 Qt 程序源码文件的结构及各个部分的作用。
2. 学会并掌握 Qt 工程管理。
3. 学会 Qt 程序的交叉编译和移植。

4. 学习并认识 Qt 及信号与槽。

二、 实验设备

- PC机、ARM网关（EBA）
- 已装有交叉编译工具（如 arm-linux-gcc）
- 已安装 Qt Embedded 库及其开发工具
- 已安装 tslib 触摸屏支持库
- 已指定环境变量指向 Qt Embedded 的安装目录及其它相关库路径

三、 实验内容

编写一个简单的 Qt 程序打开一个窗口。窗口内放置一个印有“Hello World！”字样的标签和一个实现“Close”的按钮。

四、 实验原理

编写源代码

编写 Qt 程序源代码，并保存为如下三个文件：

```
[root@localhost Hello_World_QT]# ls
formHelloWorld.cpp    formHelloWorld.h   main.cpp
```

这是些是较为标准的 Qt 源码组织形式，有些 Qt 程序还包含*.ui.h 等源码文件。

窗体头文件 formHelloWorld.h

```
#ifndef FORMHELLOWORLD_H
#define FORMHELLOWORLD_H
#include <qvariant.h>
#include <qdialog.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QSpacerItem;
class QLabel;
class QPushButton;
class FormHelloWorld : public QDialog
```

```
    {
        Q_OBJECT
        public:
        FormHelloWorld( QWidget* parent = 0, const char* name = 0, bool modal =
FALSE,WFlags fl = 0 );
        ~FormHelloWorld();
        QLabel* textLabel;
        QPushButton* pushButton;
        protected:
        protected slots:
        virtual void languageChange();
    };
    #endif // FORMHELLOWORLD_H
```

窗体源码文件 formHelloWorld.cpp

```
    #include "formHelloWorld.h"
    #include <qvariant.h>
    #include <qlabel.h>
    #include <qpushbutton.h>
    #include <qlayout.h>
    #include <qtooltip.h>
    #include <qwhatsthis.h>
    #include <qimage.h>
    #include <qpixmap.h>
    /*
    * Constructs a FormHelloWorld as a child of 'parent', with the
    * name 'name' and widget flags set to 'f'.
    *
    * The dialog will by default be modeless, unless you set 'modal' to
    * TRUE to construct a modal dialog.
    */
```

```
    FormHelloWorld::FormHelloWorld( QWidget* parent, const char* name, bool
modal,WFlags fl )
    : QDialog( parent, name, modal, fl )
    {
        if ( !name )
            setName( "FormHelloWorld" );
        setMinimumSize( QSize( 320, 240 ) );
        setMaximumSize( QSize( 320, 240 ) );
        setPaletteBackgroundColor( QColor( 170, 255, 127 ) );
        textLabel = new QLabel( this, "textLabel" );
        textLabel->setGeometry( QRect( 10, 40, 301, 80 ) );
        textLabel->setPaletteForegroundColor( QColor( 255, 0, 0 ) );
        textLabel->setPaletteBackgroundColor( QColor( 255, 170, 255 ) );
        QFont textLabel_font( textLabel->font() );
        textLabel_font.setFamily( "Nimbus Sans L" );
        textLabel_font.setPointSize( 24 );
        textLabel_font.setBold( TRUE );
        textLabel->setFont( textLabel_font );
        pushButton = new QPushButton( this, "pushButton" );
        pushButton->setGeometry( QRect( 80, 150, 160, 51 ) );
        pushButton->setPaletteForegroundColor( QColor( 85, 0, 127 ) );
        pushButton->setPaletteBackgroundColor( QColor( 255, 170, 0 ) );
        QFont pushButton_font( pushButton->font() );
        pushButton_font.setPointSize( 18 );
        pushButton->setFont( pushButton_font );
        languageChange();
        resize( QSize(320, 240).expandedTo(minimumSizeHint()) );
        clearWState( WState_Polished );
        // signals and slots connections
        connect( pushButton, SIGNAL( clicked() ), this, SLOT( close() ) );
```

```
}
/*
* Destroys the object and frees any allocated resources
*/
FormHelloWorld::~FormHelloWorld()
{
    // no need to delete child widgets, Qt does it all for us
}
/*
* Sets the strings of the subwidgets using the current
* language.
*/
void FormHelloWorld::languageChange()
{
    setCaption( tr( "HelloWorld" ) );
    textLabel->setText( tr( "<p align=\"center\">Hello World!</p>" ) );
    pushButton->setText( tr( "Close" ) );
}
```

程序入口源码文件 main.cpp：

```
#include <qapplication.h>
#include "formHelloWorld.h"
int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    FormHelloWorld w;
    w.show();
    w.adjustSize();
    a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
    return a.exec();
}
```

五、 实验步骤

1. 建立 Qt 工程

一个 Qt 程序通常会用到多个源码文件，故需要创建一个工程(生成工程文件)来对所有的源码文件进行管理。命令如下：

```
[root@localhost Hello_World_QT]# qmake –project
[root@localhost Hello_World_QT]# ls
formHelloWorld.cpp    formHelloWorld.h Hello_World_QT.pro   main.cpp
```

工程文件 Hello_World_QT.pro

```
TEMPLATE = app
INCLUDEPATH += .
# Input
HEADERS += formHelloWorld.h
SOURCES += formHelloWorld.cpp main.cpp
```

2. 配置交叉编译环境变量

交叉编译需配置相应的环境变量，内容如下：

```
export QTDIR=/root/build_qte/qte
```

（假设本机 Qt/Embedded 安装目录为/root/build_qte/qte，可根据实际情况修改此处。）

```
export QMAKEDIR=$QTDIR/qmake
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export PATH=$QMAKEDIR/bin:$QTDIR/bin:/usr/local/arm/3.4.4/bin:$PATH
export QMAKESPEC=qws/linux-arm-g++
```

通常将以上内容写入以脚本文件，需配置环境变量时使用 source 命令加载即可。

3. 生成并修改 Makefile

通过工程文件生成自动编译文件 Makefile，命令如下：

```
[root@localhost Hello_World_QT]# qmake
[root@localhost Hello_World_QT]# ls
formHelloWorld.cpp    formHelloWorld.h Hello_World_QT.pro   main.cpp      Makefile
```

由于嵌入式环境通常不使用鼠标，故我们的 Qt 程序在开发板上只能通过触摸屏来操作，因此我们需要对 Qt 程序添加对触摸屏设备的支持。这一动作可以通过修改 Makefile 来完成。

在 Makefile 找到如下原文：

```
LIBS = $(SUBLIBS) -L$(QTDIR)/lib -lqte-mt
```

改为：

```
LIBS = $(SUBLIBS) -L$(QTDIR)/lib -L/Share/tslib/lib -lts -lqte-mt
```

（假设本机 tslib 安装目录为/Share/tslib/lib，若非如此可根据实际情况修改此处。）

4. 交叉编译并移植

交叉编译源码文件，生成可执行程序。命令如下：

```
[root@localhost Hello_World_QT]# make
```

一段编译信息之后，编译完成。输入如下命令清空编译产生的临时文件并查看编译结果：

```
[root@localhost Hello_World_QT]# make clean
[root@localhost Hello_World_QT]# ls
formHelloWorld.cpp    Hello_World_QT  main.cpp
formHelloWorld.h      Hello_World_QT.pro  Makefile
```

通常编译生成的可执行程序与工程文件同名，即结果中的 Hello_World_QT 文件就是我们所要得到的 Qt Hello World 程序了。

5. 编译运行

将可执行程序拷贝到开发板中并运行。命令、参数及运行结果如下：

```
Bash-3.2# ./Hello_World_QT -qws
```

6. 运行结果

Qt Hello World 程序运行，结果如图3-20所示：

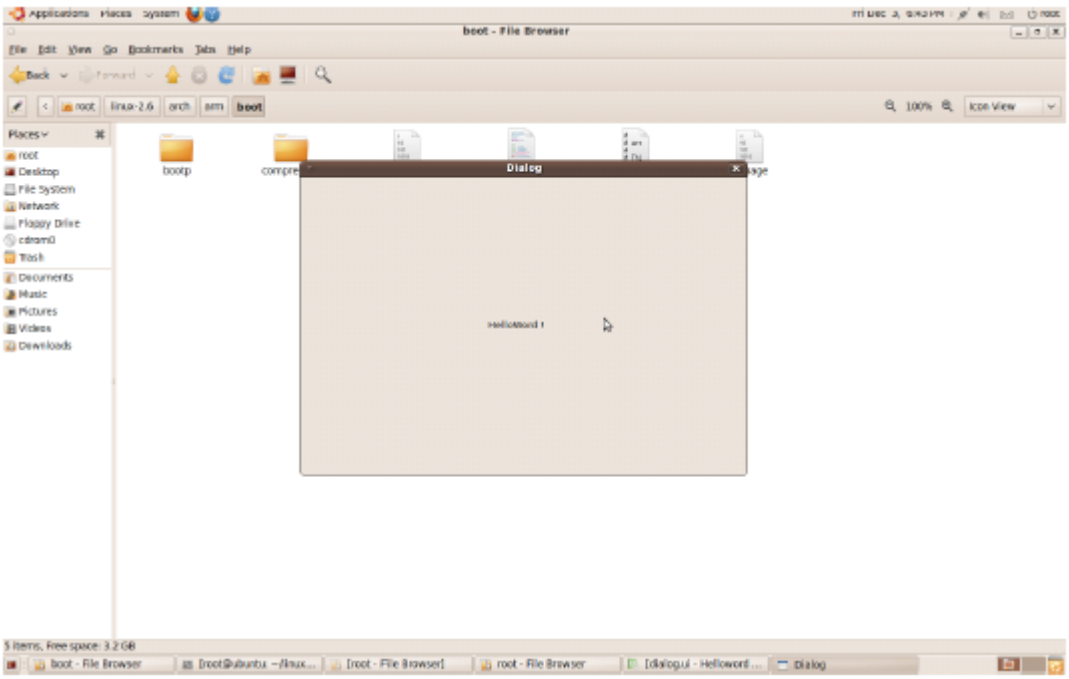


图3-20　　运行结果

六、　　拓展思考

Qt与其它框架非常不同的核心特性是那些机制？

七、　　参考阅读

第四章 物联网感知实验

4.1 感知实验概述

感知是物联网深入物理世界的第一步，是物联网中海量信息的源头，也是物联网感知互动层的最主要的技术。目前，我国在传感器核心技术、芯片、标准等方面缺少自主产权，其高端产品及市场也一直被国外企业所控制，种类繁多的传感器在工艺、形态、性能、价格等方面缺少统一的规范，不利于物联网行业间的互联互通，甚至由于缺乏统一管理而影响到国家安全。因此，很多科研机构和企业已致力于这方面的开发。

传感器和 RFID 是物联网感知的核心内容。在这一层次的主要难点是研发高精度、低成本、低功耗、稳定可靠的智能数字传感器以及物联网规模化发展所需的物理感知和通信感知系统，进一步融合及适当兼容现有国内外各种感知方法，构建物联网的标准化传感器、智能感知、物体感知和应用感知，构建我国自主产权的物联网的统一感知体系。

下面结合目前常用的几类传感器和 RFID 方案介绍物联网感知层的实验。希望能对大家深入了解传感器和 RFID 系统有所帮助。

4.2 典型传感器基础实验

4.2.1 实验三十 温湿度传感器实验

一、 实验目的

1.熟悉温湿度传感器的工作原理。

2.掌握温湿度传感器 SHT10 的使用方法。

二、 实验设备

- ZigBee模块
- 温湿度传感器（EBS-T）
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境利用单片机 CC2531 来控制温湿度传感器 SHT10 获取周围环境的温度和湿度，并将温湿度数据发送到计算机。阅读温湿度传感器 STH10 芯片的数据手册，学会阅读并分析芯片的时序图，并能够依照时序图，编写相关的驱动程序，读出传感器的温度和湿度数据，然后把数据发送到 PC。

四、 实验原理

温湿度传感器的外观图如下图 4-1 所示：

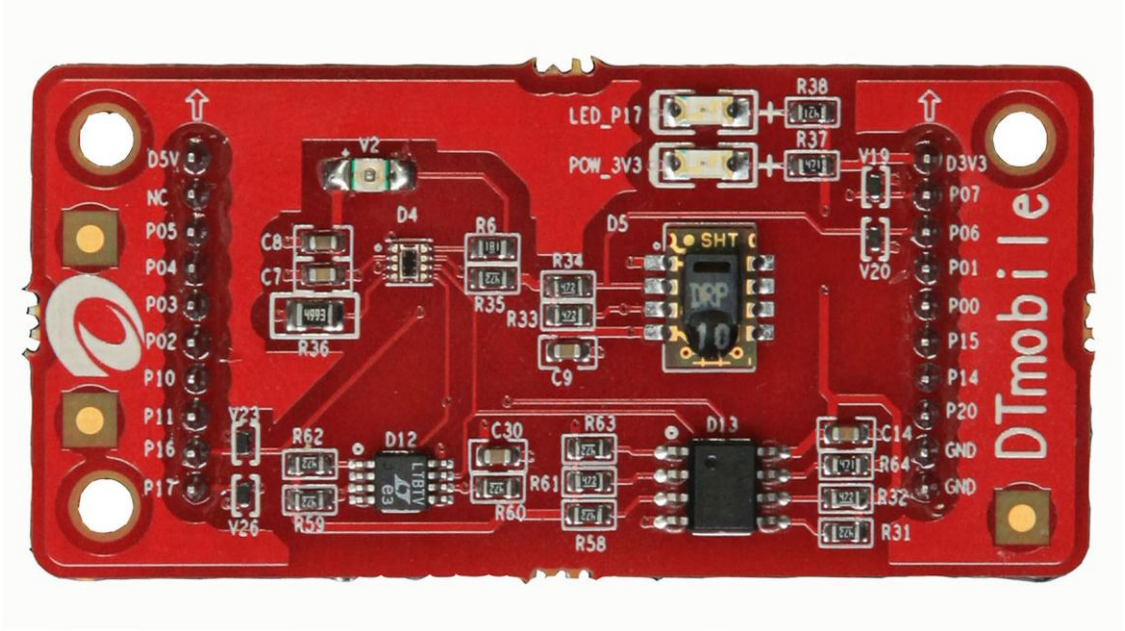


图4-1 温度传感器板卡

本实验程序主要完成的工作包括：首先，单片机发送命令给温湿度传感器，温湿度传感器再根据单片机命令，执行温度或湿度的采集；然后，单片机读出相应的温度或湿度的数据，并把相关数据发送到 PC；最后，通过 PC 获取采集到的温度和湿度数据，并计算转化为具体的温湿度值。为了便于理解，将实验内容分为 3 个部分：各模块的初始化，单片机控制温湿度传感器采集温度和湿度，把采集的数据发送到 PC 上。下面分别对以上内容进行介绍。

1. 各模块的初始化：

因为温湿度传感器只需要将 DATA 引脚和 SCK 引脚与单片机相连，所以使用单片机的 P0_6 和 P0_7 引脚分别连接到 DATA 和 SCK 口，并且给两个引脚发送相应的时序就能控制温湿度传感器。串口初始化代码：

```
void initUARTtest(void)
{
```

```
    CLKCONCMD &= ~0x40;                  //晶振
    while(!(SLEEPSTA & 0x40));           //等待晶振稳定
    CLKCONCMD &= ~0x47;                  //TICHSPD128 分频，CLKSPD 不分频
    SLEEPSTA |= 0x04;                    //关闭不用的 RC 振荡器
    PERCFG = 0x01;                       //位置 1  串口 0
    P1SEL |= 0x30;                 //P1 用作串口
    U0CSR |= 0x80;                 //UART 方式
    U0GCR |= 8;                  //baud_e
    U0BAUD |= 59;                  //波特率设为 9600
    UTX0IF = 1;
    U0CSR |= 0X40;                 //允许接收
    IEN0 |= 0x84;                  //开总中断，接收中断
}
```

2. 温度和湿度的采集：

温湿度传感器 SHT10 有 4 个引脚：GND、DATA、SCK、VDD。下图 4-2 为传感器的典型应用电路，也是它与单片机的连接方式。

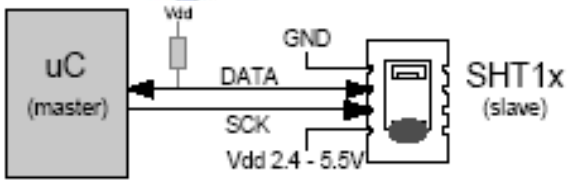


图4-2　　典型应用电路

温湿度传感器芯片采用 SHT10；

- SHT10 的 DATA 口接 CC2531 的 P0_6　//定义通讯数据端口
- SHT10 的 SCK 口接 CC2531 的 P0_7　//定义通讯时钟端口
- SHT10 的 VDD 口接 CC2531 的 VCC　//2.4~3.3V 供电（SHT10 的“电量不足”功能可监测到 VDD 电压低于 2.47V 的状态。精度为 0.05V。）
- SHT10 的 GND 口接 CC2531 的 GND　//地

在本实验中，SHT10 的 DATA 和 SCK 引脚与单片机的 P0_6 和 P0_7 引脚相连。因此，只要通过 P0_6 和 P0_7 引脚向 SHT10 发送相应的时序，就能驱动 SHT10 进行采样，并返回采样数据。传感芯片上集成了一个可通断的加热元件。接通后，可将 SHTxx 的温度提高大约 5-15°C（9-27°F）。功耗增加～8mA@5V。应用于：

- 比较加热前后的温度和湿度值，可以综合验证两个传感器元件的性能。
- 在高湿度（>95%RH）环境中，加热传感器可防止凝露，同时缩短其响应时间，提高测量精度。

警告：加热后较之加热前，SHTxx 将显示温度值略有升高、相对湿度值稍有降低。

要驱动 SHT10 进行采样必须发送如下命令：首先，向 SHT10 发送“启动传输”时序，完成数据传输的初始化。如图 4-3 所示，时序包括当 SCK 时钟高电平时，DATA 发转为低电平；紧接着 SCK 变为低电平，随后在 SCK 时钟高电平时，DATA 翻转为高电平。初始化之后，单片机便可以向 SHT10 发送命令。通常的命令包括 3 个地址位（目前只支持“000”）和 5 个命令位，具体将在后面的代码中进行介绍。SHT10 会以下述方式表示已正确的接收到命令：在第 8 个 SCK 时钟的下降沿之后，将 DATA 下拉为低电平，并且在第 9 个 SCK 时钟的下降沿之后，将 DATA 位恢复为高电平。

图4-3　　启动传输时序

启动传输程序：

```
void s_transstart(void)
//----------------------------------------------------------------------------
// 启动传输
//          _____          ________
// DATA:         |________|
//               ___     ___
// SCK : ___|    |___|    |______
{
    P0DIR = 0xC0;
    DATA=1; SCK=0;
    delay1Us(1);
    SCK=1;
    delay1Us(1);
    DATA=0;
    delay1Us(1);
    SCK=0;
```

```
        delay1Us(3);
        SCK=1;
        delay1Us(1);
        DATA=1;
        delay1Us(1);
        SCK=0;
    }
```

启动程序完成之后，SHT10 便会以串行数据的方式与单片机进行通信，时序图如图 4-4 所示。DATA 三态门用于数据的读取，DATA 在 SCK 时钟下降沿之后改变状态，并仅在 SCK 时钟上升沿有效。数据传输期间，在 SCK 时钟高电平时，DATA 必须保持稳定。为避免信号冲突，单片机应驱动 DATA 在低电平。

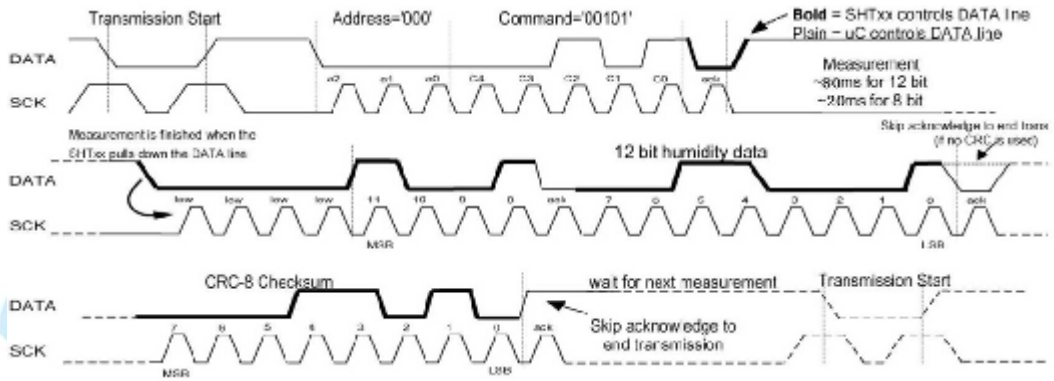


图4-4　　RH 测量时序举例：“0000’1001’0011’0001”=2353=75.79%RH

(未包含温度补偿)

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

```
char s_write_byte(unsigned char value)
// 写字节函数
{
        char i;
        char error=0;
        P0DIR= 0xC0;
        SCK=0;
        DATA=0;
        for(i=0;i<8;i++)   //发送 8 位数据，从机将在上升沿读取数据
```

```
        {
            SCK=0;
            if(value&(0x80>>i))
                DATA=1;
            else
                DATA=0;
            delay1Us(1);
            SCK=1;
            delay1Us(1);
        }
        SCK=0;      //在接下来的上升沿读取从机发送的“已收到”信号。
        P0DIR=0x80;
        delay1Us(1);
        SCK=1;
        delay1Us(1);
        error = DATA;
        delay1Us(1);
        SCK=0;
        P0DIR= 0xC0;;
        return error;
    }
```

如果实验中与 SHT10 通讯中断，下列信号时序可以复位串口：

当 DATA 保持高电平时，触发 SCK 时钟 9 次或更多。在下一次命令前，发送一个“传输启动“时序，如图 4-5 所示。这些时序只复位串口，状态寄存器内容仍然保留。

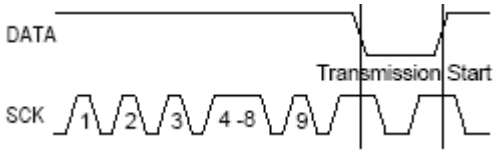


图4-5　　通信复位时序

数字信号的整个传输过程由 8bit 校验确保。任何错误数据将被检测到并清除。

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

关代码：

```
char s_read_byte(unsigned char ack)
// 读数据；
{
    unsigned char i,val=0;
    P0DIR |= 0x80;
    P0DIR &= 0xBF;
    SCK=0;
    for (i=0x80;i>0;i>>=1)                  //右移位
    {
        SCK=1;
        delay1Us(1);
        if (DATA)
            val=(val | i);          //读数据线的值
            SCK=0;
            delay1Us(1);
    }
    P0DIR |= 0xC0;
    DATA=!ack;                              //如果是校验，读取完后结束通讯；
    SCK=1;
    delay1Us(3);
    SCK=0;
    DATA=1;                                 //释放数据线
    return val;
}
```

温湿度的测量过程如下：Zigee 发布一组测量命令后（“00000001”表示测量相对湿度，“00000010”表示测量温度），传感器开始采集数据。对应 8 /12/14bit 3 种不同的测量，这个过程分别需要大约 20/80/320ms。SHT10 通过下拉 DATA 至低电平并进入空闲模式，表示测量的结束。单片机在触发 SCK 时钟来读取数据前，必须等待这个“数据备妥”信号。检测数据可以先被存储，这样单片机可以继续执行其他任务，在需要时再读出数据。在收到“数

据备妥”信号之后，传输 2B 的测量数据和 1B 的 CRC 奇偶较检。单片机需要通过下拉 DATA 为低电平，以确认每字节。所有的数据从 MSB 开始，右值有效（例如：对于 12bit 数据，从第 5 个 SCK 时钟起算作 MSB；而对于 8bit 数据，首字节则无意义）。用 CRC 数据的确认位，表明通讯结束。如果不使用 CRC-8 校验，控制器可以在测量值 LSB 后，通过保持确认位 ack 高电平，来中止通讯。在测量和通信结束后，SHT10 自动转入休眠模式。

警告：为保证自身温升低于 0.1 摄氏度，SHTxx 的激活时间不要超过 10%（例如，对应 12bit 精度测量，每秒最多进行 2 次测量）。

温湿度测量的相关代码：

```
char s_measure(unsigned char *p_value, unsigned char *p_checksum, unsigned char mode)
// 进行温度或者湿度转换，由参数 mode 决定转换内容；
{
    unsigned error=0;
    unsigned char i;
    s_transstart();                          //启动传输
    switch(mode)
    {
        case 0x02 : error+=s_write_byte(MEASURE_TEMP); break;
        case 0x01 : error+=s_write_byte(MEASURE_HUMI); break;
        default : break;
    }
    P0DIR |= 0x80;
    P0DIR &= 0xBF;
    while(DATA); //等待测量结束；
    if(DATA) error+=1;                       // 如果长时间数据线没有拉低，说明测量错误
    *(p_value)  =s_read_byte(ACK);      //读第一个字节，高字节 (MSB)
    *(p_value+1)=s_read_byte(ACK);      //读第二个字节，低字节 (LSB)
    *p_checksum =s_read_byte(noACK);   //read CRC 校验码
     return error;
}
```

3. 把采集的数据发送到 PC 上

单片机在获取了温湿度数据之后，将数据发送给 PC，并通过 PC 计算具体的温湿度值。获取数据的函数：

```
void calc_sth11(float *p_humidity ,float *p_temperature)
//-------------------------------------------------------------------------------------
// 补偿及输出温度和相对湿度
{
    const float C1=-4.0;                    // for 12 Bit 湿度修正公式
    const float C2=+0.0405;                 // for 12 Bit 湿度修正公式
    const float C3=-0.0000028;              // for 12 Bit 湿度修正公式
    const float T1=+0.01;                   // for 14 Bit @ 5V 温度修正公式
    const float T2=+0.00008;                // for 14 Bit @ 5V  温度修正公式
    float rh=*p_humidity;
    float t=*p_temperature;
    float rh_lin;
    float rh_true;
    float t_C;
    t_C=t*0.01 - 39.66;                     //补偿温度
    rh_lin=C3*rh*rh + C2*rh + C1;           //相对湿度非线性补偿
    rh_true=(t_C-25)*(T1+T2*rh)+rh_lin;     //相对湿度对于温度依赖性补偿
    if(rh_true>100)rh_true=100;             //湿度最大修正
    if(rh_true<0.1)rh_true=0.1;             //湿度最小修正
    *p_temperature=t_C;                     //返回温度结果
    *p_humidity=rh_true;                    //返回湿度结果
}
//--------------------------------------------------------------------
float calc_dewpoint(float h,float t)
//--------------------------------------------------------------------
// 计算绝对湿度值
{
    float logEx,dew_point;
```

```
        logEx=0.66077+7.5*t/(237.3+t)+(log10(h)-2);
        dew_point = (logEx - 0.66077)*237.3/(0.66077+7.5-logEx);
        return dew_point;
    }
```

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

```
    void main(void)
    {
        unsigned char error,checksum;
        unsigned char HUMI,TEMP;
        HUMI=0X01;      //0000 0001
        TEMP=0X02;
        initUARTtest();      //初始化串口
        s_connectionreset();
        while(1)
        {
            error=0;
            error+=s_measure((unsigned char*) &humi_val.i,&checksum,HUMI);   //湿度
测量
            error+=s_measure((unsigned char*) &temp_val.i,&checksum,TEMP);   //温度
测量
            UartTX_Send_String((char*)&humi_val.i,2);
            UartTX_Send_String((char*)&temp_val.i,2);
            if(error!=0)
            {
                s_connectionreset() ; //如果发生错误，系统复位
                led1 =   !led1;
                led2 =   !led2;
            }
```

```
            else
            {
                humi_val.f=(float)humi_val.i;                    //转换为浮点数
                temp_val.f=(float)temp_val.i;                    //转换为浮点数
                calc_sth11(&humi_val.f,&temp_val.f);             //修正相对湿度及温度
            }
            Delay1(50000);                          //延时
        }
    }
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将温湿度传感器模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考 \E-Box300\03- 实验源代码 \02-WSN\02- 传感器实验代码 \30_TemperatureHumi）。打开串口调试软件，选择相应的串口，并设置好波特率 9600。观察串口得到的数据。

3. 参考《2.2.4IAR 使用方法》一节中介绍的内容，设置断点进行调试，查看寄存器值，计算温度值和湿度值。

- 计算温度的公式为： Temperature = a + b * X

式中，a，b 为参数；X 为传感器采集的温度数据。

对于本实验采用 3V 电压、14 位采样分辨率，查芯片手册可知：a = --39.67，b = 0.01。带入计算即可得到以摄氏度为单位的温度值。在极端工作条件下测量温度时，可使用进一步的补偿算法以获取高精度。可参阅应用说明“相对湿度与温度的非线性补偿”。

- 计算湿度的公式为： Humidity = c + d * Y + e* Y^2

式中，c，d，e 为参数；Y 为传感器采集的湿度数据。

对于本实验采用 12 位采样精度，查芯片手册可知：c = -4，d = 0.0405，e = -2.8 * 10^-6。带入计算即可得到湿度百分比。对高于 99%RH 的那些测量值则表示空气已经完全饱和，必须被处理成显示值均为 100%^2 RH。湿度传感器对电压基本上没有依赖性。

六、 拓展思考

温湿度传感器可以应用在哪些场合？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\ Sensor\Temperature & Humidity”目录下的SHT1x_SHT7x.pdf 数据手册。

4.2.2 实验三十一 烟雾传感器实验

一、 实验目的

1.熟悉烟雾传感器的工作原理。

2.掌握利用单片机控制烟雾传感器的方法。

二、 实验设备

- ZigBee模块
- 烟雾传感器（EBS- MQ）
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境利用单片机 CC2531 来控制 MQ-2 烟雾传感器，继而利用烟雾传感器测量周围烟雾浓度。

四、 实验原理

我们实验用的典型型号 MQ-2 烟雾传感器属于半导体气敏式烟雾传感器。它由微型 AL2O3 陶瓷管、SnO2 敏感层，测量电极和加热器构成的敏感元件固定在塑料或不锈钢制定的腔体内，加热器为气敏元件提供了必要的工作条件。封装好的气敏元件有 6 只针装管脚，其中 4 个用于信号取出，2 个用于提供加热电流。MQ-2 气体传感器所使用的气敏材料是在清洁空气中电导率较低的二氧化锡（SnO2）。当传感器所处环境中存在可燃气体时，传感器的电导率随空气中可燃气体浓度的增加而增大。使用简单的电路即可将电导率的变化转换为该气体浓度相对应的输出信号。该传感器常用于家庭和工厂的气体泄漏装置，适用于液化气、

丁烷、丙烷、甲烷、酒精、氢气、烟雾等的探测，是一款适合多种应用的低成本传感器。下图 4-6 是它的外观图。

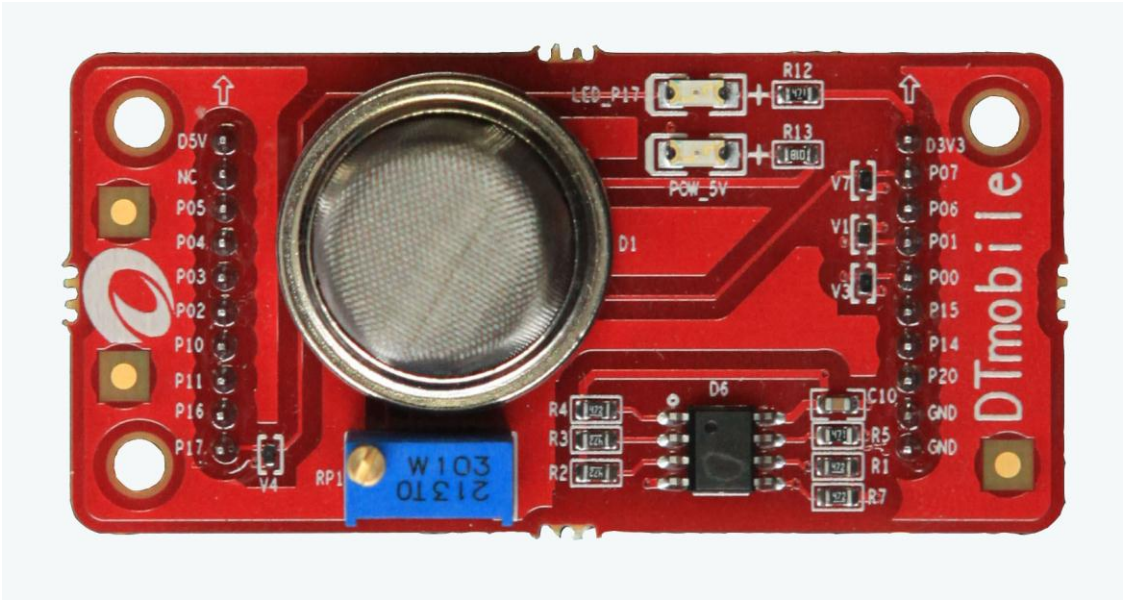


图4-6　　MQ-2 烟雾传感器外观图

烟雾传感器的工作原理图如下图 4-7 所示：

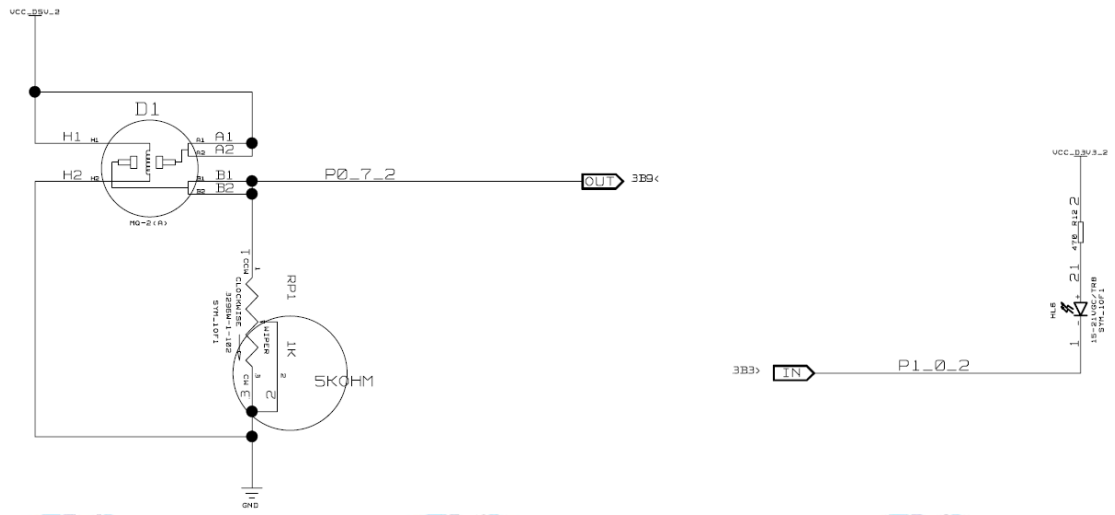


图4-7　　烟雾传感器内部原理图

烟雾传感器在 A、B 处检测周围空气的烟雾浓度，电压供烟雾传感器的加热丝工作一段时间预热后，才能正常检测烟雾。当烟雾传感器所处的环境烟雾在允许的范围内时，两端输出电极的导电率很低，则加在电极两端的电压很低，这样，输出的模拟信号电压升高。我们对输出的模拟电压信号进行 ADC 采样，利用单片机进行 ADC 转换，并将转换的到的数据通过串口传输到电脑上，形成烟雾浓度曲线图。

本实验程序主要完成的工作包括：首先，烟雾传感器探测周围空气烟雾浓度，输出模拟电压信号；然后，单片机进行 ADC 转换，读出相应的烟雾浓度数据，并把相关数据发送到

PC；最后，通过 PC 获取采集到的烟雾浓度度数据，并形成烟雾浓度曲线图。为了便于理解，将实验内容分为 2 个部分：各模块的初始化，烟雾传感器测量烟雾浓度后利用单片机进行 ADC 转换。下面分别对以上内容进行介绍。

1. 各模块的初始化

```
void initUARTtest(void)
{
    CLKCONCMD &= ~0x40;                          //晶振
    while(!(SLEEPSTA & 0x40));                   //等待晶振稳定
    CLKCONCMD &= ~0x47;                  //TICHSPD128 分频，CLKSPD 不分频
    SLEEPSTA |= 0x04;                    //关闭不用的 RC 振荡器
    PERCFG = 0x00;                 //位置 1  串口 0
    P1SEL = 0x30;                  //P1 用作串口
    U0CSR |= 0x80;                 //UART 方式
    U0GCR |= 8;                   //baud_e = 10;
    U0BAUD |= 59;                  //波特率设为 9600
    UTX0IF = 1;
    U0CSR |= 0X40;                 //允许接收
    IEN0 |= 0x84;                  //开总中断，接收中断
}
```

2. 烟雾传感器测量烟雾浓度后利用单片机进行 ADC 转换

```
uint16 Gas_Acquire(void)
{
  uint16 value;
  char msg[10] = {0};
  /* Clear ADC interrupt flag */
  ADCIF = 0;
  ADCCON3 = 0xF7;
  while ( !ADCIF );
  /* Get the result */
  value = ADCH;
```

```
    value = value << 8;
    value = value + ADCL;
     value = value >> 4;
     return value;
}
```

以上是此实验的各个模块的分析，下面介绍实验 main()函数的主要代码：

```
void main(void)
{
      P0DIR = 0x03;          //P1 控制 LED
      led1 = 1;
      led2 = 1;              //关 LED
      initUARTtest();
      while(1)
      {
            gas_temp = Gas_Acquire();
            gas_value = gas_temp * 3.3 / 2048 ;
            Delay(30000);
            led1 =!led1;                        //完成数据处理
            Delay(30000);
      }
}
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将烟雾传感器模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\02-WSN\02-传感器实验代码\31_MQ2）。打开串口调试软件，选择相应的串口，并设置好波特率 9600。观察串口得到的数据。

3. 设置断点进行调试，查看寄存器值。

六、 拓展思考

烟雾传感器可以应用在哪些场合？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\ Sensor\Gas”目录下的 MQ-2.pdf 数据手册。

4.2.3 实验三十二 加速度传感器实验

一、 实验目的

1.熟悉加速度传感器的工作原理。

2.掌握加速度传感器的使用方法。

二、 实验设备

- ZigBee模块
- 加速度传感器（EBS-A（cceleration））
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境，利用加速度传感器测量物体的三轴加速度。

四、 实验原理

我们实验用的 MMA8452 加速度传感器（如图 4-8）是一款具有 12 位分辨率的智能低功耗、三轴、电容式微机械加速度传感器。这款加速度传感器具有丰富嵌入式功能，带有灵活的用户可编程选项，可以配置多达两个中断引脚。嵌入式中断功能可以节省整体功耗，解除主处理器不断轮询数据的负担。MMA8452Q 具有 ±2g/±4g/±8g 的用户可选量程，可以实时输出高通滤波数据和非滤波数据。该器件可被配置成利用任意组合可配置嵌入式的功能生成惯性唤醒中断信号，这就使 MMA8452Q 在监控事件同时，在静止状态保持低功耗模式。MMA8452 加速度传感器的原理框图如下图 4-9。

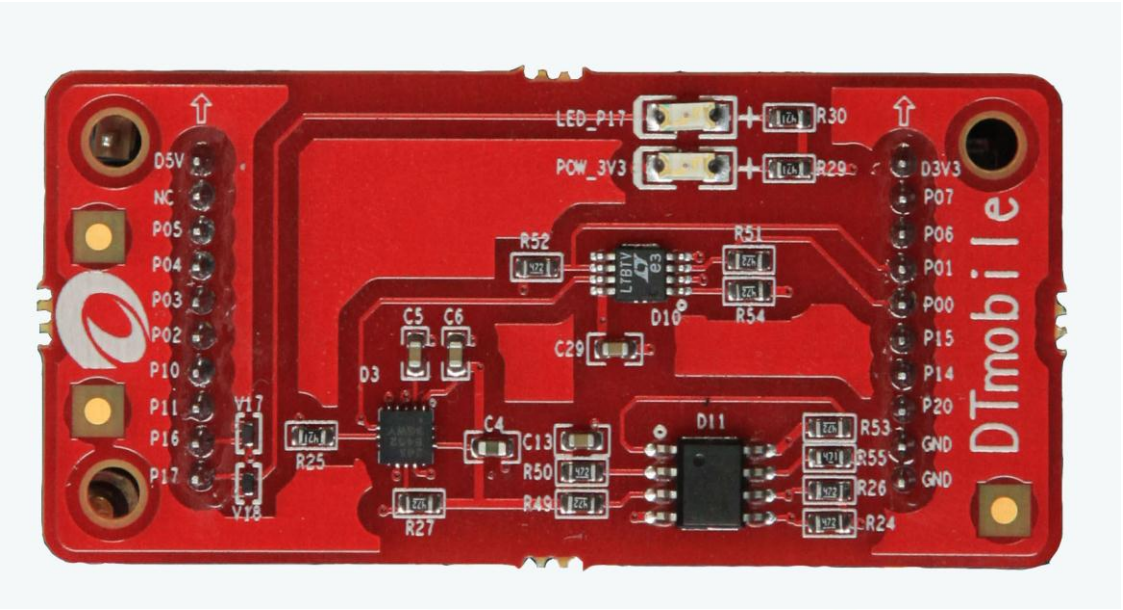


图4-8　　加速度传感器板卡

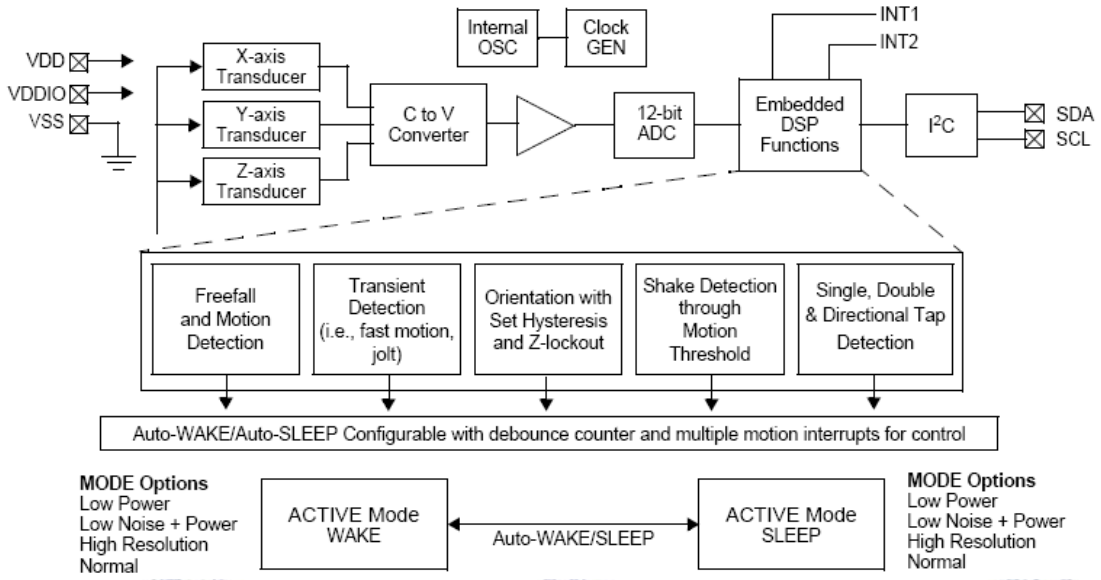


图4-9　　原理框图

加速度传感器采集到的数据与处理器之间发送数据是通过 I2C 协议实现的。我们先来了解一下 I2C 串行通信协议。

I2C 总线是两线式串行总线，用于连接微控制器及其外围设备，是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。

I2C 总线特征：

1. 只要求两条总线线路：一条串行数据线 SDA，一条串行时钟线 SCL；

2. 每个连接到总线的器件都可以通过唯一的地址和一直存在的简单的主机/从机关系软件设定地址，主机可以作为主机发送器或主机接收器；

3. 它是一个真正的多主机总线，如果两个或更多主机同时初始化，数据传输可以通过冲突检测和仲裁防止数据被破坏；

4. 串行的 8 位双向数据传输位速率在标准模式下可达 100kbit/s，快速模式下可达 400kbit/s，高速模式下可达 3.4Mbit/s；

5. 连接到相同总线的 IC 数量只受到总线的最大电容 400pF 限制。

I2C 的起始和终止条件：

SCL 线是高电平时，SDA 线从高电平向低电平切换，这个情况表示起始条件；

SCL 线是高电平时，SDA 线由低电平向高电平切换，这个情况表示停止条件。

起始和停止条件一般由主机产生，总线在起始条件后被认为处于忙的状态 ，在停止条件的某段时间后总线被认为再次处于空闲状态。

如果产生重复起始条件而不产生停止条件，总线会一直处于忙的状态，此时的起始条件（S）和重复起始条件（Sr） 在功能上是一样的。

I2C 总线的数据传输：

字节格式：发送到 SDA 线上的每个字节必须为 8 位，每次传输可以发送的字节数量不受限制。每个字节后必须跟一个响应位。首先传输的是数据的最高位（MSB），如果从机要完成一些其他功能后（例如一个内部中断服务程序）才能接收或发送下一个完整的数据字节，可以使时钟线 SCL 保持低电平，迫使主机进入等待状态，当从机准备好接收下一个数据字节并释放时钟线 SCL 后数据传输继续。

应答响应：数据传输必须带响应，相关的响应时钟脉冲由主机产生。在响应的时钟脉冲期间发送器释放 SDA 线（高）。

在响应的时钟脉冲期间，接收器必须将 SDA 线拉低，使它在这个时钟脉冲的高电平期间保持稳定的低电平。

通常被寻址的接收器在接收到的每个字节后，除了用 CBUS 地址开头的数据，必须产生一个响应。当从机不能响应从机地址时（例如它正在执行一些实时函数不能接收或发送），从机必须使数据线保持高电平，主机然后产生一个停止条件终止传输或者产生重复起始条件开始新的传输。

如果从机接收器响应了从机地址，但是在传输了一段时间后不能接收更多数据字节，主机必须再一次终止传输。这个情况用从机在第一个字节后没有产生响应来表示。从机使数据线保持高电平，主机产生一个停止或重复起始条件。

如果传输中有主机接收器，它必须通过在从机不产生时钟的最后一个字节不产生一个响应，向从机发送器通知数据结束。从机发送器必须释放数据线，允许主机产生一个停止或重复起始条件。

加速度传感器三轴方向的说明（如图 4-10、4-11）：

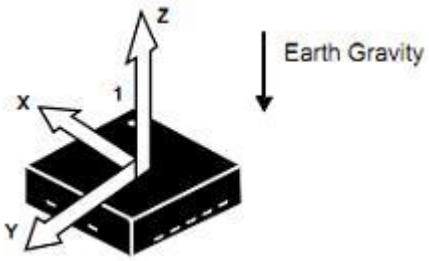


图4-10　检测加速度方向示意图

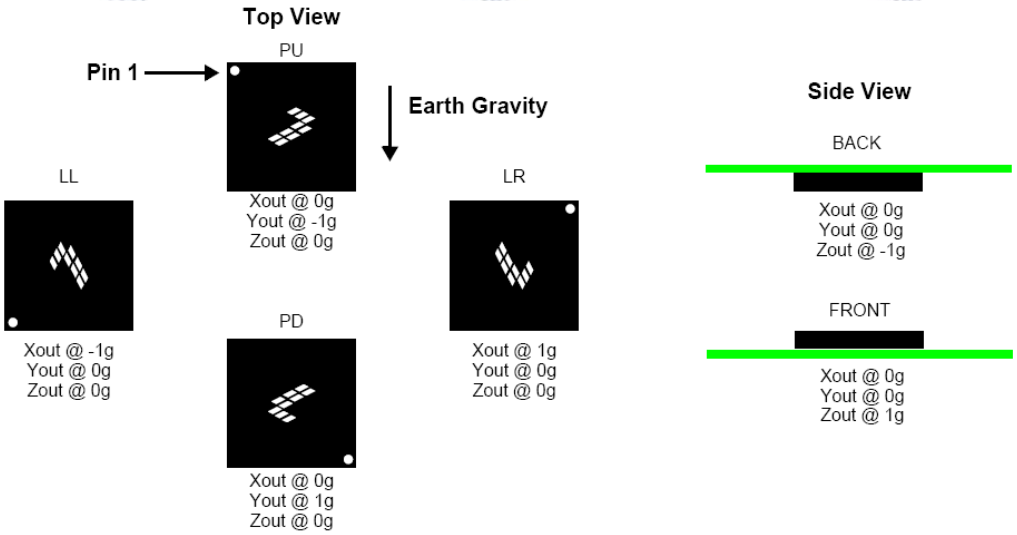


图4-11　方向说明

如上图显示了 6 个不同方向模式的设备配置，加速度传感器能够检测到所有 6 个方向的加速度值。在这个基础上，我们应用倾斜感应，能够得到更多的详细信息。物体倾斜是一种静态测量，重力作为输入计算的对象，可以确定物体的倾斜度。加速度传感器能够测量出从 1g 到-1g 这 180°范围内的倾斜度。如下图 4-12 所示，我们可以很容易从三个轴的加速度算出物体的倾斜度。设备检测的方向不再变化的角度称为“Z-锁定角”。我们检测的各个角度都能精确到±2°。

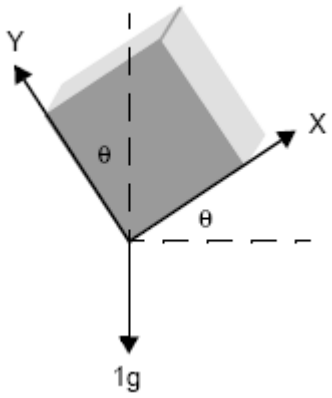


图4-12　物体倾斜度说明图

加速度传感器的应用示意图如图 4-13 所示：

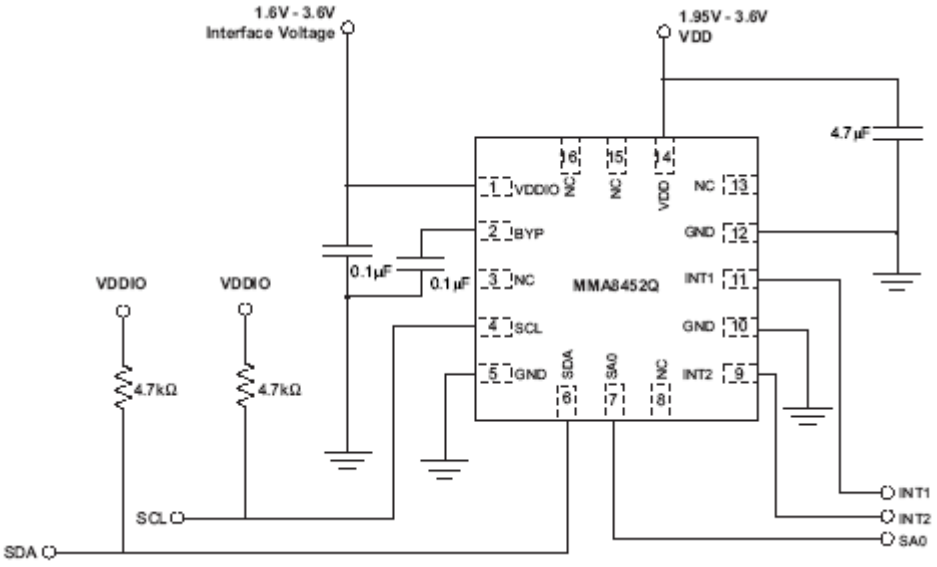


图4-13　加速度传感器应用图

本实验程序主要完成的工作包括：首先，加速度传感器测量加速度，并通过 I2C 传输数据到主处理器；然后，主处理器读出相应的加速度的数据，并把相关数据发送到 PC；最后，通过 PC 获取采集到的加速度数据。为了便于理解，将实验内容分为 2 个部分：各模块的初始化，加速度传感器测量加速度并通过 I2C 传输到主处理器。下面分别对以上内容进行介绍。

1.　各模块的初始化

0x2A 是 CTRL_REG1 寄存器（如图 4-14），除了待机模式的选择，设备必须在待机模式改变所有领域的 CTRL_REG1(0x2A)。

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ASLP_RATE1	ASLP_RATE0	DR2	DR1	DR0	LNOISE	F_READ	ACTIVE

图4-14　CTRL_REG1 寄存器

0x0E 是 XYZ_DATA_CFG 寄存器（如图 4-15），它设置动态范围和高通滤波器的输出数据，当 HPF_OUT 被设置时，当 0x01——0x6 中的数据包含高通滤波数据时，此位被设置。

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	HPF_OUT	0	0	FS1	FS0

图4-15　XYZ_DATA_CFG 寄存器

传感器数值与寄存器对应关系如下图 4-16：

Name	Type	Register Address	Auto-Increment Address		Default	Hex Value	Comment
			F_READ=0	F_READ=1			
OUT_X_MSB(1)(2)	R	0x01	0x02	0x03	Output	—	[7:0] are 8 MSBs of 12-bit sample.
OUT_X_LSB(1)(2)	R	0x02	0x03	0x00	Output	—	[7:4] are 4 LSBs of 12-bit sample.
OUT_Y_MSB(1)(2)	R	0x03	0x04	0x05	Output	—	[7:0] are 8 MSBs of 12-bit sample.
OUT_Y_LSB(1)(2)	R	0x04	0x05	0x00	Output	—	[7:4] are 4 LSBs of 12-bit sample.
OUT_Z_MSB(1)(2)	R	0x05	0x06	0x00	Output	—	[7:0] are 8 MSBs of 12-bit sample.
OUT_Z_LSB(1)(2)	R	0x06	0x00		Output	—	[7:4] are 4 LSBs of 12-bit sample.

图4-16　对应关系

```
void MMA845x_Init (void)
{
  MMA845x_Standby();
  /*
  **  Configure sensor for:
  **      - Sleep Mode Poll Rate of 50Hz (20ms)
  **      - System Output Data Rate of 200Hz (5ms)
  **      - Full Scale of +/-2g
  */
  Write_Data(CTRL_REG1, ASLP_RATE_20MS+DATA_RATE_5MS);
  Write_Data(XYZ_DATA_CFG_REG, FULL_SCALE_2G);//设置量程为 2G
}
```

2. 加速度传感器测量加速度并通过 I2C 传输到主处理器

```
void Read_Data(char reg, char *data)
{
    I2C_Start();
    WriteI2CByte(MMA8452_I2C_WRITE);
    while(Check_Acknowledge() == FALSE);
    WriteI2CByte(reg);
    while(Check_Acknowledge() == FALSE);
    I2C_Start();
    WriteI2CByte(MMA8452_I2C_READ);
    while(Check_Acknowledge() == FALSE);
```

```
    *data = ReadI2CByte();
    I2C_Stop();
}
void Write_Data(char reg, char data)
{
    I2C_Start();
    WriteI2CByte(MMA8452_I2C_WRITE);
    while(Check_Acknowledge() == FALSE);
    WriteI2CByte(reg);
    while(Check_Acknowledge() == FALSE);
    WriteI2CByte(data);
    while(Check_Acknowledge() == FALSE);
    I2C_Stop();
}
```

以上是此实验的各个模块的分析，下面介绍 main()函数的主要代码：

```
void main(void)
{
  char data_result[7] = {0x55,0x55,0x55,0x55,0x55,0x55,0x55};
  MMA845x_Init ();
  MMA845x_Active ();                                    //将控制位至 1
  Read_Data(XYZ_DATA_CFG_REG, &data_result[0]);
  while(1)
  {
      Read_Data(MMA845x_OUT_X_MSB, &data_result[1]);   //X 轴值
      Read_Data(MMA845x_OUT_X_LSB, &data_result[2]);
      Read_Data(MMA845x_OUT_Y_MSB, &data_result[3]);   //Y 轴值
      Read_Data(MMA845x_OUT_Y_LSB, &data_result[4]);
      Read_Data(MMA845x_OUT_Z_MSB, &data_result[5]);   //Z 轴值
      Read_Data(MMA845x_OUT_Z_LSB, &data_result[6]);
  }
```

```
}
```

main()函数通过 I2C 协议的读写函数得到加速度传感器采集到的 XYZ 三轴加速度，存放到 data_result 数组中，由芯片文档的计算方法最终换算得到加速度值。

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将加速度传感器模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\02-WSN\02-传感器实验代码\32_Acceleration）。打开串口调试软件，选择相应的串口，并设置好波特率 9600。观察串口得到的数据。

3. 设置断点进行调试，查看寄存器值，计算加速度值。

计算方法如下：加速度值的转换如下图 4-17 所示，以 X 轴加速度值为例：采集到的 X 轴值为两个字节（寄存器地址为：0x01，0x02），有效位数为 12 位，丢弃低四位二进制数据。另外，加速度值为 2 的补码数，数值超过 0x800 时，表明该值为负值。应该进行转换。此值先减 1，再进行按位取反，得到某负数的绝对值，其对应负数值 = 数值 - 0x1000。如果量程为±2g，则最后结果 = 测量值 * 0.001g，如果量程为±4g，则最后结果 = 测量值 * 0.002g，如果量程为±8g，则最后结果 = 测量值 * 0.0039g。

12-bit Data	Range ±2g (1 mg)	Range ±4g (2 mg)	Range ±8g (3.9 mg)
0111 1111 1111	1.999g	+3.998g	+7.996g
0111 1111 1110	1.998g	+3.996g	+7.992g
…	…	…	…
0000 0000 0001	0.001g	+0.002g	+0.004g
0000 0000 0000	0.0000g	0.0000g	0.0000g
1111 1111 1111	-0.001g	-0.002g	-0.004g
…	…	…	…
1000 0000 0001	-1.999g	-3.998g	-7.996g
1000 0000 0000	-2.0000g	-4.0000g	-8.0000g
8-bit Data	**Range ±2g (15.6 mg)**	**Range ±4g (31.25 mg)**	**Range ±8g (62.5 mg)**
0111 1111	1.9844g	+3.9688g	+7.9375g
0111 1110	1.9688g	+3.9375g	+7.8750g
…	…	…	…
0000 0001	+0.0156g	+0.0313g	+0.0625g
0000 0000	0.000g	0.0000g	0.0000g
1111 1111	-0.0156g	-0.0313g	-0.0625g
…	…	…	…
1000 0001	-1.9844g	-3.9688g	-7.9375g
1000 0000	-2.0000g	-4.0000g	-8.0000g

图4-17 加速度值转换图

六、 拓展思考

加速度传感器可以应用在哪些场合？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\Sensor\Acceleration”目录下的MMA8452Q.pdf 数据手册。

4.2.4 实验三十三 气压传感器实验

一、 实验目的

1.熟悉压力传感器的工作原理。

2.掌握压力传感器的使用方法。

二、 实验设备

- ZigBee模块
- 气压传感器（EBS-PR（essure））
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境，利用压力传感器测量周围大气压和温度。

四、 实验原理

我们实验使用的 MPL3115A2 型传感器属于智能数字压力传感器（如图 4-18）。它基于微机电系统（MEMS）技术，可以在本地处理压力和温度数据，减少了分配给应用处理器的计算量。因此，与使用由主机处理器直接管理的基本传感器的系统相比，这种压力传感器所消耗的功耗更少。该压力传感器采用 FIFO（先进/先出）内存缓冲、2 微安的待机模式和 8 微安的低功率模式，减少电流消耗，实现了最优效率，具体取决于处理器条件和所选的输出数据速率选择。

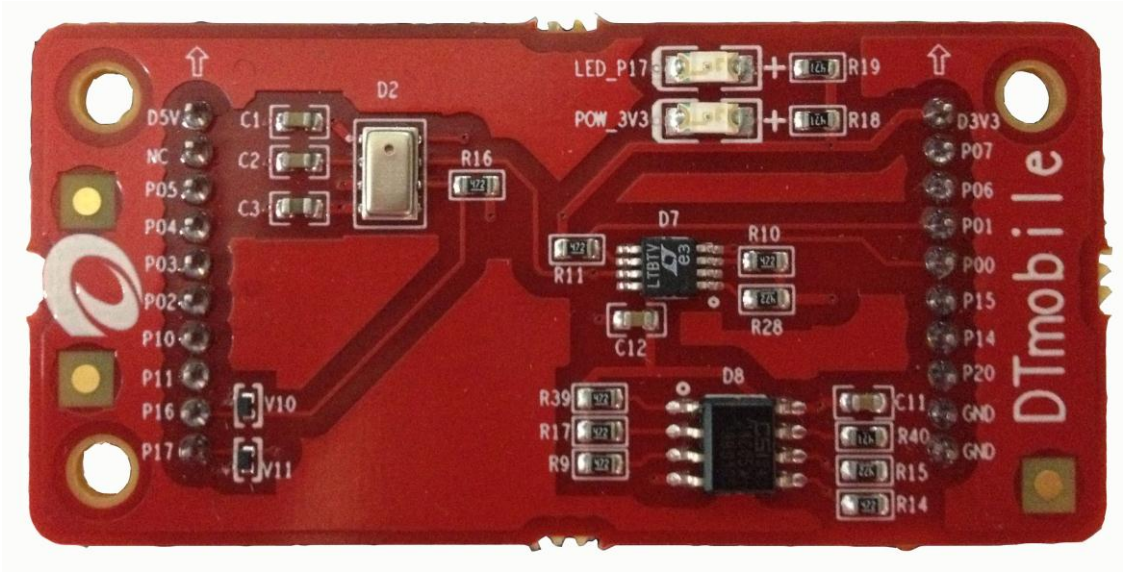


图4-18　　气压传感器板卡

MPL3115A2 压力传感器结合了高精度、高采样频率和超低功耗特性，进一步提高了性能。该器件提供了气压和高度压力检测，支持高达 30cm 的分辨率，可根据用户偏好使用米或帕斯卡为单位输出数据。MPL3115A2 传感器还包含嵌入式功能和用户可编程选项，比如温度补偿，采样频率可高达 128Hz。下图 4-19 是压力传感器的内部框图。

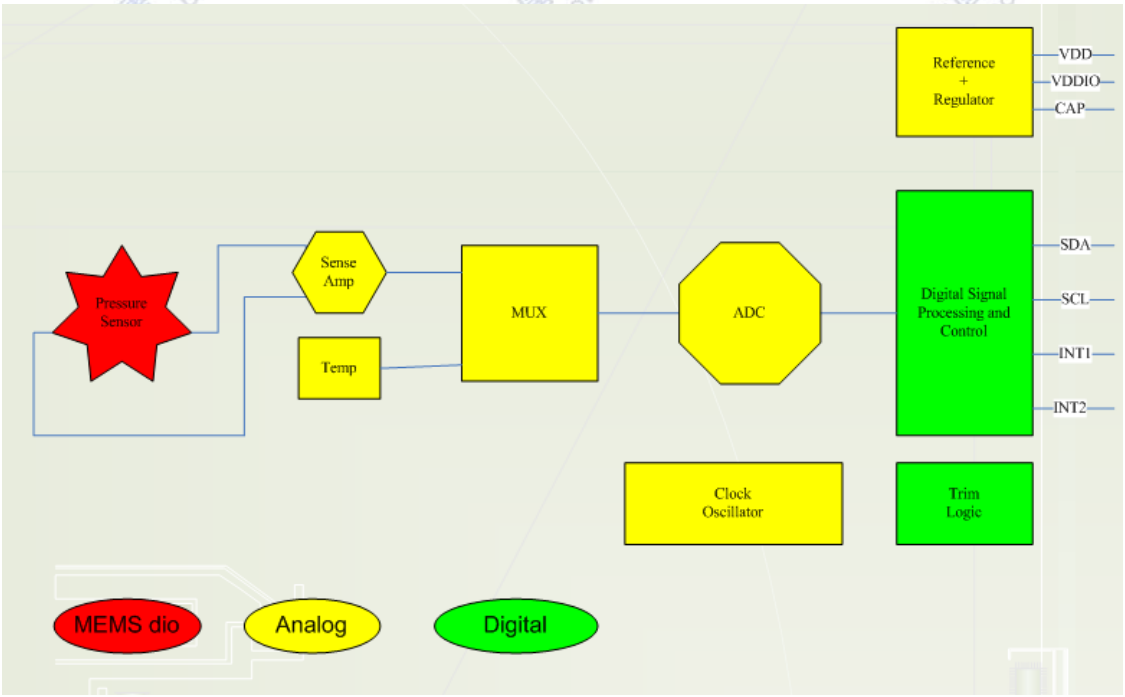


图4-19　　MPL3115A2 框图

压力传感器的原理图如下图 4-20 所示：

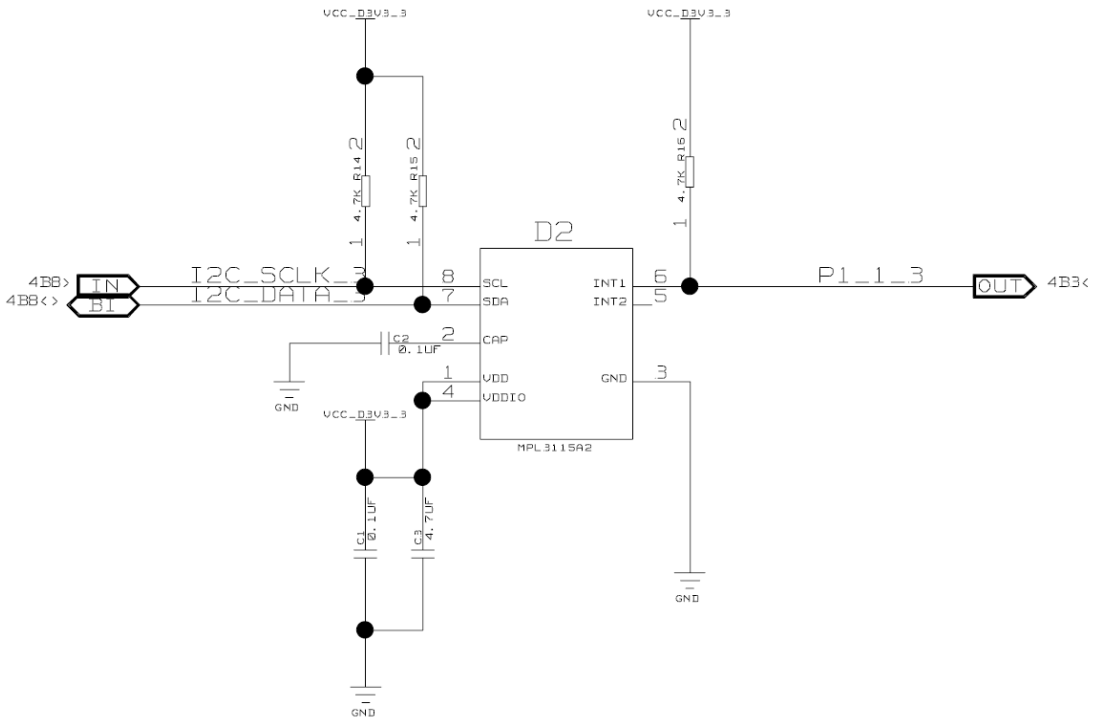


图4-20　　原理图

本实验程序主要完成的工作包括：首先，压力传感器测量周围大气压，并通过 I2C 传输数据到主处理器；然后，主处理器读出相应的大气压的数据；最后，通过 PC 获取采集到的大气压数据。为了便于理解，将实验内容分为 2 个部分：各模块的初始化，压力传感器测量周围大气压并通过 I2C 传输到主处理器。下面分别对以上内容进行介绍。

1.　各模块的初始化

0x01 是 OUT_P_MSB 寄存器。0x02 是 OUT_P_CSB 寄存器。0x03 是 OUT_P_LSB 寄存器。0x04 是 OUT_T_MSB 寄存器。0x05 是 OUT_T_LSB 寄存器。压力/海拔高度和 12 位的温度采样数据被表示为 2 的补数。这五个寄存器和 DR_STATUS 寄存器存储在 0x01 到 0x6 的自增地址中，以减少读取 6 个字节的由 20 位压力和 12 位温度数据。5 个寄存器所下图 4-21 至图 4-25 所示。

	7	6	5	4	3	2	1	0
R	PD19	PD18	PD17	PD16	PD15	PD14	PD13	PD12
W								
Reset	0	0	0	0	0	0	0	0

图4-21　　0x01

	7	6	5	4	3	2	1	0
R	PD11	PD10	PD9	PD8	PD7	PD6	PD5	PD4
W								
Reset	0	0	0	0	0	0	0	0

图4-22　　0x02

	7	6	5	4	3	2	1	0
R	PD3	PD2	PD1	PD0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

图4-23 0x03

	7	6	5	4	3	2	1	0
R	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4
W								
Reset	0	0	0	0	0	0	0	0

图4-24 0x04

	7	6	5	4	3	2	1	0
R	TD3	TD2	TD1	TD0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

图4-25 0x05

0x06 是 DR_STATUS 寄存器。状态寄存器在每个样本基础上提供采集状态信息，并实时更新反映到 OUT_P 和 OUT_T 寄存器。寄存器如下图 4-26 所示：

	7	6	5	4	3	2	1	0
R	PTOW	POW	TOW	0	PTDR	PDR	TDR	0
W								
Reset	0	0	0	0	0	0	0	0

图4-26 0x06

0x07 是 OUT_P_DELTA_MSB 寄存器。它存储 FIFO 的所有数据输出的寄存器驱动状态的数据信息。寄存器如图 4-27 所示：

	7	6	5	4	3	2	1	0
R	PDD19	PDD18	PDD17	PDD16	PDD15	PDD14	PDD13	PDD12
W								
Reset	0	0	0	0	0	0	0	0

图4-27 0x07

传感器数值与寄存器对应关系如下图 4-28 所示：

Register Address	Name	Reset	Reset when STBY to Active	Type	Auto-Increment Address		Comment	
0x01	Pressure Data Out MSB (**OUT P_MSB**)(1)(2)	0x00	Yes	R	0x02	0x01	Bits 12-19 of 20-bit real-time Pressure sample.	Root pointer to Pressure and Temperature FIFO data.
0x02	Pressure Data Out CSB (**OUT_P_CSB**)(1)(2)	0x00	Yes	R	0x03		Bits 4-11 of 20-bit real-time Pressure sample	
0x03	Pressure Data Out LSB (**OUT_P _LSB**)(1)(2)	0x00	Yes	R	0x04		Bits 0-3 of 20-bit real-time Pressure sample	
0x04	Temperature Data Out MSB (**OUT_T_MSB**)(1)(2)	0x00	Yes	R	0x05		Bits 4-11 of 12-bit real-time Temperature sample	
0x05	Temperature Data Out LSB (**OUT_T _LSB**)(1)(2)	0x00	Yes	R	0x00		Bits 1-3 of 12-bit real-time Temperature sample	

图4-28　对应关系

```
void Init_IO_INT(void)
{
    P1SEL &= 0xFD; //P11 作为普通 IO
    P1DIR &= 0xFD; //P11 作为输入
    P1INP &= 0xFD;//P11 有上拉、下拉
    PICTL |= 0X02; //下降沿
    EA = 1;
    IEN2 |= 0X10; // P1IE = 1;
    P1IEN |= 0X02; //使能 P1_1 中断
    P1IF = 0;   //清中断标志
    P1IFG = 0; //清中断标志
}
__interrupt void P1_ISR(void)
{
    if((P1IFG & 0x02) != 0) //P1_1 中断
    {
        P1IFG &= 0xFD;
        Read_Data(0x02, &INT_FLAG[0]);
        Write_Data(0x02, INT_FLAG[0] & 0x77);
        Read_Data(0x08, &data_result[1]);
        Read_Data(0x09, &data_result[2]);
        Read_Data(0x0A, &data_result[3]);
    }
    P1IFG = 0;//清中断标志
    P1IF = 0; //清中断标志
}
```

2. 压力传感器测量周围大气压并通过 I2C 传输到主处理器

```
void Read_Data(char reg, char *data)
{
```

```
        I2C_Start();
        WriteI2CByte(MPL3115A2_I2C_ADDRESS + I2C_WRITE);
        while(Check_Acknowledge() == FALSE);
        WriteI2CByte(reg);
        while(Check_Acknowledge() == FALSE);
        I2C_Start();
        WriteI2CByte(MPL3115A2_I2C_ADDRESS + I2C_READ);
        while(Check_Acknowledge() == FALSE);
        *data = ReadI2CByte();
        I2C_Stop();
}
void Write_Data(char reg, char data)
{
        I2C_Start();
        WriteI2CByte(MPL3115A2_I2C_ADDRESS + I2C_WRITE);
        while(Check_Acknowledge() == FALSE);
        WriteI2CByte(reg);
        while(Check_Acknowledge() == FALSE);
        WriteI2CByte(data);
        while(Check_Acknowledge() == FALSE);
        I2C_Stop();
}
```

以上是此实验的各个模块的分析，下面介绍 main()函数的主要代码：

```
void main(void)
{
   Write_Data(0x26, 0x39);
   Read_Data(0x26, &data_result[0]);
   while(1)
   {
      Read_Data(0x01, &data_result[1]);
```

```
        Read_Data(0x02, &data_result[2]);
        Read_Data(0x03, &data_result[3]);
        Read_Data(0x04, &data_result[4]);
        Read_Data(0x05, &data_result[5]);
      }
    }
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将温度传感器模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\02-WSN\02-传感器实验代码\33_Pressure）。打开串口调试软件，选择相应的串口，并设置好波特率 9600。观察串口得到的数据。

3. 设置断点进行调试，查看寄存器值，计算气压值。

气压的数据转换计算方法：由气压传感器芯片数据手册 MPL3115A2.pdf 中 6Register Descriptions—table8（即本节图 4-28）可知气压值表达为三个字节（寄存器地址为 0x01，0x02，0x03），假如在线调试时这三个寄存器的值分别为 0x61，0xF8，0xE0，转换时丢弃最后四位二进制位，将这三个字节组合成 0x61F8E，即 20 位二进制位。 0x61F8E = 401294，此值 401294 * 0.25 = 100323.5（帕斯卡）。

用压力传感器测温度时，数据转换方法：温度值表达为两个字节（寄存器地址为 0x04，0x05），假如在线调试时这三个寄存器的值分别为 0x1C，0x50。转换时丢弃最后四位二进制位，将这三个字节组合成 0x1C5，即 12 位二进制位。0x1C5 = 453，此值 453 * 0.0625 = 28.3125℃ 。

六、 拓展思考

气压传感器可以应用在哪些场合？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\Sensor\Pressure”目录下的 MPL3115A2.pdf 数据手册。

4.2.5 实验三十四 光传感器实验

一、 实验目的

1.熟悉光传感器的工作原理。

2.掌握利用光传感器的使用方法。

二、 实验设备

- ZigBee模块
- 光传感器（EBS-T）
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境，利用光传感器测量感应周围光的强度，继而进行工作。光传感器和温湿度传感器集成在同一块传感器板卡上（EBS-T）。

四、 实验原理

我们实验用的 ISL29028A 光传感器有接近感应和环境光感应两种功能。光传感器已经集成到温湿度传感器板卡上，如图 4-29 所示：

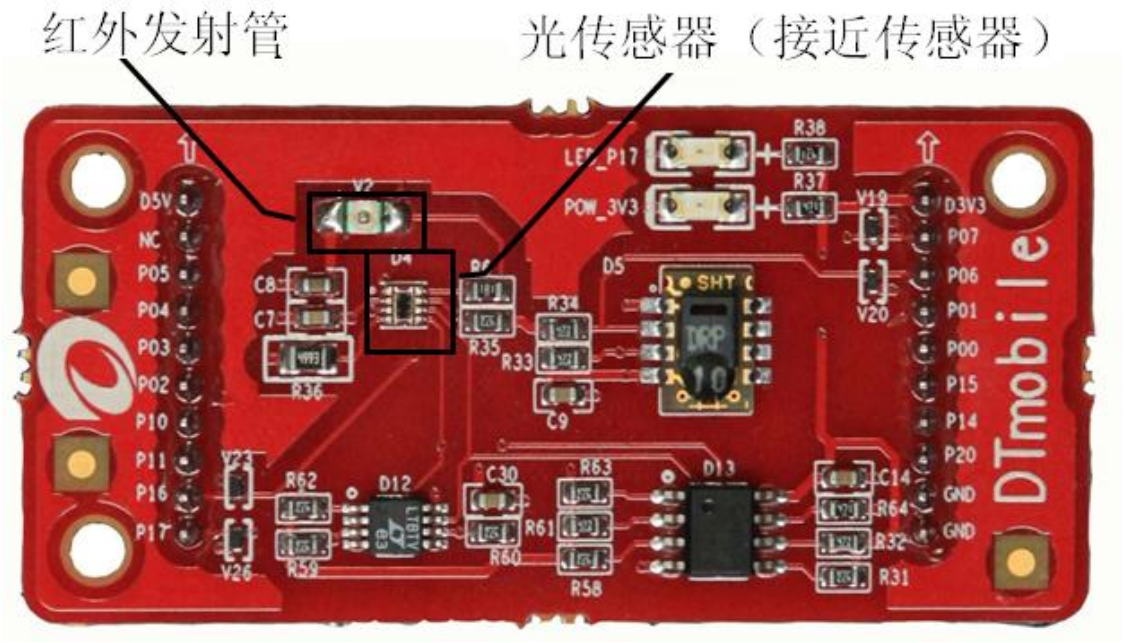


图4-29 温湿度传感器板卡（已集成光传感器）

如图 4-30 所示的 ISA29028A 内部框图，我们可以看到其内部实际上有两个独立的光电二极管，并且有双路 ADC。

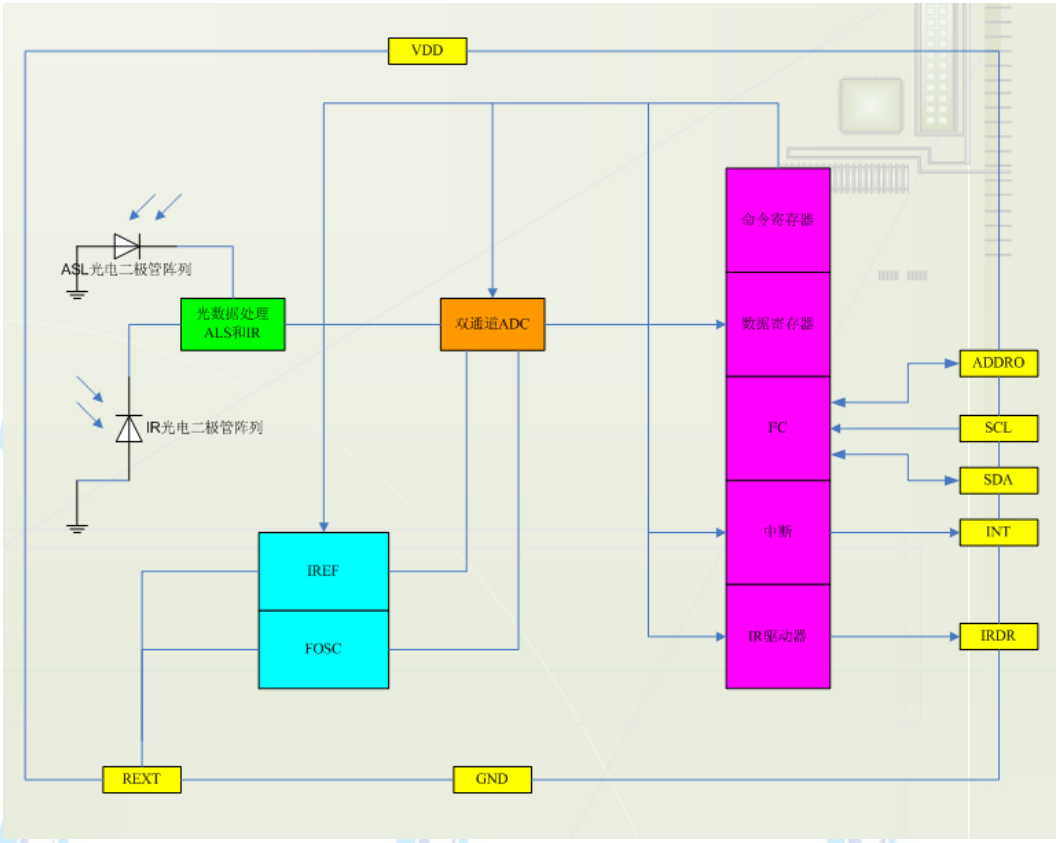


图4-30　ISL29028A 内部框图

ISL29028A 的典型电路如图 4-31 所示：

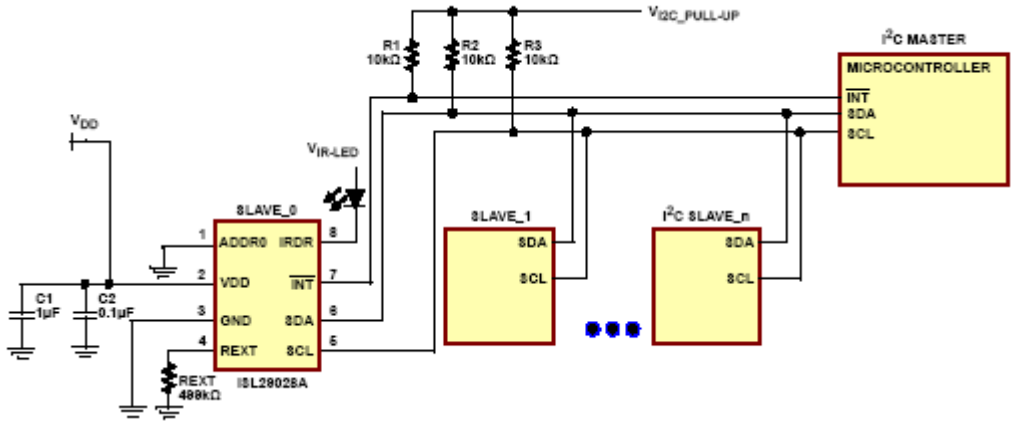


图4-31　ISL29028A 的典型电路

接近传感器的工作原理非常简单：它本质上是一个光电二极管，在其旁边放置一个红外光波长的 LED。当有物体靠近时，红外光发射的光会被物体反射回来，被接近传感器接收到，于是就感应到了物体接近。如图 4-32 所示。

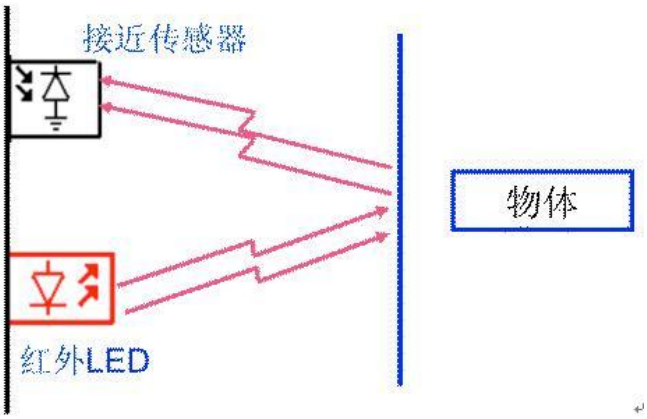


图4-32　接近传感器工作原理

本实验程序主要完成的工作包括：首先，光传感器感应光照强度，并通过 I2C 传输数据到主处理器；然后，主处理器读出相应的光照强度的数据；最后，通过 PC 获取采集到的光照强度数据。为了便于理解，将实验内容分为 2 个部分：各模块的初始化，光传感器感应周围光照强度并通过 I2C 传输到主处理器。下面分别对以上内容进行介绍。

1. 各模块的初始化

下面介绍一下 0x01——0x07 寄存器（如图 4-33—图 4-39）：

BIT #	ACCESS	DEFAULT	NAME	FUNCTION/OPERATION
7	RW	0x00	PROX_EN (Prox Enable)	When = 0, proximity sensing is disabled When = 1, continuous proximity sensing is enabled. Prox data will be ready 0.54ms after this bit is set high
6:4	RW	0x00	PROX_SLP (Prox Sleep)	For bits 6:4 = (see the following) 111; sleep time between prox IR LED pulses is 0.0ms (run continuously) 110; sleep time between prox IR LED pulses is 12.5ms 101; sleep time between prox IR LED pulses is 50ms 100; sleep time between prox IR LED pulses is 75ms 011; sleep time between prox IR LED pulses is 100ms 010; sleep time between prox IR LED pulses is 200ms 001; sleep time between prox IR LED pulses is 400ms 000; sleep time between prox IR LED pulses is 800ms
3	RW	0x00	PROX_DR (Prox Drive)	When = 0, IRDR behaves as a pulsed 110mA current sink When = 1, IRDR behaves as a pulsed 220mA current sink
2	RW	0x00	ALS_EN (ALS Enable)	When = 0, ALS/IR sensing is disabled When = 1, continuous ALS/IR sensing is enabled with new data ready every 100ms
1	RW	0x00	ALS_RANGE (ALS Range)	When = 0, ALS is in low-lux range When = 1, ALS is in high-lux range
0	RW	0x00	ALSIR_MODE (ALSIR Mode)	When = 0, ALS/IR data register contains visible ALS sensing data When = 1, ALS/IR data register contains IR spectrum sensing data

图4-33　0x01

BIT #	ACCESS	DEFAULT	BIT NAME	FUNCTION/OPERATION
7	FLAG	0x00	PROX_FLAG (Prox Flag)	When = 0, no Prox interrupt event has occurred since power-on or last "clear" When = 1, a Prox interrupt event occurred. Clearable by writing "0"
6:5	RW	0x00	PROX_PRST (Prox Persist)	For bits 6:5 = (see the following) 00; set PROX_FLAG if 1 conversion result trips the threshold value 01; set PROX_FLAG if 4 conversion results trip the threshold value 10; set PROX_FLAG if 8 conversion results trip the threshold value 11; set PROX_FLAG if 16 conversion results trip the threshold value
4	RW	0x00	*Unused* *(Write 0)*	Unused register bit - write 0
3	FLAG	0x00	ALS_FLAG (ALS FLAG)	When = 0, no ALS interrupt event has occurred since power-on or last "clear" When = 1, an ALS interrupt event occurred. Clearable by writing "0"
2:1	RW	0x00	ALS_PRST (ALS Persist)	For bits 2:1 = (see the following) 00; set ALS_FLAG if 1 conversion is outside the set window 01; set ALS_FLAG if 4 conversions are outside the set window 10; set ALS_FLAG if 8 conversions are outside the set window 11; set ALS_FLAG if 16 conversions are outside the set window
0	RW	0x00	INT_CTRL (Interrupt Control)	When = 0, set $\overline{\text{INT}}$ pin low if PROX_FLAG or ALS_FLAG high (logical OR) When = 1, set $\overline{\text{INT}}$ pin low if PROX_FLAG and ALS_FLAG high (logical AND)

图4-34 0x02

BIT #	ACCESS	DEFAULT	BIT NAME	FUNCTION/OPERATION
7:0	RW	0x00	PROX_LT (Prox Threshold)	8-bit interrupt low threshold for proximity sensing

图4-35 0x03

BIT #	ACCESS	DEFAULT	BIT NAME	FUNCTION/OPERATION
7:0	RW	0xFF	PROX_HT (Prox Threshold)	8-bit interrupt high threshold for proximity sensing

图4-36 0x04

BIT #	ACCESS	DEFAULT	BIT NAME	FUNCTION/OPERATION
7:0	RW	0x00	ALSIR_LT[7:0] (ALS/IR Low Thr.)	Lower 8 bits (of 12 bits) for ALS/IR low interrupt threshold

图4-37 0x05

BIT #	ACCESS	DEFAULT	BIT NAME	FUNCTION/OPERATION
7:4	RW	0x0F	ALSIR_HT[3:0] (ALS/IR High Thr.)	Lower 4 bits (of 12 bits) for ALS/IR high interrupt threshold
3:0	RW	0x00	ALSIR_LT[11:8] (ALS/IR Low Thr.)	Upper 4 bits (of 12 bits) for ALS/IR low interrupt threshold

图4-38 0x06

BIT #	ACCESS	DEFAULT	BIT NAME	FUNCTION/OPERATION
7:0	RW	0xFF	ALSIR_HT[11:4] (ALS/IR High Thr.)	Upper 8 bits (of 12 bits) for ALS/IR high interrupt threshold

图4-39 0x07

传感器数值与寄存器对应关系如下图 4-40 所示：

ADDR	REG NAME	BIT								DEFAULT
		7	6	5	4	3	2	1	0	
0x08	PROX_DATA	PROX_DATA[7:0]								0x00
0x09	ALSIR_DT1	ALSIR_DATA[7:0]								0x00
0x0A	ALSIR_DT2	*(Unused)*				ALSIR_DATA[11:8]				0x00

图4-40　对应关系

```
void Initialize(void)
{
    Write_Data(0x01, 0x94);  //使能渐进传感和环境光传感，红外灯 sleep 时间设置为100ms
    Write_Data(0x02, 0x23);  //4 次转换结果超过阈值，中断。渐进中断和环境光中断是逻辑"与"关系
    Write_Data(0x03, 0x30);
    Write_Data(0x04, 0xD0);
    Write_Data(0x05, 0x00);
    Write_Data(0x06, 0x03);
    Write_Data(0x07, 0xD0);
}
```

通过 I2C 协议的写数据函数，将数值写到气压传感器芯片寄存器中，进行初始化设置。

2. 光传感器感应周围光照强度并通过 I2C 传输到主处理器。

```
void Read_Data(char reg, char *data)
{
    I2C_Start();
    WriteI2CByte(ISL29028A_I2C_ADDRESS + I2C_WRITE);
    while(Check_Acknowledge() == FALSE);
    WriteI2CByte(reg);
    while(Check_Acknowledge() == FALSE);
    I2C_Start();
```

```
        WriteI2CByte(ISL29028A_I2C_ADDRESS + I2C_READ);
        while(Check_Acknowledge() == FALSE);
         *data = ReadI2CByte();
         I2C_Stop();
}
void Write_Data(char reg, char data)
{
        I2C_Start();
        WriteI2CByte(ISL29028A_I2C_ADDRESS + I2C_WRITE);
        while(Check_Acknowledge() == FALSE);
        WriteI2CByte(reg);
        while(Check_Acknowledge() == FALSE);
        WriteI2CByte(data);
        while(Check_Acknowledge() == FALSE);
        I2C_Stop();
}
```

以上是此实验的各个模块的分析，下面介绍 main()函数的主要代码：

```
void main(void)
{
   int i = 0;
   Init_IO_INT();
   Initialize();
   while(1)
   {
          for(i = 0; i < 100; i++)
         {
               Delay_1u(10000);
         }
   }
}
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将温湿度传感器模块（已集成光传感器）插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\02-WSN\02-传感器实验代码\34_Light）。打开串口调试软件，选择相应的串口，并设置好波特率 9600。观察串口得到的数据。

3. 设置断点进行调试，查看寄存器值，计算光感值。

红外反射光的数值转换如下：红外反射光表达为一个字节（寄存器地址为 0x08），人体离光感越近，该值越大，这个值不能进行定量计算，因为它和硬件调理电路有关，应用时，先预期一个报警距离阈值，该报警距离阈值所对应的字节值，即为报警阈值。画图时，可以直接用这个字节。

环境光光照强度转换如下：环境光表达为两个字节（寄存器地址为 0x9，0x0A），比如设置断点得到这两个寄存器的值分别为 0x03，0x8B。应用时将这两个字节组合成 0x38B，即 12 位二进制位，丢弃高四位二进制位。0x38B = 907。如果是小量程，907 * 0.0326 = 29.5682（单位：lux），如果是大量程，907 * 0.522 = 473.454（单位：lux）。这里，大量程和小量程可以通过上位机设置。

六、 拓展思考

光传感器可以应用在哪些场合?

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\Sensor\ Ambient Light & Proximit”目录 y 下的 ISL29028A.pdf 数据手册。

4.2.6 实验三十五 红外对管传感器实验

一、 实验目的

1.熟悉红外对管的工作原理。

2.掌握红外对管的使用方法。

二、 实验设备

- ZigBee模块

- 红外对管（EBS-IN（frared））
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境利用单片机 CC2531 来控制红外对管，继而利用红外对管工作。

四、 实验原理

红外对管传感器的外观图如下图 4-41 所示：

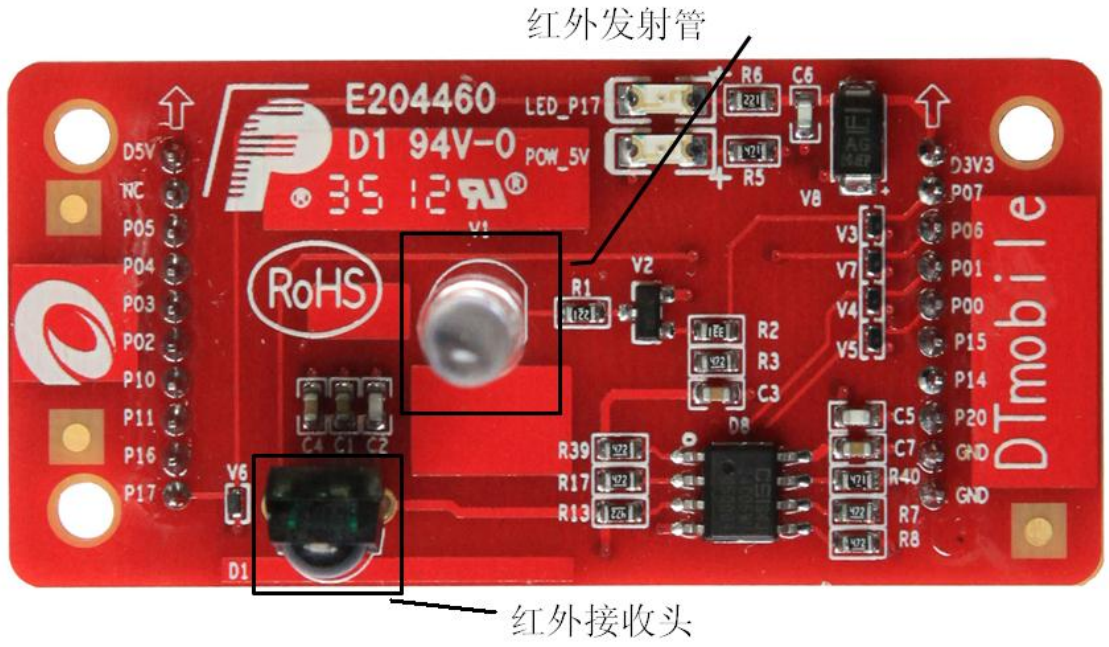


图4-41 红外传感器板卡

红外传感器工作原理并不复杂，一个典型的传感器系统各部分的实体分别是：

1. 待测目标：根据待测目标的红外辐射特性可进行红外系统的设定。

2. 大气衰减：待测目标的红外辐射通过地球大气层时，由于气体分子和各种气体以及各种溶胶粒的散射和吸收，将使得红外源发出的红外辐射发生衰减。

3. 光学接收器：它接收目标的部分红外辐射并传输给红外传感器。相当于雷达天线，常用是物镜。

4. 辐射调制器：对来自待测目标的辐射调制成交变的辐射光，提供目标方位信息，并可滤除大面积的干扰信号。又称调制盘和斩波器，它具有多种结构。

5. 红外探测器：这是红外系统的核心。它是利用红外辐射与物质相互作用所呈现出来的物理效应探测红外辐射的传感器，多数情况下是利用这种相互作用所呈现出的电学效应。此类探测器可分为光子探测器和热敏感探测器两大类型。

6. 探测器制冷器：由于某些探测器必须要在低温下工作，所以相应的系统必须有制冷设备。经过制冷，设备可以缩短响应时间，提高探测灵敏度。

7. 信号处理系统：将探测的信号进行放大、滤波，并从这些信号中提取出信息。然后将此类信息转化成为所需要的格式，最后输送到控制设备或者显示器中。

8. 显示设备：这是红外设备的终端设备。常用的显示器有示波器、显像管、红外感光材料、指示仪器和记录仪等。

红外对管传感器的工作原理图如图 4-42 所示：

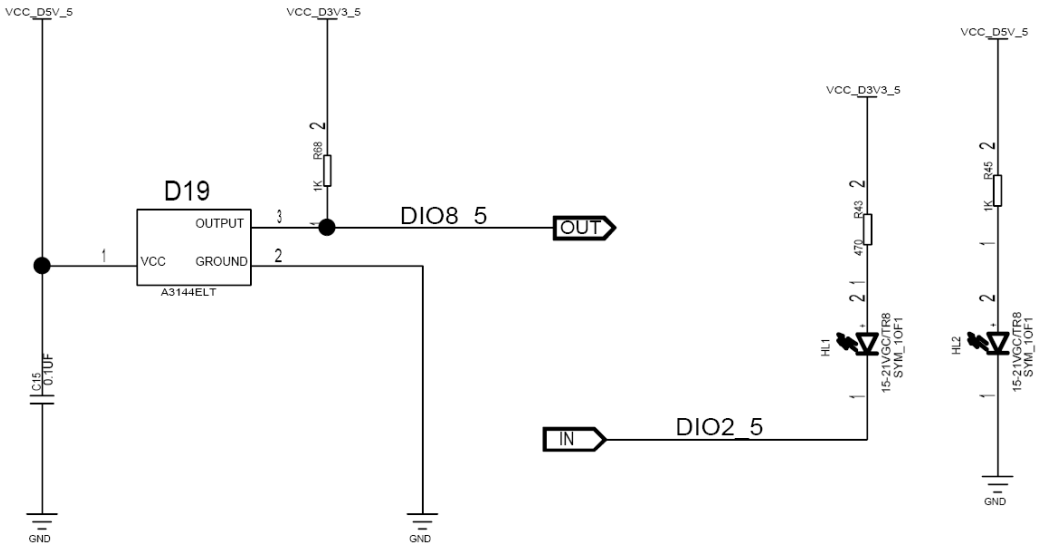


图4-42　红外对管原理图（左侧为接收头/右侧为发射管）

红外发射管：红外线发射管也称红外线发射二极管，属于二极管类。它是可以将电能直接转换成近红外 光（不可见光）并能辐射出去的发光器件。红外接收头：红外接收头是将红外线光信号变成电信号的半导体器件，它的核心部件是一个特殊材料的 PN 结，和普通的二极管相比，红外接收管 PN 结面积比较大，电极面积尽量减小，而且 PN 结的结深很浅，可以更多更大面积的接收入射光线。

红外发射模块发射 38kHz 红外线，当没有物体阻断时，红外接收模块接收到 38kHz 红外信号并转换成电信号，再根据此电信号做相应处理。当有物体阻断时，红外接收模块无法接收红外信号。

红外发射管发射 38kHz 频率的红外线，即一个高低电平的周期为 26μm，根据红外接收

管芯片数据手册（HS0038B.pdf）第 3 页 Suitable Data Format 描述，红外接收管接收到的脉冲要求至少有 20 个周期，之后有一段低电平的时间（10~70 倍周期），连续发送，脉冲波形如图 4-43 所示：

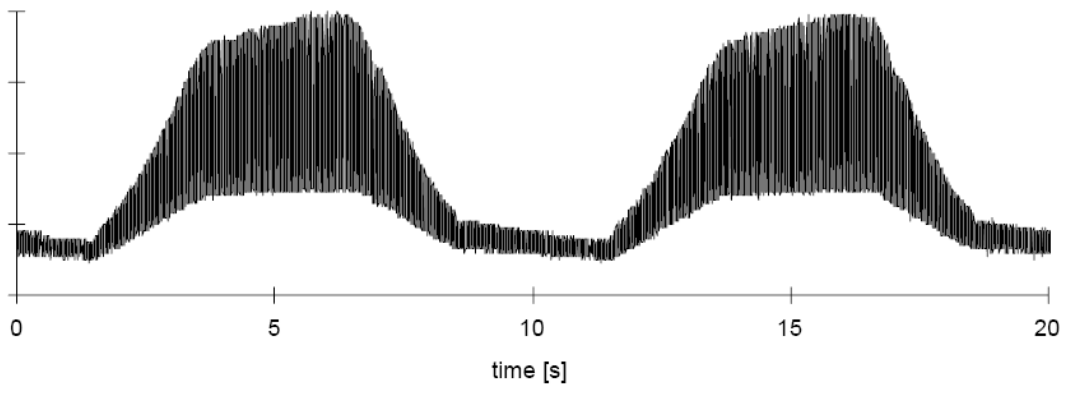


图4-43　红外接收管接收脉冲示意图

通过 2531 芯片的 T3 定时器来实现这个脉冲，函数如下：

```
#pragma vector = T3_VECTOR
__interrupt void T3_ISR(void)
{
IRCON &= 0xFD;  //清除 OVFIF
    T3IE = 0;
    counter++;
    if(counter < 40)
    {
        emit = !emit;
    }
    else if(counter == 40)
    {
        write_emit_0();
    }
    else if(counter == 80)
    {
        counter = 0;                    //计数清零
    }
```

```
    T3IE = 1;
}
```

红外接收头在接收到 38kHz 红外线时输出 1ms 方波，在未接收到红外线输出高电平，通过 T3 定时器在一定时间内检测输出管脚的高低电平变化，来判断红外接收管是否接收到红外信号。函数如下：

```
#pragma vector = T3_VECTOR
__interrupt void T3_ISR(void)
{
    IRCON &= 0xFD;
    T3IE = 0;
    counter++;
    if(counter == 5)
    {
        counter = 0;
        timer_counter++;
        F0 = RECEIVE;
        if(F0 == 1)
        {
            receiver_counter++;
        }
        else
        {
        }
        if(timer_counter == 10)
        {
            if(receiver_counter >= 8)
            {
                warn_flag = 1;
            }
```

```
                else
                {
                }
                timer_counter = 0;
                receiver_counter = 0;
            }
            else
            {
            }
        }
        T3IE = 1;
    }
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将红外对管传感器模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\02-WSN\02-传感器实验代码\35_Infrared）。把红外发射和红外接收的代码下载到两个 ZigBee 模块，一个作为红外发射，另一个作为红外接收。

3. 观察实验现象：红外接收板卡的 LED_P17 小灯不停闪烁表示没有接收到红外信号。将一块板卡的红外发射管对准另一个红外接收板卡的红外接收头（距离不要超过 30cm，角度小于 5°），如果红外接收头接收到红外线，则小灯亮。用手阻挡于红外发射管和红外接收头之间，小灯则不停闪烁。

六、 拓展思考

红外对管可以应用在哪些场合？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\Sensor\ Infrared”目录下的数据手册。

4.2.7 实验三十六 继电器控制实验

一、 实验目的

1.熟悉继电器的工作原理。

2.掌握利用单片机控制继电器的方法。

二、 实验设备

- ZigBee模块
- 红外对管（EBS-R(elay)）
- JTAG仿真器
- 电源适配器
- IAR 集成开发环境
- 串口调试软件
- 串口线

三、 实验内容

本实验使用 IAR Embedded Workbench 环境利用单片机 CC2531 来控制 G6K 继电器，继而利用继电器工作。

四、 实验原理

1. 继电器的定义

继电器（relay）是指当输入量(激励量)的变化达到规定要求时，在电气输出电路中使被控量发生预定的阶跃变化的一种电器。

继电器是一种电控制器件。它具有控制系统（又称输入回路）和被控制系统（又称输出回路）之间的互动关系。通常应用于自动化的控制电路中，它实际上是用小电流去控制大电流运作的一种“自动开关”。故在电路中起着自动调节、安全保护、转换电路等作用。继电器具有动作快、工作稳定、使用寿命长、体积小等优点。广泛应用于电力保护、自动化、运动、遥控、测量和通信等装置中。

继电器的输入量可分为电气量（如电流、电压、频率、功率等）及非电气量（如温度、压力、速度等）两大类。当输入量（如电压、电流、温度等）达到规定值时，继电器使被控制的输出电路导通或断开。

继电器的外观图如下图 4-44 所示：

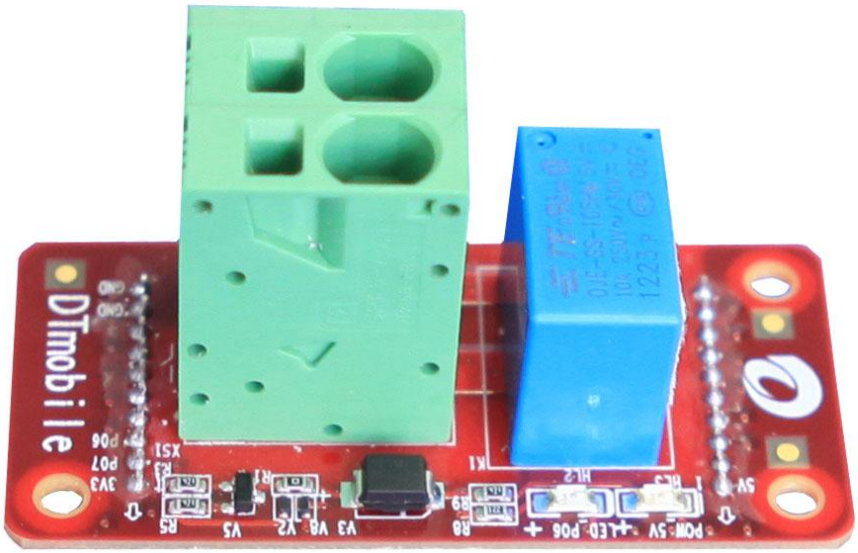


图4-44　继电器板卡

2. 继电器的作用

继电器是具有隔离功能的自动开关元件，广泛应用于遥控、遥测、通讯、自动控制、机电一体化及电力电子设备中，是最重要的控制元件之一。

继电器一般都有能反映一定输入变量（如电流、电压、功率、阻抗、频率、温度、压力、速度、光等）的感应机构（输入部分）；有能对被控电路实现“通”、“断”控制的执行机构（输出部分）；在继电器的输入部分和输出部分之间，还有对输入量进行耦合隔离，功能处理和对输出部分进行驱动的中间机构（驱动部分）。

作为控制元件，概括起来，继电器有如下几种作用：

- 扩大控制范围：例如，多触点继电器控制信号达到某一定值时，可以按触点组的不同形式，同时换接、开断、接通多路电路。
- 放大：例如，灵敏型继电器、中间继电器等，用一个很微小的控制量，可以控制很大功率的电路。
- 综合信号：例如，当多个控制信号按规定的形式输入多绕组继电器时，经过比较综合，达到预定的控制效果。
- 自动、遥控、监测：例如，自动装置上的继电器与其他电器一起，可以组成程序控制线路，从而实现自动化运行。

3. 继电器的工作原理

我们实验使用的是 G6K-2F 继电器。它的端子配置/内部连接图如下图 4-45：

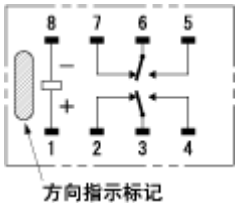


图4-45　内部链接图

当 1、8 电路不通时，2、3 为常闭，3、4 为常开；6、7 为常闭，6、5 为常开。

当 1、8 电路导通时，2、3 为常闭断开，3、4 为常开闭合；6、7 为常闭断开，6、5 为常开闭合。

我们实验就是利用 ZigBee 模块，控制继电器的开关转换，继而利用继电器来工作的。下面是继电器的内部原理图，如图 4-46：

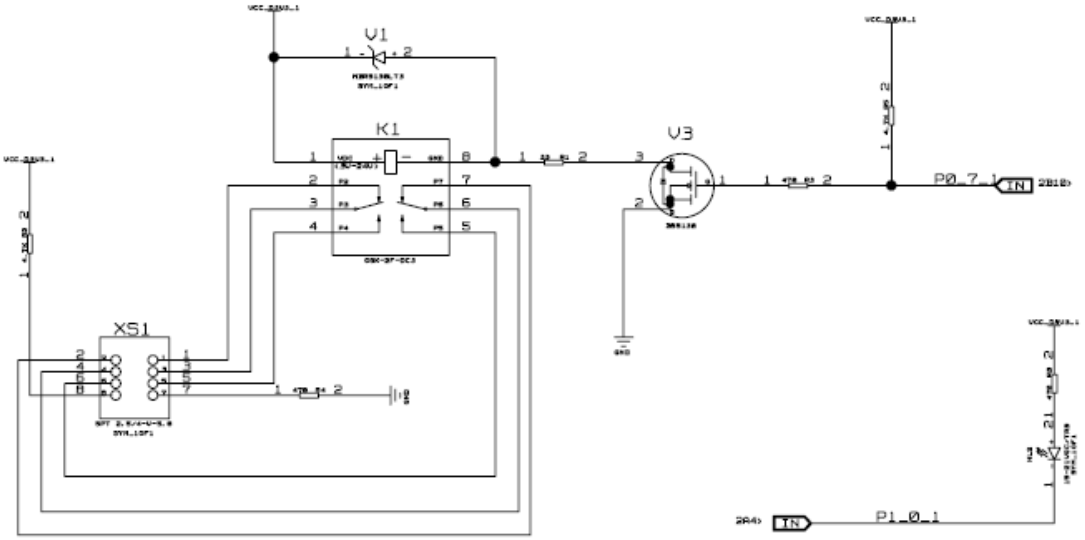


图4-46　原理图

ZigBee 模块通过 P0_7 输入高电平，使得三极管发射极和集电极处导通，这样，R 两端 1、8 导通，继电器内部连接开关状态改变，然后继电器开始工作。本实验设计为通过按键来控制继电器的通断，按下按键 K1，继电器闭合红灯亮，按下按键 K2，继电器断开黄色等亮。有关按键程序可参考实验二内容。

下面是 main（）函数的主要代码：

```
void main(void)
{
    Initial();        //调用初始化函数
    InitKey();
    Delay(4000);
```

```
    P0DIR |= 0x80;
    YLED = ON;
    relay=0;      //初始化状态继电器输出 0，断开
    while(1)
    {
        Keyvalue = KeyScan(); //扫键
        if(Keyvalue>0)
        {
          if(Keyvalue == 1) //k1 按键继电器开红灯亮
          {
            RLED = ON;            //亮灯
            YLED = OFF;
            write_relay_1();
          }
          if(Keyvalue == 2) //k2 按键继电器关黄灯亮
           {
            YLED = ON;            //亮灯
            RLED = OFF;
            write_relay_0();
          }
        }
    }
```

五、 实验步骤

1. 按要求连接好硬件，如实验一所示，将继电器传感器模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。

2. 在 IAR Embedded Workbench 环境下编写实验代码，编译、下载、运行（完整程序代码请参考\E-Box300\03-实验源代码\02-WSN\02-传感器实验代码\36_Relay）。

3.按下 K1 按键，听到继电器“嗞”的一声，观察 LED 灯状态。按下 K2 按键，听到继电器“嗞”的一声，观察 LED 灯状态

六、 拓展思考

1、继电器可以应用于哪些场合？

2、自己动手搭建可以使用继电器控制的电路，注意安全。

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\Sensor\ Relay”目录下的 OJ-OJE.pdf 数据手册。

4.3 基于 RFID 射频识别基础实验

4.3.1 实验三十七 高频 RFID 原理实验

一、 实验目的

1.了解射频读写芯片 MF RC522 的工作方式。

2. 了解 ISO 14443 Type A 标准的协议。

3.掌握射频读写芯片对标签进行读写数据操作方法。

二、 实验设备

- ZigBee模块
- RFID 模块（EBM-RF）
- ISO 14443A 标准的电子标签 1 个
- 校园一卡通系统演示程序
- 物联网实践实验平台
- 直连串口线
- 5V-1A 适配器

三、 实验内容

通过本实验，学生要了解 ISO 14443A 标准，了解 RFID 相关基础知识和 RFID 卡片构造及存储和访问方式，掌握 MCU 通过 MF RC522 控制 RFID 卡片的流程并完成从 RFID 卡片读数据与向 RFID 卡片写数据的操作过程。

四、 实验原理

1. 硬件系统组成

RFID 读写模块由三大部分组成：（1）主控 MCU；（2）射频读写芯片；（3）天线及匹配电路。如图 4-47 所示：

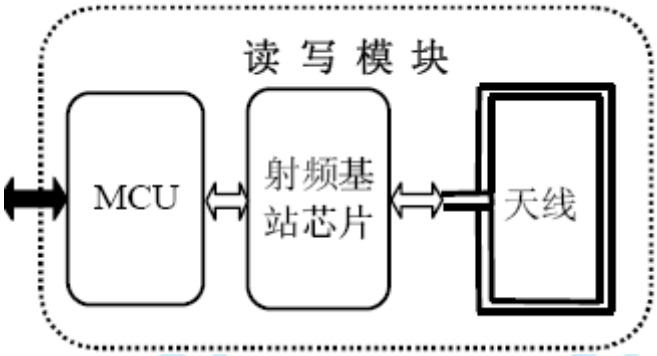


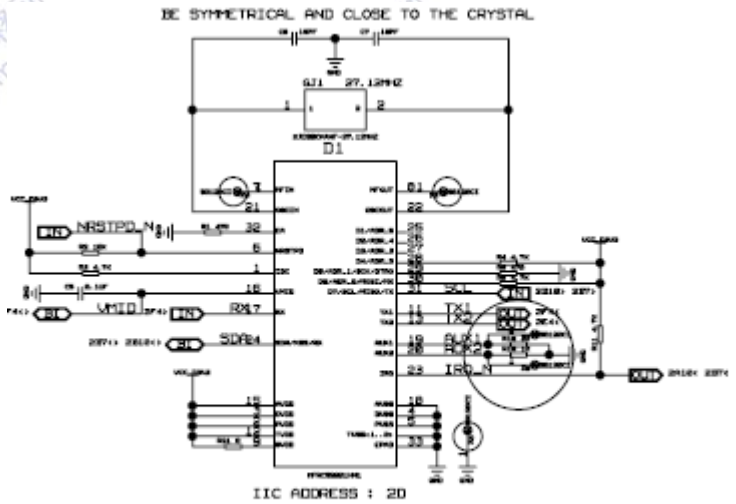
图4-47　RFID 读写模块

第一部分，主控 MCU，主要提供对射频读写芯片的控制操作。MCU 通过 SPI 控制接口与射频读写芯片连接，控制射频读写芯片的正常工作，实现与 RFID 卡的通信。主控 MCU 通过串行通信接口与 PC 方进行通信。

第二部分，射频读写芯片，它负责接收主控 MCU 的控制信息并完成与 RFID 卡的通信操作。为了正常工作，射频读写芯片须选用合适的并行接口与 MCU 连接。而为了发送、接收稳定的高频信号，射频读写芯片要通过高频滤波电路与天线部分连接。

第三部分，天线部分，包括线圈及匹配电路，这是读写模块实现射频通信必不可少的一部分。读写模块要依靠天线产生的磁通量为 RFID 卡提供电源、在读写模块与 RFID 卡之间传送信息。为使天线正常工作，天线线圈要通过无源的匹配电路连接射频读写芯片的天线引脚。

2. 硬件系统原理（图 4-48）：



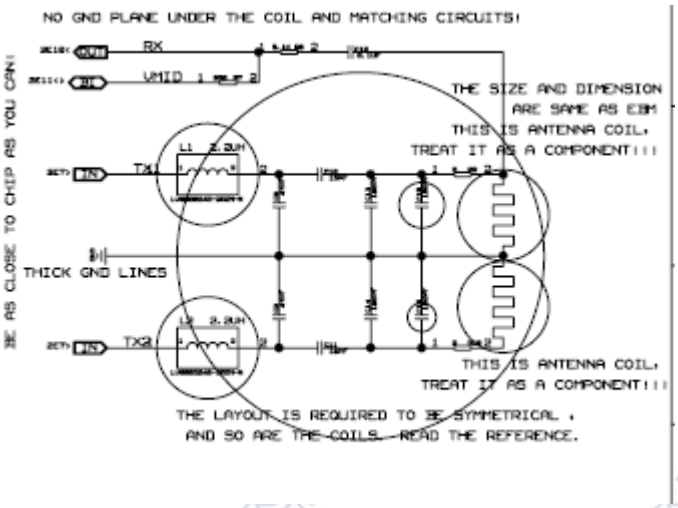


图4-48　系统硬件原理图

2.1　处理器

2.2　射频读写芯片

不同类型的 RFID 卡，由于采用的通信协议不同，相应的射频读写芯片也不同。目前在中国的市场上，RFID 卡主要的厂商有：中国的华虹、复旦微电子、以及荷兰 Philips、瑞士 LEGIC、法国意法半导体（ST）、日本索尼等，其中基于 Philips 公司 Mifare 芯片的产品在市场上占有绝对的优势。鉴于国内市场上使用 Mifare 芯片的 RFID 卡应用广泛，我们采用 Philips 公司生产的射频处理基站芯片 MF RC522。

MF RC522 是 ISO/IEC 14443 标准下低成本、高集成、高性能的 Mifare 非接触式读写芯片，基于 13.56MHz 的非接触通讯模式。MF RC500 内部有高集成的调制解调模块，内部发射器可直接驱动基于 13.56MHz 的非接触式天线，最大距离可达 10cm。主要应用于各种基于 ISO/IEC 14443A 标准的非接触式通信的应用场合，如公共交通终端、手持终端、计量设备、非接触式公用电话等。

2.3　天线及相关电路

MF RC522 根据其寄存器的设定对发送数据进行调制得到发送的信号，通过由驱动引脚 TX1 和 TX2 驱动的天线以 13.56MHz 的电磁波形式发送出去。在其射频范围内的 RFID 卡采用 RF 场的负载调制进行响应。天线接收到卡片的响应信号经过天线匹配电路送到 MF RC522 的接收引脚 RX，芯片内部的接收器对接收信号进行解调、译码，并根据寄存器的设定进行处理，最后将数据发送到 SPI 接口由处理器读取。

3.　RFID 卡－－Mifare

RFID 读写模块的操作对象是 RFID 卡片。Philips 是世界上最早研制 RFID 卡片的公司，其 Mifare 技术已经被制定为 IS0／IEC14443 TYPE A 国际标准。使用 Mifare 芯片的 RFID 卡占世界范围同类智能卡销量的 60%以上，在我国市场也占据着绝对优势。

Mifare 1 卡片除了微型芯片 IC 及一个高效率天线外，无任何其它元件。卡片电路不

用任何电池供电，工作时的能量由读写器天线发送频率为 13.56MHz 无线电载波信号，以非接触方式耦合到卡片天线上而产生电能，通常可达 2V 以上。标准操作距离高 10cm，卡与读写器之间的通信速率高 106Kbit/s。芯片设计有增／减值的专项数学运算电路，非常适合公共交通、地铁车站等行业的检票／收费系统，或充值钱包等多项应用，其典型交易时间最长不超过 100ms。

芯片内建 8K Bits 的 E2PROM 存储器。其空间被划分为可由用户单独使用的 16 个扇区。数据的擦写能力超过 10 万次以上，数据保存期大于 10 年，抗静电保护能力达 2KV。

Mifare 1 卡的芯片在制造时具有全球唯一的序列号。具有先进的数据通信加密和双向密码验证功能。具有防冲突功能，可以在同一时间处理重叠在读写器天线有效工作距离内的多张卡片。

4. PC 机和 RFID 模块的通信

PC 机与 RFID 模块之间的通信采用 UART，实现 PC 机对 RFID 模块的控制，从而实现标签的读数据、写数据操作。

PC 机和 RFID 模块的通信采用面向字节的异步通信协议数据格式。规定主机发给 RFID 模块的数据帧为命令，RFID 模块返回给主机的数据帧为响应。命令或响应数据帧是变长字节数，采用组包方法并用异或校验方法进行检错。

通信协议采用如图所示的层次结构，包括物理层、数据链路层和应用层。

应用层
数据链路层
物理层

本协议对物理层和数据链路层进行了定义。

物理层完成信号的比特数据发送与接收，RFID 模块和 PC 机之间的通信采用 RS-232 作为其物理层。其中，RS-232 接口具体设计要求如下：

- 1 位起始位、8 位数据位、1 位停止位、无奇偶校验位。
- 通信波特率设计为 9600bit/s。

数据链路层具体规定命令帧和响应帧的类型与数据格式。

D_LEN	设备类型	消息 ID	流水号	DATA	CHECK
1Byte	1Byte	2Bytes	4bytes		1byte

- D_LEN：该字段长度为 1 字节，表示 DATA 部分的长度。
- 设备类型：区分 ZigBee、RFID 等不同设备；RFID：0x01；ZigBee：0x02。

- 消息 ID，区分不同设备的消息类型，不同设备类型的 ID 可以重复；

  RFID 消息 ID 定义如下：

  上位机请求帧：

  0x00A0 读卡号

  0x00A1 读数据

  0x00A2 写数据

  0x00A3 消费

  0x00A4 充值

  0x1234 设备状态

  下位机响应帧：

  0xA000 读卡号

  0xA100 读数据

  0xA200 写数据

  0xA300 消费

  0xA400 充值

- 流水号：收到一条请求消息，要以同样的流水号回响应消息。
- DATA 为数据部分。RFID 部分数据格式定义如下：

上位机请求帧		
消息 ID （2 字节）	对应操作 类型	消息体（DATA）
00 A0	读卡号	A0 + 数据长度（1 字节）
00 A1	读数据	A1 + 密码（6 字节）+ 块号（1 字节） + 数据长度（1 字节）
00 A2	写数据	A2 + 密码（6 字节）+ 块号（1 字节） + 数据长度（1 字节）+ 数据（16 字节）
00 A3	消费	A3 + 密码（6 字节）+ 块号（1 字节） + 数据长度（1 字节）+ 数据（4 字节）
00 A4	充值	A4 + 密码（6 字节）+ 块号（1 字节） + 数据长度（1 字节）+ 数据（4 字节）

12 34	设备状态	

下位机响应帧		
消息 ID （2 字节）	对应操作 类型	消息体（DATA）
A0 00	读卡号	A0 + 数据长度（1 字节）+ 卡片序列号（4 字节）
A1 00	读数据	A1 + 块号（1 字节） + 数据长度（1 字节）+ 数据（16 字节）
A2 00	写数据	A2 + 块号+ 数据长度（1 字节）
A3 00	消费	A3 + 块号 + 数据长度（1 字节）+ 数据（4 字节）
A4 00	充值	A4 + 块号 + 数据长度（1 字节）+ 数据（4 字节）

5. 主控 MCU 对射频读写芯片的操作

主控MCU采用STC11F32X芯片，射频读写芯片采用MF RC522。STC11F32X通过对MF RC522的控制，实现RFID 卡的读写操作。STC11F32X对MF RC522的控制有三种方式：

- 设置MF RC522的状态。
- 发送命令，要求MF RC522执行相应的动作。
- 通过读MF RC522的状态标志来监测MF RC522的工作情况。

无论上述的哪一种方式，都是通过读/写MF RC522 的寄存器来实现的：配置MF RC522就是设置寄存器的某些位；执行命令要向命令寄存器写入命令代码以及通过FIFO缓冲区寄存器向缓冲区写入命令参数；监测MF RC522即读状态寄存器的标志位。因此，读写寄存器是所有操作的基础。

5.1 访问MF RC522寄存器

- MF RC522的寄存器分页机制

RC522内部共有64个寄存器，分4页，每页16个寄存器。第0页第0个寄存器的地址为0x00，第0页第1个寄存器的地址为0x01，第0页第15个寄存器的地址为0x0f，第1页第0个寄存器的地址为0x10，其它寄存器地址依此类推。

- MF RC522的FIFO缓冲区机制

MF RC522 内部有64 字节的FIFO（First In First Out,先进先出）缓冲区，是MCU与RC522之间输入和输出数据流的缓存。缓冲区中数据的流向按照先进先出的顺序进行。

与FIFO缓冲区状态关系紧密的寄存器有：

5.1.1 FIFO缓冲区数据寄存器（地址$09）：FIFOData

FIFO 缓冲区的输入输出数据总线直接连接到FIFO 缓冲区数据寄存器FIFOData上。因此，写FIFOData 寄存器即是向FIFO 缓冲区内存入一个字节并使缓冲区的内部写指针增1。连续写n次FIFOData寄存器，就向FIFO缓冲区存入了n个字节数据。读FIFOData 寄存器，就是将读指针指向的缓冲区内的内容读出并使其内部读指针增1。连续读n次FIFOData 寄存器，就从FIFO 读走n个字节数据。

5.1.2 FIFO缓冲区数据长度寄存器（地址$0A）：(1) FIFOLevel

FIFOLevel的第6位至第0位的值是存储在缓冲区中的数据的字节数，即其内部读指针与写指针之间的距离。向FIFOData 寄存器写一个字节，FIFOLevel 的值增1；读FIFOData寄存器，FIFOLevel的值减1。

FIFOLevel的第7位FlushBuffer，通过设定FlushBuffer的值为1 可复位缓冲区指针。当FlushBuffer＝1 时，相应的缓冲区长度寄存器FIFOLevel的值将变为0，错误标志寄存器（ErrorReg）的第4位BufferOvfl位将被清除，实际存放在缓冲区内的所有字节将被清除。

5.1.3 FIFO缓冲区大小寄存器（地址$0B）：WaterLevel

FIFO缓冲区的容量是64字节，在实际使用时，用户可通过WaterLevel寄存器设定需要使用的FIFO 缓冲区大小WaterLevel。WaterLevel 是判断FIFO 缓冲区是溢出还是不满的警戒线。FIFO缓冲区中实际存储的数据个数（即FIFOLevel寄存器的值）.与WaterLevel的比较，将会影响状态寄存器Status1的位第0位LoAlert标志、第1位HiAlert 标志。

若FIFOLevel≤WaterLevel，则标志位LoAlert=1；

若64－FIFOLevel≤WaterLevel，则标志位HiAlert=1。

例如，若设定WaterLevel＝4，则：

FIFOLevel＝60时，HiAlert=1；FIFOLevel＝59时，HiAlert=0

FIFOLevel＝4时，LoAlert=1； FIFOLevel＝5时， LoAlert=0

FIFO 缓冲区在程序中的一个重要作用就是传递执行MFRC522 命令时需要的参数。当MCU 启动一个命令操作时，MF RC522 到FIFO 缓冲区去取得执行这个命令的参数。实际中只有一个FIFO 缓冲区，而对缓冲区的访问有读入和取出两个方向，因此在写程序时一定要小心注意对FIFO缓冲区的访问。

5.2 MF RC522的命令

RC522内部有一个状态机，可以执行命令（Command）寄存器（地址$01）中的命令。命令的启动只需要将相关命令代码写到Command 寄存器中。执行命令所需要的变量以及数据都是通过FIFO缓冲区来传递。读这个寄存器可以知道正在执行哪条命令。

MF RC522的命令一共有10条。

命令	命令编码	功能
Idle	0x00	空闲命令，可以用此命令停止当前正在执行的命令。
Mem	0x01	将FIFO缓冲区的25字节数据传输至内部缓冲区。
Generate RandomID	0x02	产生10字节随机ID号
CalcCRC	0x03	激活CRC处理器
Transmit	0x04	将FIFO缓冲区的数据发送出去。
Nocmd Change	0x07	无命令变化，在操作其它位之前执行该命令，可以不改变命令寄存器中的命令。
Receive	0x08	激活接收电路
Transceive	0x0C	将FIFO缓冲区的数据发送至天线，并且在发送完成后自动激活接收器。
	0x0D	将来备用命令
MFAuthent	0x0E	执行MIFARE标准认证
Soft Reset	0x0F	复位MFRC522

5.3 与 Mifare 1 的射频识别通信

5.3.1 Mifare 1的状态（图4-49）

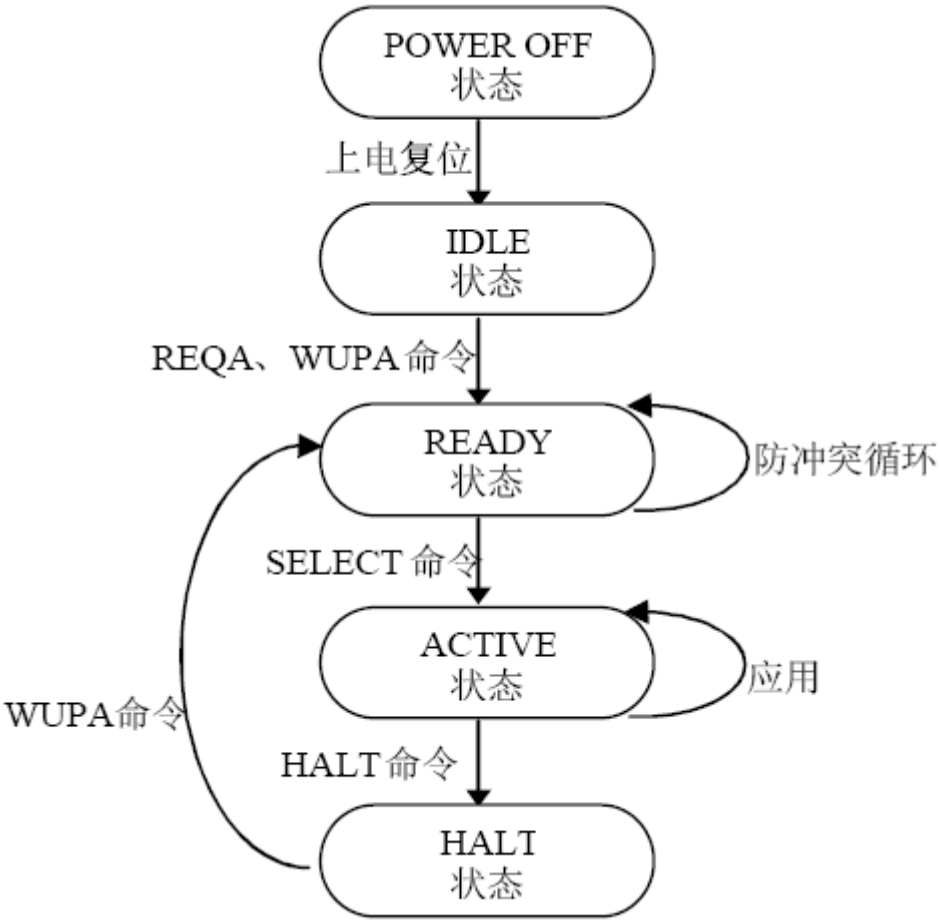


图4-49　Mifare1 状态

● POWER-OFF 状态

在POWER-OFF 状态，卡片由于缺少负载能量而处于断电状态。

● IDLE 状态

在IDLE 状态，卡片有电，可以侦听并识别出询卡命令REQA、WUPA。在

执行过询卡命令后，卡片进入READY状态。

● READY状态

在READY 状态，可应用防冲突方法得到完整的UID。当根据完整的UID，卡片被选中（SELECT）后，进入ACTIVE状态。

● ACTIVE状态

在ACTIVE 状态，卡片可执行应用操作。当接收到一个有效的挂起（HALT）命令后，卡片进入HALT状态。

● HALT状态

在HALT状态，卡片仅对WUPA 命令有反应。

5.3.2　MF1的射频通信处理流程（图4-50）

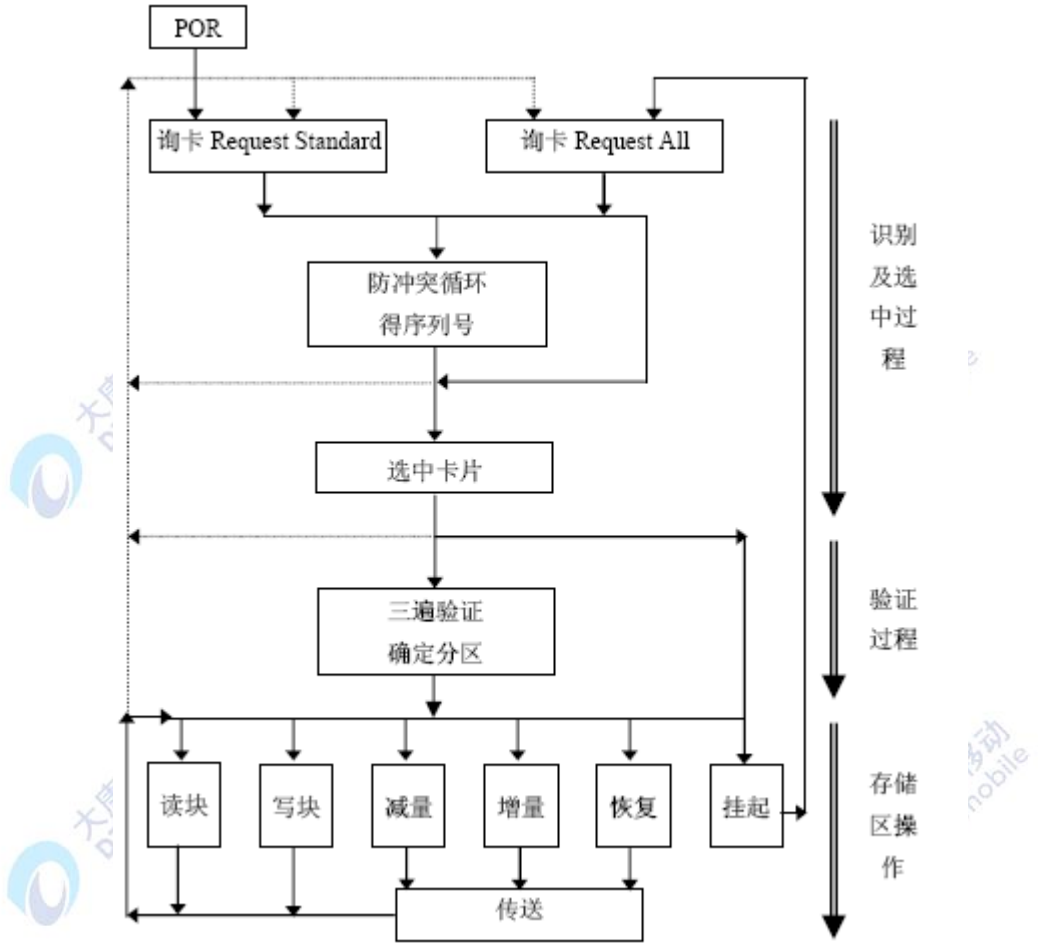


图4-50　射频读写芯片与 MF1 的通信流程

MF1 的射频通信处理流程如图4-50所示。首先，读写模块发Request询卡命令给天线工作范围内的所有卡片。卡片在上电复位（ Power OnReset,POR）后会响应这个询卡命令。在通过防冲突循环后，读写模块得到一张卡的序列号，于是根据该序列号选中一张卡。接下来，需要对准备访问的卡片的存储区的密码进行鉴别。在通过了密码验证后，读写模块可以对该存储区的数据进行读、写、增值、减值以及挂起等操作。

读写模块与MF1之间的通信需遵循上述流程进行。可以将该流程分成三个部分：识别及选中过程、（密码）验证过程以及存储区操作。

5.3.3　卡片识别及选中过程

5.3.3.1　通信数据格式

通信数据格式分为短帧、标准帧、位定向冲突帧。详细内容请参见ISO/IEC 14443-3协议的第6.2.3部分。

5.3.3.2　通信命令

在读写模块与其工作范围的卡片通信进行识别及选中的过程中，使用的通信命令有：REQA 、WUPA 、ANTICOLLISION、SELECT和HLTA。

- REQA、WUPA命令

REQA 和WUPA 命令即询卡命令，是由读写模块发送的，用于搜寻其射频范围内的A型近耦合RFID 卡，使用短帧格式发送。REQA 和WUPA 的主要区别在于，WUPA 命令可识别处在挂起（HLTA）状态的卡，而REQA 不能。其编码格式如表所示。

b7	b6	b5	b4	b3	b2	b1	Meaning
0	1	0	0	1	1	0	'26'=REQA
1	0	1	0	0	1	0	'52'=WUPA

- ANTICOLLISION、SELECT命令

这两条命令用于防冲突循环，命令组成为：选择代码SEL (1字节)，有效位数量NVB (1 字节)，以及由NVB 决定的UID CLn(第n 层序列号，0～40位)。当NVB指示其后有40个有效位时是SELECT 命令，否则为ANTICOLLISION命令。

ANTICOLLISION命令和SELECT命令的格式为：

SEL	NVB	UID CLn 数据位	BCC

- HALT命令

使用标准帧传送挂起命令HLTA，其格式包括两个字节代码及其后的CRC_A校验。在执行了HALT 命令后卡片将处于挂起状态。此时，除非使用WUPA命令将其唤醒，否则卡片将不响应读写模块的任何命令。

5.3.3.3 初始化和防冲突流程

按照ISO/IEC14443-3，对卡片的初始化和防冲突流程如图4-49所示。读写模块首先要与其工作区域的所有RFID卡建立通信连接：向卡片发送询卡命令REQA。在接收到卡片的响应ATQA后，通过防冲突循环最终得到一张卡的序列号。最后，若接收到这张卡片的SAK 响应，则表示已经完成与该卡的初始化通信连接（即选中该卡），可以继续进行下一步的密码校验过程了。

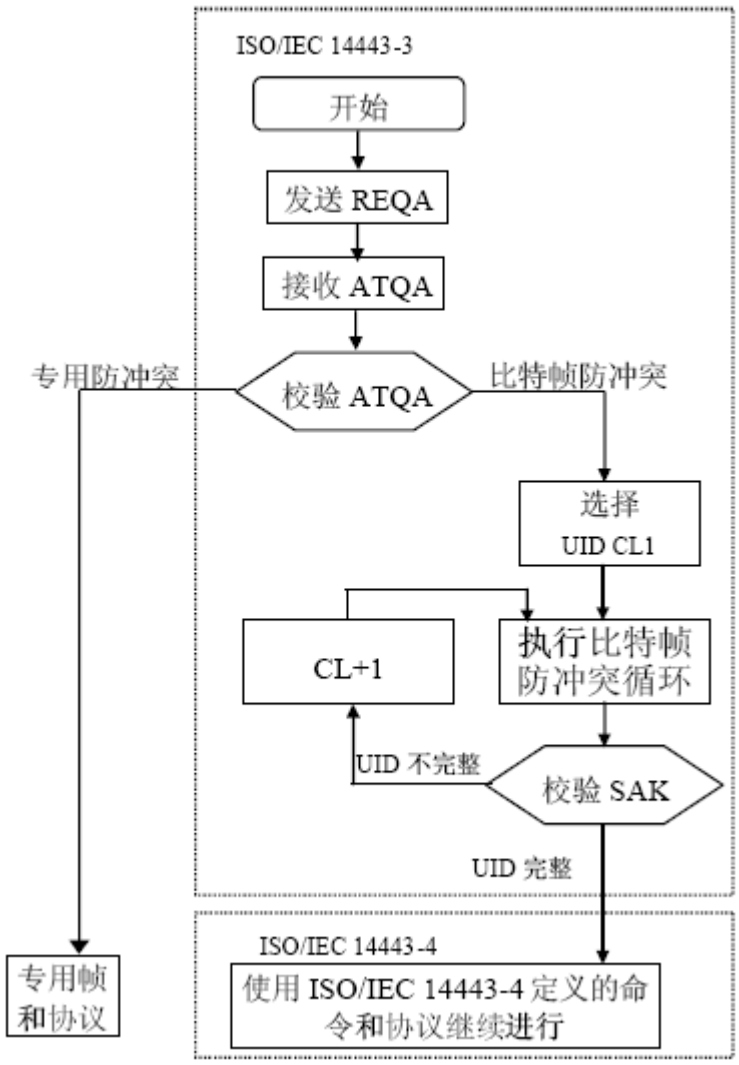


图4-51　卡片的初始化和防冲突流程图

流程图4-51中，在读写模块发送了一个询卡REQA 命令之后，所有在其工作范围处于空闲状态的卡片都会同步的以ATQA进行响应。ATQA的格式见图表

MSB　　LSB

b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1
RFU				专有代码				UID 类型		RFU	比特帧防冲突				

数据位b7和b8 的编码表示序列号UID 的类型，编码含义见表，数据位b1、b2、b3、b4和b5中若有一位被置1，表示防冲突过程使用比特帧防冲突。

b8	b7	含义
0	0	UID类型：4字节
0	1	UID类型：7字节

1	0	UID类型：10字节
1	1	RFU

卡片发送给读写模块的SAK 是一个标准帧：包括一个字节的SAK 代码和2 个字节的CRC 校验。其中，b3 位表示UID是否完整：若b3＝0，表示卡片的UID已经被读写模块确认，若b3＝1，表示还有部分UID CLn未确认，n＝2或3

5.3.3.4 初始化的具体实现

- 相关寄存器的设置：
- 执行询卡命令

清FIFO 缓冲区，向缓冲区写询卡命令代码：REQA=“52”或WUPA=“26”；

写“Transceive”（代码为“1E”）指令到命令寄存器，向卡片发送询卡命令；

- 接收卡片应答

读FIFO 缓冲区，即接收卡片的应答数据使用REQA命令询卡，也称为使用“ALL”模式询卡，该模式可以使工作范围内的所有卡片对其响应。使用WUPA 命令询卡，叫做“IDLE”模式，该模式仅处在被挂起状态的卡片有效。读写模块对MF1 卡执行询卡操作，接收卡片返回的ATQA 为2 个字节，低字节为“04”，高字节为“00”。根据ATQA 编码，MF1 卡的唯一序列号是4个字节，MF1卡的防冲突使用比特帧防冲突算法。因为所有的MF1卡对询卡操作返回的ATQA 都是这两个字节的内容，因此一般也称其为MF1 卡的类型号TagType。若询卡操作能正确得到类型号“$04$00”，说明在读写模块的工作范围内至少有一张MF1卡。

5.3.3.5 防冲突循环

ISO/IEC14443协议使用的防冲突算法称作比特帧防冲突。对应于图中“执行比特帧防冲突循环”的具体防冲突循环的流程如图4-52所示，算法步骤如下：

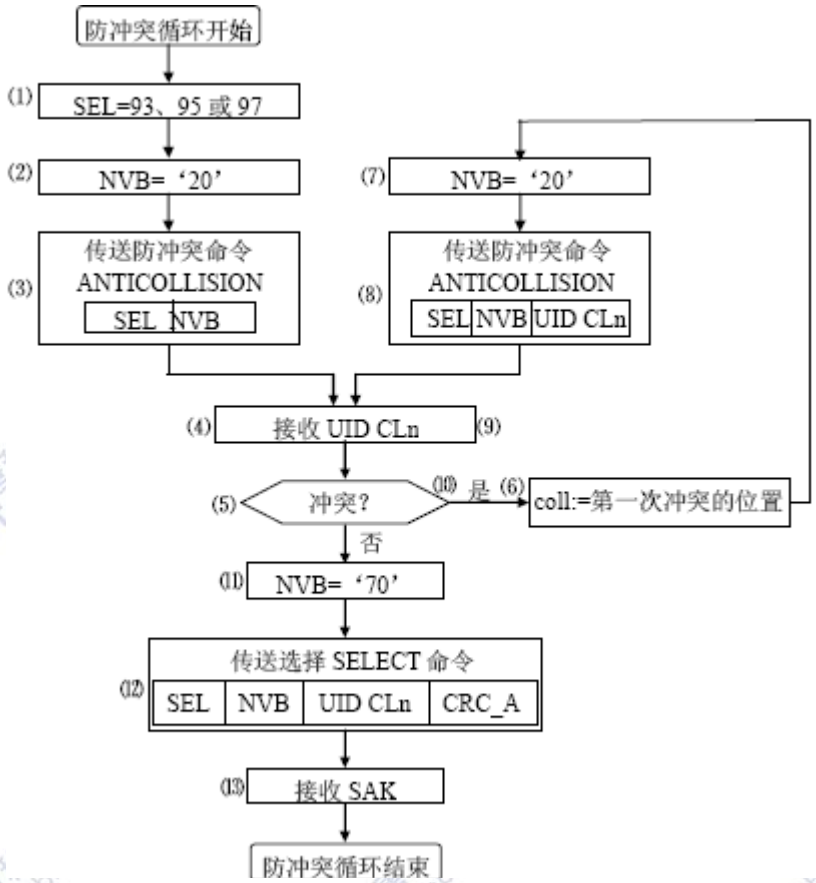


图4-52　防冲突循环流程图

● 读写模块根据唯一序列号UID的CLn 指定防冲突命令SEL的代码。CL1、CL2、CL3对应的SEL代码分别为“93”、“95”、“97”。

● 读写模块指定NVB 的值为“20”。该值表示UID CLn 部分是0 位，即不传送UID CLn，而是让射频范围内的所有RFID卡以其完整的UID Cln响应。

● 读写模块发送由SEL和NVB 组成的防冲突命令。

● 射频范围内的所有RFID卡以其完整的UID CLn 响应。

● 射频范围内的所有RFID 卡都有唯一的序列号，那么当有不止一张的PICC 进行响应，就会发生数据位冲突。如果没有发生冲突，步骤6到10 可以跳过。

● 读写模块读冲突位置寄存器CollPos的第一个冲突发生的位置。

● 读写模块指定NVB 的值为接收到的有效数据位数。有效数据位是在冲突发生前接收到的UID CLn的一部分，最后的数据位（即冲突位）可能是一个 “0”或“1”，读写模块默认其为1。

● 读写模块传送SEL和NVB，和上述的有效数据位。

● UID CLn与读写模块传送过来的有效数据位的那部分相同的RFID卡将其剩余的那部分UID CLn再传给读写模块。

- 如果再次有冲突发生，重复步骤6到步骤9。循环的最大次数可为32。
- 如果再没有冲突发生，指定NVB 的值为“70”，说明读写模块将传送完整的UID CLn。

●读写模块将传送SEL 、NVB及全部40 位的UID CLn，后面加CRC_A校验。

- 与这40位UID CLn匹配的RFID卡以其SAK 响应（至此，图4-14的防冲突流程结束）。

●接下来，根据SAK 判断唯一序列号UID 是否完整，如果UID完整，则卡片被激活，初始化通信完成，可继续对其进行密码验证等操作；否则，读写模块将增加CLn，继续进一步的防冲突循环。

5.4 Mifare 1的密码验证和加密机制

Mifare 1内部的安全加密算法叫做Crypto1，使用的密码长度是48 bits，即6个字节。Mifare卡中的数据，都有密码保护。只有使用正确的密码，才能成功进行卡密码校验，然后才能访问存储在EEPROM 中的卡片数据。在按ISO14443A 协议成功选中一张卡后，用户若要继续访问该卡，就必须首先进行卡的密码验证。Crypto1是一种三遍验证算法。MF RC500 内部将该算法进行了封装，只要执行Authent1和Authent2命令，就可自动完成这个验证过程。在卡片验证过程中，Crypto1模块开始初始化。在验证通过之后，读写模块与卡片的通信信息都将被加密。

RFID卡存储区有两种密码模式——A密码或者B密码，可供用户用于不同目的。例如，读数据块、写数据块这两种不同操作可以分别使用两个A、B 密码保护。RFID卡对A 密码进行验证的命令“AUTHENT1A”代码为“60”，对B 密码进行验证的命令“AUTHENT1B”代码为“61”。

5.5 密码验证过程

（1） MF1卡的访问存储器命令

MF1卡可以根据下列命令对存储器进行操作：

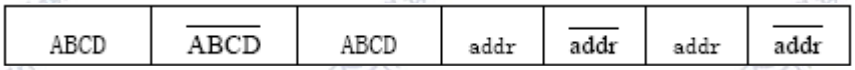
- READ 读存储区的一个数据块；
- WRITE 写存储区的一个数据块；
- DECREMENT 对存储在数值块中的数值做减法操作，并将结果存到数据寄存器；
- INCREMENT 对存储在数值块中的数值做加法操作，并将结果存到数据寄存器；
- TRANSFER 将数据寄存器的内容写入数值块；
- RESTORE 将数值块内容存入数据寄存器；

（2） MF1卡的数据块

MF1共有16个不同的存储区，每个存储区分成4个块，3个数据块以及一个控制块。MF1卡的数据块有两种：普通数据块和数值块。

普通数据块用于存储一般的16字节数据，仅可以执行READ、WRITE命令。数值块是特

殊的数据块，专门用来存储数值，不仅可以对其执行普通的READ、WRITE 命令，还可以执行用于实现电子钱包功能的命令：增值INCREMENT、减值DECREMENT、传送TRANSFER和重存RESTORE。存储在数值块中的数值为四个字节（包括符号位），按特定格式存储。设数值为ABCD（4个字节），数值块的地址（即块号）为addr，则存储格式为：“ABCD（4个字节）＋ABCD 的反码（4 个字节）＋ABCD（4个字节）＋addr（1 个字节）＋addr 的反码（1 个字节）＋addr（1 个字节）＋addr的反码（1个字节）”，参见图4-17。例如，若要将块号为5的数值块赋数值0，则用WRITE命令向数据块中写入下列数据：“00 00 00 00 FF FF FF FF 00 00 00 00 05 FA 05 FA”(16进制)。



数值块中的内容第一次由WRITE 命令写入后，以后可用INCREMENT、DECREMENT和RESTORE 命令修改内容，结果暂存在其内部的DATA 寄存器（数据寄存器）中，然后用TRANSFER命令再重新写回数值块。

（3） 对MF1卡存储区数据块的操作

读写模块对卡片数据块操作的实现过程如下。

读数据块的操作步骤为：

- 设置寄存器ChannelRedundancy($22)值为$0F,允许RxCRC,TxCRC,Parity校验；
- 设置DecoderControl寄存器,将第6位RxMultiple置1（接收1 帧以上数据）；
- 清除FIFO缓冲区，设置寄存器FIFOLevel值（>16）；
- 向FIFO缓冲区写参数：1.“读16 字节数据块”的命令代码（$30），2.块号；
- 向命令寄存器写TRANSCEIVE命令，发送READ命令代码及块号给卡片
- 等待指令完成，读FIFO 缓冲区接收16 字节数据

写数据块的操作步骤为：

- 设置寄存器ChannelRedundancy($22)值为$07, 即禁止RxCRC,允许TxCRC,Parity校验；
- 清除FIFO缓冲区，向FIFO缓冲区写参数：1.“写16字节数据块”的命令代码（$A0），2.块号；
- 向命令寄存器写TRANSCEIVE命令，发送WRITE命令代码及块号给卡片；
- 清除FIFO缓冲区,设置寄存器FIFOLevel值（>16）；
- 将要写入数据块的16 字节数据写入到FIFO缓冲区；
- 向命令寄存器写TRANSCEIVE命令，将数据发送给卡片；
- 延时等待命令完成。

五、 实验步骤

1.按要求连接好硬件，如实验一所示，将 RFID 模块插入 ZigBee 模块，注意板卡上箭头指示方向和防反插指针位置，并将直连串口线两端分别连接 ZigBee 模块和 PC 机。下载.hex 文件。（具体方法可参见实验四十二步骤 1 所示，实验代码位置为：\E-Box300\03-实验源代码\02-WSN\03-RFID 射频识别实验）

2.运行物联网实践实验平台软件，选择实践实验→串口 RFID，进入 RFID 实验界面如图 4-53 所示：

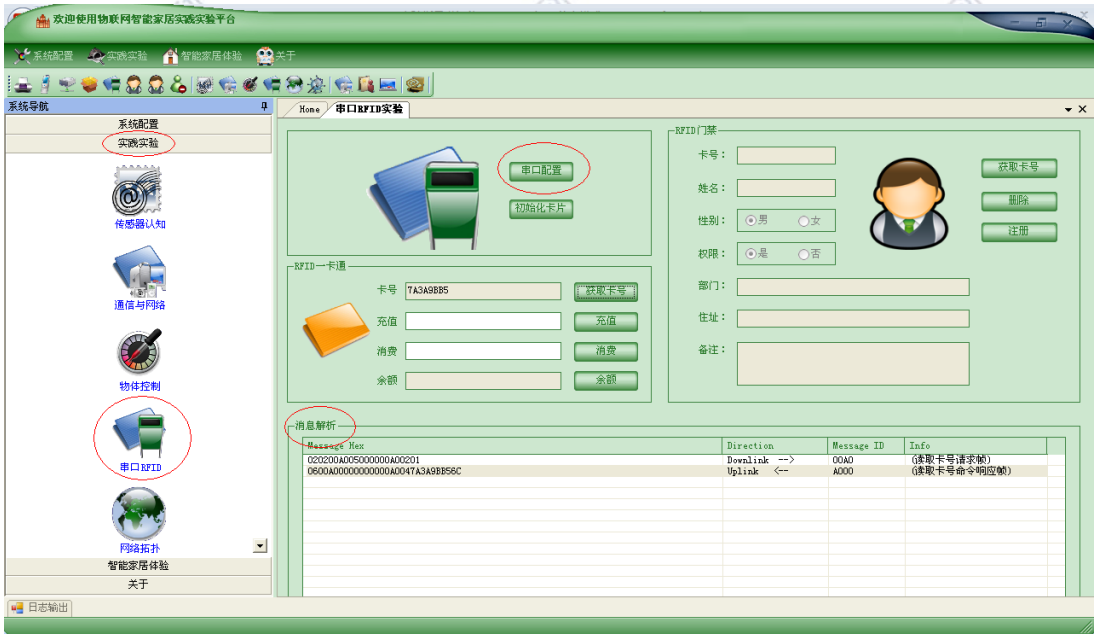


图4-53 物联网界面图

在消息解析一栏，可以观察 RFID 读写模块和 Mifare One 卡片交互数据的过程，帮助理解 RFID 读写模块的工作原理，

3. 进行串口配置。点击“串口配置”按钮，每秒位数设置为 9600，“串口号”应根据实际情况进行选择，一般是“COM1”。其它选项使用默认值。点击“打开串口”按钮如图 4-54 所示：

图4-54　串口设置

电脑串口号查看方法：右击“我的电脑”，选择管理，进入计算机管理界面，左侧树形目录中选择设备管理器，然后在右侧树形目录中选择端口（COM 和 LPT），查看通讯端口号。

4.将ZigBee模块上电，RFID板卡的LED_P11指示灯不停闪烁，表示正处于寻卡状态。刷卡操作需要先点击软件平台的操作按钮后，再将Mifare One卡片贴近RFI读写器。

5.了解Mifare One卡片存储器组织结构，如图4-55所示：

扇区	块	描述
15	63	第 15 扇区尾块
	62	数据块
	61	数据块
	60	数据块
14	59	第 14 扇区尾块
	58	数据块
	57	数据块
	56	数据块
	⋮	
1	7	第 1 扇区尾块
	6	数据块
	5	数据块
	4	数据块
0	3	第 0 扇区尾块
	2	数据块
	1	数据块
	0	厂商标志块

图4-55　Mifare One 卡片存储器组织结构

Mifare 1卡片的存储容量为8192×1位字长（即1K X 8位字长），采用E2PROM作为存储介质。整个结构划分为16 个扇区，编为扇区0～15。每个扇区有4 个块（Block）,分别为块0,块1,块2 和块3。每个块有16个字节。一个扇区共有 16 Byte X 4 =64 Byte。如图所示。每个扇区的块3（即第四块）也称作尾块， 包含了该扇区的密码A(6个字节)、存取控制(4 个字节)、密码B(6个字节)。其余三个块是一般的数据块。扇区0的块0是特殊的块，包含了厂商代码信息，在生产卡片时写入，不可改写。其中：第0～4 个字节为卡片的序列号，第5 个字节为序列号的校验码；第6 个字节为卡片的容量“SIZE”字节；第7，8个字节为卡片的类型号字节，即Tagtype字节；其他字节由厂商另加定义。

六、 拓展思考

Mifare One 卡片有多少块数据块供用户使用？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\RFID”目录下的数据手册。

4.3.2 实验三十八 充值与消费实验

一、 实验目的

1.了解射频读写芯片 MF RC522 的工作方式。

2.了解 ISO 14443 Type A 标准的协议。

3.演示程序的操作方式，能够正确地操作射频读写芯片对标签进行充值与消费操作。

二、 实验设备

- ZigBee模块
- RFID 模块（EBM-RF）
- ISO 14443A 标准的电子标签 1 个
- 校园一卡通系统演示程序
- 物联网实践实验平台
- 直连串口线
- 5V-1A 适配器

三、 实验内容

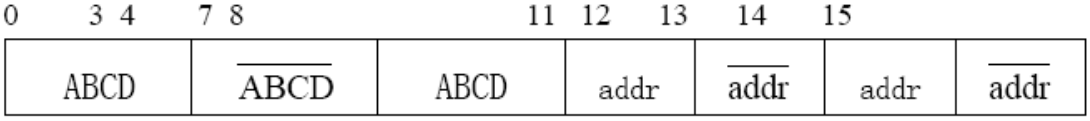
本实验要求学生了解 ISO 14443A 标准，了解 RFID 相关基础知识和 RFID 卡片构造及存储和访问方式，掌握 MCU 通过 MF RC522 控制 RFID 卡片的流程，并完成对标签进行充值与消费的操作过程。

四、 实验原理

本实验通过 PC 机与 RFID 模块的串口通信，实现 PC 机对 RFID 模块的控制和操作，从而实现对标签进行充值与消费操作。

本实验原理的内容同实验三十九，读者可以查阅实验三十九的实验原理部分。要使用充值消费功能，首先要将卡片进行初始化操作，MF1 卡片自身支持充值与消费命令，MF1 卡片

通过自身电路完成充值与消费操作，从而保证数据的安全性。MF1 共有 16 个不同的存储区，每个存储区分成 4 个块，3 个数据块以及一个控制块。MF1 卡的数据块有两种：普通数据块和数值块。普通数据块用于存储一般的 16 字节数据，仅可以执行 READ、WRITE 命令。数值块是特殊的数据块，专门用来存储数值，不仅可以对其执行普通的 READ、WRITE 命令，还可以执行用于实现电子钱包功能的命令：增值 INCREMENT、减值 DECREMENT、传送 TRANSFER 和重存 RESTORE。存储在数值块中的数值为四个字节（包括符号位），按特定格式存储。设数值为 ABCD（4 个字节），数值块的地址（即块号）为 addr，则存储格式为：“ABCD（4 个字节）＋ABCD 的反码（4 个字节）＋ABCD（4 个字节）＋addr（1 个字节）＋addr 的反码（1 个字节）＋addr（1 个字节）＋addr 的反码（1 个字节）”。例如，若要将块号为 0C 的数值块赋数值 0，则用 WRITE 命令向数据块中写入下列数据：“00 00 00 00 FF FF FF FF 00 00 00 00 0C F3 0C F3”(16 进制)。



按照上述格式把数据块初始化为数值块后，便可以调用充值与消费函数进行充值与消费操作。并且，一块数据块只需初始化一遍，但可以重复初始化，如果要更换数值块的地址，就需要重新初始化。

读写模块对卡片数值块进行充值与减值的操作的实现过程如下：

1. 设置寄存器ChannelRedundancy($22)值为$07, 即禁止RxCRC,允许TxCRC,Parity校验；
2. 清除FIFO缓冲区，向FIFO缓冲区写参数：1.命令代码（增值命令INCREMENT代码为‘C1’，减值命令DECREMENT代码为‘C0’），2.块号；
3. 向命令寄存器写TRANSCEIVE命令，发送增/减值命令代码及块号给卡片；
4. 清除FIFO缓冲区,将增/减值内容（4 个字节）写入到FIFO 缓冲区；
5. 向命令寄存器写TRANSCEIVE命令，将数据发送给卡片；
6. 延时等待命令完成；
7. 清除FIFO缓冲区，向FIFO缓冲区写参数：1.传送命令TRANSFER代码（‘B0’），2.块号；
8. 向命令寄存器写TRANSCEIVE命令，将传送指令代码及块号发送给卡片；
9. 延时等待命令完成。

五、 实验步骤

1. 按要求连接好硬件，如实验三十九所示。

2. 运行软件平台，进入串口 RFID 实验界面。串口设置与实验三十九一致，Mifare One 卡片如果是首次使用充值消费功能，需要点击“初始化卡片”按钮。初始化只需操作一次，但允许多次初始化操作。如图 4-56 所示：

图4-56 充值与消费实验界面图

3. 鼠标左键单击按钮“获取卡号”，将 Mifare One 卡片放置在读卡区上，可以在“卡号”处看到该卡片 ID。

4. 充值与消费：需要把卡片的数据块初始化为数值块，为方便实验，我们统一使用第0C数据块（点击“初始化卡片”按钮就相当于完成了此步骤）。在原理中，已经介绍过初始化方法。

在充值金额处输入一个钱数，精确到小数点后两位，比如，输入“50.69”，鼠标左键单击按钮“充值”，将 Mifare One 卡片放置在读卡区上，完成充值操作。重复查询“余额”步骤，验证充值是否成功。

在消费金额处输入一个钱数，精确到小数点后两位，比如，输入“4.73”，鼠标左键单击按钮“消费”，将 Mifare One 卡片放置在读卡区上，完成消费操作。重复查询“余额”步骤，验证消费是否成功。

5. 理解数据在数值块中的存储规则：数值块中有效数据字节为 4 字节，数据的低字节在地址前端，例如，数据 1000 对应 16 进制为“03 E8”。则数据 1000 对应 4 字节为“E8 03 00 00”。因此，数据 1000 对应数值块中的数据为“E8 03 00 00 17 FC FF FF E8 03 00 00 0C F3 0C F3”。由于数值块中的数据和有符合整型数对应，并不和浮点数对应。因此，我们以“分”作为最小单位操作电子钱包，例如，我们充值“10 元”，则等价于充值“1000 分”，向卡片发送充值命令，并命令其充值“E8 03 00 00”。

六、 拓展思考

1.如何使用一张 RFID 卡片既作为饭卡、图书卡，又作为公交卡？

2.如果采用一卡通形式，一张 Mifare One 卡片最多可以应用于多少个场合？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\RFID”目录下的数据手册。

4.3.3 实验三十九 门禁实验

一、 实验目的

1.理解门禁控制原理。

2.演示程序的操作方式，能够正确地进行门禁实验操作。

二、 实验设备

- ZigBee模块
- RFID 模块（EBM-RF）
- ISO 14443A 标准的电子标签 1 个
- 校园一卡通系统演示程序
- 物联网实践实验平台
- 直连串口线
- 5V-1A 适配器

三、 实验内容

了解 ISO 14443A 标准，了解 RFID 相关基础知识和 RFID 卡片构造及存储和访问方式。掌握 MCU 通过 MF RC522 控制 RFID 卡片的流程。通过操作软件平台完成门禁实验的操作过程。

四、 实验原理

本实验通过 PC 后台与 RFID 模块的串口通信，实现 PC 机对 RFID 模块的控制和操作，实现门禁控制。

本实验原理的基本内容同实验三十九，读者可以查阅其实验原理部分。

五、 实验步骤

1. 连按要求连接好硬件，如实验三十九所示。

2. 运行软件平台，进入门禁实验界面。串口设置与实验三十九一致。如图 4-57 所示：

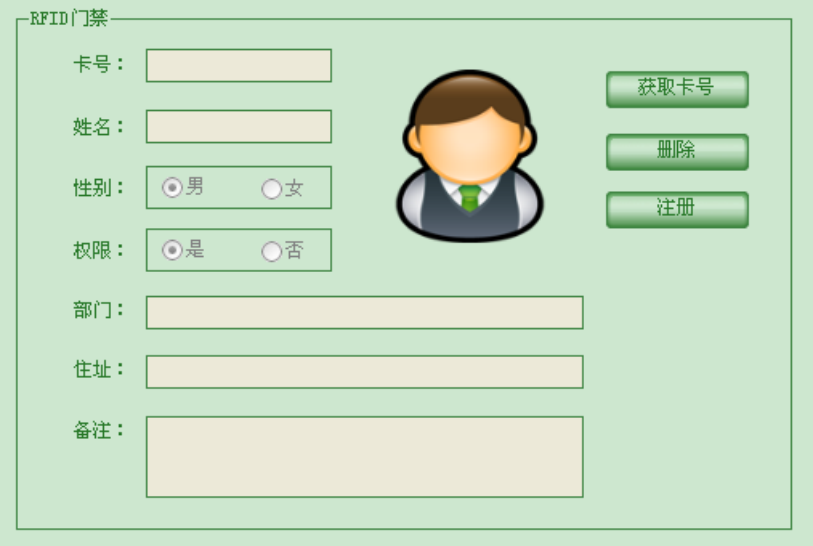


图4-57 门禁界面

门禁管理的实质是区分不同的 Mifare 1 卡片的 ID 号。上位机要获取 Mifare 1 卡片的 ID 号，需要读者在 ctrlprocess(void)函数中把 Mifare 1 卡片的 ID 号发送给上位机。

3. 鼠标左键单击“获取卡号”，如果此卡注册过门禁，在“卡号”处可以看到卡片的 ID号，以及卡片包含的信息，并弹出“通过权限认证，请通行”界面；如果此卡未注册过，并弹出“没有该卡的注册信息，禁止通行”界面。可以鼠标左键单击“注册信息”，为该卡片注册用户信息。选中权限“是”按钮，则使用该卡片刷卡，可以通过门禁系统。选中“否”按钮，则使用该卡片刷卡时，无法通过门禁系统。

六、 拓展思考

门禁系统分为单机系统和联机系统，请查阅资料，单机系统和联机系统有什么差别？

七、 参考阅读

请参考阅读“E-Box300\05-芯片数据手册\RFID”目录下的数据手册。

第五章 物联网无线传感网络与传输实验

5.1 无线传感网络与传输实验概述

互联网和移动通信网（包括移动互联网）是物联网应用和发展的核心基础设施，而传感器网络、RFID 等短距离网络又是其重要组成部分，这些网络的融合是物联网发展的关键内容之一。物联网的通信和网络技术的目标是构建物联网发展所需的开放、分层、可扩展的网络体系结构，解决物联网中 IP 地址、网络标识、通信、存储及计算等基础资源管理与服务技术。未来的网络是属于异构异种网络融合网络，已经不再严格区别究竟是互联网还是通信网，它是基于多种网络大融合的，具有虚拟化、自治计算、高可信、移动泛在、高效节能和高可扩展的物联网网络和接入平台。

5.2 ZIGBEE 网络实验

5.2.1 实验四十 点对点通信实验

一、实验目的

正确使用IAR软件和单片机CC2531，通过实验了解CC2531点对点通信操作流程，了解如何对zigbee设备进行射频配置并完成空口数据传输。

二、实验设备

- ZigBee模块（两块）
- IAR 集成开发环境
- JTAG仿真器

三、实验内容

使用两块CC2531zigbee模块实现点对点的无线通信，学习如何对zigbee设备进行射频配置并完成空口数据收发。

四、实验原理

本实验实现了一个简单的点对点通信程序，即由一个zigbee设备通过2.4GHz无线信道发送数据包到另外一个zigbee设备，这两个匹配的zigbee设备需要配置相同的PANID和信道号，使用的ZigBee开发环境为 IAR Embedded Workbench。

CC2531是一颗真正的系统芯片 (SoC)CMOS解决方案。这种解决方案能够提高性能并满足2.4GHz波段应用对低成本，低功耗的要求。它结合一个高性能2.4GHz DSSS(直接序列扩频)射频收发器核心和一颗工业级小巧高效的8051控制器。CC2531内部结构图如下图5-1所示：

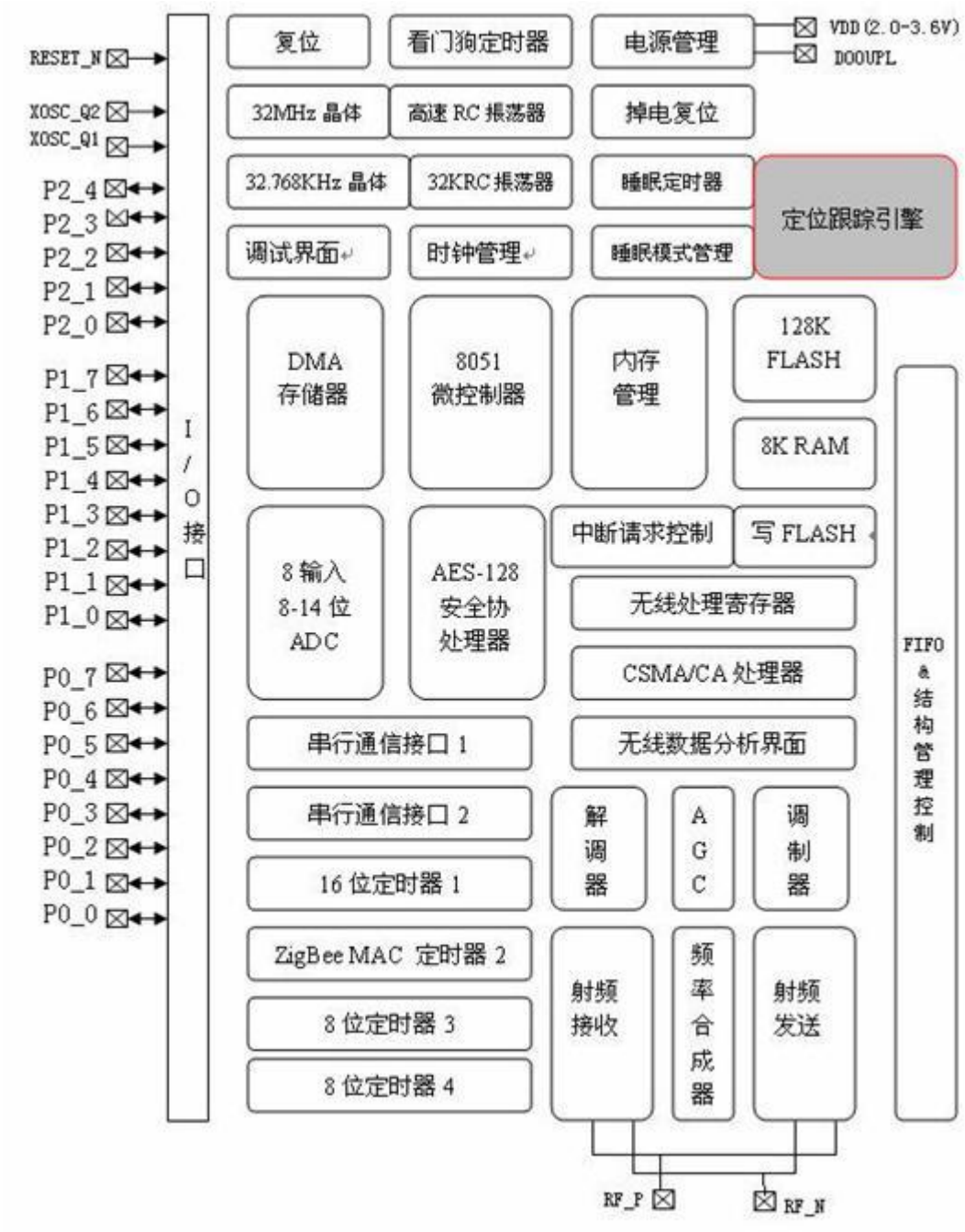


图5-1　　CC2531内部结构

CC2531的发送器基于上变频器。接收数据存放在一个接收先进先出（区别于发送先进先出）的数据缓冲区内。发送数据帧的前导符和开始符由硬件生成，通过数模转换把数字信号转换成模拟信号发送出去。CC2531要发送的数据存放在128字节的TXFIFO之中（与

RXFIFO彼此分隔），要发送的帧引导序列和帧开始定界符由硬件产生。每个符号(4位)使用IEEE802.15.4扩展序列扩展为32位码片序列，输出到DAC之中。

CC2531的无线接收器是一个低中频的接收器。接收到的射频信号通过低噪声放大器放大而正交降频转换到中频。在中频2MHz中，当ADC模数转换时，输入的正交调相信号被过滤和放大。CC2531通过硬件校验CRC，将接收信号强度指示器(RSSI)的相关数值附加到数据帧之中；在接收模式下，通过中断提供空闲信道评估(CCA)。

本实验要实现一个ZigBee模块以直接序列扩频方式发送数据，另一个ZigBee模块对数据进行接收，即下图5-2所示结构图：

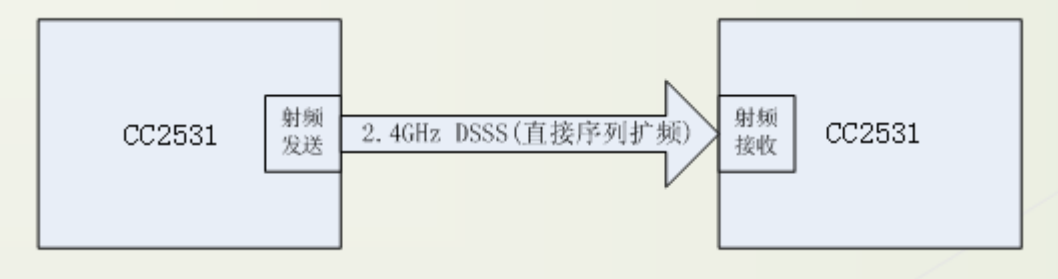


图5-2　　点对点发射结构图

直接序列扩频(Direct Sequence Spread Spectrum) 是发送端将要发送的信息用伪随机码（PN 码）扩展到一个很宽的频带上去，在接收端，用与发端扩展用的相同的伪随机码对接收到的扩频信号进行相关处理，恢复出发送的信息。

直接序列扩频通讯的主要技术特点是：

- 抗干扰性强
- 隐蔽性好
- 易于实现码分多址（CDMA）
- 抗多径干扰
- 直扩通信速率高

1. 点对点通信的IAR目录，如图5-3所示：

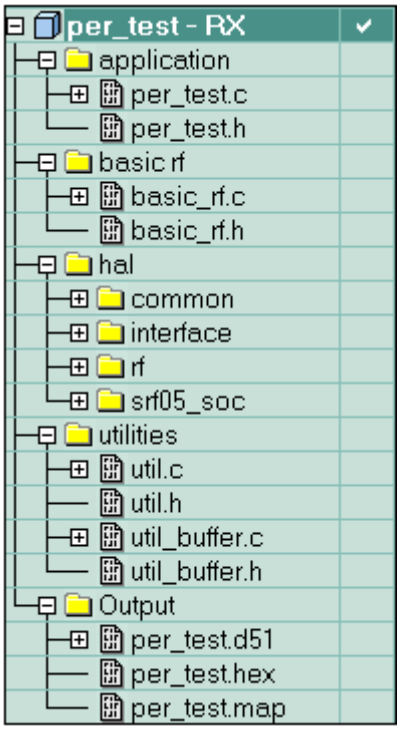


图5-3　　协议栈目录

其中：

- application：应用层目录，这是用户创建各种不同工程的区域，在这个目录中包含了应用层的内容和这个项目的主要内容；
- basic rf：射频配置目录，这是用户为了使用 CC2531 的无线通信，而对射频部分进行配置和初始化等操作，这个目录中包含了 CC2531 射频部分的内容。
- hal：硬件层目录，包含有与硬件相关的配置和驱动及操作函数；
- utilities：公用函数目录，为用户提供了一些例如反转buf中字节顺序等功能函数;
- Output：输出文件目录，这个是EW8051 IDE 自动生成的；

2. 几个重要函数

2.1 射频初始化函数：uint8 halRfInit(void)

2.1.1 功能描述：射频初始化，对RF相关寄存器进行配置，是CC2531能够通过RF进行空口数据的接收和发送。

2.2 发送数据包函数：uint8 basicRfSendPacket(uint16 destAddr, uint8* pPayload, uint8 length)

2.2.1　　功能描述：发送 length 字节的数据（最多128个字节）到目的地址。。

2.2.2　　参数描述：

（1）destAddr：destination short address，即要发送的目的地址；

（2）pPayload ：指向payload buffer，此有效载荷缓冲区必须由应用层分配；

（3）length：有效载荷的长度；

2.3 接收数据包函数：void sppReceive(SPP_RX_STRUCT *pReceiveData)

2.3.1 功能描述：这个函数使能CC2531接收最多128个字节的数据包并将其存储到pRxData中。

（1）参数描述：pRxData：指向存储接收数据的缓冲区，此缓冲区由用户创建分配；

（2）len：读入到pRxData的数据长度；

（3）函数返回值：实际存入到接收缓冲区的数据长度。

3. 程序实现

3.1 射频初始化应用函数：

```
uint8 halRfInit(void)
{
//帧协议控制寄存器FRMCTRL0配置，使能自动回复ACK和CRC校验
    FRMCTRL0 |= (AUTO_ACK | AUTO_CRC);
    //发送滤波寄存器配置
    TXFILTCFG = 0x09;
    AGCCTRL1 = 0x15;
    FSCAL1 = 0x00;
    // 使能RX接收中断
    halRfEnableRxInterrupt();
    return SUCCESS;
}
```

首先对发送滤波寄存器和帧协议控制寄存器FRMCTRL0按照下表所示进行配置，配置发送滤波器并使能CC2531自动回复ACK和CRC校验。具体寄存器配置请参考CC2531芯片手册。

表5-1 发送滤波寄存器

Register Name	New Value (Hex)	Description
AGCCTRL1	0x15	Adjusts AGC target value.
TXFILTCFG	0x09	Sets TX anti-aliasing filter to appropriate bandwidth.
FSCAL1	0x00	Reduces the VCO leakage by about 3 dB compared to default setting. Default setting is recommended for optimal EVM.

表5-2 帧协议控制寄存器**FRMCTRL0**

Bit No.	Name	Reset	R/W	Description
7	APPEND_DATA_MODE	0	R/W	When AUTOCRC = 0: Don't care When AUTOCRC = 1: 0: RSSI + The crc_ok bit and the 7-bit correlation value are appended at the end of each received frame 1: RSSI + The crc_ok bit and the 7-bit SRCRESINDEX are appended at the end of each received frame. See Table 19-1 for details.
6	AUTOCRC	1	R/W	In TX 1: A CRC-16 (ITU-T) is generated in hardware and appended to the transmitted frame. There is no need to write the last 2 bytes to TXBUF. 0: No CRC-16 is appended to the frame. The last 2 bytes of the frame must be generated manually and written to TXBUF (if not, TX_UNDERFLOW occurs). In RX 1: The CRC-16 is checked in hardware, and replaced in the RX FIFO by a 16-bit status word which contains a CRC OK bit. The status word is controllable through APPEND_DATA_MODE. 0: The last two bytes of the frame (crc-16 field) are stored in the RXFIFO. The CRC check (if any) must be done manually. Note that this setting does not influence acknowledgment transmission (including AUTOACK).
5	AUTOACK	0	R/W	Defines whether the radio automatically transmits acknowledge frames or not. When autoack is enabled, all frames that are accepted by address filtering, have the acknowledge request flag set, and have a valid CRC are automatically acknowledged 12 symbol periods after being received. 0: Autoack disabled 1: Autoack enabled
4	ENERGY_SCAN	0	R/W	Defines whether the RSSI register contains the most-recent signal strength or the peak signal strength since the energy scan was enabled. 0: Most-recent signal strength 1: Peak signal strength
3:2	RX_MODE[1:0]	00	R/W	Set RX modes 00: Normal operation, use RXFIFO. 01: Reserved 10: RXFIFO looping ignore overflow in RXFIFO, infinite reception. 11: Same as normal operation except that symbol search is disabled. Can be used for RSSI or CCA measurements when it is undesired to find symbol.
1:0	TX_MODE[1:0]	00	R/W	Set test modes for TX 00: Normal operation, transmit TXFIFO 01: Reserved. Should not be used 10: TXFIFO looping ignore underflow in TXFIFO and read cyclic, infinite transmission. 11: Send pseudorandom data from CRC, infinite transmission.

3.2 发送状态函数：

```
static void appTransmitter()
{
    uint32 burstSize=0;
    uint8 appTxPower;
    uint8 n;

    // Initialize BasicRF
    basicRfConfig.myAddr = TX_ADDR;
    if(basicRfInit(&basicRfConfig)==FAILED)
 {
```

```
        HAL_ASSERT(FALSE);
        }
        // 设置发送功率
        //HAL_RF_TXPOWER_0_DBM, see hal_rf.c for definations
        halRfSetTxPower(HAL_RF_TXPOWER_2_5_DBM);

        // 设置burst size
        burstSize=BURST_SIZE_1;

        //关闭射频接收功能
        basicRfReceiveOff();

        // 初始化 packet payload
        txPacket.seqNumber = 0;

        for(n = 0; n < 8; n++)
        {
            txPacket.padding[n] = Send_data[n];
        }

        while(1)            // 主循环
        {
            LED1 = 1;
            basicRfSendPacket(RX_ADDR, (uint8*)&txPacket, PACKET_SIZE);
            halMcuWaitMs(800);
            LED1 = 0;
            halMcuWaitMs(800);
         }
    }
```

在发送状态调用了 basicRfSendPacket函数来向目的设备发送数据，其中源发送地址为TX_ADDR，发送的目的地址为RX_ADDR。

3.3 接收处理函数：

```
static void appReceiver()
{
    int16 rssi;
    uint8 RxStatus = 0;
    // 初始化射频
    basicRfConfig.myAddr = RX_ADDR;
    if(basicRfInit(&basicRfConfig)==FAILED)
    {
      HAL_ASSERT(FALSE);
    }
    basicRfReceiveOn();
     while (TRUE)
    {
      LED1 = 1;
      LED2 = 1;
      if(basicRfReceive((uint8*)&rxPacket, MAX_PAYLOAD_LENGTH, &rssi)>0)
      {
          if(rxPacket.padding[1] != 0)
          {
            RxStatus = 1;
            memset((uint8*)&rxPacket, 0, MAX_PAYLOAD_LENGTH);
          }
          while(RxStatus == 1)
          {
              LED1 = !LED1;
              halMcuWaitMs(50);
              RxStatus = 0;
```

```
            }
        }
        halMcuWaitMs(100);
        LED2 = 0;
        halMcuWaitMs(100);
    }
}
```

本接收处理函数中调用basicRfReceive来对空口数据包进行接收解析。。

3.4 点对点无线通信主函数：

```
void main (void)
{
    uint8 appMode;

    appState = IDLE;
    appStarted = FALSE;

    // RF射频配置
    basicRfConfig.panId = PAN_ID;
    basicRfConfig.ackRequest = FALSE;

    // 板卡硬件初始化
    halBoardInit();

    // 初始化RF射频
    if(halRfInit()==FAILED) {
      HAL_ASSERT(FALSE);
    }

    halMcuWaitMs(350);
```

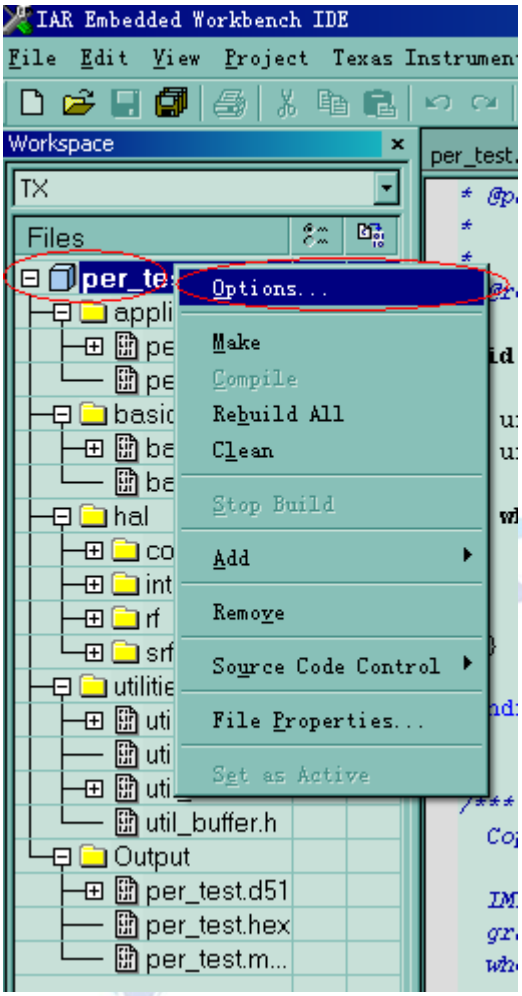
```
        // 设置通信信道，2.4GHz的zigbee基于IEEE802.15.4标准，一共定义了16个信道
        //basicRfConfig.channel = appSelectChannel();  //此处可以通过配置按键来自己定义选择信道
        basicRfConfig.channel = Channels[2]; //Channel 13, 2415MHz


    #ifdef TX
        appTransmitter();              //调用发送函数
    #else
        appReceiver();                 //调用接收函数
    #endif
        HAL_ASSERT(FALSE);
    }
```

注意这里的条件编译#ifdef TX，即：如果在IAR工程中的Options中预编译了TX，则调用函数appTransmitter（），否则调用函数appReceiver()；目的是为了区分在同一个程序中同时编写 的TX（发送） 和 RX（接收） 程序。

在IAR工程中增加预编译选项TX：在 IAR的Workspace选中TX 工程，如图5-4所示：

图5-4　　RX工程

首先在IAR工程的工程名上鼠标右击，然后选择“Options”选项，会弹出一个如图5-5的对话框，然后按照如图5-5所示增加条件编译选项T：


IAR Embedded Workbench IDE
File Edit View Project Texas Instrumen
Workspace
TX
Files
per_te
appli
pe
pe
basic
ba
ba
hal
co
int
rf
srf
utilitie
uti
uti
uti
util_buffer.h
Output
per_test.d51
per_test.hex
per_test.m...
Options...
Make
Compile
Rebuild All
Clean
Stop Build
Add
Remove
Source Code Control
File Properties...
Set as Active
per_test.

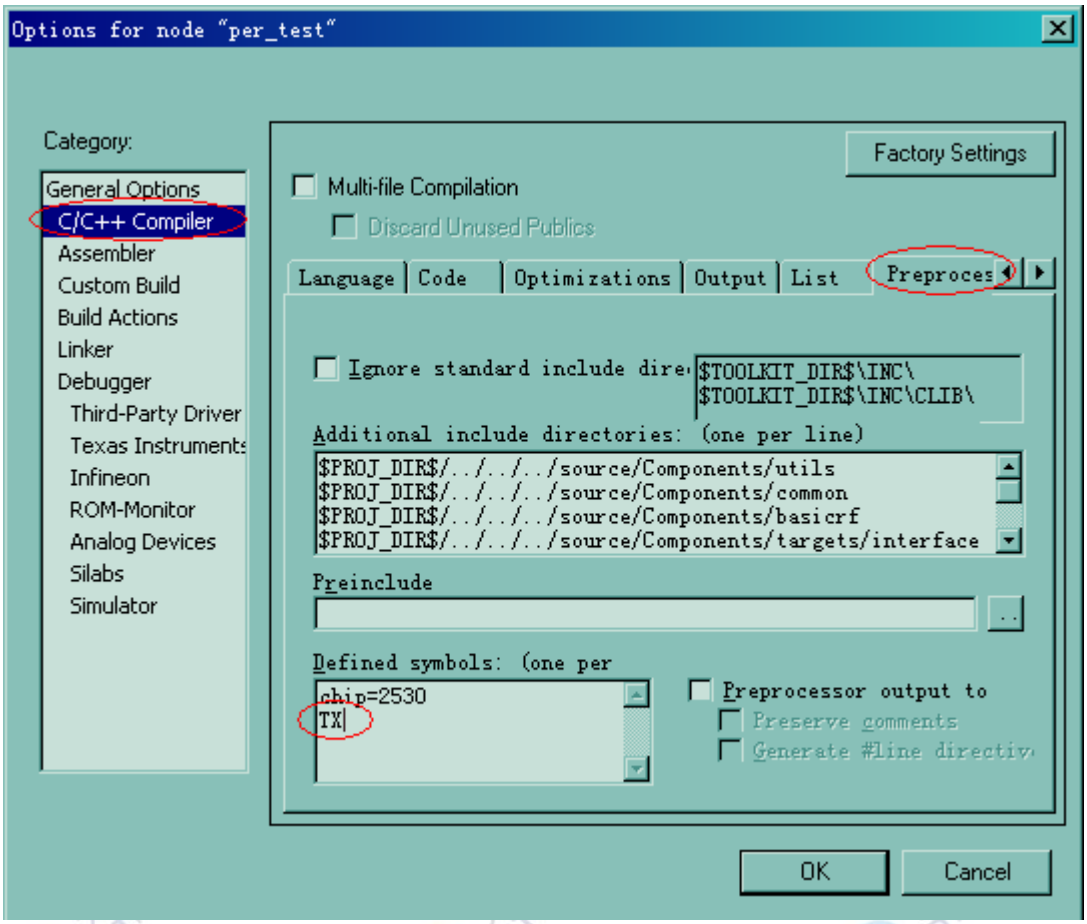


图5-5　　条件编译

表运行发送状态。在IAR工程中另一个 RX 就是接收状态。

4. 实验步骤

1、按照如下相对工程路径：“…ide\srf05_cc2530\iar”打开如图5-6所示的IAR工程“per_test.eww”；

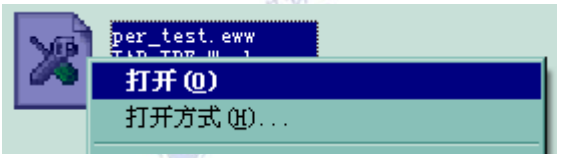


图5-6　　打开文件

2、打开工程文件后，出现如图5-7所示的工程界面：

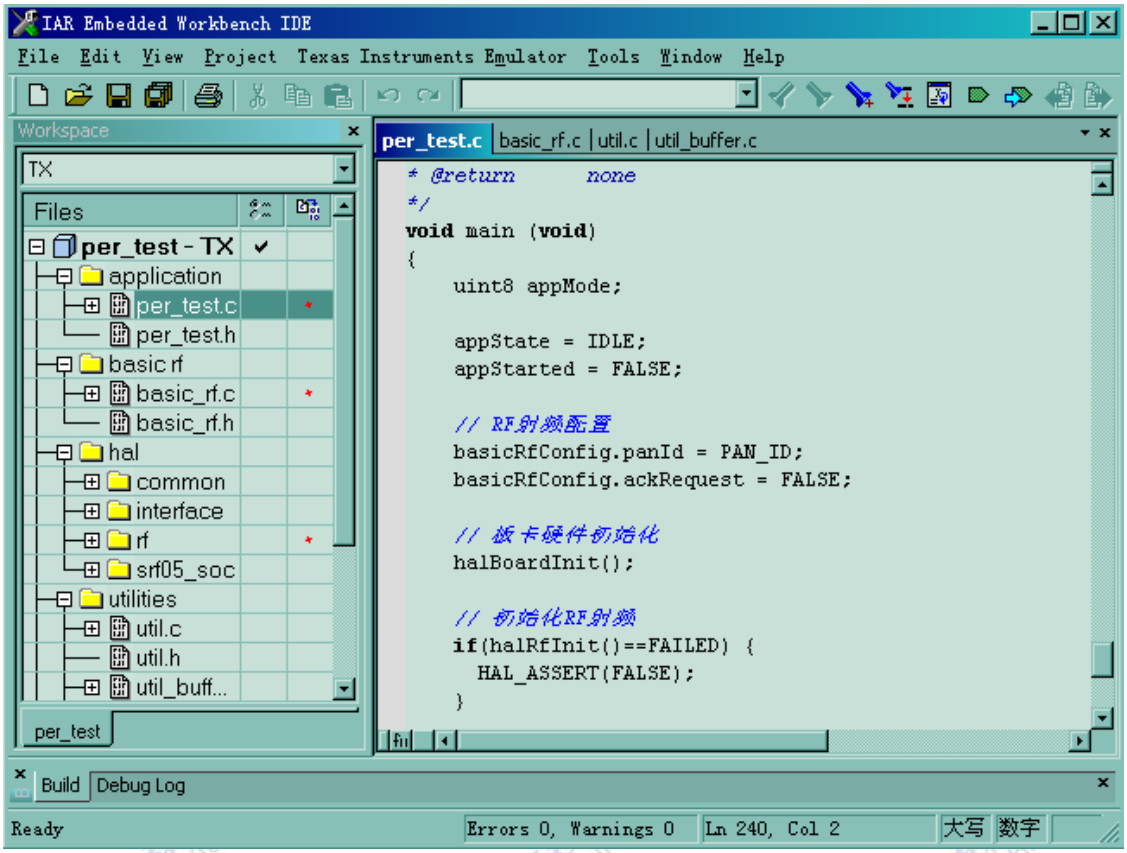


图5-7　　工程文件界面

3、在IAR工程中选择选择RF状态后，分别下载到两个zigbee模块中：

（1）选择发送状态TX如图5-8所示，然后下载到zigbee模块中，作为点对点无线通信的发送方；

图5-8　　发送状态

（2）选择接收状态RX如图5-9所示，然后下载到zigbee模块中，作为点对点无线通信的接收方；

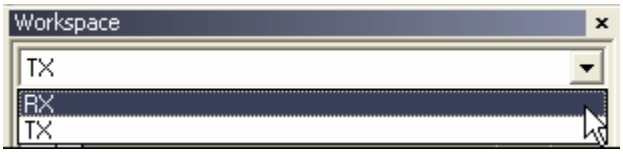


图5-9　　接收状态

4、实验结果：

运行的结果是发送和接收模块上的小灯交替闪烁。 从程序上也能分析出实验现象：

1. zigbee数据发送模块； zigbee发送模块上电后，黄灯开始闪烁，表示数据已经正常发送，黄灯每闪烁一次，表示数据成功发送出去一次；

2. zigbee数据接收模块：zigbee接收模块上电后， 红灯一直闪烁，表示接收模块正常工作，准备好了接收工作；若接收模块的黄灯闪烁，则表示接收到了zigbee数据发送模块的数据，黄灯闪烁一次表示成功接收到一次空口数据。

5. 拓展思考

1、如何在zigbee接收数据后，给源zigbee节点返回确认，即实现通信握手，完成zigbee数据确认收发的机制。

2、如何通过修改点对点通信工程中的参数进行来实现多对zigbee模块都下载点对点通信程序后，而通信组模块之间不会相互串扰，即每一对点对点通信的zigbee模块的工作都是相对独立的。（提示：可以通过修改通信的PAN_ID[小于0x3FFF]和信道号Channels[2.4GHz一共有16个zigbee信道]来区分不同的点对点工作组）。

七、参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

实验代码请参考\E-Box300\03-实验源代码\02-WSN\04-无线传输实验\40_CC2531点对点通信.rar。

5.2.2 实验四十一 点对多点通信实验

一、 实验目的

1.在对点对点无线通信理解的基础上，学习如何利用 CC2531 实现 zigbee 模块的点对多点的无线通信。

2.学习如何使用 IAR 的工程选项区分相同功能模块的不同处理，即一个工程中如果想有多个 RX 该如何进行分别的处理。

二、 实验设备

- ZigBee模块三块（可扩展为多个）
- IAR 集成开发环境
- JATG仿真器

三、 实验内容

让两台接收机，接收同一台 zigbee 模块发送的数据，收到数据后通过小灯闪烁表示成功接收或发送数据。

四、 实验原理

本实验实现了一个简单的点对多点通信程序，即由一个zigbee设备通过2.4GHz无线频段发送数据包到另外两个zigbee设备，这两个匹配的zigbee接收设备需要配置不同的接收地址，但是需要配置成与发送数据端相同的PANID和信道号，使用的ZigBee开发环境为 IAR Embedded Workbench。

实验原理与点对点无线通信相似，不同点在于在处理多个zigbee数据接收时的不同处理。

1. 点对多点主程序如下：

```
void main (void)
{
    uint8 appMode;

    appState = IDLE;
    appStarted = FALSE;

    // RF射频配置
    basicRfConfig.panId = PAN_ID;
    basicRfConfig.ackRequest = FALSE;

    // 板卡硬件初始化
    halBoardInit();

    // 初始化RF射频
    if(halRfInit()==FAILED) {
      HAL_ASSERT(FALSE);
    }

    halMcuWaitMs(350);
```

```
        // 设置通信信道，2.4GHz的zigbee基于IEEE802.15.4标准，一共定义了16个信道
        //basicRfConfig.channel = appSelectChannel();  //此处可以通过配置按键来自己定义选择信道
        basicRfConfig.channel = Channels[2]; //Channel 13, 2415MHz

        // Set mode
        //appMode = appSelectMode();

    #ifdef TX
        appTransmitter();              //调用发送函数
    #elif defined RX
        appMultiReceiver();        //调用接收函数1
    #else
        appReceiver();                 //调用接收函数2
    #endif
        HAL_ASSERT(FALSE);
    }
```

注意这里的条件编译#ifdef TX，即：如果在IAR工程中的Options中预编译了TX，则调用函数appTransmitter（），如果定义了RX，则调用函数appMultiReceiver ()；否则调用appReceiver（），目的是为了区分在同一个程序中同时编写的TX（发送）、RX1和RX2（接收）程序。

2. 发送函数程序如下：

```
    static void appTransmitter()
    {
        uint32 burstSize=0;
        uint8 appTxPower;
        uint8 n;

        // Initialize BasicRF
        basicRfConfig.myAddr = TX_ADDR;
```

```
if(basicRfInit(&basicRfConfig)==FAILED) {
  HAL_ASSERT(FALSE);
}

// 设置发送功率
//HAL_RF_TXPOWER_0_DBM, see hal_rf.c for definations
halRfSetTxPower(HAL_RF_TXPOWER_2_5_DBM);

// 设置burst size
burstSize=BURST_SIZE_1;

//关闭射频接收功能
basicRfReceiveOff();

// 初始化 packet payload
txPacket.seqNumber = 0;

for(n = 0; n < 8; n++)
{
  txPacket.padding[n] = Send_data[n];
}

// 主循环
while(1)
{
  LED1 = 1;
  basicRfSendPacket(RX_ADDR, (uint8*)&txPacket, PACKET_SIZE);
  halMcuWaitMs(800);
  basicRfSendPacket(RX1_ADDR, (uint8*)&txPacket, PACKET_SIZE);
  LED1 = 0;
```

```
            halMcuWaitMs(800);
        }
    }
```

发送程序主要功能为循环发送数据，程序开始同样是初始化CC2531射频部分和内部CPU后，开始循环对不同的目的地址发送数据，当数据发送成功后，黄灯闪烁一次，表示数据成功轮询发送一次。本发送函数循环调用函数basicRfSendPacket()向地址RX_ADDR和地址RX1_ADDR循环发送数据，具体实现请参照源码。

3. 数据接收：

接收程序主要功能为接收zigbee发送过来的数据，当zigbee接收设备上电准备好接收数据后，红灯会一直闪烁，当接收到数据后黄灯闪烁，且接收到一条数据后黄灯闪烁一次。RX1和RX2分别调用函数appMultiReceiver()和appReceiver()来进行不同的数据接收处理。

4. 在IAR工程中为zigbee发送节点增加预编译选项TX：

在 IAR的Workspace选中TX 工程，如图5-10所示：

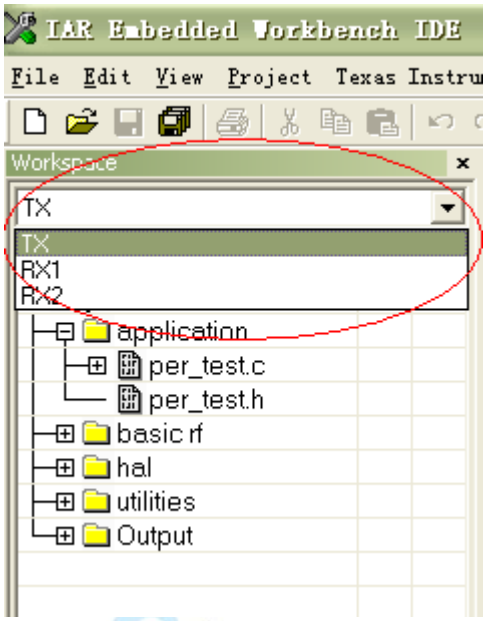
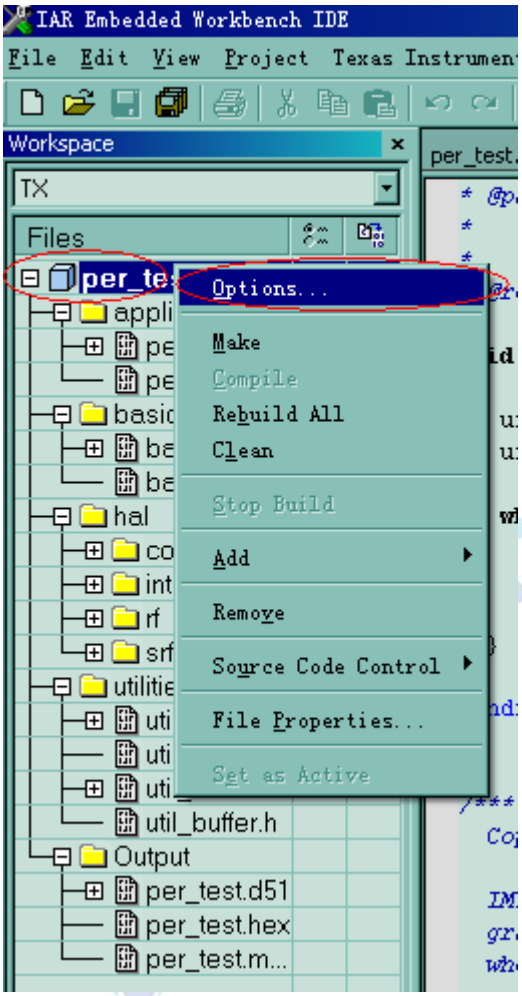


图5-10 RX工程

首先在IAR工程的工程名上右击鼠标，然后选择“Options”选项，会弹出一个如图5-11的对话框，然后按照如图5-11所示增加条件编译选项TX：


IAR Embedded Workbench IDE
File Edit View Project Texas Instrumen
Workspace
TX
Files
per_te
appli
pe
pe
basic
ba
ba
hal
co
int
rf
srf
utilitie
uti
uti
uti
util_buffer.h
Output
per_test.d51
per_test.hex
per_test.m...
Options...
Make
Compile
Rebuild All
Clean
Stop Build
Add
Remove
Source Code Control
File Properties...
Set as Active
per_test.

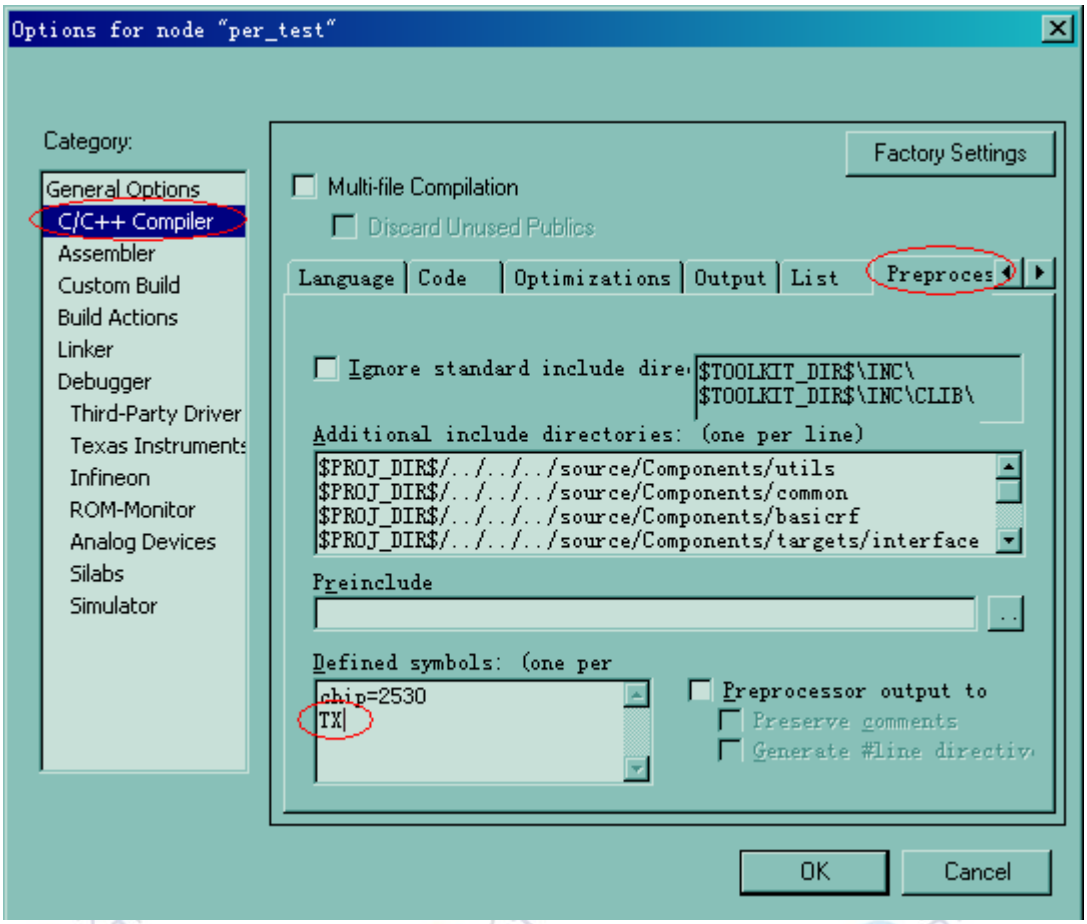


图5-11　条件编译

另外对两个接收模块 RX1 和 RX2 的预编译修改与 TX 相同，此处不再赘述。

五、　实验步骤

1、按照如下相对工程路径：“…ide\srf05_cc2530\iar”打开如图5-12所示的IAR工程“per_test.eww”；

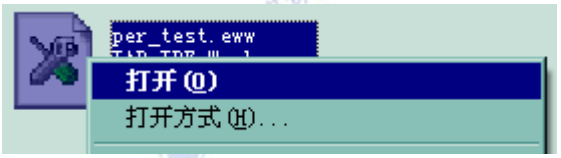


图5-12　图5-7

2、打开工程文件后，出现如图5-13所示的工程界面：

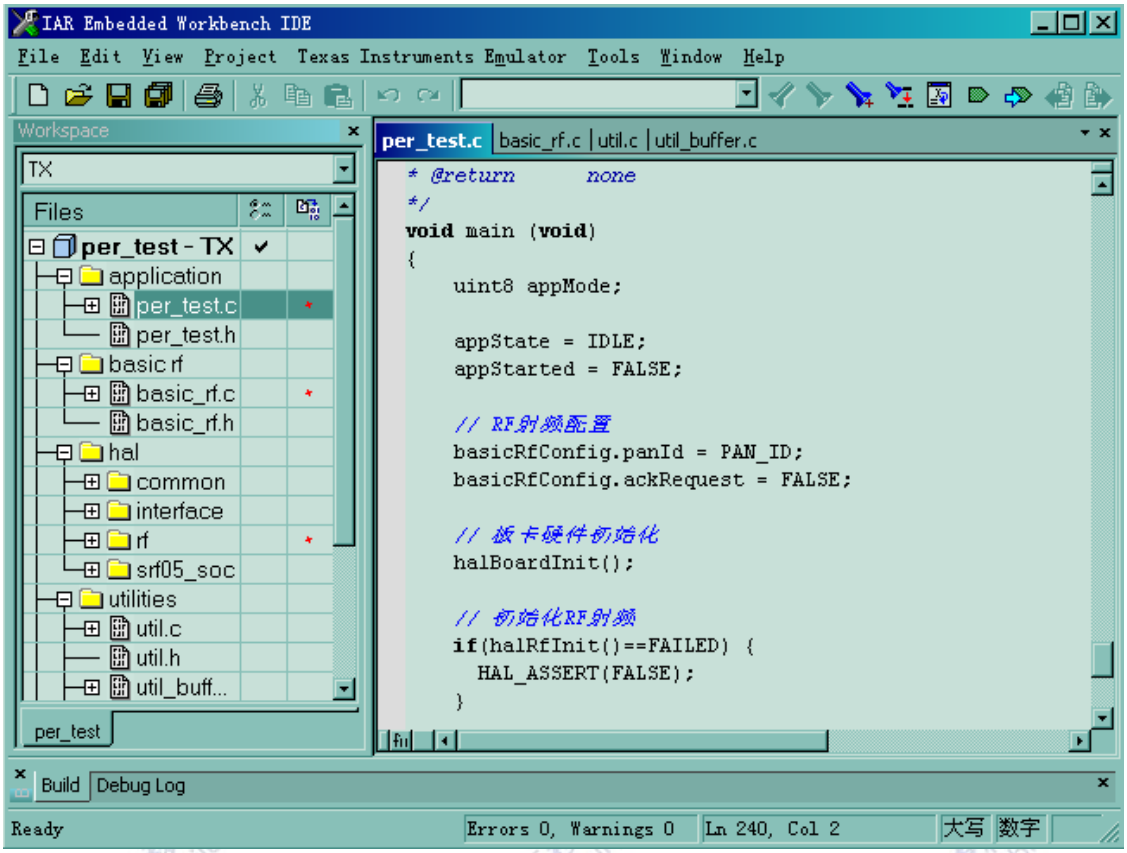


图5-13　工程文件界面

3、在IAR工程中选择选择RF状态后，分别下载到三个zigbee模块中：

（1）选择发送状态TX如图5-14所示，然后下载到zigbee模块中，作为点对多点无线通信的发送方；

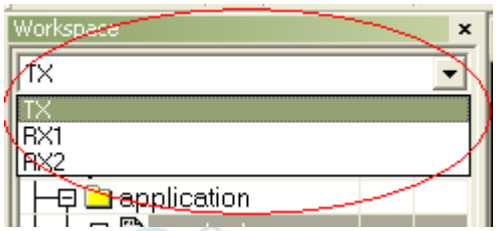
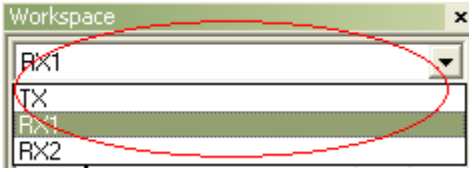


图5-14　图发送状态

（2）分别选择接收状态RX1和RX2，如图5-15所示，然后分别下载到zigbee模块中，作为点对多点无线通信的两个接收方。



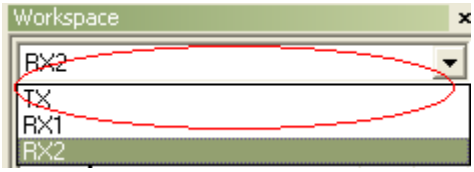


图5-15　接收状态

4、实验结果：

运行的结果是发送和接收模块上的小灯交替闪烁。 从程序上也能分析出实验现象：

1） zigbee数据发送模块：zigbee发送模块上电后，黄灯开始闪烁，表示数据已经正常发送，黄灯每闪烁一次，表示数据成功向两个zigbee模块轮训发送出去一次；

2） zigbee数据接收模块：zigbee接收模块RX1和RX2上电后，红灯一直闪烁，表示接收模块正常工作，准备好了接收工作；若接收模块的黄灯闪烁，则表示接收到了zigbee数据发送模块的数据，黄灯闪烁一次表示成功接收到一次空口数据。

六、　拓展思考

本实验是通过在发送端对要发送的目的地址进行增加，然后轮询进行发送，实现了点对多点的 zigbee 无线通信，本实现还可以采用 FDMA（频分多址）来实现，即改变数据收发的频率，让 zigbee 设备切换不同的信道去监测空口数据或者利用不同的信道对数据进行发送，此种实现请同学根据点对多点的源码进行修改。

提示：修改点在 basicRfConfig 的 channel 变量配置和 basicRfInit()函数中调用 halRfSetChannel(pConfig->channel)对信道的设置。

七、　参考阅读

请参考阅读“\E-Box300\05-芯片数据手册\CC2531”目录下的CC253x数据手册。

实验代码请参考\E-Box300\03-实验源代码\02-WSN\04-无线传输实验\ 41_CC2531点对多点通信.rar。

5.2.3　实验四十二 无线自组网实验

一、　实验目的

1.了解无线自组网工作原理。

2.掌握利用 ZigBee 协议栈和传感器组建无线传感网络的方法。

二、　实验设备

- ARM网关（EBA）
- ZigBee模块（7个）
- 温湿度传感器（EBS-T）（已集成光传感器）
- 烟雾传感器（EBS-MQ）
- 加速度传感器（EBS-A）
- 气压传感器（EBS-PR）
- 霍尔传感器（EBS-HALL）
- RFID模块（EBS_RF）
- 继电器（EBS-R）
- 天线（8根）
- IAR Embedded Workbench开发环境

三、 实验内容

本实验使用 IAR Embedded Workbench 环境和物理地址烧写软件 SmartRF Flash Programmer，利用 ZigBee 协议栈和传感器组建无线传感网络，学习 zigbee 网络组成过程以及各个传感器模块的工作原理和功能。

四、 实验原理

1.无线自组网简介

无线自组织网络，是一种不同于传统无线通信网络的技术。传统的无线蜂窝通信网络，需要固定的网络设备如基站的支持，进行数据的转发和用户服务控制。而无线自组织网络不需要固定设备支持，各节点即用户终端自行组网通信。这种网络形式突破了传统无线蜂窝网络的地理局限性，能够更加快速、便捷、高效地部署，适合于一些紧急场合的通信需要，如战场的单兵通信系统。举一个简单的例子就可以说明这个问题，当一队伞兵空降后，每人持有一个ZigBee网络模块终端，降落到地面后，只要他们彼此间在网络模块的通信范围内，通过彼此自动寻找，很快就可以形成一个互联互通的ZigBee网络。他们预先无法判断谁是谁，而且，由于人员的移动，彼此间的联络还会发生变化。在这种情况下，无线自组网模块可以通过重新寻找通信对象，确定彼此间的联络，对原有网络进行刷新，方便快捷的实现了通信。在实际工业现场，由于各种原因，往往并不能保证每一个无线通道都能够始终畅通，就像城市的街道一样，可能因为车祸，道路维修等，使得某条道路的交通出现暂时中断，此时由于我们有多个通道，车辆（相当于我们的控制数据）仍然可以通过其他道路到达目的地。而这一点对工业现场控制而言则非常重要。

无线自组织网络一般是由几十到上百个节点组成、采用无线通信方式、动态组网的多跳

移动性对等网络。其目的是通过动态路由和移动管理技术传输具有服务质量要求的多媒体信息流。通常节点具有持续的能量供给。

无线自组网的工作原理简单来讲就是动态路由。所谓动态路由是指网络中数据传输的路径并不是预先设定的，而是传输数据前，通过对网络当时可利用的所有路径进行搜索，分析它们的位置关系以及远近，然后选择其中的一条路径进行数据传输，如传不通，再使用另外一条稍远一点的通路进行传输，以此类推，直到数据送达目的地为止。在实际工业现场，预先确定的传输路径随时都可能发生变化，或者因各种原因路径被中断了，或者过于繁忙不能进行及时传送。动态路由结合网状网拓扑结构，就可以很好解决这个问题，从而保证数据的可靠传输。

无线自组织网络也存在网络带宽受限、对实时性业务支持较差、安全性不高的弊端。目前，国内外有大量研究人员进行此项目研究。

本实验的网络结构图如下图5-16所示：

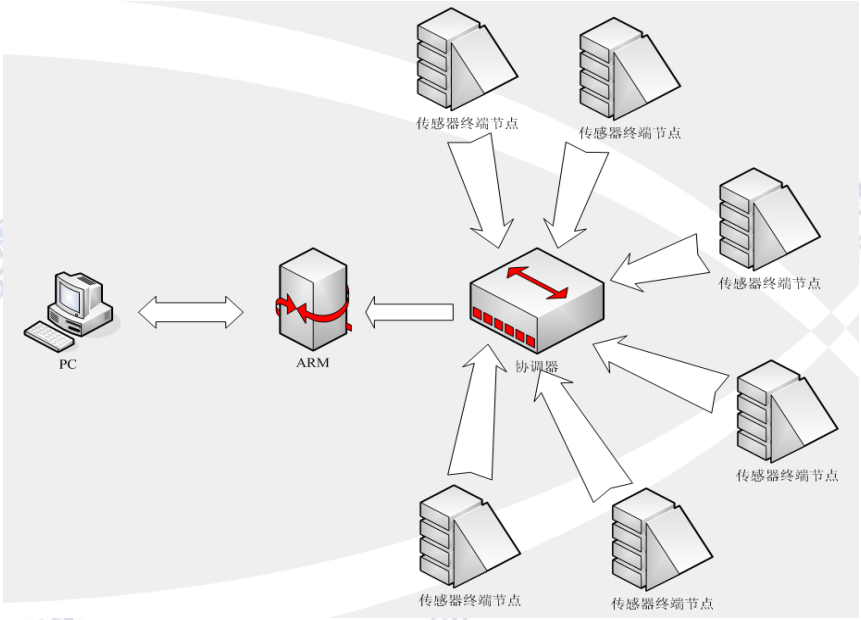


图5-16　网络结构图

实验中有两种设备被配置：zigbee协调器和zigbee传感器终端。

整个实验流程是：zigbee传感器终端节点加入到zigbee网络并正常工作后，采集各种传感数据，并将它们发送到协调器进行处理。这里为了实验简单，设置一个协调器收集这些信息，处理后通过串口发送给ARM网关，继而传输到计算机。

这个实验实现了：

- 多个Zigbee模块自动形成一个zigbee无线传感网络
- 各个zigbee传感器终端模块能够正常进行相应传感数据的采集并通过zigbee网络上报给zigbee协调器。

2、协调器

在zigbee网络中，zigbee节点分为协调器、路由器和终端节点三种角色。其中协调器负责对整个zigbee网络进行建立管理，负责PAN网络的初始化，确定PAN的ID号和PAN操作的物理信道并统筹短地址分配，充当信任中心和储存安全密钥，与其他网络的连接等。一个网络中只能有一个协调器。

协调器的工作流程如下图5-17所示：首先zigbee协调器上电后对zigbee协议栈系统进行初始化，然后通过MAC层对信道的扫描选择zigbee通信信道和网络PANID后，完成对网络的建立。当zigbee设备完成网络的建立后，开始对无线信号进行监测，当监测到zigbee终端的入网请求信息后，会对终端的入网进行处理，允许或禁止该终端的入网请求。同时协调器的会时时的检测串口，当有ARM网关的数据请求或控制请求时，会进行相应的处理。

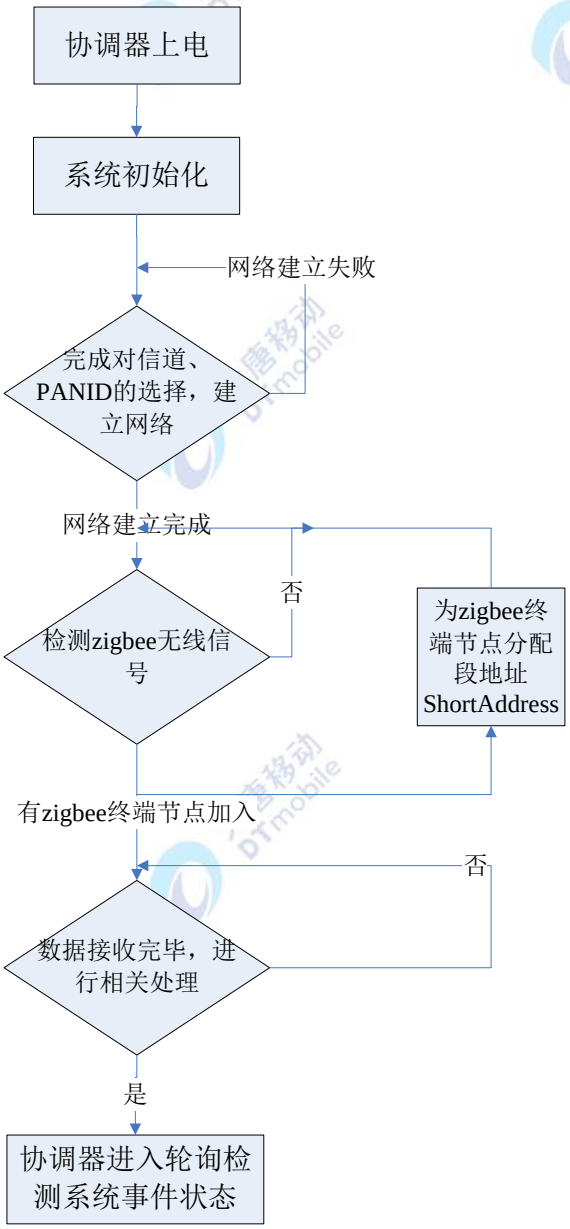


图5-17　协调器工作流程图

3、zigbee 传感器终端节点

zigbee 传感器终端节点完成对各种传感数据的采集以及对传感器进行控制等功能。如下图 5-18 所示，为 zigbee 传感器终端节点的处理流程。

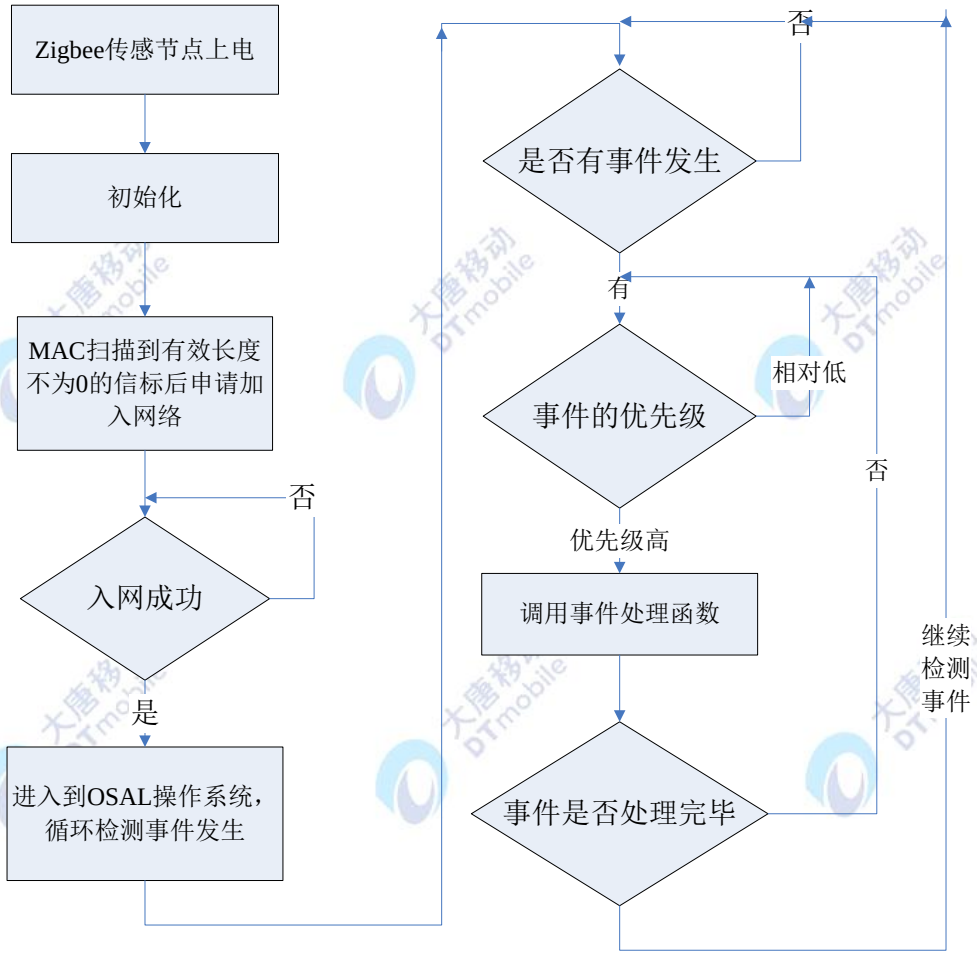


图5-18　传感器终端节点处理流程图

Zigbee 传感器终端节点的入网流程：

1、 传感器终端节点通过 MAC 层连接程序对网络进行连接，首先终端节点的 MAC 层对无线信道进行扫描，当扫描到有效长度不为零的信标 beacon 时（信标中包含信标的设备地址、网络是否允许连接以及信标载荷），验证是否与 IEEE802.15.4 协议中规定的 zigbee 协议标识符匹配，若匹配，则将信标中的信息复制到本设备的邻居表中。

2、 然后网络层通过 NLME_JION.request 原语申请加入到 zigbee 网络后，当协调器允许该入网请求后会根据该终端节点的 64bit 物理地址为该节点分配一个 16bit 的 shortAddr（短地址），在以后与协调器直接通过该短地址进行通信。

3、 在传感器节点成功连接到网络后，传感器节点需要主动报告自己的 64bit 的物理地址、16bit 的网络地址和自己的传感器类型。

4、 编译选项中增加 NV_RESTORE 的编译选项，以便在 NV 中保存 zigbee 的网络状态，

当终端重启后会读取 NV_RESTORE 中存储的网络状态直接对网络进行连接，不需要在通过 MAC 层扫描信道、检测信标帧等步骤，提高了 zigbee 终端节点入网的速度和稳定性。

5、 传感器节点依据 zigbee 协议栈中的 OSAL 机制循环对任务进行监测，当检测到任务时，则调用相应的回调函数对任务进行处理。传感器节点需要处理的事件：

（1）空口事件：接收到空口的消息后，解析出空口消息包中不同的消息 ID，从而进行不同的消息处理；

（2）按键事件：可以通过中断或轮训方式进行检测处理；

（3）串口事件：波特率是 115200；

（4）自定义事件：用户根据自身需要自定义的事件。

五、 实验步骤

1. 给 zigbee 模块下载程序：

1.1. 使用 JTAG 仿真器连接 zigbee 模块和 PC 机；

1.2. 打开软件 SmartRF Flash Programme（物理地址烧写软件）：

如果连接正常，在 SmartRF Flash Programme 的 Chip type 中可以看到芯片类型，如图 5-19 所示，显示的芯片类型为 CC2531：

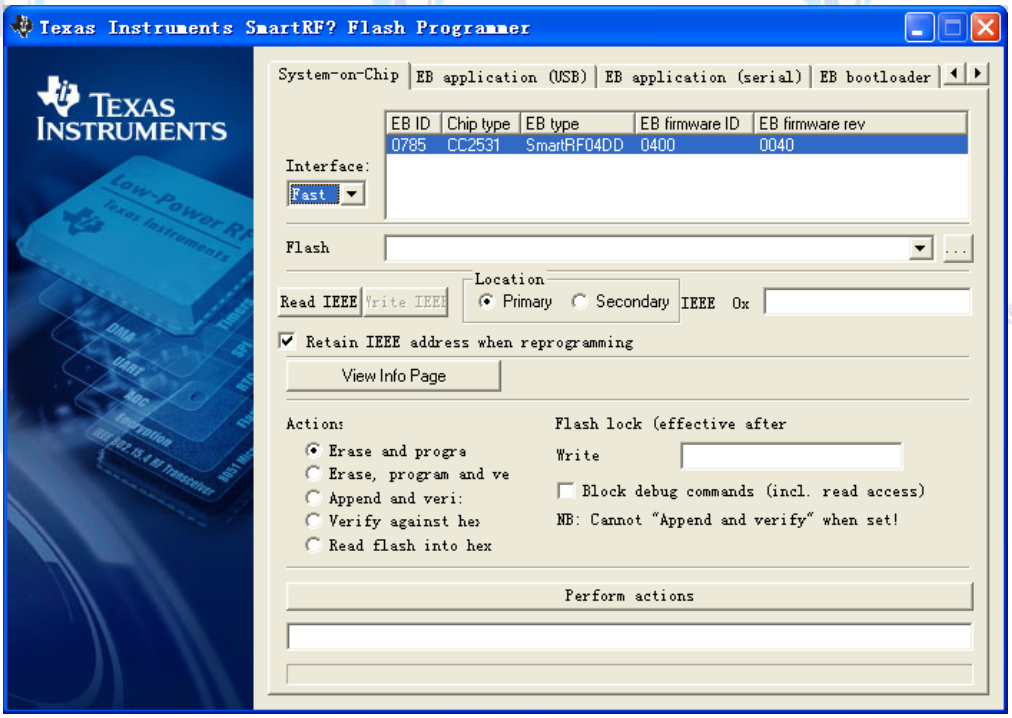


图5-19 SmartRF Flash Programmer软件界面

1.3. 下载*.hex 文件

找到下载所需的程序（*.hex 文件），分别为协调器程序以及各个传感器板卡程序（程序位置在：\E-Box300\03-实验源代码\02-WSN\04-无线传输实验）。使用 SmartRF Flash Programmer 软件将*.hex 文件下载到协调器以及各个传感器模块中（协调器模块已经集成在 ARM 网关上）。

首先，使用 SmartRF Flash Programmer 软件打开将要下载的*.hex 文件，如图 5-20 所示：

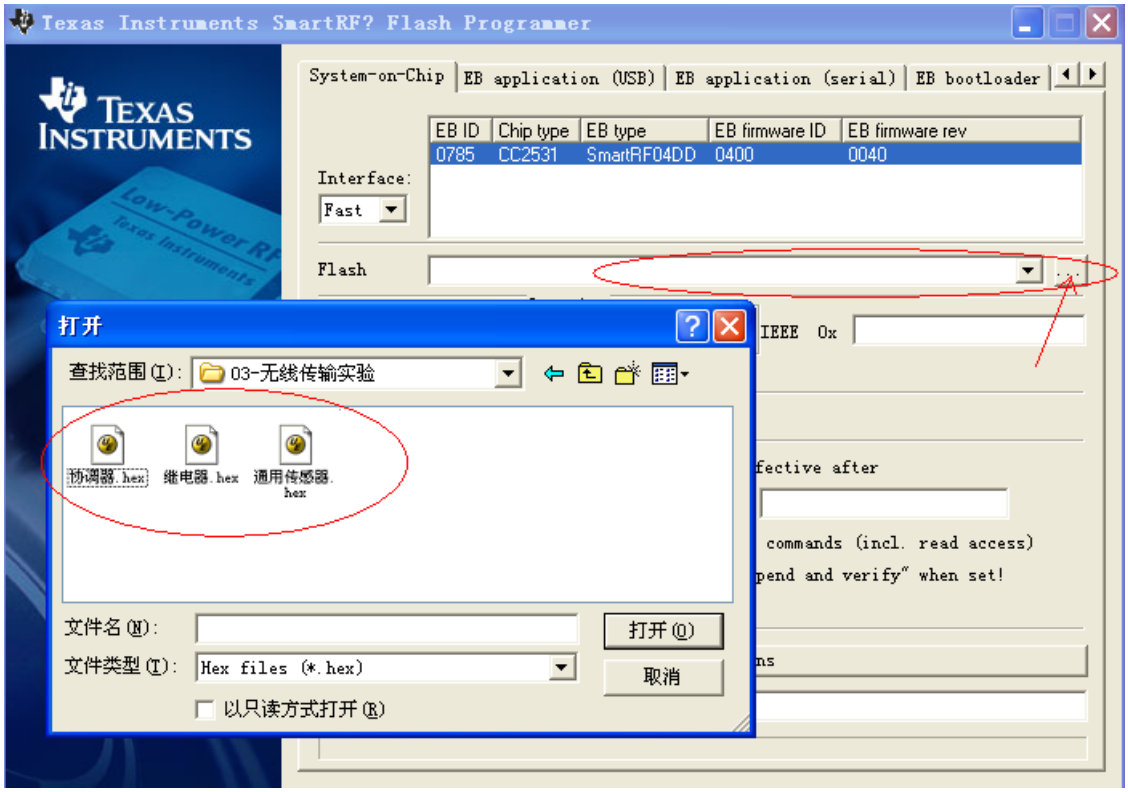


图5-20　打开将要下载的*.hex文件

然后，打开“协调器.hex”文件后，点击 Perform actions，如图 5-21 所示：

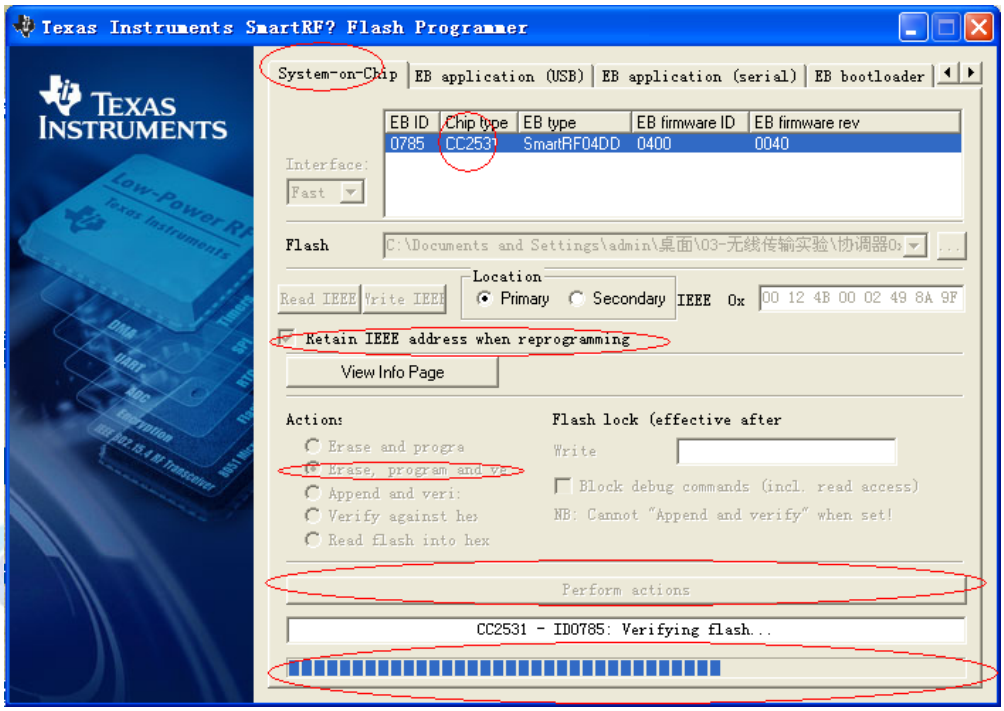


图5-21　执行对*.hex文件的烧写

最后，完成对*.hex 文件的烧写，如图 5-22 所示，即完成了对协调器程序文件的烧写：

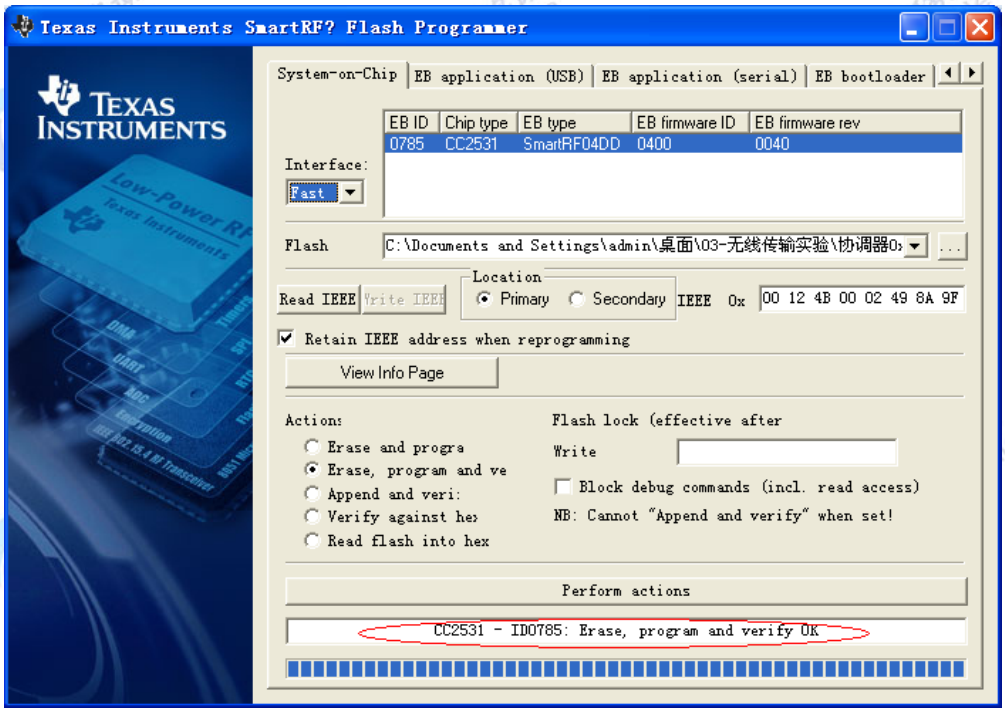


图5-22　完成对*. hex文件的烧写

用同样的方法分别对其他传感器终端节点模块进行程序烧写。

注：传感器节点的 hex 文件分为通用传感器.hex 和继电器.hex，即传感器模块除了继电器需要下载“继电器.hex”外，其他的传感器模块均下载“通用传感器.hex”。

2.　将 PC 机 IP 修改为：192.168.1.120，子网掩码为： 255.255.255.0。

3. 启动 ARM 网关，与 PC 建立连接。选择 ARM 网关 DTmobile 标签下的 ARMGEATWAY 应用程序，并点击“运行”按钮。ARM 网关作为 zigbee 网络与 PC 机之间的桥梁负责采集数据转发给 PC 机，并且将 PC 机的请求发送给 zigbee 协调器。

4. 打开上位机软件，如下图所示，选择“网络拓扑”菜单选项，启动无线传感器网络上位机，传感器节点会自动搜索并加入到指定 PANID 标识的网络，zigbee 传感器终端模块黄灯不停闪烁表示此传感器节点正在寻找网络并请求加入，黄灯和红灯都亮表示此传感器节点已经加入网络。当传感器节点成功加入到网络后，网络拓扑图上会显示出各种传感器终端节点。如图 5-31 所示。

图5-23　网络拓扑图

5. 观察实验结果：右击每个传感器图标，查看传感器实时采集的数据值，图 5-24 至图 5-16 是 PC 机收到的各传感器的检测结果。

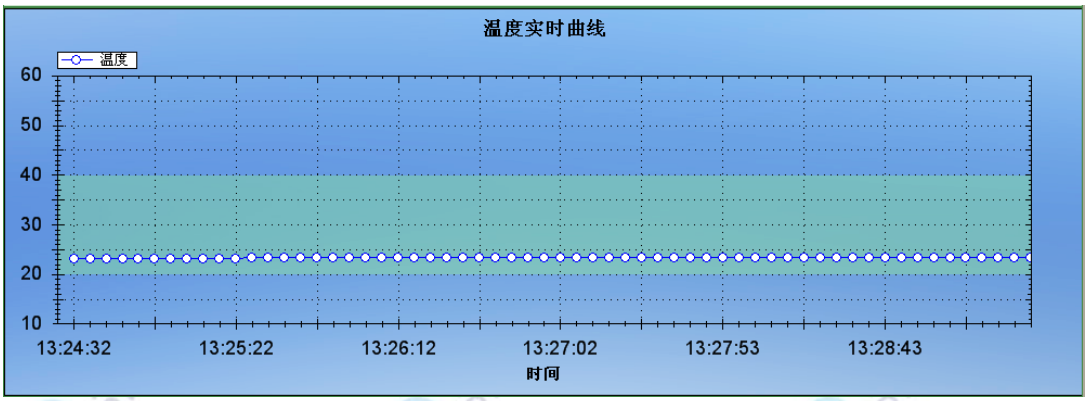


图5-24 温度传感器数据采集结果

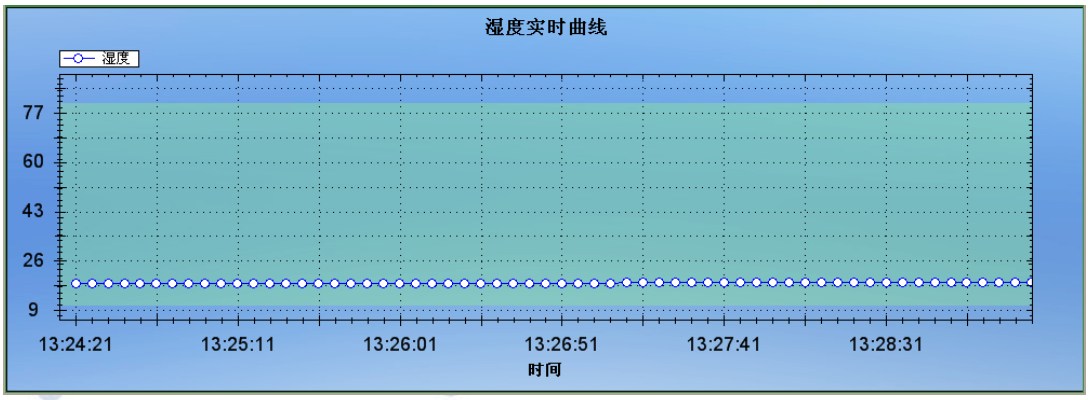


图5-25 湿度传感器数据采集结果

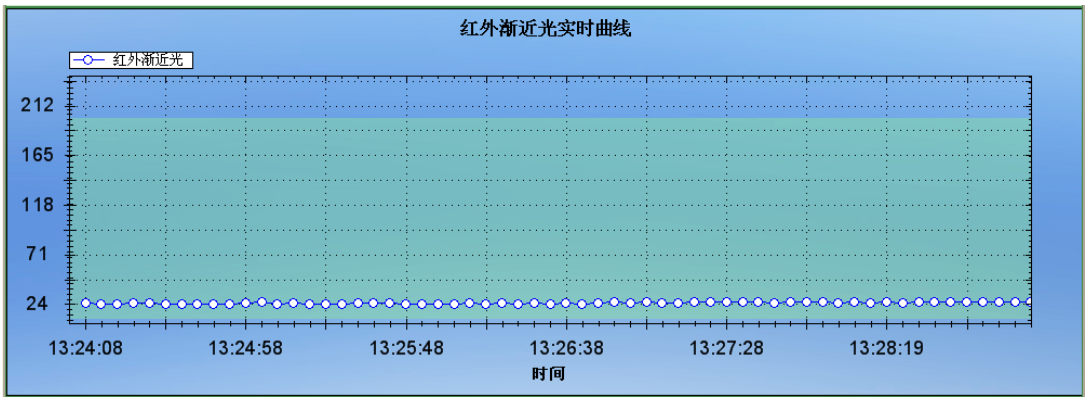


图5-26 红外渐进光数据采集结果

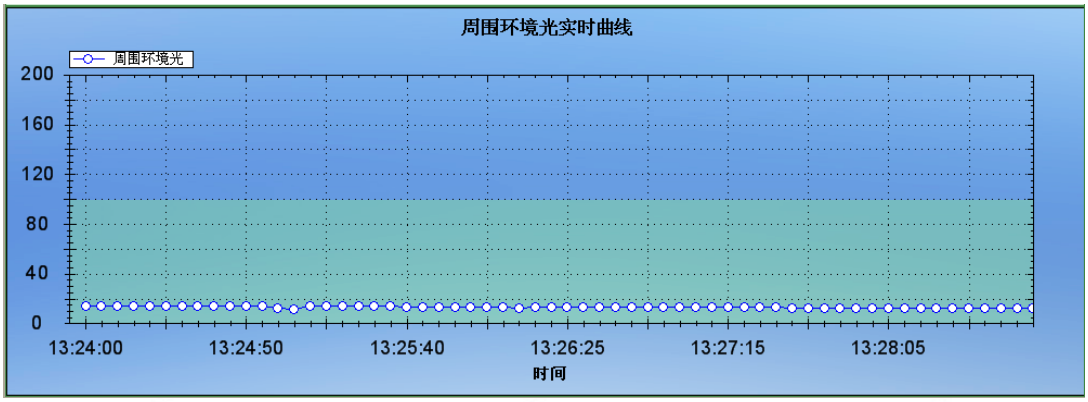


图5-27　周围环境光数据采集结果

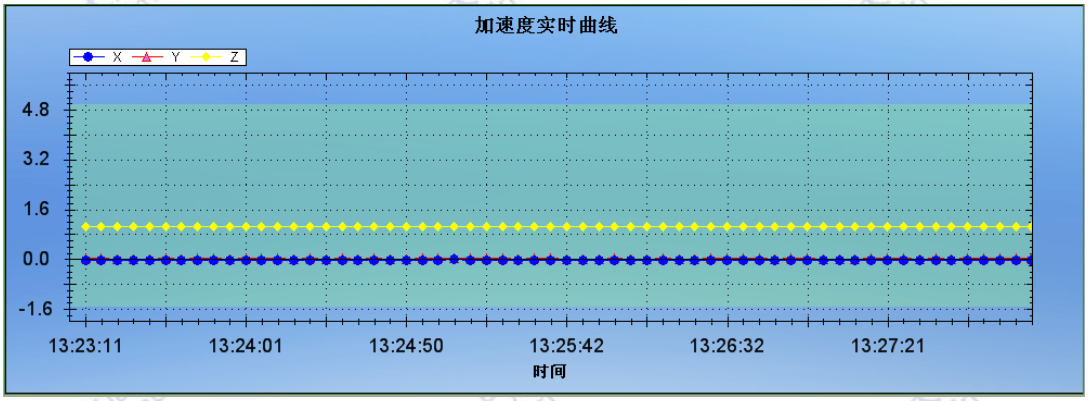


图5-28　加速度传感器数据采集结果

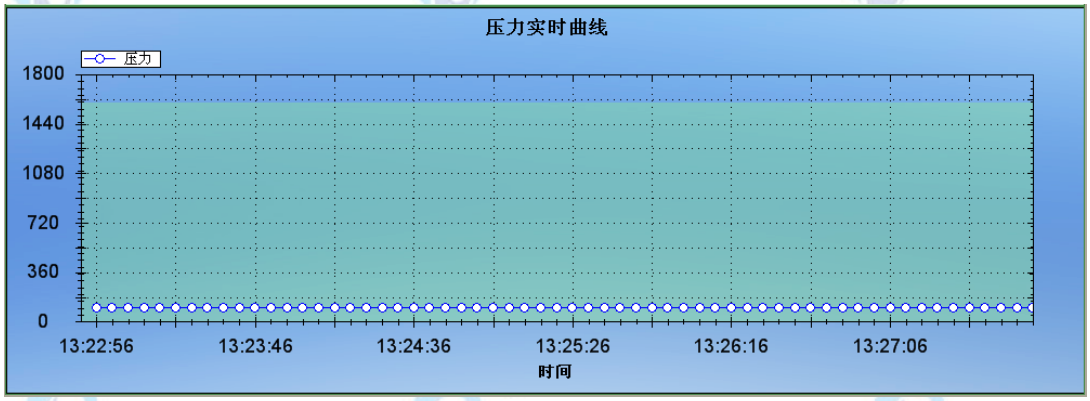


图5-29　气压传感器数据采集结果

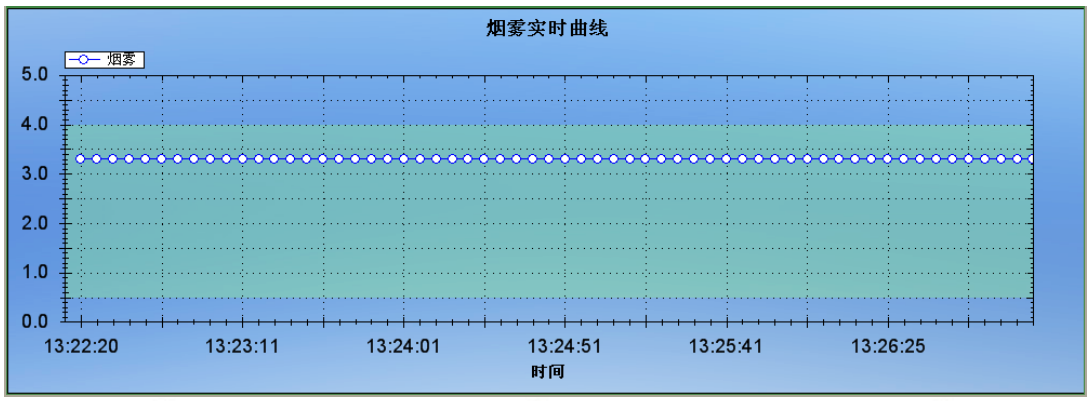


图5-30　烟雾传感器数据采集结果

图5-31　继电器状态查询结果并更改状态

6. 在做无线 RFID 充值消费实验的操作步骤如下：

1）在做 RFID 无线充值消费时，首先选中 RFID 节点后右击鼠标，选中弹出的“查看 RFID 信息”选项，如图 5-32 所示：

图5-32　查看RFID信息

2）获取 IC 卡卡号：如图 5-33 所示，点击“获取卡号”，上位机通过协调器会向 RFID 模块请求 IC 卡的卡号，RFID 模块收到卡号数据请求后，红灯常亮，黄灯灭；当把 IC 卡放到 RFID 模块的射频天线处，RFID 会读取到 IC 卡的卡号，然后会上发 IC 卡卡号给协调器并通过上位机显示出来，RFID 模块成功读取卡号并发送出数据后，蜂鸣器会响一声并且红灯和黄灯都是关闭状态。

图5-33　RFID界面

3）充值：在充值框中输入要给 IC 卡充值的金额，然后点击充值，待 RFID 收到充值请求后（红灯亮），将 IC 卡贴近（相距约 1cm~2cm）RFID 模块的 RFID 射频线圈处，待红灯灭且蜂鸣器响一声后，再将 IC 卡拿开，充值结果如图 5-34 所示：

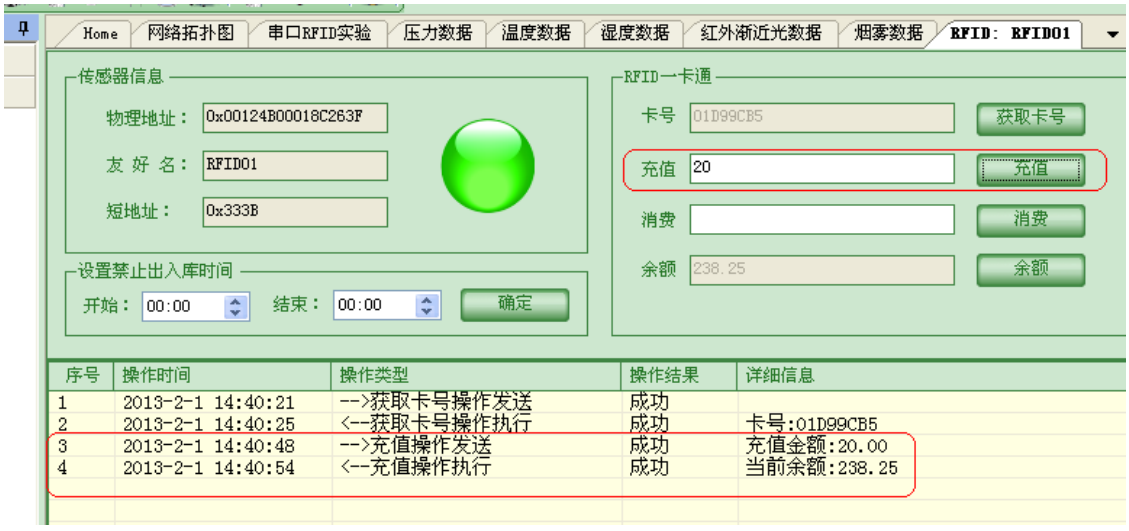


图5-34　充值结果

4）消费和余额查询：操作步骤同步骤三中的充值一样，结果如图 5-35 所示：

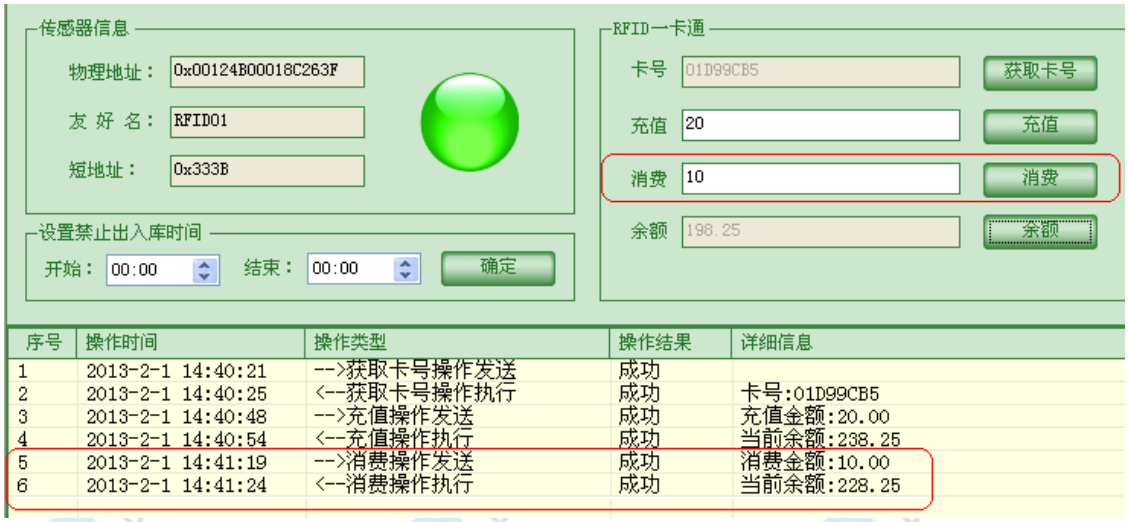


图5-35　消费和余额查询

5）无线 RFID 充值消费结束。在做实验时需要注意，在对 RFID 刷卡操作时，需要贴近 RFID 射频线圈（约 1cm~2cm），待蜂鸣器响一声后且红灯灭再将 IC 卡拿开。

注：本实验中集成到 ARM 网关的协调器硬件组成与单独的 zigbee 模块并无区别，为了能更直观的了解 zigbee 组网的过程，我们通过上位机软件将组网的最终结果以图形化的方式展现。如做其他不需要上位机显示的扩展实验，也可以用单独的 zigbee 模块作为协调器，通过设置串口打印信息进行调试。

六、　拓展思考

基于此 zigbee 无线传感网络可以做哪些实际应用；譬如说智能安防、仓库管理等。如果让你去设计一个这样的应用场景，你会如何去设计。提示：在实际的应用中有些传感器模块间需要建立绑定联系，如温度达到一定的温度后需要使用继电器打开风扇开关等。

七、　参考阅读

1.zigbee 协议栈源码：目录位置：E-Box300\03-实验源代码\02-WSN\05-2007 协议栈代码；
2.zigbee 协议栈文档：目录位置： E-Box300\03-实验源代码\02-WSN\05-2007 协议栈代码\Texas Instruments\ZStack-CC2530-2.2.0-1.3.0\Documents

5.3 WIFI 网络实验

5.3.1　实验四十三 WIFI 接入实验

一、　实验目的

1.了解 WIFI 相关基础知识。

2. 完成 WIFI 数据传输的流程和方法。

二、 实验设备

- 天线
- ARM 网关（EBA）

三、 实验内容

本实验主要实现在Linux下通过WIFI进行数据传输的应用程序开发流程与方法。在该实验中使用UDP协议，使用Socket编程方式，通过WIFI实现无线传输字符串。

四、 实验原理

1.UDP

UDP是一个面向数据报和无连接的简单传输层协议．它不像TCP那样通过握手过程建立服务器与客户端的连接才可以工作。在网络通信质量较好的情况下，UDP体现出高效率，这适合于传送少量报文的应用。linux系统是通过套接字结构来进行网络编程的，应用程序通过对套接字的几个函数调用，会返回一个用于通信的套接字描述符，而Linux应用程序在进行任何形式的I／O操作时，程序实际上是在读写一个文件描述符。因此Linux下的套接字编程，可以看成是对普通文件描述符的操作，这些操作与被使用的硬件平台无关，这是linux设备无关性的优点。

2.Socket

常用的Socket类型有两种：流式Socket（SOCK_STREAM）和数据报式Socket（SOCK_DGRAM）。流式是一种面向连接的Socket，针对于面向连接的TCP服务应用；应用流式Socket 可以保证数据传输中的完整性、正确性和单一性，可类比于现实生活中的打电话。数据报式Socket是一种无连接的Socket，对应于无连接的UDP服务应用。数据报式Socket可以象流式Socket 一样进行数据的双向传输，但无法保证传输数据的完整性、正确性和单一性。可以类比于现实生活中的寄信。

Socket并不是以面向对象的方式提供的。Socket事实上并不是留给编程开发的，而是留给网络操作系统实现协议栈的一个支持模型。例如berkeley Socket和windows Socket。至于编程开发时使用的Socket模型和相应的API，则是操作系统服务API。事实上Socket也不是tcp/udp服务的全部而是部分。如果用电话来对应网络会话的话，端口应该正好比分机号。

raw Socket其实就是跳过TCP，UDP这一层。用Socket我们不光是只发数据了，还要TCP和UDP的头。 大家也可以自己一个字节一个字节来构建自己的IP数据报文，并非一定需要

通过Socket接口。（通过流行的winpcap，你可以抛开Socket，发送自己构建的任意的IP报文） 。Socket是一个大家都认为好用的构建和解析IP协议的接口而已（当然Socket并非仅仅能够构建和解析ip协议），其实有很多嵌入式设备还有其他的网络通讯接口（比如LWIP）。只不过在大部分操作系统上Socket更加通用而已。

五、 实验步骤

1.ARM网关断电。将天线插入wifi模块的SMA_WIFI部分，如图5-36所示：

图5-36 wifi插入天线位置

2. ARM网关上电，启动Linux系统。

3.本小节中是一个WIFI连接路由的示例。由于网络环境的不同，所以在您做本实验时，请根据实际情况进行设置。

4.在终端内依次执行下面的命令连接路由器。

```
#ifconfig –a （检测开发板所有网卡状况）
```

```
[root@DTmobile]# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:90:90:90:90:90
          inet addr:192.168.0.232  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:108 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:27:13:ED:27:A0
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

DM9000

本地环路

SDIO WIFI

#ifconfig eth0 down （关闭dm9000 网卡）

```
[root@DTmobile]# ifconfig wlan0 down
```

#ifconfig wlan0 up （启动SDIO WIFI）

```
[root@DTmobile]# ifconfig wlan0 up
```

#iwlist wlan0 scan （使用SDIO WIFI 扫描无线网络设备）

```
[root@DTmobile]# iwlist wlan0 scan
wlan0     Scan completed :
          Cell 01 - Address: 00:21:27:65:77:5E
                                    Frequency:2.437 GHz (Channel 6)
                    Quality=65/70  Signal level=-45 dBm
                    Encryption key:on
                    ESSID:"TP-LINK_65775E"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                              12 Mb/s; 24 Mb/s; 36 Mb/s
                    Bit Rates:9 Mb/s; 18 Mb/s; 48 Mb/s; 54 Mb/s
                    Mode:Master
                    Extra:tsf=00000045d2e3c9b
                    Extra: Last beacon: 590ms ago
                    IE: Unknown: 000E54502D4C494E4B5F363537373545
                    IE: Unknown: 010882848B960C183048
                    IE: Unknown: 030106
                    IE: Unknown: 2A0100
                    IE: Unknown: 32041224606C
                    IE: Unknown: DD0900037F01010008FF7F
                    IE: Unknown:
DD1A00037F03010000000212765775E02212765775E64002C010808
```

#iwconfig wlan0 essid “E-Box” (设置essid)

```
[root@DTmobile]# iwconfig wlan0 essid "E-Box"
```

如果路由器不支持自动获取IP功能，则输入命令：

```
#ifconfig wlan0 192.168.0.232 （设置SDIO WIFI 的IP）
```

```
[root@DTmobile]# ifconfig wlan0 192.168.0.232
```

若路由器支持自动获取IP功能，那么无需设置SDIO WIFI的IP，输入下面的命令：

```
#udhcpc –i wlan0
```

设置路由器访问密码，若路由器无密码则无需此命令。

```
#iwconfig wlan0 key “123456789”
```

```
[root@DTmobile]# iwconfig wlan0 key "123456789"
```

```
#route add default gw 192.168.0.201 （设置网关，路由器支持自动获取IP则无需此命令）
#ping 192.168.0.201 （ping 网关）
```

```
[root@DTmobile]# route add default gw 192.168.0.201
[root@DTmobile]# ping 192.168.0.201
PING 192.168.0.201(192.168.1.201):56 data bytes
64 bytes from 192.168.0.201:seq=1 ttl=64 time=3.806 ms
64 bytes from 192.168.0.201:seq=2 ttl=64 time=3.051 ms
64 bytes from 192.168.0.201:seq=3 ttl=64 time=3.981 ms
64 bytes from 192.168.0.201:seq=4 ttl=64 time=3.013 ms
64 bytes from 192.168.0.201:seq=5 ttl=64 time=3.051 ms
64 bytes from 192.168.0.201:seq=6 ttl=64 time=3.037 ms
```

本小节测试的路由器可以访问互联网，所以可以使用QTOPIA浏览器上网冲浪。

六、 拓展思考

编写 shell 脚本，一键完成 wifi 启动，接入路由及测试。

七、 参考阅读

5.3.2 实验四十四 WIFI 交互实验

一、 实验目的

1.掌握 Linux，学会利用 Linux 进行编程。

2.完成 WIFI 交互式数据传输的流程和方法。

二、 实验设备

- 天线
- ARM 网关（EBA）两个
- SD 卡

三、 实验内容

本实验主要实现在Linux下通过WIFI进行服务器与客户端之间交互式数据传输的应用程序开发流程与方法。在该实验中使用 TCP/IP 协议，使用 Socket 编程方式，通过 WIFI 实现无线传输字符串。

四、 实验原理

1. Socket

TCP/IP（Transmission Control Protocol/Internet Protocol）即传输控制协议/网间协议，是一个工业标准的协议集，它是为广域网（WANs）设计的。TCP/IP协议族包括运输层、网络层、链路层。Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议。Socket在其中的位置如图5-37所示：

TCP/IP仅仅是一套协议、规范，并没有实体存在，是完全抽象的；Socket模型则是清晰而具体的。而大部分情况下，门面模型是可有可无的，只是为了满足封装一个简单的外部门面而存在的，TCP/IP和Socket并没有相提并论的前提，根本是为了解决同一个问题的两个步骤，是完全分离的。抓包程序基本上是使用raw Socket(原始套接字)实现的,这是Socket的高级功能，并不是因为对TCP/IP协议很了解，而是对Socket很了解(当然不了解TCP/IP也无法理解高级的Socket)。离开特定操作系统(或者虚拟机)讨论TCP/IP的编程是没有意义的。

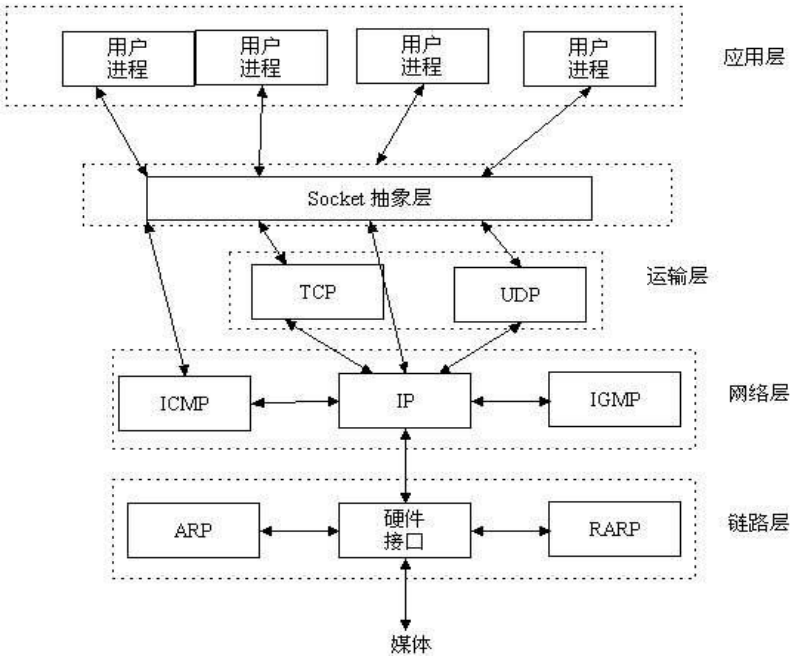


图5-37 Socket在网络协议中的位置

服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept

阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。如图5-38所示：

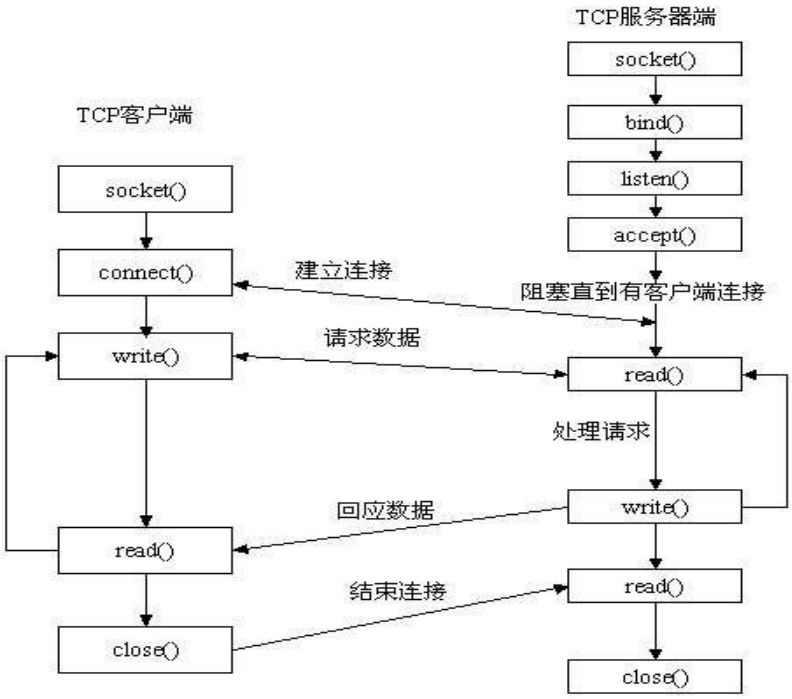


图5-38　服务器与客户端之间交互过程

1.1Socket 建立

为了建立 Socket，程序可以调用 Socket 函数，该函数返回一个类似于文件描述符的句柄。Socket 函数原型为：

```
int Socket(int domain, int type, int protocol);
```

domain 指明所使用的协议族，通常为 PF_INET，表示互联网协议族（TCP/IP 协议族）；type 参数指定 Socket 的类型：SOCK_STREAM 或 SOCK_DGRAM，Socket 接口还定义了原始 Socket（SOCK_RAW），允许程序使用低层协议；protocol 通常赋值"0"。 Socket()调用返回一个整型 Socket 描述符，你可以在后面的调用使用它。

两个网络程序之间的一个网络连接包括五种信息：通信协议、本地协议地址、本地主机端口、远端主机地址和远端协议端口。Socket 数据结构中包含这五种信息。

Bind 函数将 Socket 与本机上的一个端口相关联，随后你就可以在该端口监听服务请求。Bind 函数原型为：

```
int bind(int sockfd,struct sockaddr *my_addr, int addrlen);
```

Sockfd 是调用 Socket 函数返回的 Socket 描述符,my_addr 是一个指向包含有本机 IP 地址及端口号等信息的 sockaddr 类型的指针；addrlen 常被设置为 sizeof(struct sockaddr)。

struct sockaddr 结构类型是用来保存 Socket 信息的：

```
struct sockaddr
{
    unsigned short sa_family; /* 地址族，AF_xxx */
    char sa_data[14]; /* 14 字节的协议地址 */
};
```

sa_family 一般为 AF_INET，代表 Internet（TCP/IP）地址族；sa_data 则包含该 Socket 的 IP 地址和端口号。

另外还有一种结构类型：

```
struct sockaddr_in
{
    short int sin_family; /* 地址族 */
    unsigned short int sin_port; /* 端口号 */
    struct in_addr sin_addr; /* IP 地址 */
    unsigned char sin_zero[8]; /* 填充 0 以保持与 struct sockaddr 同样大小 */
};
```

这个结构更方便使用。sin_zero 用来将 sockaddr_in 结构填充到与 struct sockaddr 同样的长度，可以用 bzero()或 memset()函数将其置为零。指向 sockaddr_in 的指针和指向 sockaddr 的指针可以相互转换，这意味着如果一个函数所需参数类型是 sockaddr 时，你可以在函数调用的时候将一个指向 sockaddr_in 的指针转换为指向 sockaddr 的指针；或者相反。

使用 bind 函数时，可以用下面的赋值实现自动获得本机 IP 地址和随机获取一个没有被占用的端口号：

```
my_addr.sin_port = 0; /* 系统随机选择一个未被使用的端口号 */
my_addr.sin_addr.s_addr = INADDR_ANY; /* 填入本机 IP 地址 */
```

通过将 my_addr.sin_port 置为 0，函数会自动为你选择一个未占用的端口来使用。同样，通过将 my_addr.sin_addr.s_addr 置为 INADDR_ANY，系统会自动填入本机 IP 地址。

注意在使用 bind 函数是需要将 sin_port 和 sin_addr 转换成为网络字节优先顺序；而 sin_addr 则不需要转换。

面向连接的客户程序使用 Connect 函数来配置 Socket 并与远端服务器建立一个 TCP 连接，其函数原型为：

```
int connect(int sockfd, struct sockaddr *serv_addr,int addrlen);
```

Sockfd 是 Socket 函数返回的 Socket 描述符；serv_addr 是包含远端主机 IP 地址和端口号的指针；addrlen 是远端地质结构的长度。 Connect 函数在出现错误时返回-1，并且设置 errno 为相应的错误码。进行客户端程序设计无需调用 bind()，因为这种情况下只需知道目的机器 的 IP 地址，而客户通过哪个端口与服务器建立连接并不需要关心，Socket 执行体为你的程序自动选择一个未被占用的端口，并通知你的程序数据什么时候到打断口。

Connect 函数启动和远端主机的直接连接。只有面向连接的客户程序使用 Socket 时才需要将此 Socket 与远端主机相连。无连接协议从不建立直接连接。面向连接的服务器也从不启动一个连接，它只是被动的在协议端口监听客户的请求。

Listen 函数使 Socket 处于被动的监听模式，并为该 Socket 建立一个输入数据队列，将到达的服务请求保存在此队列中，直到程序处理它们。

```
int listen(int sockfd， int backlog);
```

Sockfd 是 Socket 系统调用返回的 Socket 描述符；backlog 指定在请求队列中允许的最大请求数，进入的连接请求将在队列中等待 accept()它们。Backlog 对队列中等待服务的请求的数目进行了限制，大多数系统缺省值为 20。如果一个服务请求到来时，输入队列已满，该 Socket 将拒绝连接请求，客户将收到一个出错信息。

当出现错误时 listen 函数返回-1，并置相应的 errno 错误码。

accept()函数让服务器接收客户的连接请求。在建立好输入队列后，服务器就调用 accept 函数，然后睡眠并等待客户的连接请求。

int accept(int sockfd, void *addr, int *addrlen);

sockfd 是被监听的 Socket 描述符，addr 通常是一个指向 sockaddr_in 变量的指针，该变量用来存放提出连接请求服务的主机的信息（某台主机从某个端口发出该请求）；addrten 通常为一个指向值为 sizeof(struct sockaddr_in)的整型指针变量。出现错误时 accept 函数返回-1 并置相应的 errno 值。

首先，当 accept 函数监视的 Socket 收到连接请求时，Socket 执行体将建立一个新的 Socket，执行体将这个新 Socket 和请求连接进程的地址联系起来，收到服务请求的初始 Socket 仍可以继续在以前的 Socket 上监听，同时可以在新的 Socket 描述符上进行数据传输操作。

数据传输

Send()和 recv()这两个函数用于面向连接的 Socket 上进行数据传输。

Send()函数原型为：

```
int send(int sockfd, const void *msg, int len, int flags);
```

Sockfd 是你想用来传输数据的 Socket 描述符；msg 是一个指向要发送数据的指针；Len 是以字节为单位的数据的长度；flags 一般情况下置为 0。

recv()函数原型为：

```
int recv(int sockfd,void *buf,int len,unsigned int flags);
```

Sockfd 是接受数据的 Socket 描述符；buf 是存放接收数据的缓冲区；len 是缓冲的长度。Flags 也被置为 0。Recv()返回实际上接收的字节数，当出现错误时，返回-1 并置相应的 errno 值。

如果你对数据报 Socket 调用了 connect()函数时，你也可以利用 send()和 recv()进行数据传输，但该 Socket 仍然是数据报 Socket，并且利用传输层的 UDP 服务。但在发送或接收数据报时，内核会自动为之加上目地和源地址信息。

结束传输

当所有的数据操作结束以后，你可以调用 close()函数来释放该 Socket，从而停止在该 Socket 上的任何数据操作：

```
close(sockfd);
```

（1）服务器端代码分析

服务器端一直在监听是否有客户端连接，如有连接，处理客户端的请求，给出回应，然后继续监听。

因为IP地址，端口号等信息需要自己输入，所以main函数需要参数。

```
int main(int argc, char **argv)
```

将第一个输入的字符串参数转化为整形数后，作为端口号赋值给myport，默认参数为7838

```
    if (argv[2])            myport = atoi(argv[2]);
    else   myport = 7838;
    printf("Port is ok!\n");
```

创建Socket，如果返回值为-1，则创建成功，否则创建失败

```
    if ((sockfd = Socket(PF_INET, SOCK_STREAM, 0)) == -1)
    {     printf("Creat Socket succeed !\n");
          perror("Socket");
          exit(1); }
    else
          printf("Creat Socket failed !\n");
```

将 my_addr空间清空，且包括“、0”

```
bzero(&my_addr, sizeof(my_addr));
```

设置地址族

```
my_addr.sin_family = PF_INET;
```

设置端口号 ，以大端方式赋值（htons()：把 16 位值从主机字节序转换成网络字节序）

```
my_addr.sin_port = htons(myport);
```

设置IP地址，将IP地址字符参数3，转换为32位网络序列的IP地址，如果没有参数3，则自动填入本机IP地址

```
if (argv[3])
    my_addr.sin_addr.s_addr = inet_addr(argv[1]);
else
    my_addr.sin_addr.s_addr = INADDR_ANY;
printf("Set IP succeed !\n");
```

调用bind函数将其与本机地址以及一个本地端口号绑定

```
if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr))== -1)
{
    printf("Fail to bind !\n");
    perror("bind");
    exit(1);
}
printf("Succeed to bind !\n");
```

下面是收发程序的主体，为了实现循环收发，使用了while死循环，需要断开时按ctrl+C

```
char recv[30];
char *send = “hello,this is server!”;
while (1)
{
    memset(recv, 0, sizeof(recv));
    recvfrom(sockfd, recv, sizeof(recv), 0, (struct sockaddr*)&client_addr, &addr_len);
    printf(“%s\n”, recv);
    sendto(sockfd, send, strlen(send), 0, (struct sockaddr*)&client_addr, &addr_len);
```

```
}
关闭Socket，返回
close(sockfd);
return 0;
```

（2）客户端代码分析

因为IP地址，端口号等信息需要自己输入，所以main函数需要参数

```
int main(int argc, char **argv)
```

如果输入参数不够，则退出

```
if (argc != 3)
{
    printf("Peremeter erro ! For example : \n\t\t%s IP port\n\t:\t%s 127.0.0.1 80\n",argv[0], argv[0]);
    exit(0);
}
else printf("\nProgram starts normally!\n");
```

创建一个 Socket 用于 tcp 通信

```
if ((sockfd = Socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("Fail to creat Socket!\n");
    perror("Socket");
    exit(errno);
}
else printf("Creat Socket succeed!\n");
```

初始化服务器端（对方）的地址和端口信息

```
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0)
{
    printf("Fail to initial ip address and port!\n");
```

```
        perror(argv[1]);
        exit(errno);
    }
    else printf("Initial ip address and port succeed!\n");
```

连接服务器

```
    printf("\n     Connecting......\n");
    connum = connect(sockfd, (struct sockaddr *) &dest, sizeof(dest));
    printf("Connect fail becasue of %d %d\n",connum,errno);
    if ( connum != 0)
    {
        printf("Connect fail! \n");
        perror("Connect ");
        exit(errno);
    }
    else printf("Connect succeed!\n");
    printf ("\nEverything is ready.Please type in and press enter to send messages!\n");
```

下面是收发程序的主体，为了实现循环收发，使用了while死循环，需要断开时按ctrl+C

```
    char recv[30];
    char *send = “hello, this is client!”;
    while (1)
    {
        memset(recv, 0, sizeof(recv));
        write(sockfd, send, strlen(send));
        read(sockfd, recv, sizeof(recv));
        printf(“%s\n”, recv);
        sleep(5);
    }
```

关闭连接，返回

```
    close(sockfd);
    return 0;
```

五、 实验步骤

1. 代码：\E-Box300\03-实验源代码\01-ARM 嵌入式网关\02-wifi 网络实验，本实验的两个 ARM 网关分别作为服务器和客户端进行通信。

2. 将 severt 程序拷贝到 ARM 网关中：将代码拷贝到 SD 卡中，将 SD 卡插入 ARM 网关，启动 ARM 网关，在 ARM 网关的终端中将路径改为 SD 卡。

```
cd sdcard
./server  IP 号  端口号  （例如./server  196.168.1.1 4444）
```

注意：服务器的 IP 号要与此 ARM 网关的 IP 号一致（ARM 网关的 IP 号查看方法：DTmobile 标签→ARMGATEWAY→Network）。

3. 将 client 拷贝到 ARM 网关中：与步骤 2 一致。

```
cd sdcard
./client  IP 号  端口号 （IP 号与端口号与 sever 设置的一致，之后就可以将字符串发送给另一端，并在另一端终端上显示出来）。
```

六、 拓展思考

编写简易的聊天程序，控制数据内容及数据发送。

七、 参考阅读

5.4 实验四十五 视频采集实验

一、 实验目的

1.了解视频采集的工作原理。

2.掌握使用摄像头进行视频采集的方法。

二、 实验设备

- PC 机
- ARM 网关（EBA）
- EBM-CC(摄像头模块)

三、 实验内容

本实验主要实现在 Linux 下进行 CMOS 图像传感器驱动及应用程序开发流程与方法。

四、 实验原理

随着技术和需求的发展，视频采集技术在视频监控、电视会议、计算机图像处理和多媒体通信等技术领域起到关键性作用并得到广泛的应用，并且采集质量、效率和集成度上要求越来越高。传统的视频采集系统一般采用基于 PC 机平台加上视频采集卡的形式，该方案体积大、成本高，在远距离、多点系统中实现困难。嵌入式视频采集系统由于体积小、性能稳定的特点，逐步受到广泛应用。目前图像采集常用的两种图像传感器为 CCD 与 CMOS 图像传感器。CCD 一般输出带制式的模拟信号，需要经过视频解码器得到数字信号才能传入微处理器中， 而 CMOS 图像传感器直接输出数字信号，可以直接与微处理器进行连接。不同的 CMOS 图像传感器有不同的性能，主要表现在图像分辨率大小不同、帧速率不同、曝光方式不同等，CMOS 图像传感器可直接通过 I2C 来设置图像分辨率大小及曝光、增益等参数，而 CCD 图像传感器则需要对视频解码器进行设置来控制图像的曝光、增益等参数信息。相对于 CCD 图像传感器，CMOS 图像传感器具有低功耗、小体积、高速数据传输和方便控制等优点，因此，CMOS 图像传感器更适用于嵌入式系统应用中。在设计和提高图像采集速度时，CMOS 图像传感器具有成本低、集成度高、体积小、超低功耗等优点。

什么是video4linux? Video4linux（简称V4L),是linux中关于视频设备的内核驱动,现在已有Video4linux2，还未加入linux内核，使用需自己下载补丁。在Linux中，视频设备是设备文件，可以像访问普通文件一样对其进行读写，摄像头在/dev/videoN下，N可能为0，1，2，3... 一般为0。另外，推荐一个用于播放从摄像头采集到的raw数据的播放器RawPlayer，只需要把采集的数据保存到文件***.yuv就OK了。

V4L2采集视频流程 ：

- 打开设备文件。 int fd=open(”/dev/video0″,O_RDWR);
- 取得设备的 capability，看看设备具有什么功能，比如是否具有视频输入,或者音频输入输出等。VIDIOC_QUERYCAP,struct v4l2_capability
- 选择视频输入，一个视频设备可以有多个视频输入。VIDIOC_S_INPUT,struct v4l2_input
- 设置视频的制式和帧格式，制式包括 PAL，NTSC，帧的格式个包括宽度和高度等。VIDIOC_S_STD,VIDIOC_S_FMT,struct v4l2_std_id,struct v4l2_format.
- 向驱动申请帧缓冲，一般不超过 5 个。struct v4l2_requestbuffers
- 将申请到的帧缓冲映射到用户空间，这样就可以直接操作采集到的帧了，而不必去复制。Mmap
- 将申请到的帧缓冲全部入队列，以便存放采集到的数据.VIDIOC_QBUF,struct v4l2_buffer
- 开始视频的采集。VIDIOC_STREAMON

- 出队列以取得已采集数据的帧缓冲，取得原始采集数据。VIDIOC_DQBUF
- 将缓冲重新入队列尾,这样可以循环采集。VIDIOC_QBUF
- 停止视频的采集。VIDIOC_STREAMOFF
- 关闭视频设备。close(fd);

实验代码分析：

1. 打开视频设备

在V4L2中，视频设备被看做一个文件。使用open函数打开这个设备：

// 用非阻塞模式打开摄像头设备

```
int cameraFd;
cameraFd = open("/dev/video0", O_RDWR | O_NONBLOCK, 0);
```

// 如果用阻塞模式打开摄像头设备，上述代码变为：

```
//cameraFd = open("/dev/video0", O_RDWR, 0);
```

应用程序能够使用阻塞模式或非阻塞模式打开视频设备，如果使用非阻塞模式调用视频设备，即使尚未捕获到信息，驱动依旧会把缓存（DQBUFF）里的东西返回给应用程序。

2. 设定属性及采集方式

打开视频设备后，可以设置该视频设备的属性，例如裁剪、缩放等。这一步是可选的。在Linux编程中，一般使用ioctl函数来对设备的I/O通道进行管理：

```
int ioctl (int __fd, unsigned long int __request, .../*args*/) ;
```

在进行V4L2开发中，常用的命令标志符如下(some are optional)：

- VIDIOC_REQBUFS：分配内存
- VIDIOC_QUERYBUF：把 VIDIOC_REQBUFS 中分配的数据缓存转换成物理地址
- VIDIOC_QUERYCAP：查询驱动功能
- VIDIOC_ENUM_FMT：获取当前驱动支持的视频格式
- VIDIOC_S_FMT：设置当前驱动的频捕获格式
- VIDIOC_G_FMT：读取当前驱动的频捕获格式
- VIDIOC_TRY_FMT：验证当前驱动的显示格式
- VIDIOC_CROPCAP：查询驱动的修剪能力
- VIDIOC_S_CROP：设置视频信号的边框
- VIDIOC_G_CROP：读取视频信号的边框
- VIDIOC_QBUF：把数据从缓存中读取出来

- VIDIOC_DQBUF：把数据放回缓存队列
- VIDIOC_STREAMON：开始视频显示函数
- VIDIOC_STREAMOFF：结束视频显示函数
- VIDIOC_QUERYSTD：检查当前视频设备支持的标准，例如 PAL 或 NTSC。

2.1 检查当前视频设备支持的标准

在亚洲，一般使用PAL（720X576）制式的摄像头，而欧洲一般使用NTSC（720X480），使用VIDIOC_QUERYSTD来检测：

```
v4l2_std_id std;
do
{
   ret = ioctl(fd, VIDIOC_QUERYSTD, &std);
}
while (ret == -1 && errno == EAGAIN);
switch (std)
 {
      case V4L2_STD_NTSC:
      case V4L2_STD_PAL:
}
```

2.2 设置视频捕获格式

当检测完视频设备支持的标准后，还需要设定视频捕获格式，结构如下：

```
struct v4l2_format fmt;
memset ( &fmt, 0, sizeof(fmt) );
fmt.type                = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width       = 720;
fmt.fmt.pix.height      = 576;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
if (ioctl(fd, VIDIOC_S_FMT, &fmt) == -1)
{
   return -1;
```

```
}
v4l2_format结构如下：
struct v4l2_format
{
    enum v4l2_buf_type type;      // 数据流类型，必须是
                                  // V4L2_BUF_TYPE_VIDEO_CAPTURE
    union
    {
        struct v4l2_pix_format      pix;
        struct v4l2_window          win;
        struct v4l2_vbi_format      vbi;
        __u8       raw_data[200];
    }
    fmt;
};
struct v4l2_pix_format
{
    __u32           width;          // 宽，必须是16的倍数
    __u32           height;         // 高，必须是16的倍数
    __u32           pixelformat;    // 视频数据存储类型，例如是YUV4：2：2还是RGB
    enum v4l2_field   field;
    __u32           bytesperline;
    __u32           sizeimage;
    enum v4l2_colorspace      colorspace;
    __u32           priv;
};
```

2.3 分配内存

接下来可以为视频捕获分配内存：

```
struct v4l2_requestbuffers   req;
if (ioctl(fd, VIDIOC_REQBUFS, &req) == -1)
```

```
    {
        return -1;
    }
    v4l2_requestbuffers  结构如下：
    struct v4l2_requestbuffers
    {
        __u32 count;  // 缓存数量，也就是说在缓存队列里保持多少张照片
        enum v4l2_buf_type type;  // 数据流类型，必须是
                                  // V4L2_BUF_TYPE_VIDEO_CAPTURE
        enum v4l2_memory memory; // V4L2_MEMORY_MMAP 或 V4L2_MEMORY_USERPTR
        __u32 reserved[2];
    };
```

2.4 获取并记录缓存的物理空间

使用VIDIOC_REQBUFS，我们获取了req.count个缓存，下一步通过调用VIDIOC_QUERYBUF命令来获取这些缓存的地址，然后使用mmap函数转换成应用程序中的绝对地址，最后把这段缓存放入缓存队列：

```
    typedef struct VideoBuffer
    {
        void    *start;
        size_t   length;
    }
    VideoBuffer;
    VideoBuffer*    buffers = calloc( req.count, sizeof(*buffers) );
    struct v4l2_buffer buf;
    for (numBufs = 0; numBufs < req.count; numBufs++)
    {
        memset( &buf, 0, sizeof(buf) );
        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_MMAP;
```

```
        buf.index = numBufs;
        // 读取缓存
        if (ioctl(fd, VIDIOC_QUERYBUF, &buf) == -1)
        {
            return -1;
        }
        buffers[numBufs].length = buf.length;
        // 转换成相对地址
        buffers[numBufs].start = mmap(NULL, buf.length, PROT_READ | PROT_WRITE,
MAP_SHARED,fd, buf.m.offset);
        if (buffers[numBufs].start == MAP_FAILED)
        {
            return -1;
        }
        // 放入缓存队列
        if (ioctl(fd, VIDIOC_QBUF, &buf) == -1)
        {
            return -1;
        }
    }
```

2.5 视频采集方式

操作系统一般把系统使用的内存划分成用户空间和内核空间，分别由应用程序管理和操作系统管理。应用程序可以直接访问内存的地址，而内核空间存放的是 供内核访问的代码和数据，用户不能直接访问。v4l2捕获的数据，最初是存放在内核空间的，这意味着用户不能直接访问该段内存，必须通过某些手段来转换地址。一共有三种视频采集方式：使用read、write方式；内存映射方式和用户指针模式。 read、write方式，在用户空间和内核空间不断拷贝数据，占用了大量用户内存空间，效率不高。内存映射方式：把设备里的内存映射到应用程序中的内存控件，直接处理设备内存，这是一种有效的方式。上面的mmap函数就是使用这种方式。用户指针模式：内存片段由应用程序自己分配。这点需要在 v4l2_requestbuffers里将memory字段设置成V4L2_MEMORY_USERPTR。

2.6 处理采集数据

V4L2 有一个数据缓存，存放req.count数量的缓存数据。数据缓存采用FIFO的方式，当应用程序调用缓存数据时，缓存队列将最先采集到的视频数据缓存送出，并重新采集一张视频数据。这个过程需要用到两个ioctl命令,VIDIOC_DQBUF和VIDIOC_QBUF：

```
struct v4l2_buffer buf;
memset(&buf,0,sizeof(buf));
buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory=V4L2_MEMORY_MMAP;
buf.index=0;
//读取缓存
if (ioctl(cameraFd, VIDIOC_DQBUF, &buf) == -1)
{
    return -1;
}
//……视频处理算法
//重新放入缓存队列
if (ioctl(cameraFd, VIDIOC_QBUF, &buf) == -1)
{
    return -1;
}
```

3. 关闭视频设备

使用close函数关闭一个视频设备

```
close(cameraFd)
```

实际代码解析：

打开摄像头，如果不能打开则提示打开失败，否则提示打开成功

```
v4l2_fd = open(V4L2_DEV_NODE, O_RDWR);
printf("already open the devicise codec \n");
if (v4l2_fd < 0)
{
    printf(" open video ERR\n");
    return 0;
```

```
}
```

打开液晶屏设备，如果不能打开则提示打开失败，否则提示打开成功

```
printf("Before openning FB \n");
fb_fd = open(FB_DEV_NODE, O_RDWR);
if (fb_fd < 0)
{
        printf(" open FB ERR\n");
        return 0;
}
```

定义显示屏所需要的对应空间的大小，一个像素占两个字节

```
screensize = fb_xres * fb_yres * 16 / 8;   // 16bpp
```

映射空间，将screensize大小的空间映射给fb_buf

```
fb_buf=(__u16*)mmap(0,screensize,PROT_READ|PROT_WRITE,MAP_SHARED,fb_fd, 0);
```

将映射区域的首地址赋值给 fb_buf_bak，以便需要首地址时使用

```
fb_buf_bak = fb_buf;
```

如果映射函数的返回值是-1，则表示映射失败

```
if ((int)fb_buf == -1)
{
        printf("Error: failed to map framebuffer device to memory.\n");
        close(fb_fd);
        return -1;
}
```

获取摄像头的信息

```
if((n =(ioctl(v4l2_fd, VIDIOC_QUERYCAP, &caminfo)))<0)
 {
        printf("Error: failed to get camera info.%d\n",n);
        return 0;
}
```

设置图象模式

```
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width = COLS;                                   //图像宽
fmt.fmt.pix.height = ROWS;                                  //图像高
fmt.fmt.pix.depth = 16;                                 //像素对应的位长
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB565;     //将格式设置为RGB565
```

如果返回值小于0，则表示图像模式设置失败

```
printf("VIDIOC_S_FMT is %x\n",VIDIOC_S_FMT);
if((ioctl(v4l2_fd, VIDIOC_S_FMT, &fmt))<0)
{
        printf("Error: failed to set video format.\n");
        return 0;
}
```

分配宽*高*3个字节的空间供lcd显示使用，并将空间的首地址赋值给buf_bak保存

```
buf = malloc(COLS*ROWS*3);
buf_bak = buf;
```

开始视频的采集，如果不能采集则提示失败，否则提示开始采集

```
printf("VIDIOC_STREAMON is %x\n",VIDIOC_STREAMON);
if((ioctl(v4l2_fd, VIDIOC_STREAMON, &type))<0)
{
        printf("error ioctl streamon\n");
        return 0;
}
else
      printf("start to get pic %d\n",(int)fmt.fmt.pix.sizeimage);
```

为了获得视频，所以采用了while的死循环。

```
while(1){}
```

每次显示图片前，都将摄像头存储图像的区域指针，

```
fb_buf = fb_buf_bak;
buf = buf_bak;
```

将COLS*ROWS*2个字节的内容从v4l2_fd的空间中复制到buf的空间中

```
n=read(v4l2_fd, buf, COLS*ROWS*2);
```

跳过屏幕的前14行不显示

```
fb_buf += (ScreenWidth_7*14); //
```

显示函数主体：

```
for( y=0; y<ROWS; y++)
{
    for( x=0; x<COLS; x++)
    {                                    //
        calculate 2 time
         fb_buf[x]     = *buf++ ; //将buf中的像素显示在液晶屏上
     }
     fb_buf += ScreenWidth_7 – COLS;//跳过一行中超出图片宽度的部分
}
```

五、 实验步骤

1.将 EBM-CC（摄像头模块）插入 ARM 网关 CAMERA 模块的 xp5 接插件内，如图 5-39 所示，注意摄像头插入方向， CAMERA 模块有箭头的方向与摄像头方向一致。

图5-39 摄像头插入方向

2. 启动 ARM 网关，选择 DTmobile 标签下的 ARMGATEWAY 应用程序，点击“Camera”按钮，点击“open”按钮，得到视频画面。

六、 拓展思考

通过 socket 编程，将摄像头采集的数据进行网络传输。

七、 参考阅读

5.5 3G 网络实验

5.5.1 实验四十六 常用 AT 指令实验

一、 实验目的

1.了解常用的AT指令。

2.掌握常用AT指令的使用方法。

二、 实验设备

- ARM网关（EBA）
- 3G模块
- SIM卡
- 天线

三、 实验内容

本实验需要向linux内核中添加GSM驱动，并对串口信息进行读写，要求了解常用的AT指令，并用AT指令进行简单操作。

四、 实验原理

1. M2M概述

20世纪90年代中后期，随着各种信息通信手段（如Internet、遥感勘测、远程信息处理、远程控制等）的发展，加之世界上各类设备的不断增加，人们开始越来越多的关注于如何对设备和资产进行有效监视和控制，甚至如何用设备控制设备，这也就是“M2M”理念由此起源。

M2M（Machine/Man-to-Machine/Man）是一种以机器智能交互为核心的、网络化的应用与服务。简单的说，M2M是指机器之间的互联互通。广义上来说，M2M可代表机器对机器、

人对机器、机器对人、移动网络对机器的连接与通信，它涵盖了所有实现在人、机器、系统之间建立通信连接的技术和手段。M2M技术综合了数据采集、GPS，远程监控、电信、信息技术，是计算机、网络、设备、传感器、人类等的生态系统，能够使业务流程自动化，集成公司资讯科技 (IT)系统和非IT设备的实时状态，并创造增值服务。

M2M 是一种理念，也是所有增强机器设备通信和网络能力的技术的总称。人与人之间的沟通很多也是通过机器实现的，例如通过手机、电话、电脑、传真机等机器设备之间的通信来实现人与人之间的沟通。另外一类技术是专为机器和机器建立通信而设计的，如许多智能化仪器仪表都带有RS-232接口和GPIB通信接口，增强了仪器与仪器之间，仪器与电脑之间的通信能力。随着科学技术的发展，越来越多的设备具有了通信和连网能力，网络一切（Network Everything）逐步变为现实。人与人之间的通信需要更加直观、精美的界面和更丰富的多媒体内容，而M2M的通信更须要建立一个统一规范的通信接口和标准化的传输内容。

现有的M2M标准，都涉及到5个重要的技术部分：机器、M2M终端、通信网络、中间件、应用。

1.1 智能化机器

实现M2M的第一步就是从机器/设备中获得数据，然后把它们通过网络发送出去。使机器“开口说话”（talk），让机器具备信息感知、信息加工（计算能力）、无线通信能力。使机器具备“说话”能力的基本方法有两种：生产设备的时候嵌入M2M 硬件；对已有机器进行改装，使其具备通信/联网能力。

1.2 M2M终端

M2M终端是使机器获得远程通信和联网能力的部件。主要进行信息的提取，从各种机器/设备那里获取数据，并传送到通信网络。有一部分硬件已经封装了M2M协议 。

1.3 通信网络

将信息传送到目的地。通信网络在整个M2M技术框架中处于核心地位，包括：广域网（3G网络、GPRS、卫星通信网络、Internet、公众电话网）、局域网（以太网、无线局域网WLAN、Bluetooth）、个域网（ZigBee、传感器网络）。

1.4 中间件

中间件包括两部分：M2M 网关、数据收集/集成部件。网关是M2M系统中的“翻译员”，它获取来自通信网络的数据，将数据传送给信息处理系统。主要的功能是完成不同通信协议之间的转换。

1.5 应用

数据收集/集成部件是为了将数据变成有价值的信息。对原始数据进行不同加工和处理，并将结果呈现给需要这些信息的观察者和决策者。这些中间件包括：数据分析和商业智能部

件,异常情况报告和工作流程部件,数据仓库和存储部件等。

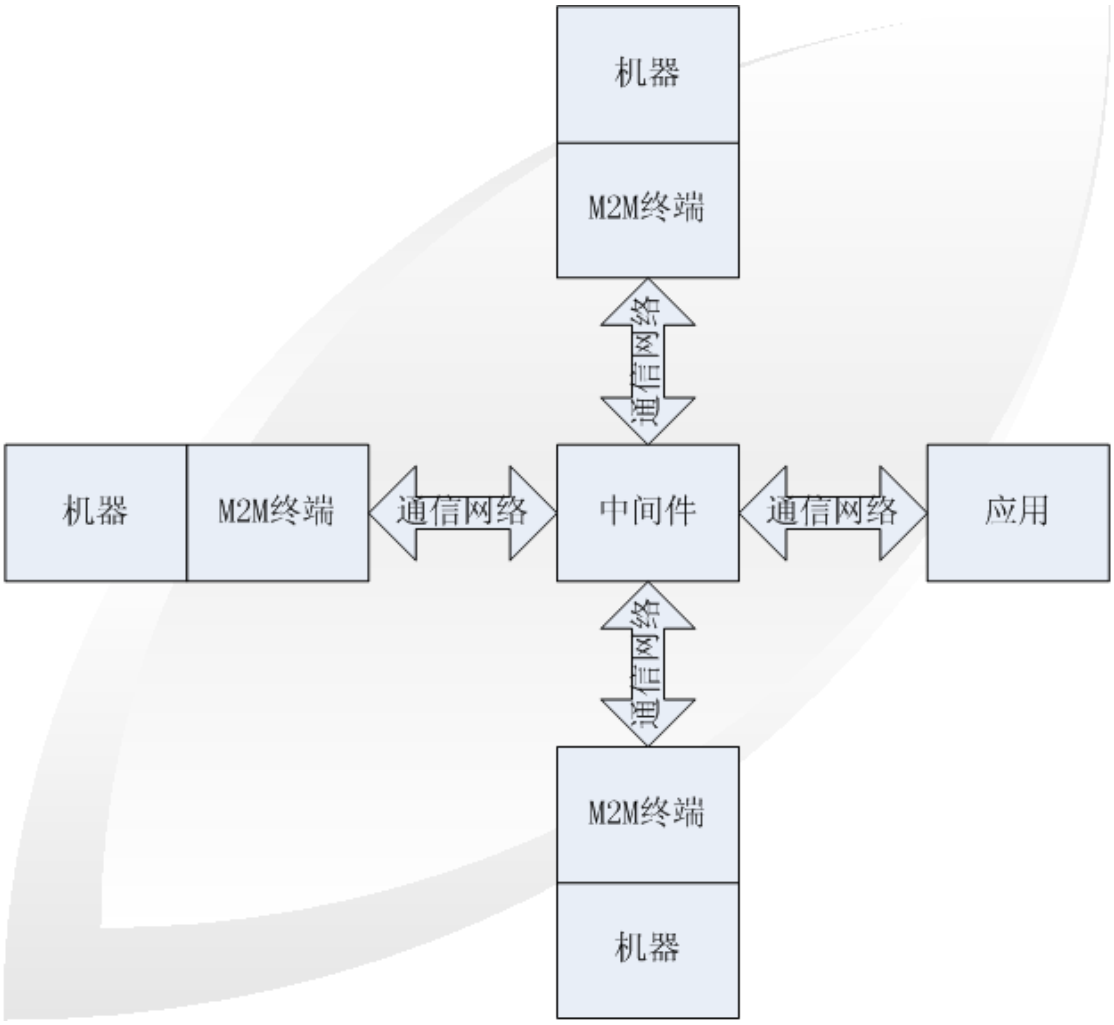


图5-40　M2M技术组成部分

通信网络技术的出现和发展，给社会生活面貌带来了极大的变化。人与人之间可以更加快捷地沟通， 信息的交流更顺畅。但是目前仅仅是计算机和其他一些IT类设备具备这种通信和网络能力。众多的普通机器设备几乎不具备联网和通信能力，例如家电、车辆、自动售货机、工厂设备等。M2M技术的目标就是使所有机器设备都具备连网和通信能力，其核心理念就是网络一切（Network Everything）。M2M技术具有非常重要的意义，有着广阔的市场和应用，推动着社会生产和生活方式新一轮的变革。

2.　3G模块系统框图（如图5-41）：

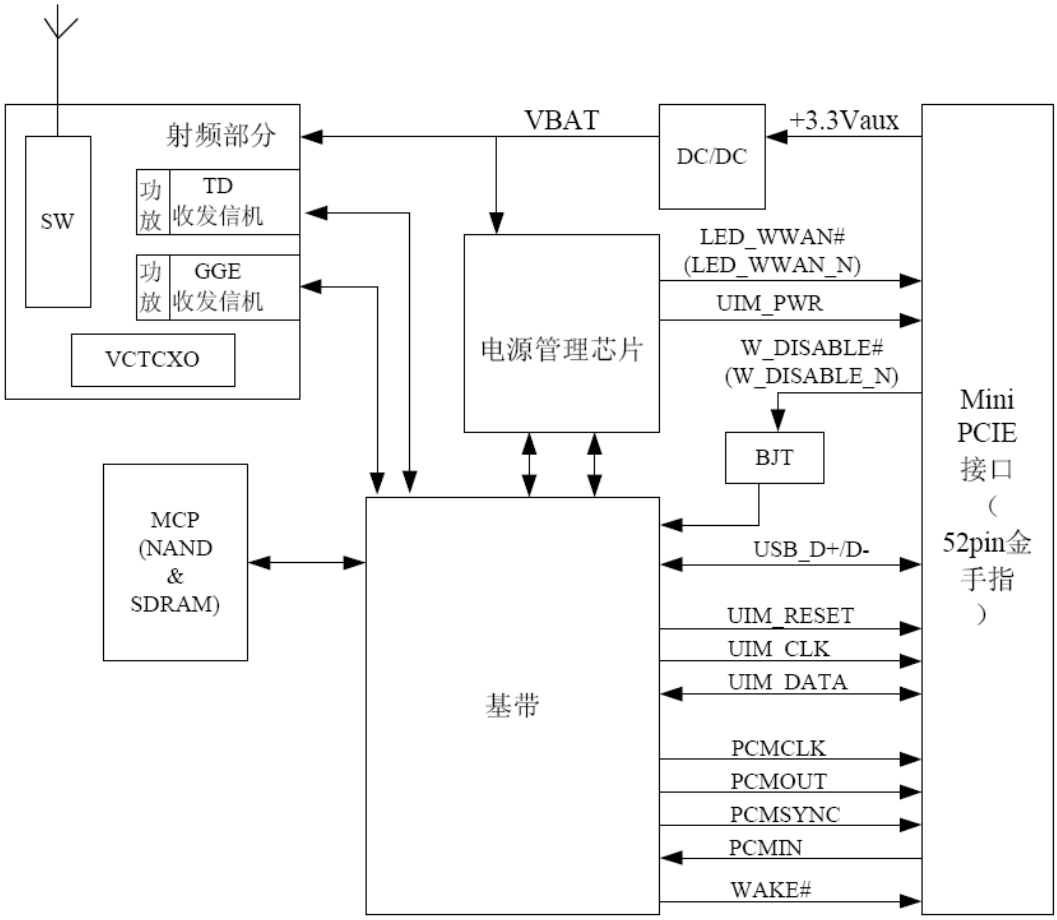


图5-41　3G模块系统框图

3.AT指令

AT 即Attention，AT指令集是从终端设备(Terminal Equipment，TE)或数据终端设备(Data Terminal Equipment，DTE)向终端适配器(Terminal Adapter， TA)或数据电路终端设备(Data Circuit Terminal Equipment，DCE)发送的。通过TA，TE发送AT指令来控制移动台(Mobile Station，MS)的功能，与GSM网络业务进行交互。用户可以通过AT指令进行呼叫、短信、电话本、数据业务、传真等方面的控制。90年代初，AT指令仅被用于Modem操作。没有控制移动电话文本消息的先例，只开发了一种叫SMS BlockMode的协议，通过终端设备(TE)或电脑来完全控制SMS。几年后，主要的移动电话生产厂商诺基亚、爱立信、摩托罗拉和HP共同为GSM研制了一整套AT指令，其中就包括对SMS的控制。AT指令在此基础上演化并被加入GSM07．05标准以及现在的GSM07．07标准，完全标准化和比较健全的标准。如：对SMS的控制共有3种实现途径：最初的BlockMode；基于AT指令的TextMode；基于AT指令的PDUMode。到现在PDUMode已经取代BlockMode，后者逐渐淡出。GSM模块与计算机之间的通信协议是一些AT指令集，AT指令是以AT作首，字符结束的字符串，AT指令的响应数据包在中。每个指令执行成功与否都有相应的返回。其他的一些非预期的信息(如有人拨号进来、线路无信号等)，模块将有对应的一些信息提示，接收端可做相应的处理。

示例：CDMA modem DTE

AT< CR>

< LF> OK < LF>

ATTEST< CR>

< CR> ERROR < LF>

如果AT指令执行成功，“OK”字符串返回；

如果AT 指令语法错误或AT 指令执行失败，

“ERROR”字符串返回。

五、 实验步骤

1.3G 模块的安装及天线的安装，如图 5-42 所示，图中 1 为天线的接插口；2 为 3G 模块与 ARM 网关 EBA 的接插口。

图5-42 3G模块

图 5-43 中，1 为 ARM 网关（EBA）与 3G 模块的接插口；2 为耳机接插口，在语音通话时候可使用；3 为天线的接口 SIM_3G。

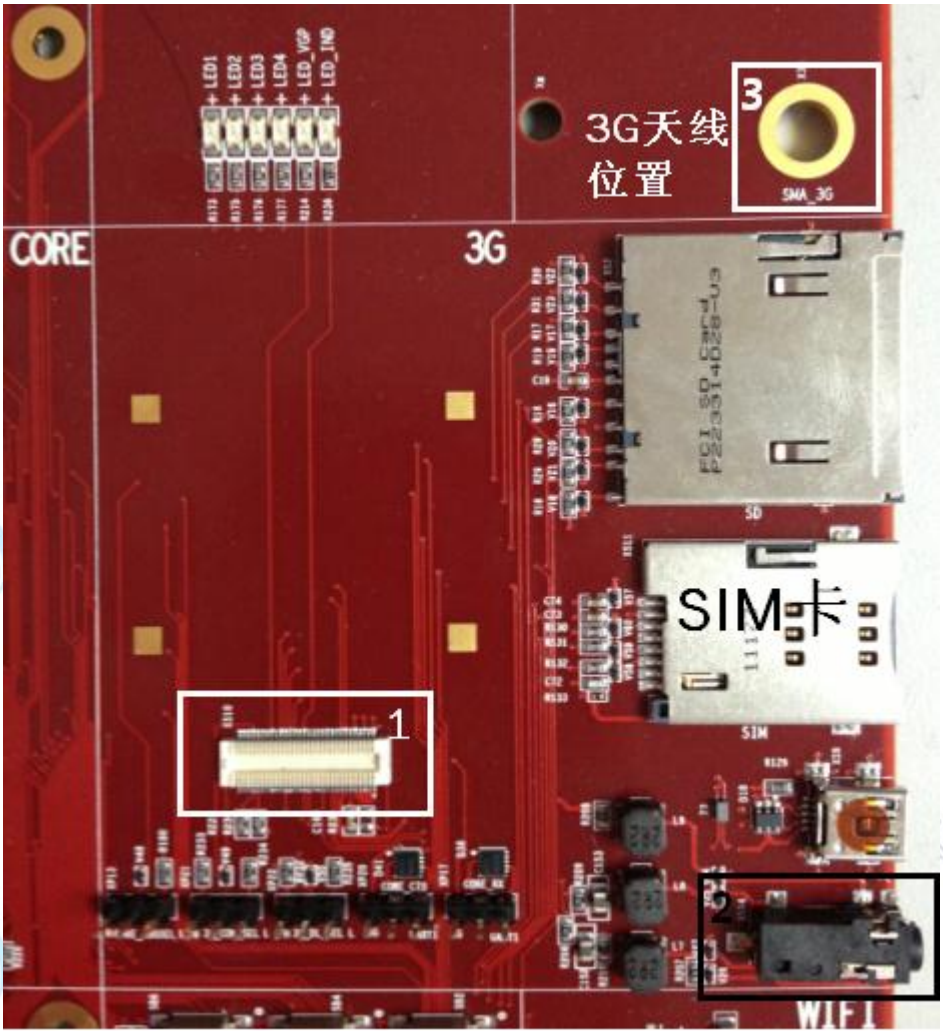


图5-43　ARM网关3G模块

图 5-25 中，天线的位置 1 与 3G 模块相接，天线的位置 2 穿过 ARM 网关的位置 3 与天线接口 SIM_3G 相连接。安装时，将 3G 模块与 ARM 网关的接插口连接上，将天线安装好即可，3G 模块按照好的效果图如图 5-44 右侧所示：

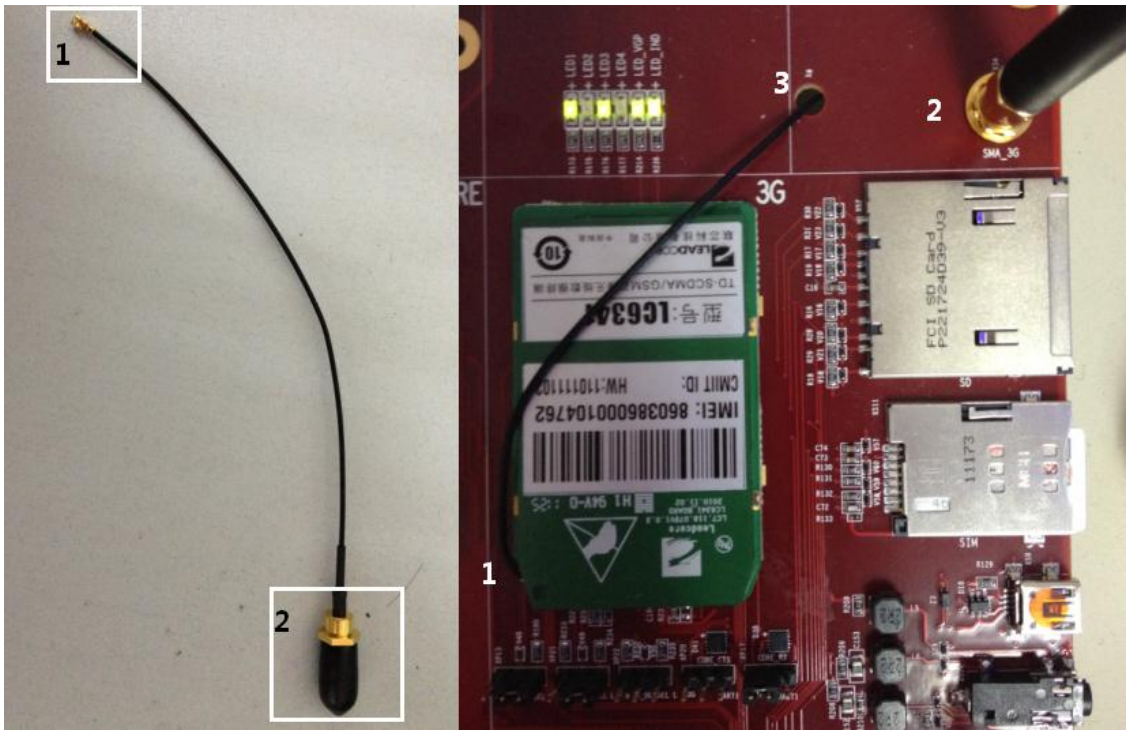


图5-44　3G天线与安装完成后效果图

2.跳线的链接

EBA 跳线对 3G 模块的跳线规则如下：

2.1 对外 UART1 接插件和 3G 模块的 UART 复用同一个 UART 与 S3C6410 通信（S3C6410 的 UART1 口）。此时工作模式选择通过跳线 XP17\XP18\XP19\XP20 实现。具体描述如下：（板卡正放时，从左到右的管脚分别为 1、2、3）

	对外 UART1 接口	3G 模块 UART 接口
XP17(UART_RX)	2-3 管脚连接	1-2 管脚连接
XP18(UART_TX)	2-3 管脚连接	1-2 管脚连接
XP19(UART_RTS)	2-3 管脚连接	1-2 管脚连接
XP20(UART_RTS)	2-3 管脚连接	1-2 管脚连接

2.2 3G 模块下载模式及通讯口选择，通过 XP21\XP22 实现。具体描述如下：

	1-2 管脚连接	2-3 管脚连接
XP21 （通讯口选择信号）	以 USB 为通讯口，工作在正常模式下	以 UART1 为通讯口，工作在正常模式下
XP22 (下载模式选择信号)	通过 UART 或 USB 口下载	正常启动

选择以 UART1 为通讯口，工作在正常模式下正常启动，3G 模块 UART 接口，跳线连接如图 5-45 所示：

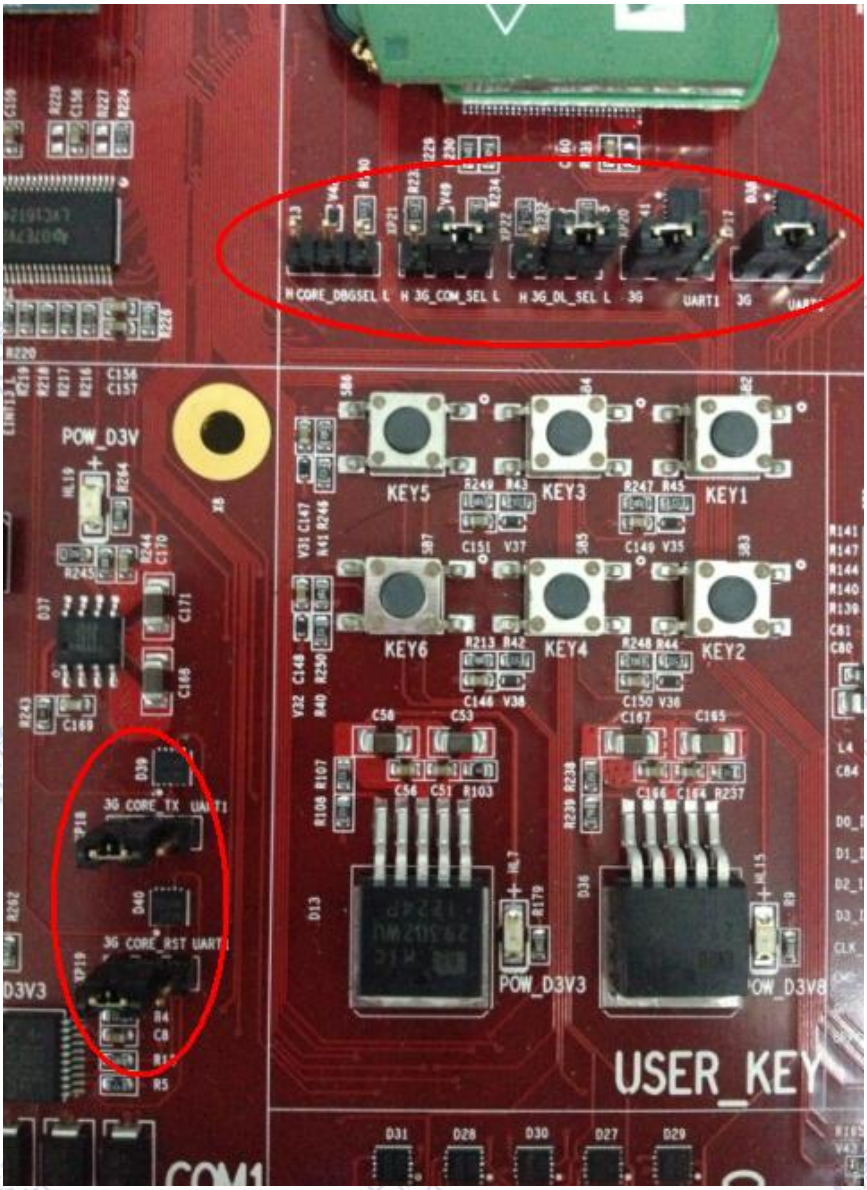


图5-45　3G实验跳线帽接法图示

2.将SIM卡插入ARM网关SIM卡槽中，注意SIM卡的插入方向，如图5-46所示：

图5-46　插入SIM卡

3.运行ARM网关的3G程序。启动ARM网关，选择DTmobile标签下的3G-test，进入3G界面如图5-47所示（启动3G程序需要等待月15秒钟），在发送AT指令后面的窗体中输入AT指令点击发送，输出结果在右边的窗体中显示。熟悉AT指令。

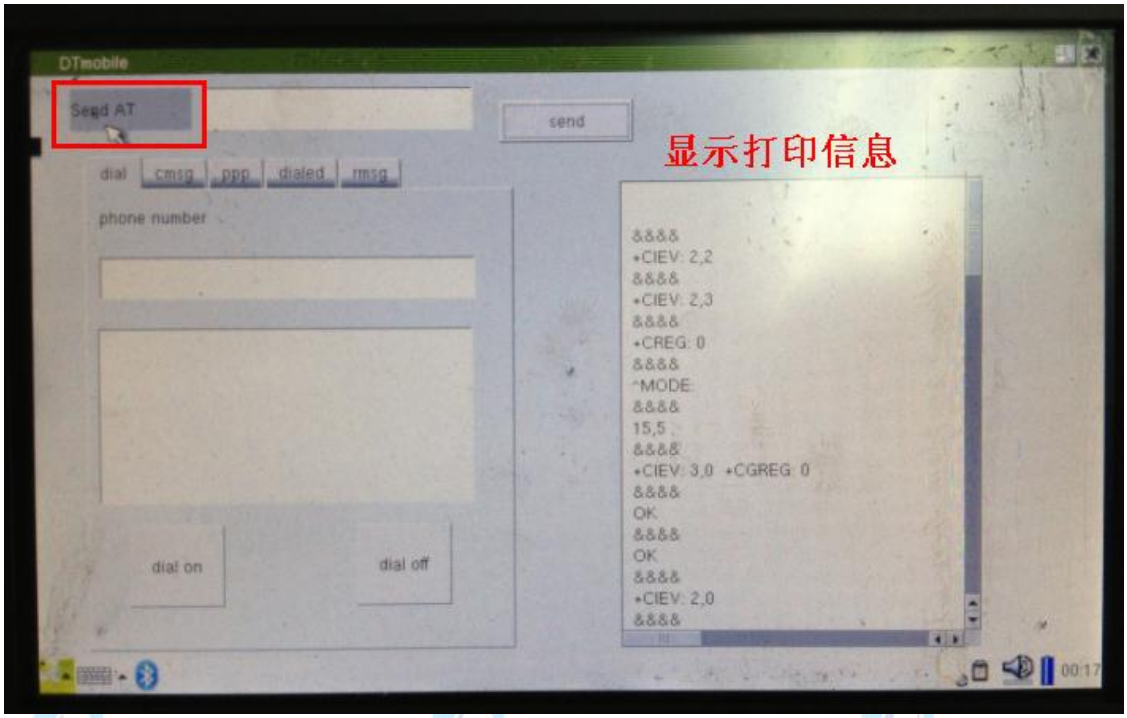


图5-47　3G测试程序界面

备注：常见的 AT 如下表：

功能	AT指令	详细
验证串口	AT	返回OK表示连接串口成功
拨号	ATD	后面紧跟拨打电话号码
挂机	ATH	挂机
短信息格式	AT+GMGF	选择短消息模式（TEXT或PDU）
读取短消息	AT+CMGR	读取短消息
新消息提示	AT+CNTI	选择新消息到来时的提示方式
发送短消息	AT+CMGS	发送短消息
TCP/UDP连接初始化	AT+CGDCONT	初始化TCP/UDP连接
建立TCP/UDP连接	AT+IPSTART	与设定的IP建立TCP/UDP连接

服务器侦听命令	AT+IPLISTEN	打开服务器侦听功能
TCP/UDP数据发送	AT+IPSEND	选定链路发送数据
TCP/UDP接收数据缓存查询	AT+IPGETDATA	查询某链路是否有数据到达
TCP/UDP数据到达指示	AT+CIPDATA	当有数据到达时可主动上报
关闭TCP/UDP连接	AT+CIPCLOSE	关闭指定的连接

六、参考阅读

请参考阅读 “E-Box300\05-芯片数据手册\3G”目录下的数据芯片手册

5.5.2 实验四十七 语音通话实验

一、 实验目的

1.了解 3G 模块开关机流程，语音通话流程。

二、 实验设备

- ARM 网关（EBA）
- 3G 模块
- SIM 卡
- 天线

三、 实验内容

利用 3G 模块实现语音拨打电话及接受来电电话。

四、 实验原理

1. 正常开机流程（图 5-48）：

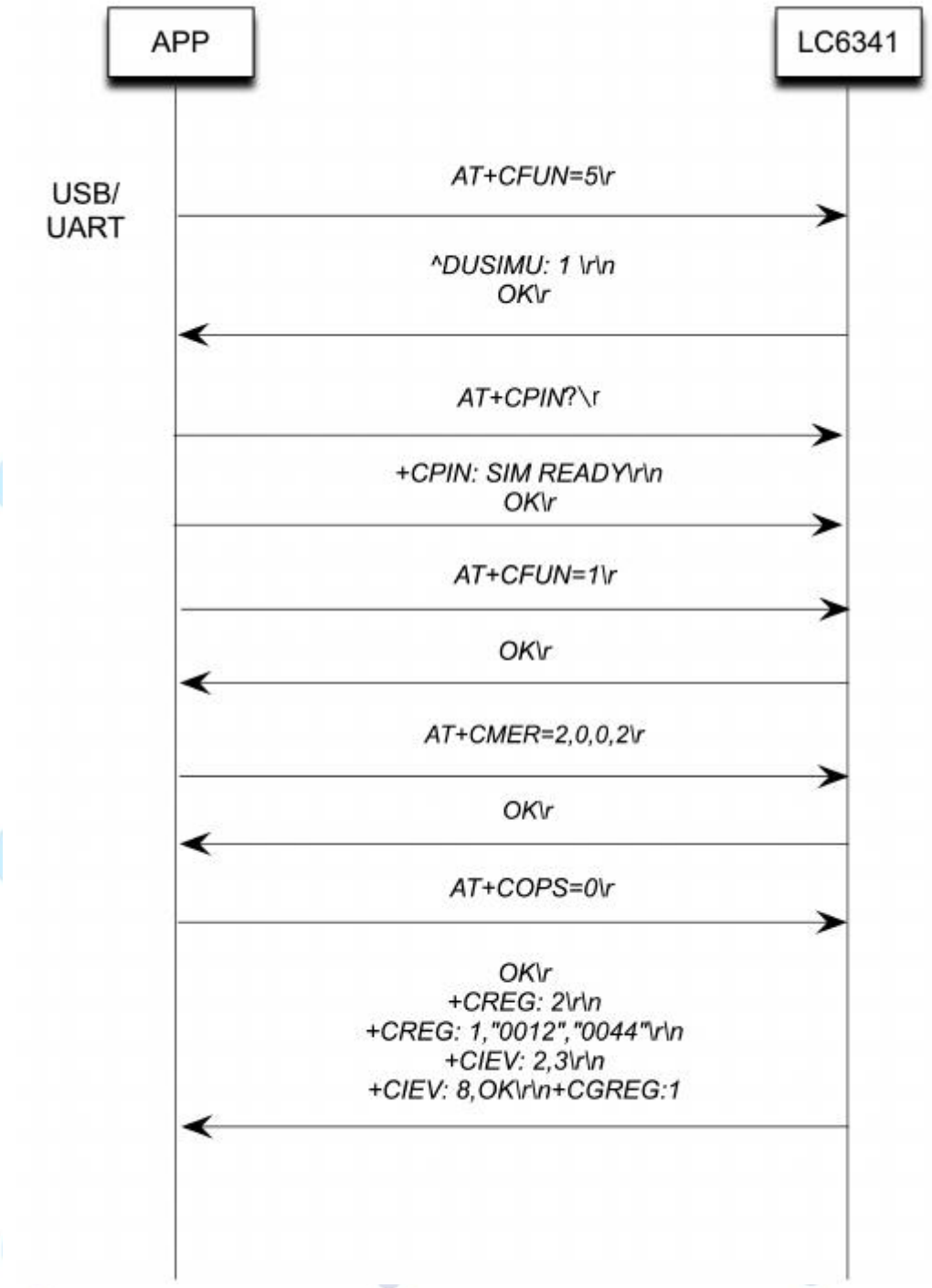


图5-48　开机流程

1）APP端输入“AT＋CFUN=5\r”，返回“OK\r”，（此指令为激活0USIM）；

2）APP端输入”AT+CPIN?”查询SIM状态，返回”+CPIN：SIM READY\r\n OK\r”。

3）APP端输入“AT＋CFUN=1\r”，返回“OK\r”，（此指令为激活协议栈）；

4）APP端输入“AT+CMER=2,0,0,2\r”，返回“OK\r”（此指令为开启协议栈事件CIEV：<ind>,<value>上报）；

5）APP端输入“AT+COPS=0\r” ，返回“0\r”，(此指令为搜网指令，搜网驻留结束后返回ok),此后LC6341会返回很多数据，直到返回“+CREG：1”表示开机成功（“^DACTI:2”表示当前接入技术是TD; “+CIEV：2,2”表示信号等级为2级； “+CIEV ： 8,0” 表示短消息存储未满；“+CREG ： 2” 表示搜网中；“+CREG：1”表示搜网成功并注册成功。【说明：“+CREG:n”其中“n”的值可为0[未注册]，1[注册成功]，2[搜网]，3[网络拒绝注册]，4[网络注册状态未知]，5[漫游],开机后只有当CREG返回的是1或5时,才能做LC6341所支持的业务】;鉴于对CREG：4的协议理解，建议当+CREG：4时，APP处理器在UI显示为可提供服务状态，即等同于+CREG：1或+CREG:5】,同样CGREG有类似上报,上报CGREG:1或CGREG:5表示ATTACH成功）。

2. 正常关机流程（图 5-49）：

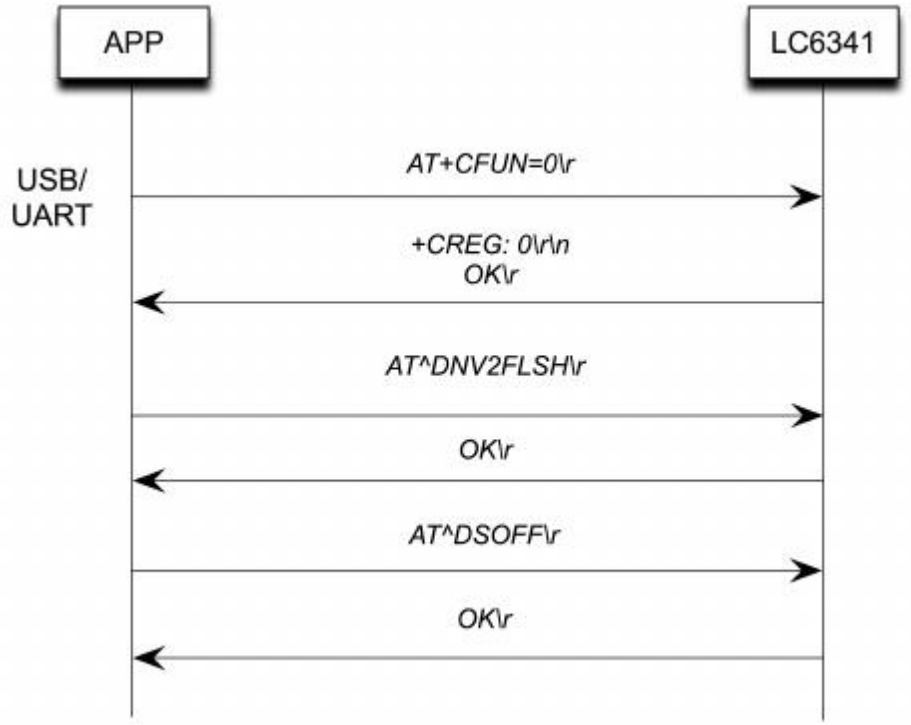


图5-49 关机流程

1）APP端输入“AT+CFUN=0\r”，返回“OK\r”（此指令为去注册网络、去激活协议栈、SIM/USIM）；

2）APP端输入“AT^DNV2FLSH\r”，返回“OK\r”(此指令为将内存缓冲的数据刷新到Flash中保存，防止Flash被异常破坏)；

3）APP端输入“AT^DSOFF\r”，返回“OK\r”（此指令为模块软关机）；

3.主叫(MOC)（图 5-50）

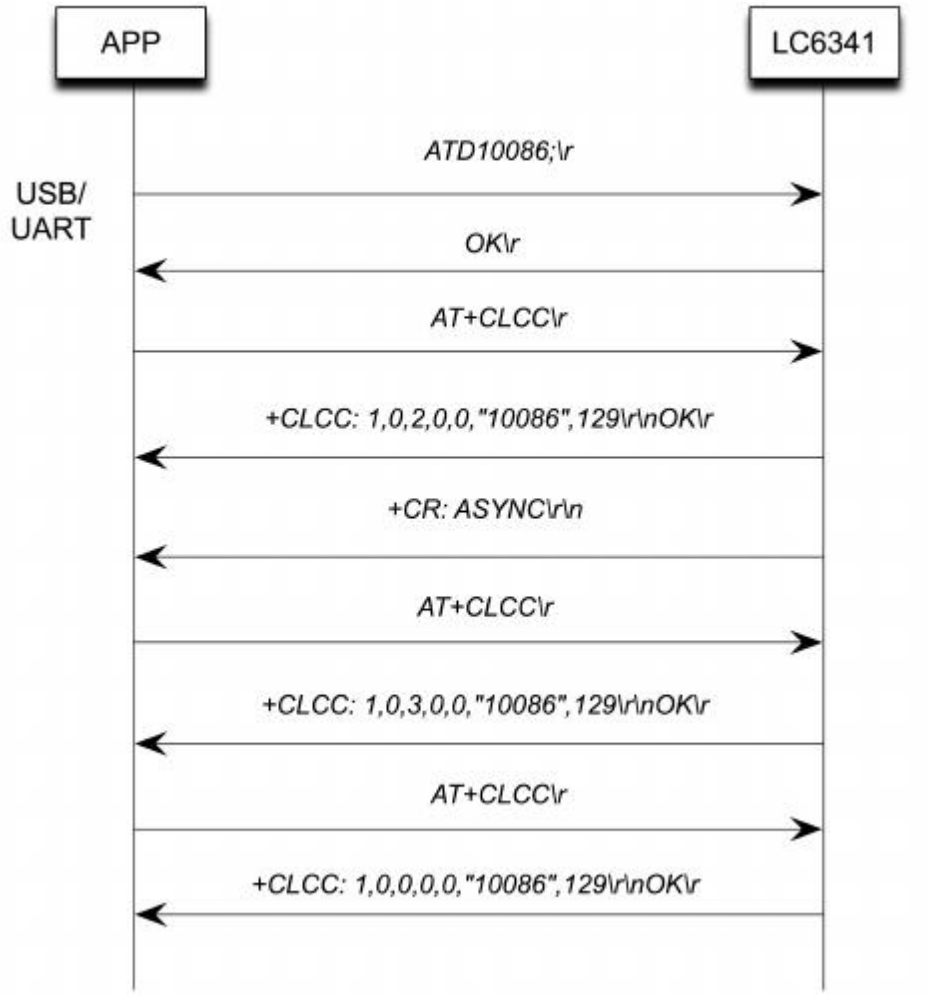


图5-50　主叫MOC

1）APP端输入“ATD10086;\r”,返回“OK\r”，（此指令为拨号语音电话10086）；

2）APP端输入“AT+CLCC\r”，此为查询呼叫状态的AT指令，（如果返回“1,0,2,0,0,"10086",129\r\n”，表示当前状态为正在呼叫状态；如果返回“1,0,3,0,0,"10086",129\r\n”，表示当前状态为被叫振铃状态；如果返回“1,0,0,0,0,"10086",129\r\n”，表示当前状态为建立连接状态；而在其他情况下，如果此指令无问题，则会返回“OK\r”，在被叫振铃状态上报之前，会有“+CR: ASYNC\r\n”主动上报通知APP端）；

以上流程为模拟口输出，如果需要通过PCM管脚IO，可在拨打电话前通过AT^DUSC2PCM设置

4.被叫(MTC)（图 5-51）

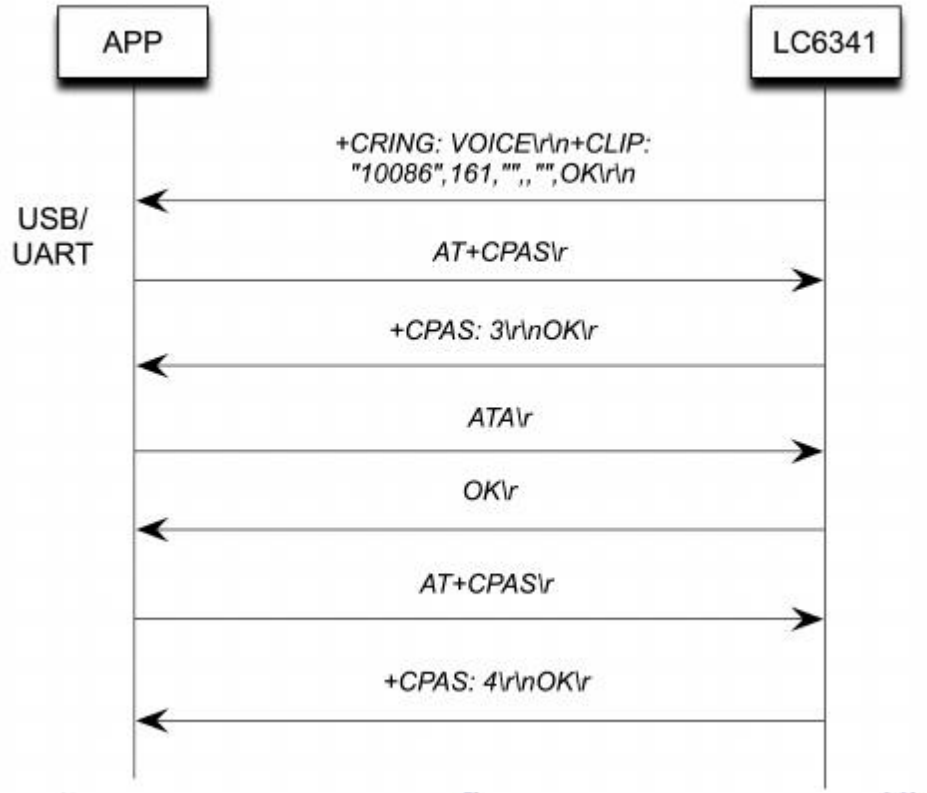


图5-51　被叫（MOC）

1）LC6341 收到 10086 被叫寻呼后，给 APP 端上报“ +CRING: VOICE\r\n+CLIP:"10086",161,"",,"",OK\r\n”,APP依据此指令进行相关操作。

2）APP端输入“AT+CPAS\r”，此为查询 LC6341的呼叫状态的 AT指令，（其中上报“+CPAS: 3\r\nOK\r”表示当前处于振铃状态，上报“ +CPAS:4\r\nOK\r”表示当前处于电话连接状态，可选输入指令）；

5.主动挂断（图 5-52）

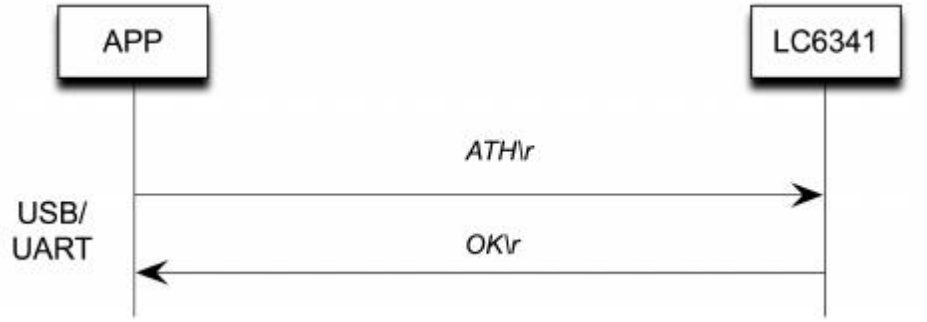


图5-52　主动挂断

1）APP 端输入“ATH\r”，返回“OK\r”；

2）对于挂断 AT 指令，AT＋CHUP（挂断当前激活 CS 链路）和 AT＋CHLD 也可作为可选 AT 指令。

6.对端挂断（图 5-53）

图5-53　对端挂断

1）LC6341 会给 APP 上报“NO CARRIER”,表示对端主动挂断，APP 依此进行相关操作。

五、　实验步骤

1．按照实验四十六搭建起 3G 环境。运行 ARM 网关的 3G-test 程序。选择“dial”标签，在呼叫号码下的编辑框中输入想要拨打的电话号，点击左下的“dial on”按钮（拨打电话）。右侧显示信息界面会打印对应命令和码流信息，如图 5-54 所示。

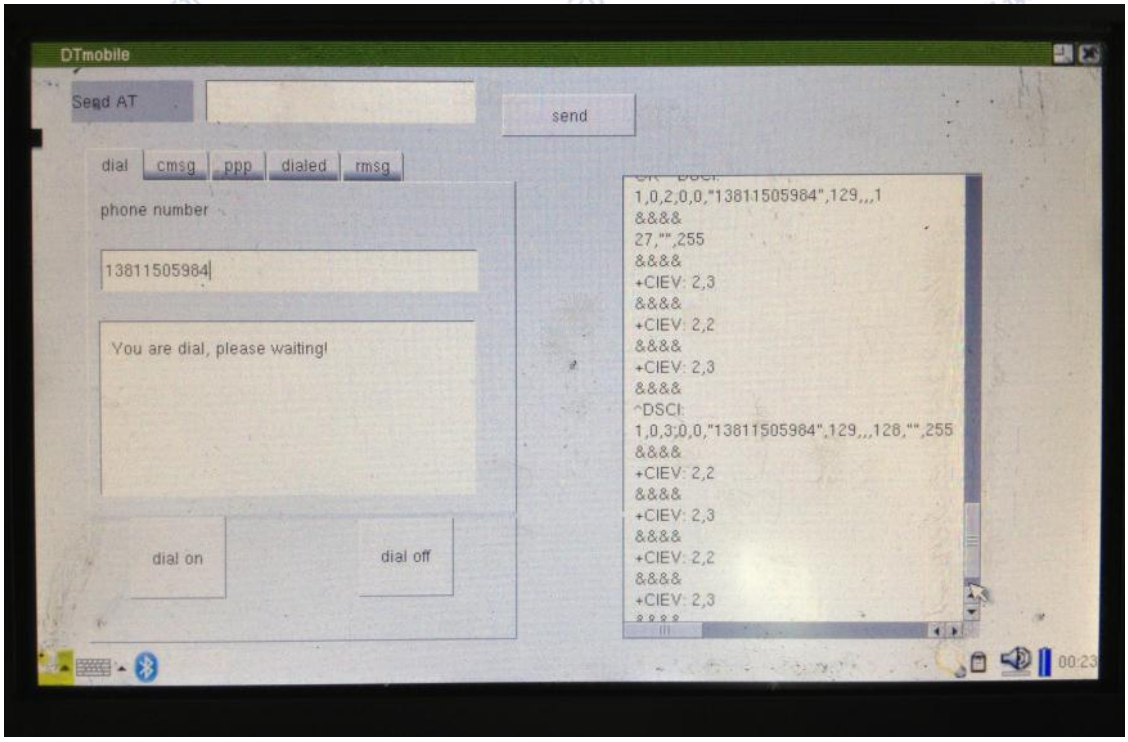


图5-54　拨打电话

使用软键盘的方法：点击界面左下角小箭头图标，如图 5-55 所示，选择“Keyboard”按钮，在界面最下方显示软键盘，可以输入数字和英文等。再点击键盘图标，软键盘界面则隐藏。

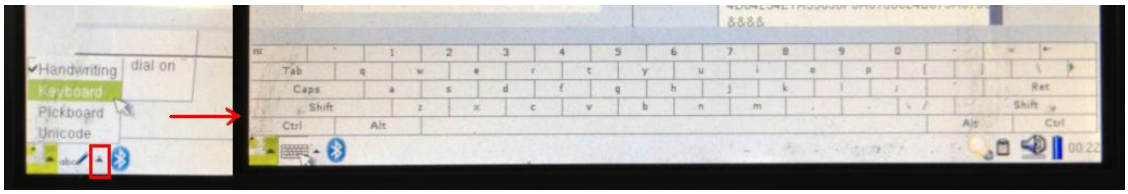


图5-55　　软键盘

2．观察被叫号码是否接通，进行通话或者挂断操作（“dial off”按钮为挂断按钮）也可将耳机插入 ARM 网关 3G 模块的 XS14 接插件（即耳机插件）进行语音通话功能。

3．利用其他 3G 终端或手机拨打该 SIM 卡卡号，接通后，3G 界面会自动弹到“dialed”标签，并显示来电号码。如图 5-56 所示：

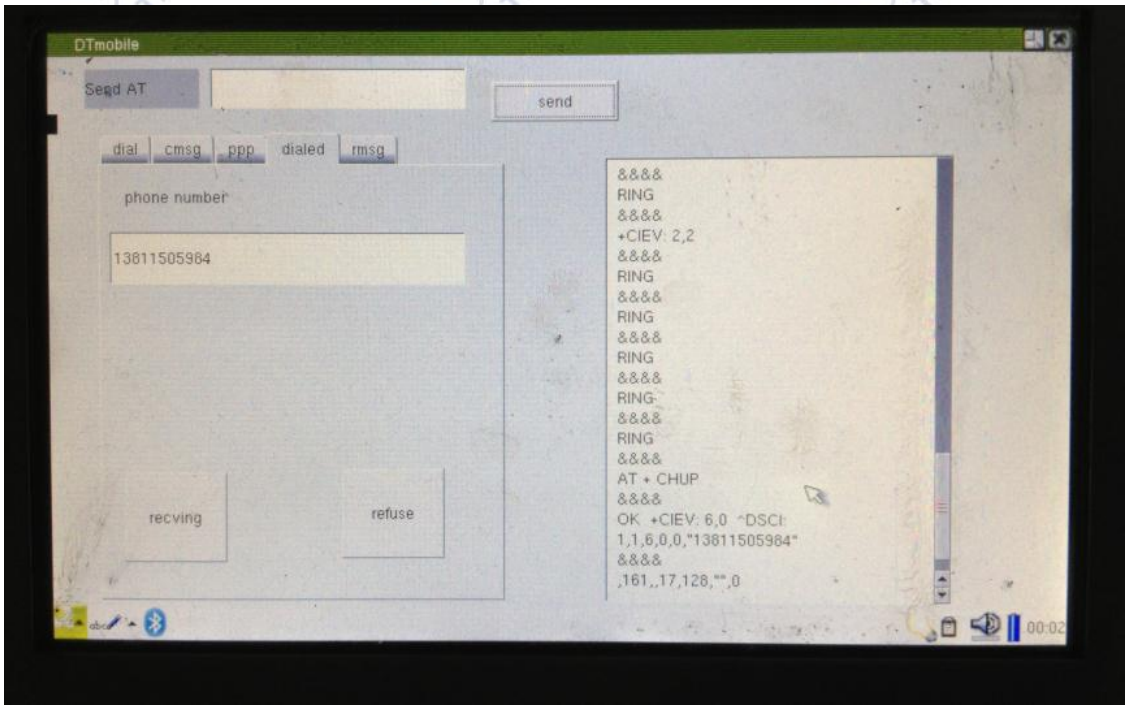


图5-56　　“dialed”标签

4．点击接电话“recving”或者挂断电话“refuse”图标，查看码流信息。

六、　拓展思考

思考手机中的呼叫转移功能是如何实现的，尝试实现此功能。

七、　参考阅读

请参考阅读 E-Box300\05-芯片数据手册\3G 内的数据芯片手册

5.5.3 实验四十八 数据业务实验

一、　实验目的

1.了解 3G 模块上网方法。

2.了解 PPP 拨号。

二、 实验设备

- ARM 网关（EBA）
- 3G 模块
- SIM 卡
- 天线

三、 实验内容

利用 3G 模块实现上网功能。

四、 实验原理

1.首先分析3G模块的上网方式，我们使用的是pppd方式进行上网，也就是通过ppp选项和ppp脚本的设置进行上网，其中，/etc/ppp/peers中含有PPP选项，/etc/ppp/chat中含有ppp脚本，运行的时候命令行为pppd call 后面加脚本文件名。

下面是ppp选项，位于/etc/ppp/peers内，我们采用cmnetN和cmwapN（N代表第几个终端）的方法区别不同的选项文件，基本配置如下，后期对于业务的不同将有相应调整，其中第一行指向所需上网终端的端口号。最后一行指向拨号脚本：

/dev/ttySAC1
115200
nocrtscts
nocdtrcts
local
nobsdcomp
nodeflate
novj
usepeerdns
defaultroute
lock
connect "/usr/sbin/chat -v -t 50 -f /etc/ppp/peers/cmtc-isp-cmnet"
ppp拨号脚本

----编辑/etc/ppp/chat /cmtc-isp-net配置文件，输入如下内容后，保存退出。其中AT相关部分为向串口的指令，对于不同的网络要求可以进行初始化的更改。

ABORT 'BUSY'
ABORT 'NO CARRIER'

"" AT

""ATZ

""AT+CGDCONT=1,"IP","cmnet"

""AT+CGEQREQ=1,2,128,2048,0,0,0,0, "0E0","0E0",,0,0

""ATS0=0

""ATDT*98*1#

此处为实现组合业务，""AT+CGDCONT=1,"IP","cmnet"为设定上网方式，这里通过变换cmnet和cmwap变换上网方式。

ppp 拨号：这里我们通过调整不同文件，即 pppd call 不同文件，而不同文件指向不同脚本来在程序中控制不同方式，不同终端的上网效果。

五、 实验步骤

1．按照实验四十六搭建起 3G 环境。运行 ARM 网关的 3G-test 程序。选择“ppp”标签。在“Ping”按钮后，输入需要测试的网址，然后点击“pppd on”按钮，“internet link”下方显示区域会显示连接信息，右侧显示信息界面会打印对应命令和码流信息。如图 5-57 所示：

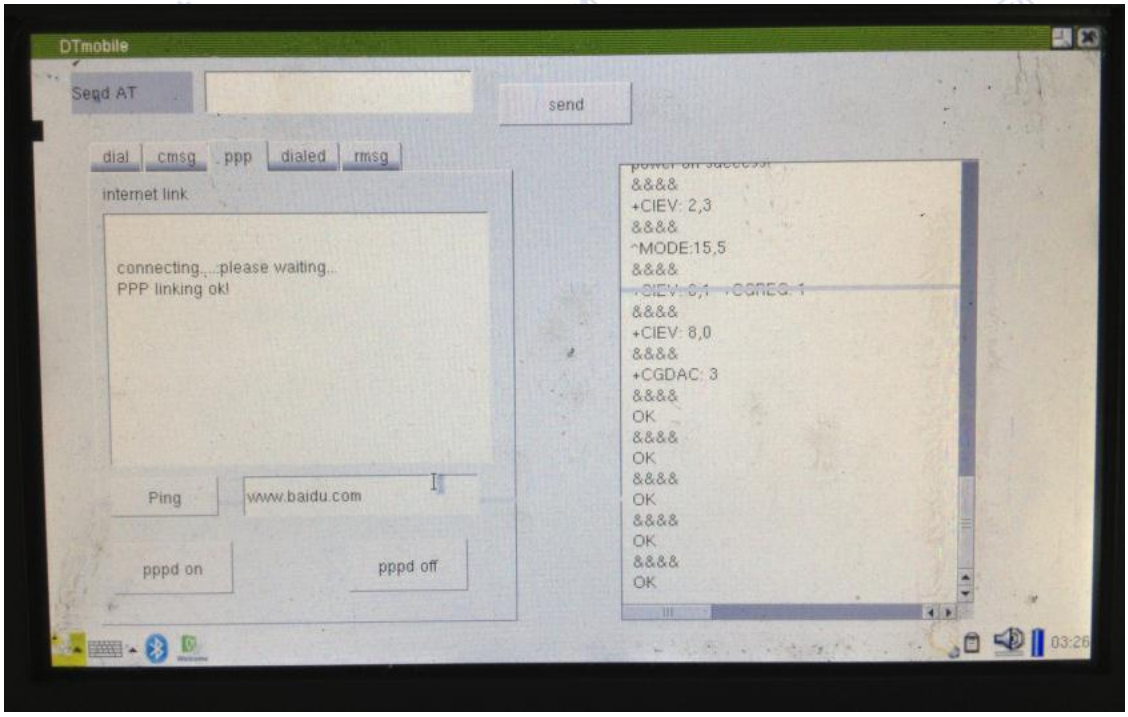


图5-57　ppp界面

2. 点击下方“pppd off”按钮，断开连接，查看提示信息以及打印的码流信息，如图 5-58 所示：

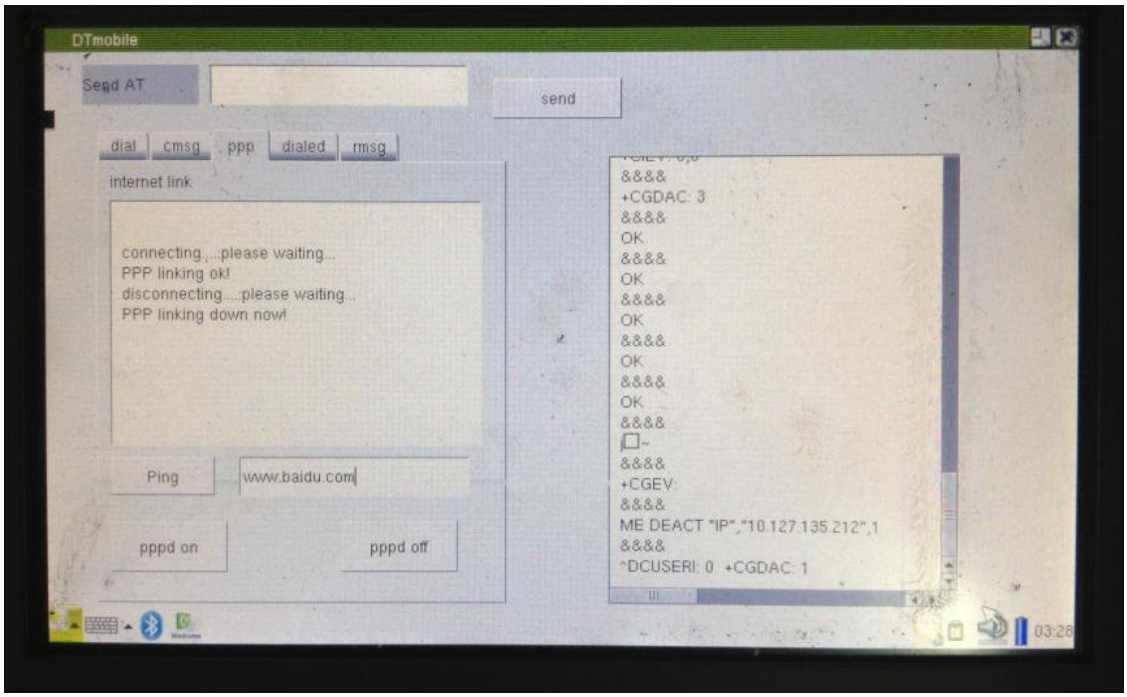


图5-58　断开PPP界面

六、　拓展思考

利用上网功能获取 IP 之后建立与上位机的 Socket 通信。

七、　参考阅读

请参考阅读 “E-Box300\05-芯片数据手册\3G”目录下的数据芯片手册

5.5.4 实验四十九 短信发送与解析实验

一、　实验目的

1.了解 3G 模块短信发送接收流程。

二、　实验设备

- ARM 网关（EBA）
- 3G 模块
- SIM 卡
- 天线

三、　实验内容

利用 3G 模块实现短信发送，短信接收，回复短信。收发短信仅限于英文。

四、 实验原理

1.短信业务(TEXT 模式)（图 5-59）

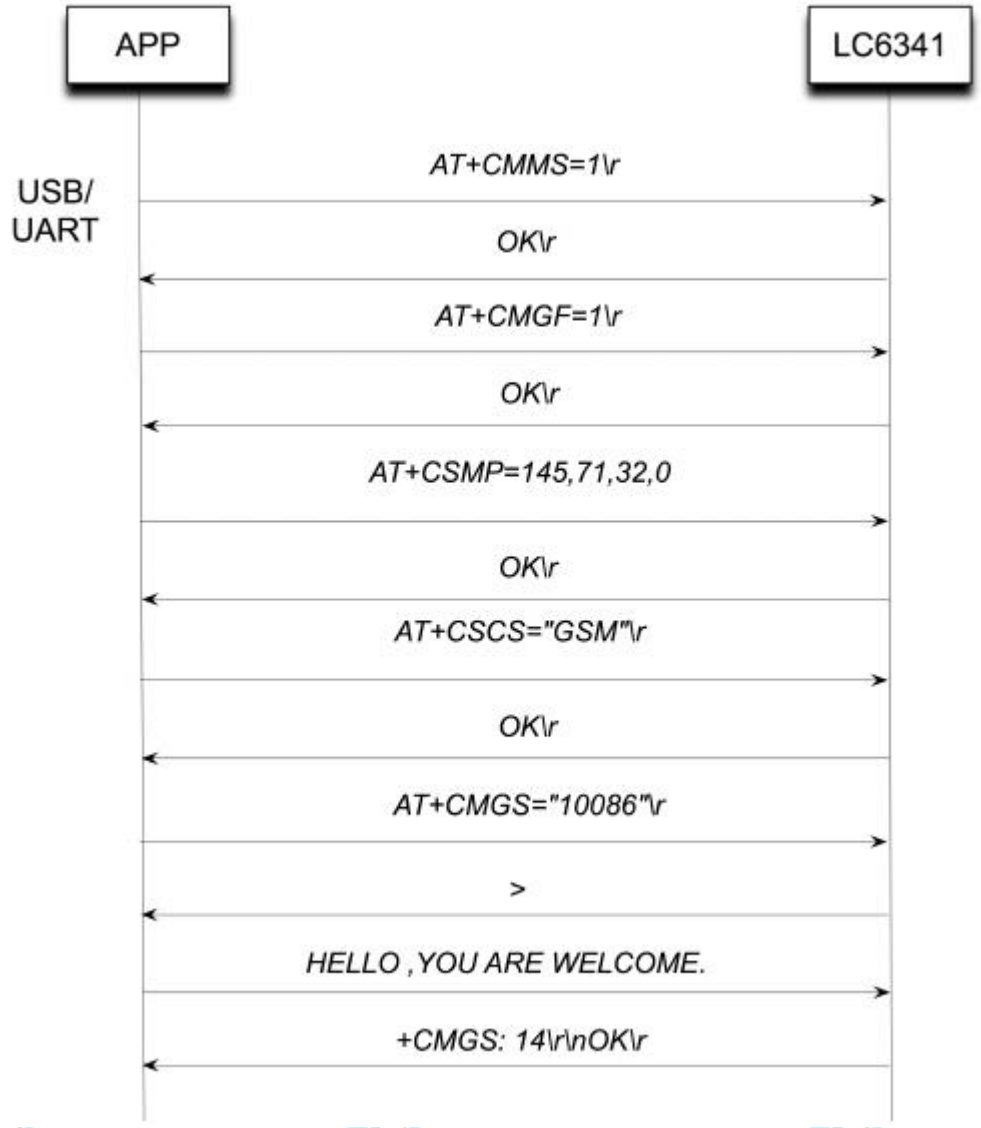


图5-59 短信业务流程

1）APP端输入“AT+CMMS=1\r” ，返回“OK\r”，（此指令为发送长短信的激活设置，需 APP端在15S内将下一部分 SMS发送给LC6341，可选输入指令）；

2）APP端输入“AT+CMGF=1\r” ，返回“OK\r”，（此指令为设置短消息发送格式为文本发送格式）；

3）APP端输入“AT+CSMP=145,71,32,0\r” ，返回“OK\r”，（此指令为设置发送模式参数，4）第四个参数为终端字符集格式参数（0代表GSM编码格式，4代表HEX编码格式，8代表UCS2

编码格式）；

5）APP端输入“AT+CSCS="GSM"\r”，返回“OK\r”，（此指令为设置终端字符集格式为GSM编码）；

6）APP端输入“AT+CMGS="10086"\r”，10086为对应的目标号码，返回“>”

然后APP端传输短信内容，以Ctrl+Z作为短消息内容的结束符结束；

7）如果短信息发送成功，则会有如上相似的上报信息。

2.AT常用短消息命令

1)AT+CSMS 选择消息服务。支持的服务有GSM-MO、SMS-MT、SMS-CB。

2)AT+CNMA 新信息确认应答。

3) AT+CPMS 优先信息存储。这个命令定义用来读写信息的存储区域。

4)AT+CMGF 优先信息格式。执行格式有TEXT方式和PDU方式。

5)AT+CSAS 保存设置。保存+CSAS和+CSMP的参数。

6) AT+CRES 恢复设置。

7) AT+CSDH 显示文本方式的参数。

8)AT+CNMI 新信息指示。这个命令选择如何从网络上接收短信息。

9) AT+CMGR 读短信。信息从+CPMS命令设定的存储器读取。

10)AT+CMGL 列出存储的信息。

11)AT+CMGS 发送信息。

12)AT+CMGW 写短信息并存储。

13)AT+CMSS 从存储器中发送信息。

14) AT+CSMP 设置文本模式的参数。

15) AT+CMGD 删除短信息。删除一个或多个短信息。

16)AT+CSCA 短信服务中心地址。

17)AT+CSCB 选择单元广播信息类型。

18)AT+WCBM 单元广播信息标识。

19)AT+WMSC 信息状态（是否读过、是否发送等等）修正。

20) AT+WMGO 信息覆盖写入。

21) AT+WUSS 不改变SMS状态。在执行+CMGR或+CMGL后仍保持UNREAD。

五、 实验步骤

1.按实验四十六搭建起 3G 环境。运行 ARM 网关的 3G-test 程序。.点击中间的短信标签（cmsg），在短信号码（cmsg number）下的编辑框中输入想要发送短信的电话号码，在短信内容下的编辑框中输入想要发送的短信内容，如图 5-60 所示：

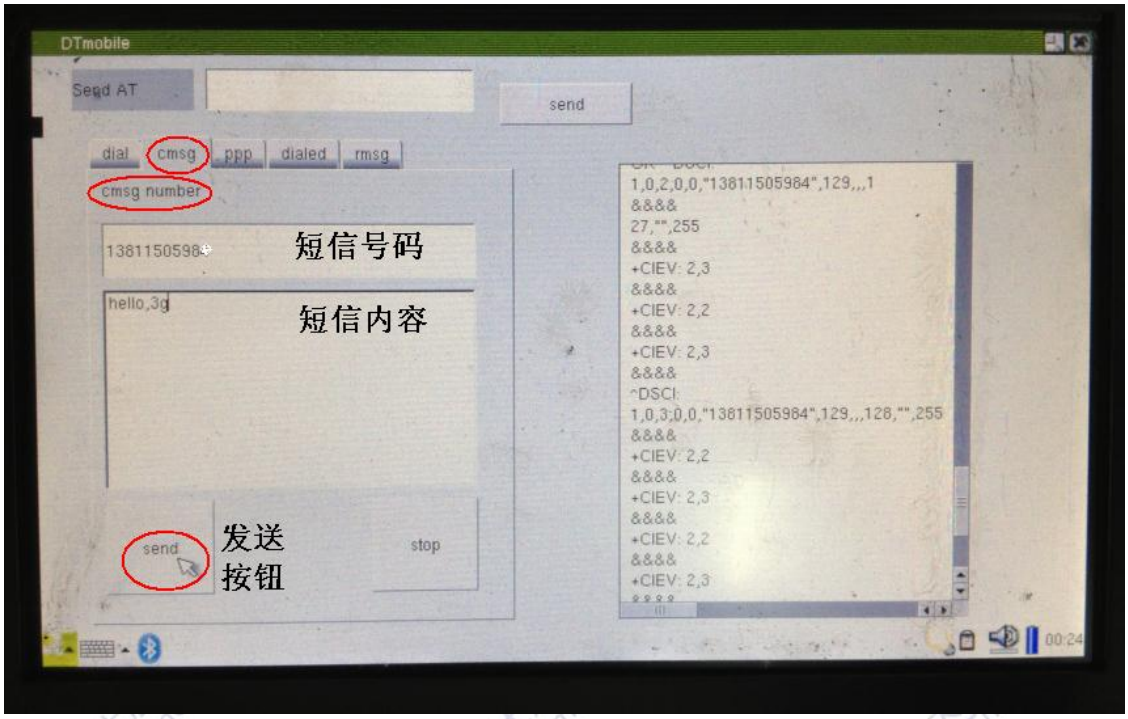


图5-60　　cmsg标签界面

3.点击发送短信按钮，观察此手机号终端是否接收到短信消息。

4.利用其他 3G 终端或手机给该 SIM 卡号发送短信，当此 SIM 卡接收到短消息后，会自动显示短信接收窗体。

六、　拓展思考

手机发送指定的短信内容，解析短信，自动回复指定内容。

七、　参考阅读

请参考阅读 “E-Box300\05-芯片数据手册\3G”目录下的数据芯片手册

第六章物联网基础应用实验

6.1 演示一 环境监测系统演示实验

一、 实验目的

1.掌握基于 ZigBee 的无线传感器网络对室内环境监测的原理。

2.实现室内环境监测实时数据采集和信息交互。

二、 实验设备

- ZigBee 无线传感网络
- 环境监测系统软件

三、 实验内容

室内环境监测系统以ZigBee星型网络为核心，充分利用ZigBee技术低功耗、自组网稳定、可靠性高的技术优势，为监测系统提供可靠、稳定的无线通信基础，实现了室内环境的自动化监测。在掌握整个系统的工作原理后，运行系统，得到监测数据。

四、 实验原理

1、系统结构

基于ZigBee无线通信技术的室内环境监测系统结构如图6-1所示。室内环境监测系统是一个由ZigBee通信模块组成的无线传感器网络，拓扑结构采用以监测主机为中心的星型网络，适用于室内环境监测传输距离较近、环境干扰大、实时性要求高、数据传输量大的特征。

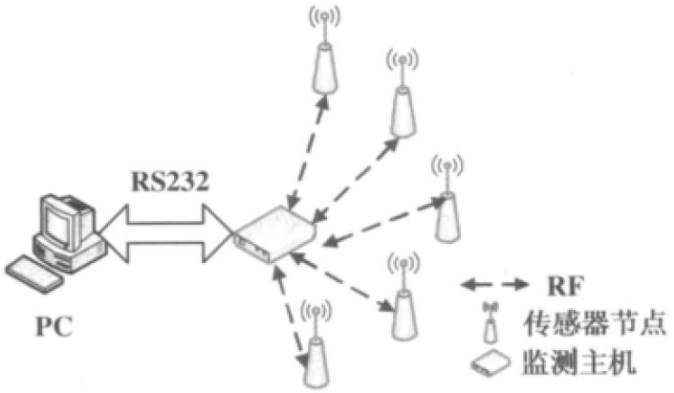


图6-1 室内环境监测系统结构图

2、硬件系统设计

系统中的微处理器单元（micro- controller united，MCU）选用Chipcon公司的CC2531芯片。CC2531是一种成熟的ZigBee系统芯片CMOS解决方案，满足ZigBee在2.4GHz ISM波段对低成本、低功耗的要求。CC2531包括了一个工业增强型的8位8051控制器和一个高性能的2.4GHz DSSS（直接序列扩频）射频收发器核心。增强型8051内核使用标准的8051指令集，具有8倍于标准8051内核的性能，满足CC2531协议栈、网络和应用软件对MCU的要求。

传感器节点的硬件设计原理如图6-2所示。传感器节点主要由处理器模块、无线通信模块、传感器模块和能量供应模块组成。传感器节点是室内环境监测系统的基本单元，负责采集室内环境数据，然后通过ZigBee网络上传给监测主机。由于室内环境复杂，传感器需要经过保护隔离单元然后再接入A/D转换单元。考虑到传感器节点的可移动性，电源模块采用5V-2A/h的锂电池供电，通过低压差稳压芯片AMS1117_3.3为CC2531提供直流3.3V工作电压。

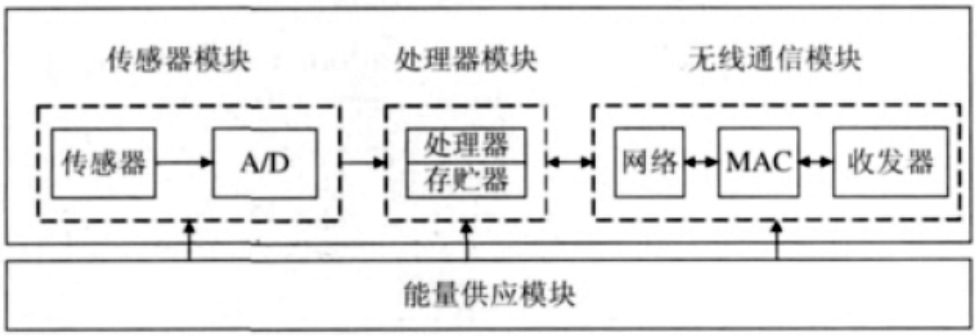


图6-2　　传感器节点硬件结构

监测主机的硬件设计原理如图6-3所示。监测主机主要由处理器模块、无线通信模块、串行接口模块和能量供应模块组成。监测主机通过无线通信模块和传感器节点进行无线通信，获取的监测数据被临时存储在128M的FLASH芯片中，这些缓存数据被定时开启的串口通信模块发送到PC机上。监测主机带有RS232数据转存接口，可以将储存的数据转存到PC机上。监测主机设计为可充电锂电池供电，也可以用AC220V供电。当使用AC220V供电时，同时也是给内置锂电池充电。当没有AC220V电源时监测主机会自动启动锂电池供电，这样既可以作便携式仪器使用，也可作为在线仪器使用。

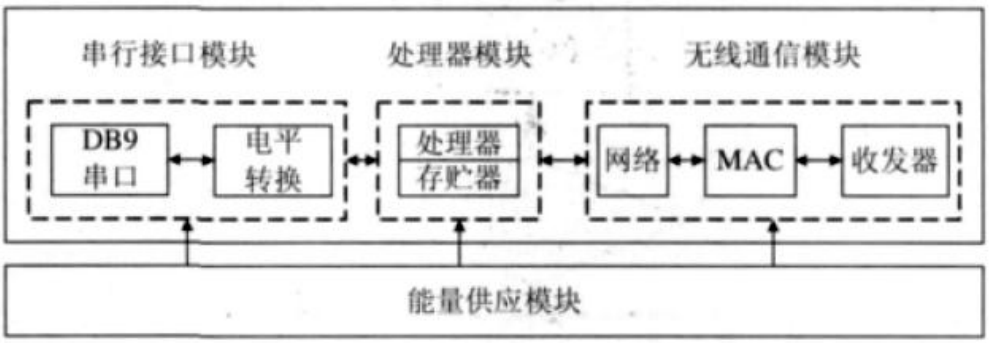


图6-3　　监测主机硬件结构

3 软件系统设计

3.1 监测主机软件设计

监测主机工作流程如图6-4所示。监测主机首先初始化CC2531，接着初始化协议栈和打开中断源，随后格式化一个网络。如果网络格式化成功，监测主机进入无线监测模式。程序的主循环较为简单，当接收到空气中的无线信号后，监测主机首先验证其合法性，然后判断传感器节点发送的信号类型。如果是新的传感器节点请求加入网络，则监测主机给该节点分配网络号，允许加入网络，然后进入无线监控状态。如果是传感器节点发送的监测数据，则监测主机将接收的数据存入FLASH中，同时也可选择将其通过串口实时地传送到PC机，最后进入无线监测模式。

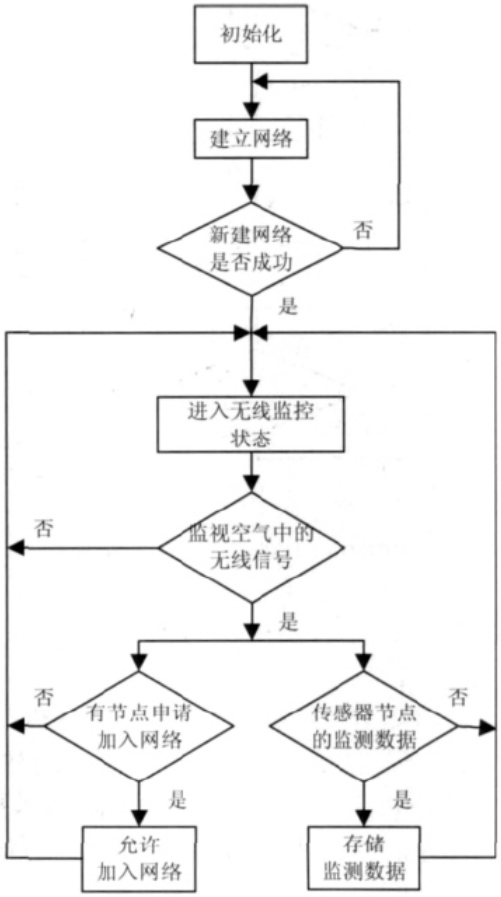


图6-4　　监测主机程序流程图

3.2 传感器节点软件设计

传感器节点的工作流程如图6-5所示。传感器节点程序首先初始化CC2531，然后初始化ZigBee协议栈，开始发送加入网络信号，等待主机响应。如果加入网络成功，传感器节点进入低功耗的休眠状态。当传感器节点收到设定的定时中断后，首先退出休眠状态，然后初始化A/D转换单元和传感器，接着进行数据采集，随后将采集到的数据发送到协调节点，最后关闭传感器，重新驱动CC2531进入低功耗状态，准备再次接收定时中断信号。

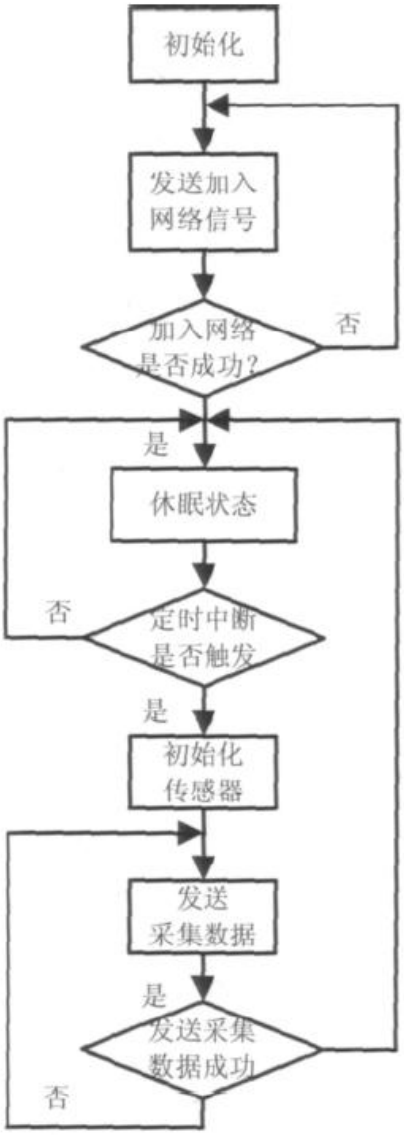


图6-5　　传感器节点程序流程图

五、　实验步骤

1.搭建ZigBee无线传感网络，运行环境监测系统。

2.实验测试：系统在标准两室一厅（面积<100m2）的居民住房内进行了测试，在两个房间和客厅分别布置6个传感器节点，将监测主机布置在客厅，通过PC机对监测的数据进行存储、分析和处理，并记录实验监测数据。

3.分析实验数据：观察传感器节点测得的监测数据（室内温度与甲醛浓度）与实际数据是否一致，分析是否满足室内环境监测的实际需求。

4.生成实验报告。

六、　拓展思考

1. 调研通用环境监测系统都使用哪些传感器。

2. 如何在 ZigBee 网络中新增一个传感器节点。

6.2 演示二 校园一卡通系统演示实验

一、 实验目的

本实验校园一卡通系统的功能是通过基于 ISO 14443 Type A 标准的 13.56MHz 的 RFID 卡片实现的，通过对校园一卡通系统演示实验使学生对高频 RFID 有一个直观、整体的了解，熟悉和掌握高频 RFID 卡片的特点。

二、 实验设备

- RFID 射频读写器
- Mifare One 卡片
- 校园一卡通系统演示程序

三、 实验内容

了解 ISO 14443A 标准，了解 RFID 相关基础知识，包括 RFID 卡片构造、存储及访问方式。掌握基于 RFID 的校园一卡通系统的使用，并能够使用实验系统对 Mifare One 卡片进行操作。

四、 实验原理

1. 概述

随着电子信息技术的发展，智能卡（IC 卡）已经在我们的生活中随处可见。射频识别卡正逐渐取代传统的接触式IC 卡，成为智能卡领域的新潮流。研究、开发射频识别卡的读写技术与读写设备，对其推广有着重要的实际意义。

RFID 是目前应用比较广泛的一种标识技术，其利用射频信号自动识别目标对象并获取相关信息，RFID 按频率分类，主要包括低频 125KHz、高频 13.56MHz 及超高频 915MHz。一个完整的 RFID 系统由读卡器、应答器（Tag）及应用软件系统组成。

13.5MHz 的 RFID 发展比较早，相关标准也比较成熟，主要国际标准有 ISO/IEC 14443 和 ISO/IEC 15693 两种，13.56MHz 的 RFID 技术已经在国内得到广泛应用，主要集中于身份识别、公共交通管理等领域。

本实验平台中的 RFID 读卡器模块采用 NXP MF RC522 的射频基站，完全集成了在

13.56MHz 下所有类型的被动非接触式通信方式和协议，支持 ISO14443A 的多层应用，通过读写器天线与 Mifare 卡和应答机通信。

2. 关于射频识别技术

射频识别（Radio Frequency Identification, RFID）技术是一种非接触自动识别技术，利用射频信号通过空间耦合(电感或电磁耦合)实现无接触信息传递并通过所传递的信息达到识别目的。射频识别技术的显著优点在于非接触性，因此完成识别工作时无须人工干预，能够实现识别自动化且不易损坏；可识别高速运动物体并可同时识别多个射频标签，操作快捷方便；射频标签不怕油渍、灰尘污染等恶劣的环境。当前，射频识别技术在国内最广泛的应用是射频识别卡。

3. 射频识别卡

射频识别卡（简称射频卡、RFID卡），也被称作非接触式IC卡（Contactless Smart Card，CSS）或非接触IC卡、非接触卡、感应卡，诞生于20世纪90年代初。由于成功地结合射频识别技术和IC卡技术，解决了无源（卡内无电池）和免接触的难题，RFID卡拥有磁卡和接触式IC卡不可比拟的优点。其一问世便立即引起广泛关注，并以惊人的速度得到推广应用。RFID卡由IC芯片、感应天线组成，完全密封在一个标准PVC卡片中，无外露部分，如图6-6所示。

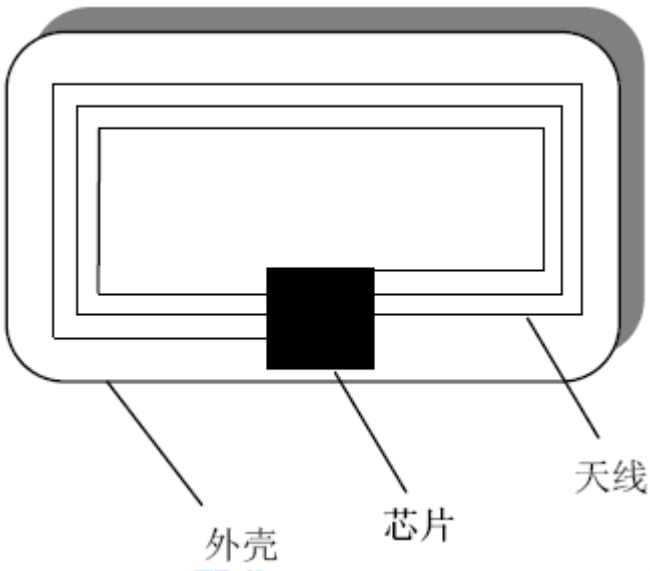


图6-6 RFID 卡结构示意图

4. RFID卡的优点

与接触式IC卡相比，RFID卡具有以下优点：

- 高可靠性：由于无触点，避免了由接触读写而产生的各种故障。提高了抗静电和环境污染(如油烟、灰尘、水汽等)的能力，因此提高了使用的可靠性、读写设备和卡片的使用寿命。

- 易用性：操作方便、快捷，无需插拔卡，完成一次操作只需0.1～0.3秒。使用时，

卡片可以任意方向掠过读写设备表面。

- 高安全性：序列号是全球唯一的．出厂后不可更改。卡与读写设备之间采用双向互认验证机制：即读写器验证卡的合法性，同时卡验证读写器的合法性。通讯过程中所有的数据都加密。卡片上不同分区的数据可用不同的密码和访问条件进行保护。
- 高抗干扰性：对有防冲突电路的RFID 卡，在多卡同时进入读写范围内时，读写设备可一一对卡进行处理，抗干扰性高。
- 一卡多用：卡片上的数据分区管理，可以很方便的实现一卡多用、一卡通。
- 多种工作距离：作用距离从几厘米到几米，适应不同的应用场合。

5. RFID卡的应用

RFID 卡以其方便交易、速度快、应用领域广而增长迅速，从长远角度看，RFID卡将会替换目前广泛使用的接触式IC卡。在国内，RFID卡主要应用在公共交通、身份识别、门禁控制等领域。

5.1公共交通：RFID卡应用潜力最大的领域之一就是公共交通领域。例如公交、地铁，乘客将RFID卡做的电子车票放在钱包或者包里就可以检票，方便快捷。公交经营者也宜于管理、减少支出。

5.2身份识别：使用RFID卡作为身份识别方式，比一般的证件卡片具有更高的防伪性，存储更多信息，便于管理。我国第二代公民身份证即采用RFID 卡，卡中输入生物特征信息及身份信息，以进一步加强防伪，同时便于全国实时管理。

5.3门禁控制：采用基于RFID卡的控制系统，可以自动检查每个人进入大楼、管理区的准入权限，并记录出入时间。

另外，还有高速公路收费、停车场收费、加油站收费、智能卡水表、电表及煤气表等应用，使用非接式IC卡都是首选。

6. RFID卡读写设备

RFID卡读写设备（或称阅读设备、读写器）是连接RFID卡与应用系统间的桥梁，是RFID卡应用中至关重要的一个环节。RFID 卡读写设备的基本任务就是启动RFID卡，与RFID卡建立通信，在应用系统和卡片间传递数据。RFID卡读写器将要发送的信息编码后加载到一个固定频率的载波上，当RFID（卡片内有一个谐振电路，其频率与读写器发送的载波频率相同）进入读写器的工作区域后，谐振电路产生共振并产生电荷积累，当电荷积累到一定数值时，就能为RFID 卡内的电路提供工作电压，使IC 卡内的芯片开始正常工作，处理读写器发送的数据信息。一般来说，完整的RFID卡读写设备的基本结构包括以下几个部分，如图6-7所示。

MCU：MCU 是读写设备的数据处理控制核心。它不仅要控制射频处理模块完成对RFID卡的读写，还要负责通过通信接口与主机或应用系统进行通信以及对键盘、显示设备等其他外部设备的控制。

射频处理模块：射频处理模块负责射频信号的处理和数据的传输，完成对RFID卡的读写。射频处理模块可以采用厂商提供的专用模块或射频基站芯片。

天线：天线的作用就是产生磁通量，为卡片提供电源，在读写设备和卡片之间传送信息。天线的有效电磁场范围就是系统的工作区域。

MCU与主机的通信接口以及键盘、LED/LCD显示等其它外部设备。

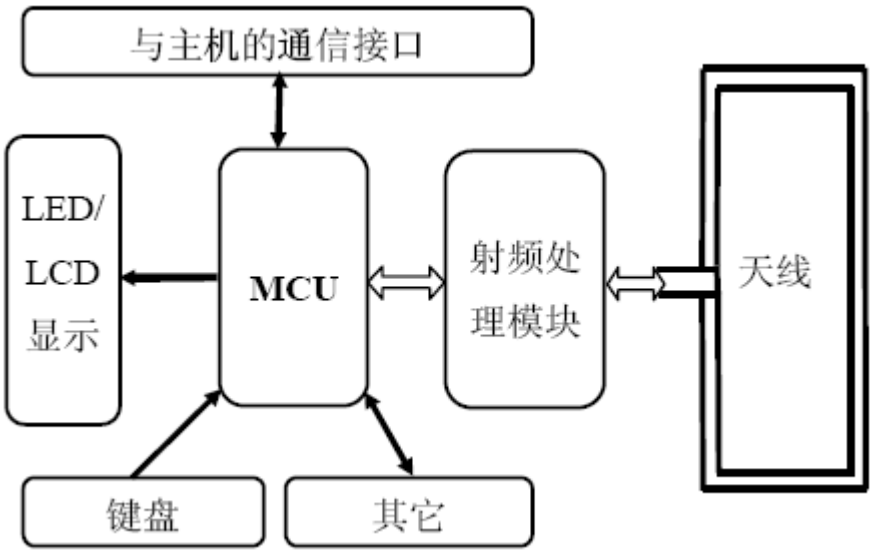
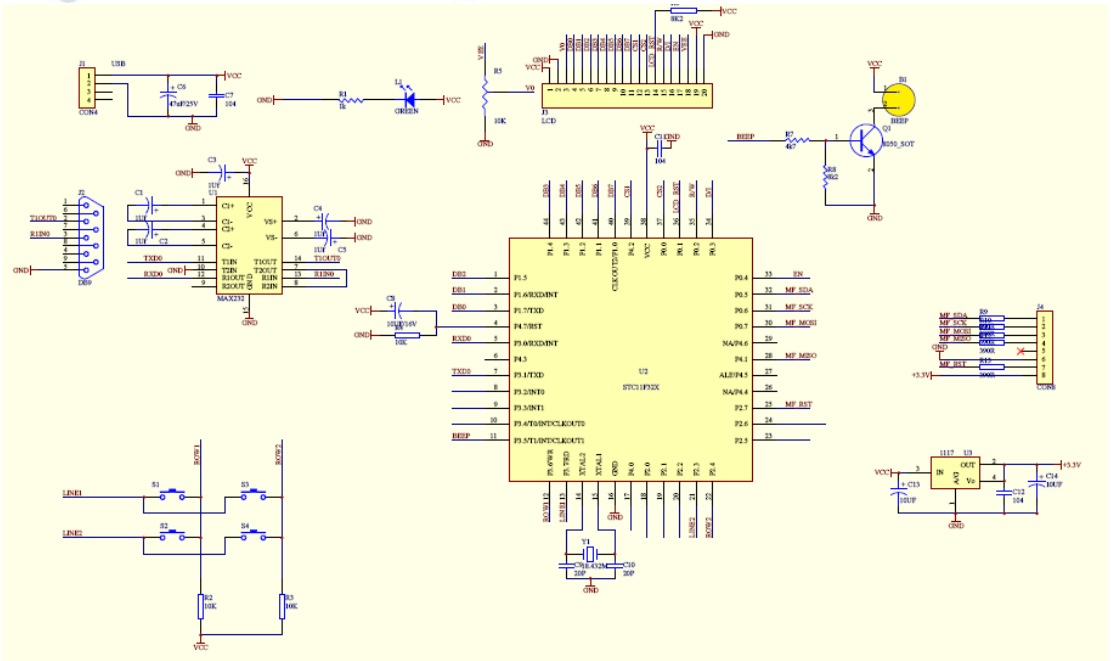


图6-7　　RFID 卡读写设备系统组成图

7. RFID硬件原理



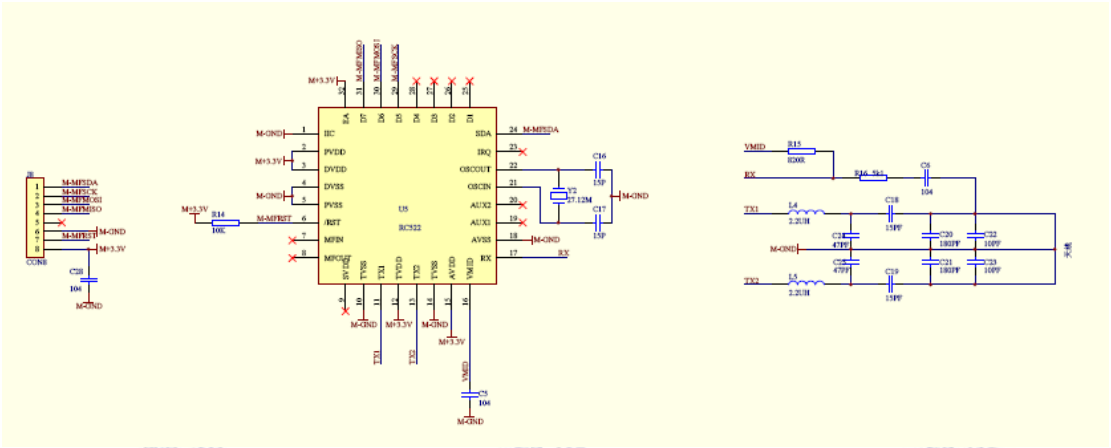


图6-8　　RFID 卡读写设备系统硬件原理图

如上图所示，整个读写模块由三大部分组成：主控MCU；射频读写芯片；天线及匹配电路。

7.1 主控MCU采用STC11F32X芯片，主要实现对射频基站芯片的控制操作。MCU通过IO管脚与射频基站芯片连接，通过SPI串行通信方式实现对射频基站芯片的控制和数据交换。

7.2 射频读写芯片采用MF RC522，其内部结构如图6-9所示，它负责接收主控MCU 的控制信息并完成与RFID 卡的通信操作。为了发送、接收稳定的高频信号，射频基站芯片要通过高频滤波电路与天线部分连接。

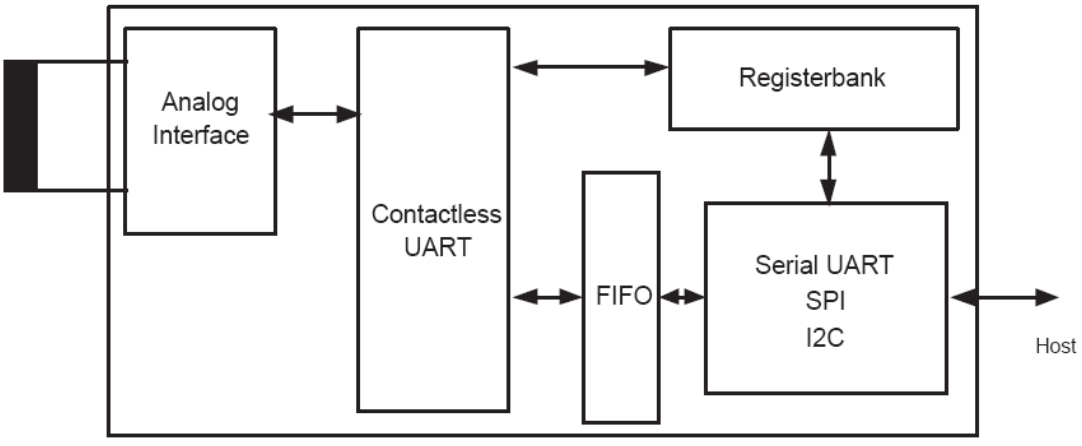


图6-9　　MF RC522 结构框图

主控MCU对MF RC522的控制是通过对其内部寄存器的读写来实现的。MF RC522 内部共有64个寄存器，分成4页，每页16个寄存器。关于MF RC500的寄存器描述请参见MF RC522技术手册。

7.3 天线部分，包括线圈及匹配电路，这是读写模块实现射频通信必不可少的一部分。读写模块要依靠天线产生的磁通量为RFID卡提供电源、在读写模块与RFID 卡之间传送信息。为使天线正常工作，天线线圈要通过无源的匹配电路连接射频读写芯片的天线引脚。

MF RC522 根据其寄存器的设定对发送数据进行调制得到发送的信号，通过由天线驱动引脚TX1 和TX2 驱动天线以13.56MHz 的电磁波形式发送出去。在其射频范围内的RFID卡采用RF 场的负载调制进行响应。天线接收到卡片的响应信号经过天线匹配电路送到MF RC522的接收引脚RX，芯片内部的接收器对接收信号进行解调、译码，并根据寄存器的设定进行处理，最后将数据发送到串行接口由微控制器读取。

8. RFID软件原理

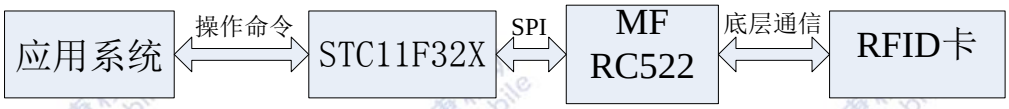


图6-10 RFID 软件原理框图

从图6-10可知，主控MCU芯片STC11F32X和射频读写芯片MF RC522及它们的外围电路共同构成读写模块，读写模块在射频识别应用中是应用系统与RFID卡之间数据交换的接口。

五、 实验步骤

1.连接实验箱电源，并开启电源。启动校园一卡通演示程序。应用程序如图 6-11 所示：

图6-11 校园一卡通演示程序界面图

2.操作软件：点击按钮“查询余额”，将 Mifare One 卡片放置在读卡区上，可以在“余额”

处看到卡片的余额。在“充值金额”处输入希望充值的金额，鼠标左键单击“充值”。鼠标左键单击按钮“查询余额”，观察“余额”处显示的余额。重复上述步骤，可以验证充值和消费的结果。

点击图书注册，我们使用另外一张 Mifare One 卡片（卡片 B）当作图书的条形码，为其注册图书信息。选择图书类别和图书名称，点“读取卡片”，将 Mifare One 卡片放置在读卡区上，当看到卡号时，表明读取卡片成功，点“确定注册”，将图书信息注册到 Mifare One 卡片。

点击借书区域的“扫描书签”，将 Mifare One 卡片（卡片 B）放置在读卡区上，此时可以看到界面上所显示的图书信息。

点击“借书”，将 Mifare One 卡片（卡片 A）放置在读卡区上，此时，校园一卡通系统演示程序把 Mifare One 卡片（卡片 B）的图书信息写入 Mifare One 卡片（卡片 A），借书过程完成。

点击“借书信息”，可以观察到借书信息。

点击还书区域的“扫描书签”，此时可以看到界面上所显示的图书信息。鼠标左键单击“还书”，还书过程完成。

六、 拓展思考

如何使用一张 RFID 卡片既作为饭卡、图书卡，又作为公交卡？

6.3 演示三 基于 ZIGBEE 的 RFID 读写器演示实验

一、 实验目的

1.掌握 RFID 读写的工作原理

2.掌握射频读写芯片对标签进行读写数据操作方法。

二、 实验设备

- ZigBee 无线网络
- RFID 软件操作系统
- Mifare One 卡片

三、 实验内容

了解RFID射频技术和ZigBee协议的无线局域网(WPAN)这两种种技术的特点、标准协议及实现的关键环节，掌握RFID读写器的工作原理，运行系统完成实验。

四、 实验原理

1.系统模型

基于ZigBee技术的RFID系统模型如图6-12所示，本系统是属于有源RFID系统，其通信频率为2.4GHz。系统后端为网络终端设备，网络终端设备完成与读写器的通信、控制以及与用户的直接交互式的工作，它通过网络对相连的读写器发出控制信息，或从相连的读写器读取数据，做出适当处理,包括对标签数据的过滤、汇集以及计算，最终生成图片信息和有效数据并友好地显示给用户。系统前端分成读写器和应答器两部分。读写器由读写器微程序控制器(MCU)、读写器ZigBee模块组成。读写器MCU接收到网络终端设备的控制信号后，做出相应的处理，并实时和读写器ZigBee模块保持通信,它是通过串行外围接口(SPI)与Zig Bee模块进行通信。ZigBee模块则作为RFID主模块实现对目标的ID识别，并通过中断将ZigBee模块的状态反馈给MCU。应答器则是由应答器MCU、应答器ZigBee模块组成。应答器MCU完成对应答器ZigBee模块的控制，应答器ZigBee模块完成接收识别认证请求,并发送自身ID码和基本信息。此外,为了体现ZigBee系统低功耗的优势,本系统选择高效低耗的MCU，例如飞思卡尔的HCS08系列的微处理器和Tl公司的MSP430F161xMCU。

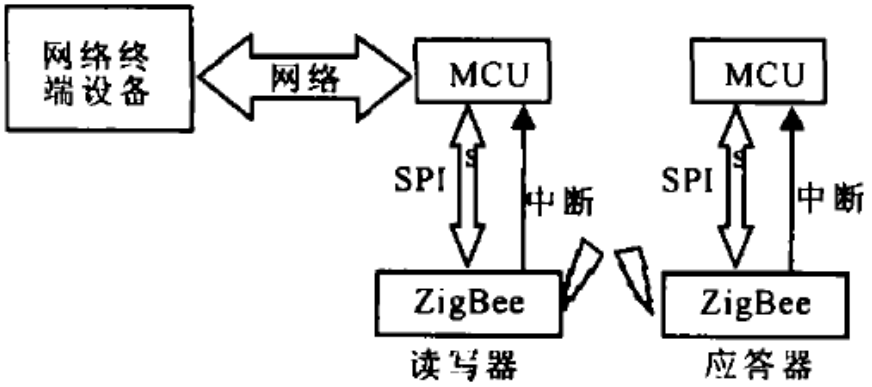


图6-12 基于 ZigBee 技术的 RFID 系统模型

2 系统主要功能及实现方法

2.1ZigBee模块状态转换

为了体现ZigBee技术的低功耗优势，系统的ZigBee芯片采用冬眠、睡眠和空闲三个状态转换模式，如图6-13所示。当读写器处于识别状态时，读写器ZigBee芯片必须一直处于循环发射并接收信号状态，以保证所有应答器都可接收到识别请求。而当读写器处于信息写入状态时，则读写器ZigBee芯片进人到冬眠状态，每当进行一次信息写入，读写器MCU便发送激活信息使读写器ZigBee芯片转变到空闲状态，随后进入到发送状态发送写入信息到应答器，确认信息写入完毕后，读写器ZigBee芯片重新进入到冬眠状态。在读写器处于射频识别的状

态时，应答器ZigBee芯片可以根据系统要求的识别速度来设置从睡眠状态到空闲状态的状态转换时间,每隔一定时间就转换到空闲状态，并连续接收信号数次，如果接收到读写器的识别请求，则连续发射识别信息以保证读写器ZigBee芯片能同步接收到完整信息，并在接收到读写器确认信息后重新转换成睡眠状态。

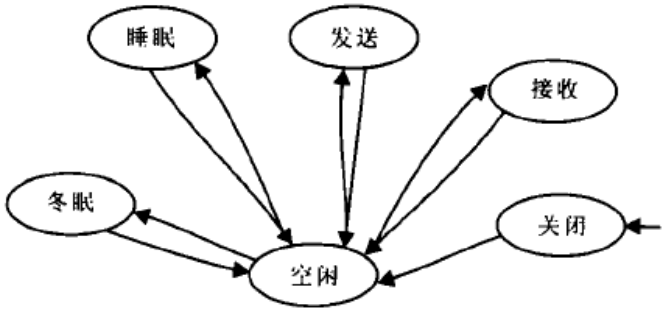


图6-13　ZigBee 芯片状态转换

2.2 ZigBee技术的通信确认机制

ZigBee技术采用完全确认的数据传输模式，每个发送的数据包都必须等待接收方的确认信息，保证了信息传送的可靠性，ZigBee协议规定的确认帧就是专门完成此项工作的。应答器处于信息写入状态时，在读写器ZigBee芯片发出请求后，接收到应答器的确认帧，便可停止发送写入请求。读写器在进行射频识别的状态时，作为应答器的ZigBee芯片,在发送自身的识别信息时也是在接收到读写器的确认帧后方可重新进入到睡眠状态。

2.3 应答器信息写入

应答器信息写入即由读写器将物品信息写入应答器中，也可以不断地对应答器内部信息进行擦写。成功写入信息后，应答器会返回成功确认信息。

应答器信息写入的具体实现步骤是用户在网络终端设备上选择信息写入按钮，输入需要写入应答器的物品信息以及识别代码，点击确认按钮将控制信息、识别代码以及物品信息通过网络写入读写器MCU，初始化读写器为写入信息到应答器状态；读写器MCU通过SPI接口再把控制信息、识别代码以及物品信息写入ZigBee芯片，并由ZigBee芯片发送出去，发送信息时需不断地重复发送同一请求，以保证应答器ZigBee芯片可同步接收到完整信息；应答器ZigBee芯片接收到信息后，通过中断通知应答器MCU，并将这些信息全部通过SPI接口传送给应答器MCU，应答器MCU将识别代码及物品信息通过SPI接口送入并存储在ZigBee芯片内。应答器处理完毕后控制应答器ZigBee芯片发送写入成功信息通知读写器，应答器信息写入完成。

2.4 射颇识别

射频识别帮助用户对物品进行ID识别，通过识别得到物品相关信息。具体的实现步骤是用户在网络终端设备上选择识别按钮,网络终端设备通过网络把控制信息传递给读写器MCU ，初始化读写器为射频识别状态，读写器MCU通过SPI接口发出控制信息控制ZigBee芯片发送识别请求；在读写器处于射频识别状态时，ZigBee芯片必须循环发射并接收信号，应答器ZigBee芯片接收识别信息，通过中断告知应答器MCU,应答器MCU控制应答器ZigBee芯片发送

物品信息给读写器;读写器ZigBee芯片接收信息,读写器MCU通过中断响应,将接收到的信息返回给网络终端设备,网络终端设备的中间件对数据进行加工,最终在屏幕上显示相关信息。

五、 实验步骤

1.搭建ZigBee无线网络以及RFID终端节点，烧写组网以及RFID代码，运行软件系统。
2.实验测试：进行刷卡操作，观察软件系统的显示信息。修改 Mifare One 卡片的信息，并保存，实现对于 Mifare One 卡片的读写操作。
3.记录实验数据，生成实验报告。

六、 拓展思考

6.4 演示四 无线传感网络网关演示实验

一、 实验目的

1.掌握无线传感网络网关工作原理。

2.掌握烧写网关系统的操作方法。

二、 实验设备

- ZigBee 无线传感网络
- 无线传感网络网关

三、 实验内容

本实验通过网关为基于嵌入式系统的无线传感器网络网关节点，使传感器网络中的各种数据信息可以通过Internet网络传输到远程终端，学生通过实验获取传感器数据，并对数据进行分析、处理等操作，从而掌握网关的工作原理。

四、 实验原理

1.WSN网关特点

WSN（无线传感器网络）是一种使用传感器协作地监控物理或环境条件、具有Ad hoc（多跳自组织）特性的的无线网络。WSN可用于汇聚大量终端节点的感知数据， 为感知技术提供了一种新的组播手段。由于传感器节点的“极简”特性，节点的数据存储能力十分有限。同时,传感器节点一般内置低功率的射频收发器，因此它其与其他传感器节点的通信范围有限。

WSN通常部署在偏远、危险或者敏感的环境中，因此我们需要一种有线或无线媒介，通过这种媒介设备，我们可以远程地监控和配置该WSN。而网关正是这样一种设备，通过网关，我们可以达到远程对WSN进行操作的目的。

假如没有网关节点的存在，用户需要在WSN覆盖区域的射频范围内才能对WSN进行实时监控和操作，因为WSN中节点的通信能力有限。在实验室或教学实验中， 我们首先对Mote模块进行编程，再将Mote模块与PC机直接相连（通过USB或RS232接口），这样，Mote模块可以充当PC机的“射频功能模块”，使用户通过PC机与WSN进行信息交互。然而，当WSN被部署在实时性很高的环境中时，这种WSN网关节点的模式显然不能满足要求。

通常，网关连接两个异构网络。而具体到这里，GWN（WSN网关节点）连接用户和WSN。GWN使用户能够远程管理和操作WSN，同时，使远程的数据存储和解析成为可能。感知数据可能通过GWN及时地传输到用户侧并存储在“离线”服务器中。当然，GWN同时需要具备将用户侧的数据（通常是用户的操作指令）传输到WSN中的能力，即双向通信能力。无线传感网网关与IEEE标准中用于无线通信的其他类型网关（或网络接入点）有所不同，GWN使用的是IEEE 802.15.4协议， 而其他普通的WiFi装置使用的是IEEE 802.11协议。WSN中使用的GWN需要具有低功耗的特性，同时需要高度接口化，以满足部署在特殊环境中的需要。

2．网关节点的设计原理

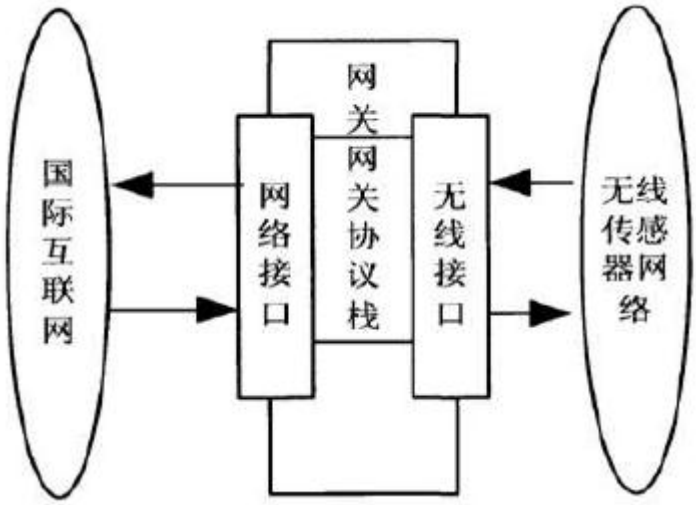


图6-14　无线传感器网络网关节点结构图

网关节点是一个特殊并且及其重要的节点，传感器节点可以通过它与外部网络进行通信。同样，外部网络中的用户也可以通过网关节点来对传感器网络内部的各个传感器节点进行查询和操作。因此，网关节点最重要的任务就是负责两个不同网络之间的数据转换，实现传感器网络与外部网络之间的信息交互。在无线传感器网络中，节点之间的数据传输一般采用短距离的无线通信技术，因此传感器节点与网关通信时，使用的是基于共享信道的无线通讯方式。

3．硬件设计思想

网关节点根据硬件结构来分，主要有以下两种：

1）网关节点是没有环境信息采集功能，仅带有无线通信接口、以太网通信接口的嵌入式网关设备。该网关节点可以接收无线传感器网络中其他传感器节点采集和处理后的数据，再通过以太网接口将数据最终传送到用户终端。

2）网关节点还可以是在普通传感器节点上加装串口通信模块、USB通信模块等装置与PC进行连接，使用PC来作为传感器网络的网关节点，来收集无线传感器网络中的其他传感器节点采集和处理后的数据。

网关硬件部件的具体组成与其在GWN中的作用如图6-15所示.

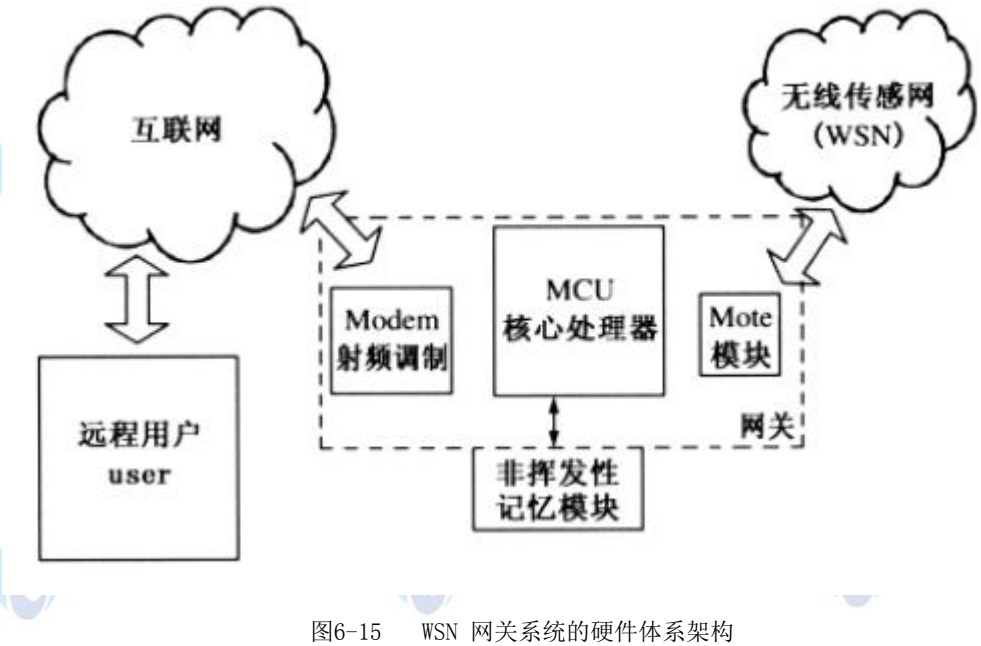


图6-15　WSN 网关系统的硬件体系架构

1）GWN与用户的互连

GSM模块为GWN与Internet 间提供了连接。MCU（核心处理器模块）配有Ethernet接口，倘若在以后有研究需要，将MCU连入LAN（局域网）是一件很容易的事情。GSM模块的选择在很大程度上受到MCU可用接口和连接器的影响，为了便于操作，我们通常选择通过RS232接口将GSM模块与MCU模块相连。

2）GWN与WSN的互连

通常，我们使用Mote模块（射频模块，内置WSN网络协议栈ZigBee，802.15.4的延伸版本）来作为连接GWN与WSN的媒介，Mote模块可以说是WSN网络技术工程实现的基础。Mote模块通过USB端口与GWN直接相连， 所以在选择GWN节点的具体硬件时，需要考虑到硬件节点的通用性，需要与各种封装802.15.4的WSN网络相兼容(即互通)。使用协议完整的Mote模块(例如ZigBee)比使用单独的802.15.4 （802.15.4协议只定义了物理层和MAC层）射频收发器要有优势。首先是简单，因为使用协议完整的Mote模块不需要再添加额外的硬件。同时所有与WSN相关的任务都可以由Mote来负责处理，Mote可以用来解析和过滤WSN数据包， 从中选出目的地地址为GWN节点的数据。

无线传感器网络嵌入式网关节点的硬件平台应该具有以下特点：

1）嵌入式无线传感器网络网关节点具有较强的网络控制能力。这是网关节点系统结构最重要的体现。网关节点的任务就是完成Internet和无线传感器网络两种异构网之间的信息交互，没有强大的网络控制能力，就无法妥善的完成网关节点的设计目标。

2)较强的信息处理及任务调度能力。作为网关节点会随时收到来自不同网络的数据传递任务，如果没有强大的信息处理及任务调度能力，那么势必会造成网关节点在不同的任务需求中来回切换，以至不能完成异构网之间的信息交互任务。

3）更好地支持网络通信协议。由于传统传感器节点硬件平台数据处理能力较弱，所以其很难实现高性能的网络特性。

4）更大的存储空间。网关节点为了便于将数据融合、处理，便于远端用户查询及定期打包发送，一般都需要较高容量的存储空间。

4.软件设计思想

操作系统是无线传感器网络网关节点软件的灵魂，由于嵌入式网关节点的特殊性，导致其对操作系统的需求相对于传统操作系统有较大的差异。无线传感器嵌入式操作系统一般需要满足以下几个特性：

1）实时性：只有操作系统具有较好的实时性，网关节点才能够及时响应不同网络的请求并及时反馈，按照相应的任务调度策略完成不同任务之间的调度，完成无线传感器网络和以太网之间的数据收发任务。

2）健壮性和容错性：网关节点上的操作系统必须拥有较好的健壮性和容错性，能够及时发现节点因为能量而失效的情况，并通过向网络发出调节请求来保证整个网络的正常工作。

3）剪裁性：网关节点操作系统需要有较强的针对性，其内核应能够自由配置，对于一些不需要的模块可以剪裁，从而适应网关节点对于不同硬件资源的要求。

4）网络支持：用于网关节点主要担负不同异构网之间的信息交互，因此需要在网关节点的嵌入式操作系统中提供必要的网络协议栈支持。

5）代码量：由于嵌入式网关节点的存储空间相对有限，因此网关节点的操作系统的核心代码量必须较小。

6）功能可扩展：未来，根据具体应用环境的不同，无线传感器网络嵌入式网关节点的功能也会多样化。

WSN网关系统的软件体系架构如图6-16所示。

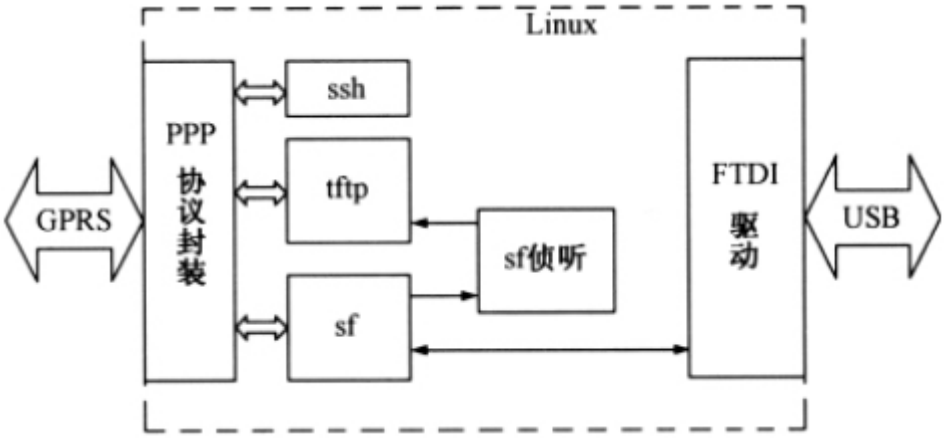


图6-16　WSN网关系统的软件体系架

1）嵌入式Linux

在诸多嵌入式操作系统中，嵌入式Linux是应用较为广泛的一种，嵌入式Linux中所有用于开发的软件都是开源和免费的。Linux提供了强大的文件系统、网路功能、GUI（图形用户界面）等软件模组支持，而且这些功能都是可以剪裁的，同时它还提供了标准的驱动程序接口和软件开发接口， 便于用户编程和程序维护。

在GWN节点中使用嵌入式Linux使整体网关具有很大的可扩展性，如果需要，可以让GWN充当系统Web服务器， 直接将感知数据转发到Internet。同时，GWN可以用来汇聚WSN中的数据并转发给远程数据库服务器。

2）工具链

嵌入式Linux 自身具备一整套工具链（GNUGCC），包括编译和调试工具，开发时可以自行建立嵌入式系统的开发环境和交叉运行环境，并且可以跨越在嵌入式系统开发中仿真工具（ICE）的障碍。

五、　实验步骤

1. 烧写网关程序，搭建无线传感网络环境。

2. 启动系统获取数据信息，分析并记录实验数据。

六、　拓展思考

6.5 演示五 低功耗的无线传感网络演示实验

一、　实验目的

1.掌握 ZigBee 低功耗网络工作原理。

二、　实验设备

- ZigBee 无线传感网络
- 传感网络数据监测软件

三、 实验内容

本实验为基于ZigBee的可配置无线数据采集平台，利用ZigBee通信标准来实现各采集点和管理终端的无线传输。在该平台的实现过程中，重点考虑节点的多功能化、低功耗特性，以及管理终端软件对网络的管理和配置。

四、 实验原理

1.系统的总体设计

ZigBee网络的拓扑结构分为3 种：星状结构、树状结构和网状结构，网络中的节点按功能可分为终端节点（EP）、路由器节点（RP）和协调器节点（CP）3 种。其网络拓扑结构如图6-17所示。

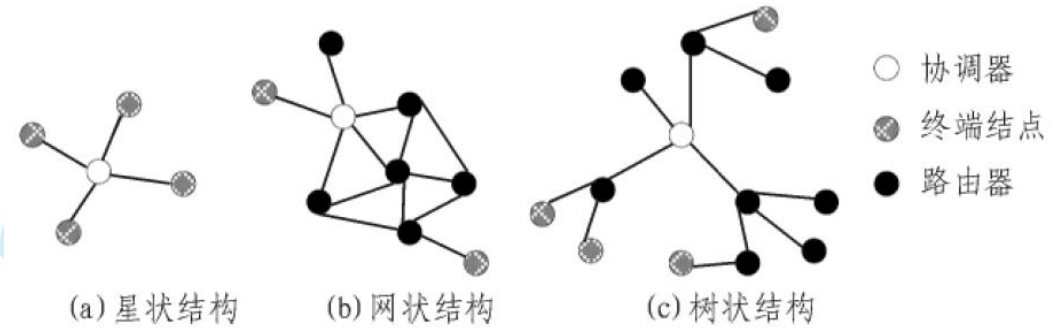


图6-17 ZigBee 网络拓扑结构图

本系统采用星状结构进行组网，包含一个协调器节点和若干个由无线传感器构成的终端节点，其中协调器节点通过网口与上位机连接，通过ZigBee协议与每个终端节点通信，各个终端节点之间无通信。

2.系统硬件设计

本系统硬件结构分为两部分：无线传感器和协调器节点。

2.1 无线传感器

无线传感器的功能是采集室内的各项参数，并通过基于ZigBee 协议的无线通信网络将数据传输的PC 上位机。需要根据所采集参数的不同选择合适的传感器，根据所选择传感器确定调理电路和供电电路。无线传感器结构图如图6-18所示。

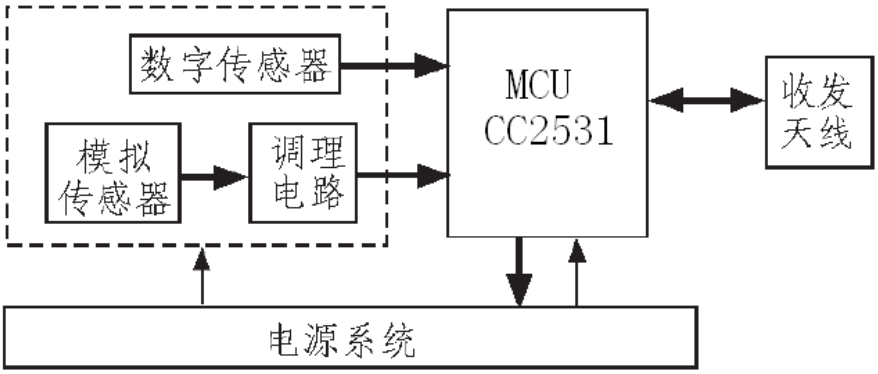


图6-18　无线传感器原理框图

传感器的选择需要根据所采集的参数的不同来进行，一般来说， 家居环境中需要采集的参数有以下几种： 温度、湿度、二氧化碳浓度及光照强度等。

根据小体积、低功耗、安装方便等原则选取无线传感器，选用SHT10 温湿一体化传感器测量家居环境中的温度和湿度，其供电电压为2.4~5.5 V，可直接以无线传感器的电源供电，测湿度精度为±4.5%，室温下测量温度的测量精度为±0.5℃，满足智能家居数据采集的精度要求；选用英国GSS公司的C20传感器测量家居环境中的二氧化碳浓度，其输出量为RS232串口数字输出，感应时间小于4s，供电电压5V和3.3V可选，功耗小于100 mW，符合低功耗要求；选用MG41-21 光敏电阻测量家居环境中的光照强度，其暗电阻大于等于0.1 MΩ，亮电阻小于等于1kΩ，满足测量精度要求。

CC2531外部有20个通用I/O 口，其中P0口8个管脚可以直接连接外部模拟量输入，内部有14位高速ADC，满足各类传感器的输入和A/D 转换的需求。CC2531与各类传感器的连接电路如图6-19和图6-20所示。

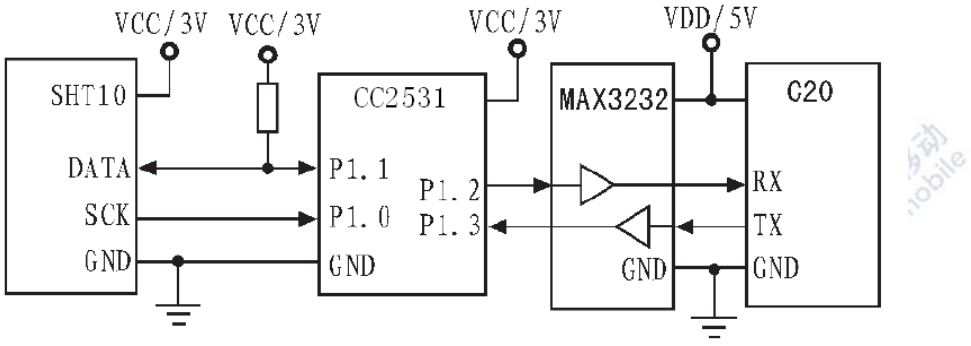


图6-19　数字传感器与 CC2531 连接示意图

在温湿一体化传感器SHT10与CC2531连接时，双方均为3V供电，数据线需加装上拉电阻以保证数据的正常传输。二氧化碳传感器C20采用5V电源供电和串口通信，为达到与CC2531的匹配，需在数据线路中加装MAX3232以进行电平的匹配，MAX3232供电范围为3~5.5 V，可选用3V和5V任意电源供电。由于二氧化碳传感器需要预热时间，所以数据采集时需要留出5s的余量，待稳定后方可进行数据读取。

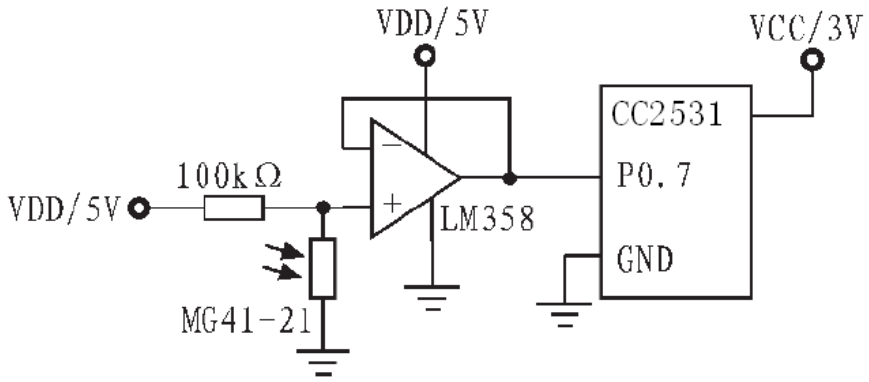


图6-20　光敏电阻与CC2531连接示意图

光敏电阻根据不同光照强度，电阻值在1~100kΩ宽范围内波动，在电路中增加100kΩ的分压电阻，在不同光照条件下采集MG41-21的两端电压，根据电压值的不同即可采集到不同的光照强度。在电路中加入LM358 运算放大器，接成电压跟随模式，可方便的完成光照强度的采集。无线传感器采用电池供电，为保证电池的使用寿命，需要对整个系统进行全面低功耗设计，其中包括休眠模式设计和电源模式设计。CC2531外接两个晶振：32 MHz 晶振与32.768 kHz 晶振。32 MHz晶振在正常的工作条件下使用，保证数据采集和处理的速度。32.768 kHz 晶振用于休眠模式，在此模式下降频休眠可进入低功耗模式。CC2531具有4种电源模式，可根据不同需求选择以降低功耗。

在电源系统设计中，采用输出可关断的MAX619 芯片来进行电源的控制，在休眠模式下切断传感器和调理电路的电源，保证系统整体低功耗。MAX619可通过配置接口接收CC2531的关断指令和恢复指令，在系统唤醒后可迅速恢复传感器供电，快速进入数据采集状态。由于在日常家居生活中不需要实时监测二氧化碳的浓度，且二氧化碳传感器所消耗功耗较大，故在软件设计中使用MAX619控制C20传感器的电源供应，在得到指令后方开启。

2.2 协调器节点

协调器节点的功能是连接上位机系统与ZigBee网络硬件系统的接口，其构造比较简单，仅需一片CC2531外部引出天线及RS232串口，采用5V电源适配器输出供电，不需要进行低功耗设计。由于CC2531的电平与上位机串口电平不匹配，需在中间加入MAX3232 进行电平匹配。

3 系统软件设计

3.1 无线传感器软件设计

无线传感器在上电后进行初始化操作，并且向协调器节点申请入网， 根据ZigBee协议的自组织网络特性完成入网注册，分配地址等操作。以上操作结束后即进入正常工作阶段，正常工作阶段由休眠唤醒后打开传感器和调理电路电源，进行传感器预热。预热结束后开始进行数据采集，将采集到的数据进行初步处理后发送给ZigBee网络的协调器节点，协调器节点回复接收成功后关闭传感器与调理电路电源，系统再次进入休眠阶段。数据采集流程图如图6-21所示。

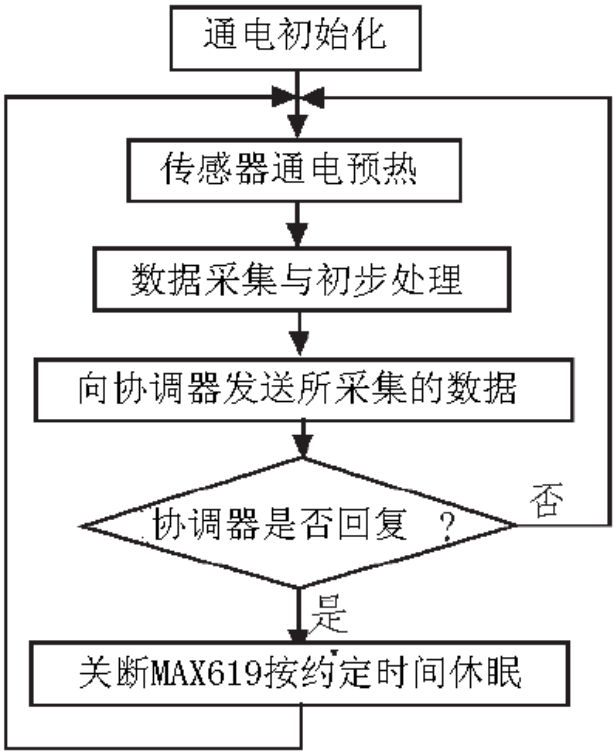


图6-21　无线传感器数据采集流程

3.2 上位机软件设计

上位机系统采用个人计算机来实现，采用Windows Server 2003 操作系统，方便数据库的连接和远程访问。软件开发语言采用Microsoft公司的C#。C#同时兼备了VC功能强大和VB简单易学的特点，具有高稳定性的特点，适合工业控制领域内的应用，同样适合智能家居系统的程序开发。数据库系统选用SQL Server 2005，此数据库系统稳定性好，连接方式多样，可以存储242字节级别的数据， 且使用SQL语言可以完成所有的操作，与C#兼容性好，可将SQL语言嵌入其中。上位机系统接收到ARM网关所传输的数据之后，对数据格式进行检验，检验完毕之后对其进行入库操作。上位机软件主要用于监测和处理各种传感器采集参数值，如空气温湿度、气压、光照度及气体浓度等。软件功能模块主要有串口配置、发射功率配置、模拟通道配置、数字通道配置、节点休眠配置、节点配置查询、启动系统、报表生成、查看和帮助菜单等功能。

五、　实验步骤

1.搭建无线传感网络环境，运行上位机系统。

2.采集精度测试：测量温度和湿度值，记录实验数据，计算采集误差。

3.功耗测试：无线传感器采用两节5号电池供电，供电电压3V。设定1 min采集一次数据，温湿度传感器数据采集期间工作50ms，工作电流约为30mA，休眠期间MAX619关断输出，电流在1μA左右。测量传感器模块工作是否满足低功耗要求。

4.传输性能测试：传输性能测试主要测试无线传感器与协调器的通信可靠性，在智能家居系统中往往有多间房屋，通信过程中由于墙壁的阻隔不能将数据实时发送。设定无线传感器数据采集周期为1min，测试时长为12H（720min），采用两节5号电池供电，测试对象为5只无线传感器，测量发送次数与正确接收次数是否一致，记录实验数据。

六、 拓展思考

1. 分析各传感器的功耗。

2. 分析系统的传输性能，如果测试中有发送次数与接收次数不一致现象，分析原因。

6.6 演示六 智能终端远程控制演示实验

一、 实验目的

1.了解 wifi 传输机制。

2.学会使用智能终端实现对智能家居的远程控制。

二、 实验设备

- 智能家居软件平台
- 智能家居模型沙盘
- 智能家居网关
- 智能控制终端

三、 实验内容

我们开发的智能远程控制终端是基于安卓系统来开发的，可以通过wifi和3G两种通讯方式与智能家居网关连接，从而控制智能家居内所有传感器节点，得到各传感器采集的实时数据，同时还可以通过服务器端设置与PC机相连，控制ZigBee协调器的打开和关闭。

四、 实验原理

1.系统组成

整个系统主要包括3个部分：远程控制终端（Android手机或者平板电脑），服务器和家电控制器。先由远程终端发送带有控制命令的数据包到服务器，当服务器收到控制指令之后，再由服务器发送控制命令到相应的家电控制器上。如图6-22所示

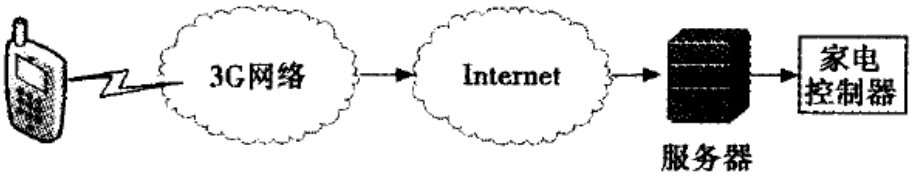


图6-22　远程终端控制系统结构图

服务器收到命令数据时，通过设备类型和子设备号来识别是哪个设备的控制命令，然后将相应的控制命令发送到相应的控制器上。对于家电控制器，分为两种。一种是简单控制电路的通断来控制电器的控制器，主要可控制灯光、冰箱等家电；一种是模拟红外遥控器发出信号的控制器，主要可控制器空调、电视机等使用红外遥控器的家电。我们有了家电控制器之后，就可以在不对家电做任何改动的情况下，对家电进行远程控制。

2.终端程序设计

2.1 传输协议的选择

TCP和UDP作为i目前最常用到的网络通信协议，TCP是基于连接的协议，UDP是一个无连接的、不可靠的协议，相对于基于流传输的TCP而言，UDP是基于消息传输的，整体上具有传输速度快等优点。通过研究可以看出家用电器的控制信息特点是数据量小，控制信息简单等，适合用于传送少量数据、对可靠性要求不高的应用环境，因此更适合采用简单的、面向数据报的UDP协议。因此在智能家居系统中，采用UDP协议作为智能远程终端和服务器之间的通信协议。

2.2 应用层帧格式的定义

由于目前智能家居通信协议没有统一的标准，各个生产厂家都有自己定义的通信协议。表6-1是本系统应用层报文帧格式的定义。

表6-1 系统应用层报文帧格式

报头		数据长度	路由地址		设备类型		子设备号	命令数据
66	AD	Len	31	XX	0Xf4	31	XX	

（1）报头：固定为0x66、0xAD两字节，表示一个报文帧的开始。

（2）数据长度：从报文段0x开始算到命令数据段的最后一个字节，表示整个帧的字节数。

（3）路由地址：这个数据位是对路由进行选择。根据部署的网络不同的位置，选择链路质量最好的路由节点进行数据传输。

（4）设备类型：表示家居设备的种类，我们将家电控制系统固定为0Xf4。

（5）子设备号：此字段为家电控制系统里的设备分配唯一的地址，这样我们可通过设备类型字段和子设备号两个字段唯一表示某个家电设备，便于控制信息的准确有效送达到控制器。

（6）命令数据：装载具体的家电控制命令，告诉家电控制器收到数据之后需要对家用电器做哪些操作。

2.3 socket 套接字

在网络层通过传输层进行数据通信时，常常会遇到多个应用程序提供并发服务的问题，为了区别不同的应用程序进程间的网络通信和连接，就需要使用socket套接字这个接口。对

于使用不同的协议通信，TCP是使用socket对象来实现，而UDP是使用DatagramSocket对象来实现。DatagramSocket是用于连接两个端点的分组投递服务，是由Android系统提供的一个公共类。根据编程人员所给的参数，它可以建立于任意可用端口或者与给定端口的连接。

这里我们需要了解的方法有close()、receive(DatagramPacket pack)和send(DatagramPacket pack)。它们分别是关闭套接字、接收数据报和发送数据报。在发送和接收数据报时，我们接收到的是一个DatagramPacket对象。DatagramPacket有多个构造方法，但是它的每个构造方法中一定包括byte[]buf和int length这两个参数。buf是用来存储所要传输的数据，length是表示传输的数据的长度。当然你在创建DatagramPacket对象时，还可以指定接收端的IP地址和端口号。

发送数据报的代码为：

```
DatagramSocket datagramSocket = new DatagramSocket(portNumber);
try{
DatagramPacket packSend = new DatagramPacket(message,message.length,
serverAddr,portNumber);
datagramSocket.send(packetSend);
}
catch(IOException e){
e.printStackTrace();
}
```

其中portNumber是指服务器的端口号，message是根据应用层帧格式得到的控制命令，serverAddr是指服务器的IP地址。

2.4 软件设计流程

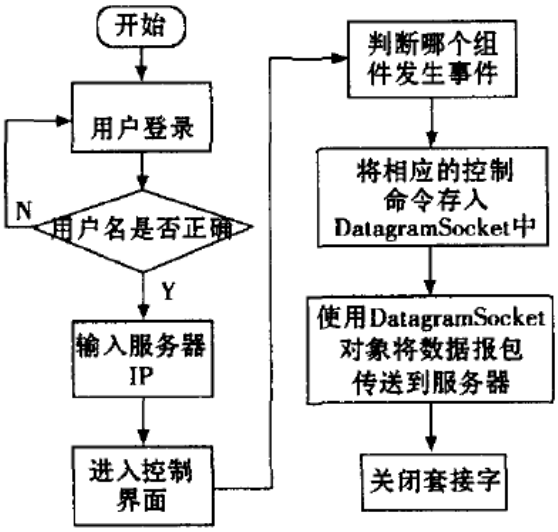


图6-23　软件设计流程图

图6-23是系统软件设计流程图。为了方便地使用于不同的家电控制系统，我们将服务器

的IP地址设计成可以由用户输入的方式，这样当在一个新的地方安装整套系统时，在无需改变Android安装包的情况下，自己输入新系统的IP即可实现远程控控制。

五、 实验步骤

1. 搭建智能终端控制环境，运行系统软件。
2. 操作终端实现对家居电器设备的控制。观察实验现象，记录实验结果。

六、 拓展思考

思考如何使用基于 TCP 协议的 socket 实现远程控制终端和服务器的通信。

6.7 演示七 物联网多传感器综合数据处理平台

一、 实验目的

1.了解多传感器数据处理机制。

2.掌握多传感器数据采集工作原理。

二、 实验设备

- 传感器模块
- 综合数据处理平台

三、 实验内容

物联网中多传感器数据管理中间件，可实现连接会话管理，数据解析、指令回发、数据缓存等功能，具有适应性和扩展性。系统解决了大量传感器连接下数据采集的稳定性和数据解析的正确性的问题。

四、 实验原理

1 多传感器数据管理中间件的分析与设计

多传感器数据管理中间件要解决系统与各种传感器的通讯。使用户可以通过对业务处理类进行扩展，来获取数据，进行数据的业务处理。它屏蔽TCP 服务器所需要处理的问题。如内存池、连接状态、连接上下文管理，封装了设备连接服务器的各种事件，使网络层通讯对用户透明，使用户可以专注于业务的开发。

1.1多传感器数据管理中间件的功能分析

多传感器数据管理中间件实现了多传感器数据采集的主要通用功能，包括会话管理、数

据解析、数据缓存、指令回发、定时循环等。这些功能在服务器里具有通用性，并提供可扩展的接口。多传感器数据采集的过程主要是传感器设备连接服务器后，系统会先给它分配一个上下文类，用于记录设备的信息，并分配内存，然后就会等待其发送数据，接收到数据后，开始遍历解析器，查找一个能解析该数据的解析器。当找到适合的解析器后，上下文会获得它的引用，方便以后数据直接调用该解析器。未验证的设备会先生成一个认证信息，主要是获取其序列号，然后引发验证设备事件。在上层的处理器会获取这个认证信息，然后返回认证结果。认证成功的设备将会再生成数据包，引发数据接收事件，交给上层处理器处理。认证失败的设备将会断开连接。

另外，有3个异常流程：①设备超时未认证，服务器会断开其连接；②设备超时未发送数据，服务器会断开其连接；③设备连接后发现已有相同连接，服务器会断开原来的连接，并把原来的上下文分配给新的连接。

系统数据采集和数据回流交互的流程：外部传感器把数据以字节流的形式发送给系统，通过用户上下文来获取它的数据解析器，解析成数据包，再通过采集器的接口发送给设备管理器，设备管理器会根据数据包的设备ID把数据路由到相应的处理器上进行业务处理。在一些协议里，需要设备回发指令的，可以调用处理器的数据回发函数，把指令回发给设备。系统数据缓存及批量写入流程：数据库缓存写入模块以泛型的方式适配不同的数据库表，一般是交给处理器类来调用的，调用它需要重写其ORM函数。它启动后由一个线程来管理，通过信号量激活，以队列的形式向数据库批量写入数据。这种写入方式，解决了大量数据写入的瓶颈，同时也解决了不同设备同时向数据库同一张表写入数据时造成的冲突。由于批量写入函数的限制，目前这个模块仅支持SQL 2000 以上的数据库。

1.2 多传感器数据管理中间件主功能设计

(1) 会话管理

每一个接入系统的连接会被视为会话，由UserToken类来管理其上下文。由于TCP/IP传送的是字节数组，如果服务器每次接受数据，都新建一个字节数组来存放数据，然后再向业务层发送，就会造成大量的内存碎片，从而导致操作系统频繁地作内存页调度，.net的资源回收频繁启动，导致性能大幅下降。要解决这个问题，就要用到内存池技术。内存池是在真正使用内存之前，先申请分配一定数量的、大小相等(一般情况下)的内存块留作备用。当有新的内存需求时，就从内存池中分出一部分内存块，若内存块不够再继续申请新的内存。这样做的一个显著优点是尽量避免了内存碎片，使得内存分配效率得到提升。UserToken 还需要解决的一个问题是TCP/IP传输的数据无边界。处理这种数据无边界的问题，一般来讲有3种方法：①规定消息的长度，每次接收固定长度的消息，作为一条数据；②设置客户端Socket不缓存，及时发送数据；③设定消息首尾边界符，用来区分一条消息的开始和结尾。由于本系统用于支持多传感器，在不确定传感器的情况下，前两种方法都不可能采用。因此采用了第3种方法。而目前接触的传感器，都是有规定数据格式的，在服务器上会按照它的格式匹配和解析数据。在UserToken里，除了接收缓存外，还有Temp-Buff 和HelpBuff 两个内存块。TempBuff 和HelpBuff没有采用内存池的技术，因为它们是随着UserToken新建的，在连接

有效过程中也是可复用的。同时使用了对象池技术，使得UserToken可以重用。会话管理还需对连接的有效性和合法性进行验证。在连接里，当客户端正确关闭socket 连接时，服务器端产生一个字节长度为0的接收事件，表示客户端中断连接。这时服务器端也会正确中断连接。不过这些都是发生在应用层的情况。驱动层会发送reset 数据包，服务器收到这个数据包可以正常关闭了。

当连接在物理层断开的情况下，结果证明，如果把网线拔去，无论服务器还是客户端都无法检测连接是否有效。在网络情况不好的环境里，这种情况经常发生，直接导致服务器资源挂起无法使用。如何及时有效地检测到一方的非正常断开，有两种技术可以运用：①御用是TCP 协议层实现的Keepalive；②运用应用层自己实现的心跳包。TCP 默认并不开启Keepalive功能，因为开启Keepalive功能需要消耗额外的宽带和流量，尽管这微不足道，但在按流量计费的环境下增加了费用，另一方面，Keepalive设置不合理时可能会因为短暂的网络波动而断开健康的TCP连接．另一种技术，由应用程序自己发送心跳包来检测连接的健康性。客户端可以在一个Timer中或低级别的线程中定时向发服务器发送一个短小精干的包，并等待服务器的回应。客户端程序在一定时间内没有收到服务器回应即认为连接不可用，同样，服务器在一定时间内没有收到客户端的心跳包则认为客户端已经掉线。

在本系统中，通过设置响应超时，来判断连接是否有效，超时时间可以根据传感器的通讯间隔设置。这样无效连接会由定时检查机制清理。另外，还需设置一个验证超时的项，当客户端连接服务器后，超过一定时间未验证成功，服务器将视为非法连接，由定时检查机制主动断开。

(2) 数据解析

系统定义了IAnalyzer 接口，用户根据业务需要开发解析器，解析器会被添加到连接管理器里，在连接的数据第一次到达时，通过遍历验证，来确认使用哪一个解析器。

(3) 指令回发

要做到设备管理，客户端也必须能接收由服务器转发的用户消息，因此系统留出了数据回发的接口。上层处理器可以向采集器发送指令，指令通过设备ID 路由给相应Socket，通过异步机制向客户端发送。

(4) 数据缓存机制

设备数据在通过业务处理后，通常会写入到数据库，用于日后做轨迹查询或者数据统计甚至数据挖掘用。假设传感器设备每秒向服务器发送一条数据(测试情况，正常设备一般不会这么频繁)，一天就会产生86 400 条数据。大量的数据写入数据库，成为了系统的主要性能瓶颈，数据库读写能力通常是评价一个系统性能的主要指标，而数据库读写能力的优化，可以从多个角度入手。

对于读，主要就是使用索引，这点根据业务不同而设计。写的方面，本系统采用了SqlBulkCopy类数据导入技术。Microsoft SQL Server提供一个称为bcp的流行的命令提示符实用工具，用于将数据从一个表移动到另一个表(表既可以在同一个服务器上，也可以在不

同服务器上)。 SqlBulkCopy类允许编写提供类似功能的托管代码解决方案。还有其他将数据加载到SQL Server 表的方法(例如，INSERT语句) ，但相比之下SqlBulkCopy提供明显的性能优势。实验证明，使用SqlBulkCopy的插入速度，是普通插入方式的36 倍。当然这种效率优势，要在大数据量得情况下才能体现出来，而这正是传感器管理系统的情况。

多个设备对应一张表，虽然有种种优势，但是也带来了一个问题。系统是采用异步机制的，当不同设备的多个线程同时向数据表写入数据时，就会导致假锁状态。直接导致等待超时及数据丢失甚至程序崩溃。为此，数据缓存模块建立了一个缓存队列，通过唯一的数据写入线程来向数据库写入数据，从而解决冲突。

(5) 定时循环

在实际应用中，有些设备是处于被动状态的，必须由服务器来主动请求数据。这时服务器就需要设置一个定时器，来定时向设备请求数据。定时循环机制封装了系统的Timer类，实现基于线程池的定时循环，同时它有设备处理器的引用，方便在它的处理函数里获得处理器的操作。

五、 实验步骤

1.搭建多传感器环境，运行系统软件。

2.启动各个传感器模块，获得传感器实时数据信息，分析发送和接收到的次数是否一致，记录实验数据并进行分析。

六、 拓展思考

1．思考系统是如何使用数据库，高效实现大量数据的存储。

6.8 演示八 基于 ZIGBEE 技术的智能安防系统演示实验

一、 实验目的

1.了解安防系统工作原理。

二、 实验设备

- 无线传感网络
- 红外报警模块
- 智能安防软件系统

三、 实验内容

本系统主要对智能家居的总体设计、硬件设计、软件设计、通信协议设计进行了研究，基于这些设计内容，实现了家庭安防预警、电源远程控制等智能应用。学生要掌握整个系统的工作原理。

四、 实验原理

1.系统总体结构

本系统的智能应用包括家庭的电源控制、家用电器的智能开关、家庭安防等。系统要求满足或解决以下问题。

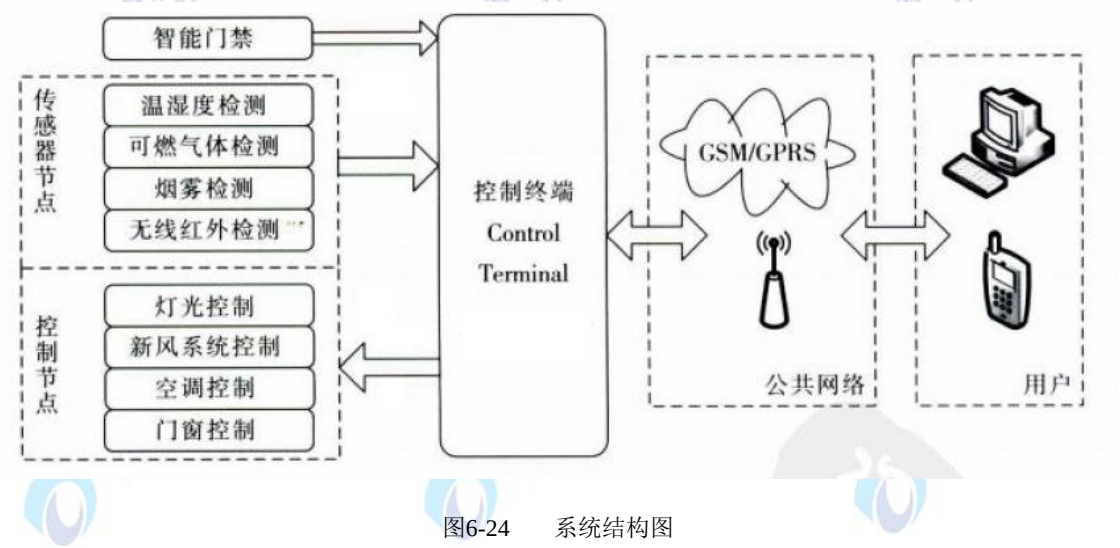


图6-24 系统结构图

1.1PDA 远程控制功能

用户可以通过PDA 终端，远程控制家庭中的家电设备，如家电的开关、电源的通断、三表的抄录等。在软件设计上，要求突出功能的可扩展性。

1.2 电源控制功能

用户随时随地通过终端设备实现对家庭所有电器的电源控制。需要解决的问题有：①实现对现有的家电设备的远程电源控制。②当小区的某个业主添加或更换电器设备，系统要提供对新增、更换后的家电设备的远程电源控制。

1.3 家庭安防预警功能

在人们的生活中，经常会遇到忘记带钥匙、忘记关煤气或是电源的情况，虽然可以在家里安装相关的探测设备、电子门禁，但这终究是事后救急。倘若能在异常情况发生之前，通过相关的消息提示，如在用户终端发送短信等方式提醒用户，那么必能做到防患于未然。因此，目标系统要求能实现这种智能风险预警功能，科学、适用地服务于广大用户。

2.系统功能模块设计

各功能模块如下：

①电源控制子系统：电源控制子系统用于实现对家庭电源的远程控制。是确保家庭用电安全、节约节能的主要措施。

②安防预警子系统：安防预警子系统主要实现对采集的数据进行处理，从中提炼有价值的信息，以供用户参考，从而实现对用户的有效预警。

③远程抄表系统：通过数据采集将电表、水表以及燃气表的数据写入到统一控制平台的数据库中，从而实现远程抄表功能。

④基础数据维护子系统：集中管理家庭、家庭成员等基本信息。

3．ZigBee技术

ZigBee基于IEEE802．15．4无线通信标准，IEEE802．15．4制定了Phy(物理层)和MAC(介质访问层)的标准，而ZigBee联盟则制定了NWK(网络层)和APP(应用层)的协议标准。ZigBee作为一种无线通信技术，可工作在2.4GHz(全球流行)、868MHz(欧洲流行)和915MHz(美国流行)3个频段上，它的传输距离在10—75m的范围内。ZigBee具有如下特点：(1)功耗低：ZigBee的传输速率最高为250kbit／s，发射功率仅为lmw，且采用了休眠模式，所以ZigBee设备非常省电。据估算ZigBee设备仅靠两节5号电池就可以维持长达6个月到2年左右的使用时间。(2)成本低：ZigBee模块价格低廉，并且ZigBee协议是免专利费的。

4．基于ZigBee技术的智能家居安防系统的设计方案

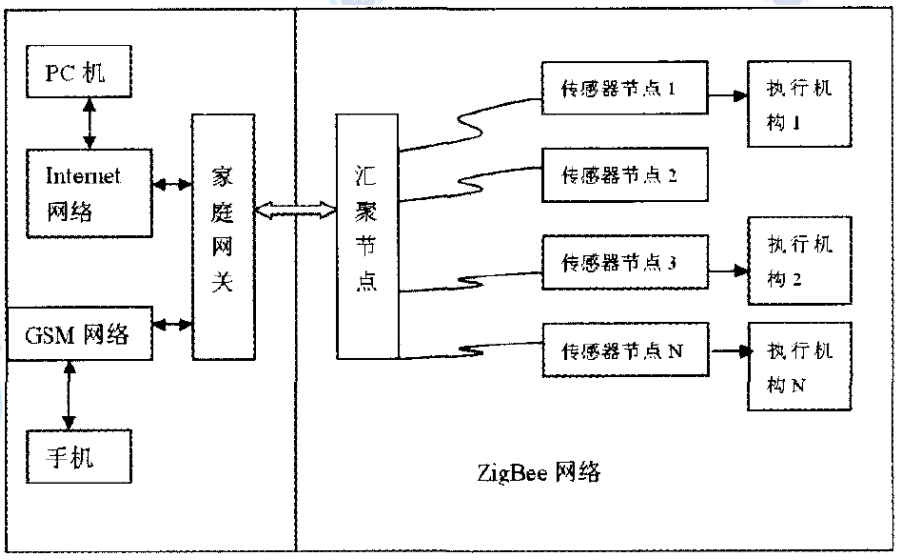


图6-25　智能家居安防系统整体设计方案

智能家居安防系统的整体设计方案如图6-25所示，系统包括两大部分：ZigBee无线模块组成的家庭无线监控网络和家庭网关。

其中ZigBee网络包括执行机构、传感器节点和汇聚节点。执行机构负责各种家用安防设备的启动，比如发生煤气泄漏时关闭燃气管道上的电磁阀。传感器节点通过自组织的方式组

建无线通信网络，负责采集和传递各路监控数据，并控制执行机构。汇聚节点是整个网络的核心，选用结合了高性能2.4GHz射频收发器和8051控制器的系统芯片CC2531，其满足系统对低成本、低功耗的要求。CC2531芯片上的8051MCU控制整个节点的工作，CC2420RF收发器负责节点的数据收发。整个ZigBee网络的建立和管理也是由汇聚节点负责，同时汇聚节点通过RS232串口模块负责和家庭网关进行通信，以作进一步的处理。家庭网关通过与汇聚节点之间的通信，利用Internet网络和GSM(Global System ofMobile communication)，全球移动通讯系统)网络发出报警信号，并能对家中的安防情况进行远程监控。设计时考虑将安防情况做好记录以备查询，需要组建数据库。

五、 实验步骤

1. 搭建智能安防系统演示环境，运行系统软件。
2. 操作安防系统实现对家庭环境的安防控制。
3. 观察实验现象，记录实验结果。

六、 拓展思考

1. 如何提高安防系统的数据安全性。

2. 分析安防系统的应用前景。

6.9 演示九 基于 ZIGBEE 的仓储监测演示实验

一、 实验目的

1.了解仓储监测系统工作原理。

6. 实验设备

- ZigBee 模块
- 射频模块
- ARM 网关

7. 实验内容

利用 RFID 采集数据结合 ZigBee 无线数传组网的技术来实现对仓库的无人监测，射频读写器即可采集数据又能及时处理，并实时数传实现多条组网的优点，通过检测多个网络节点定位检测，保障仓库系统的安全性。

8. 实验原理

1. 系统结构

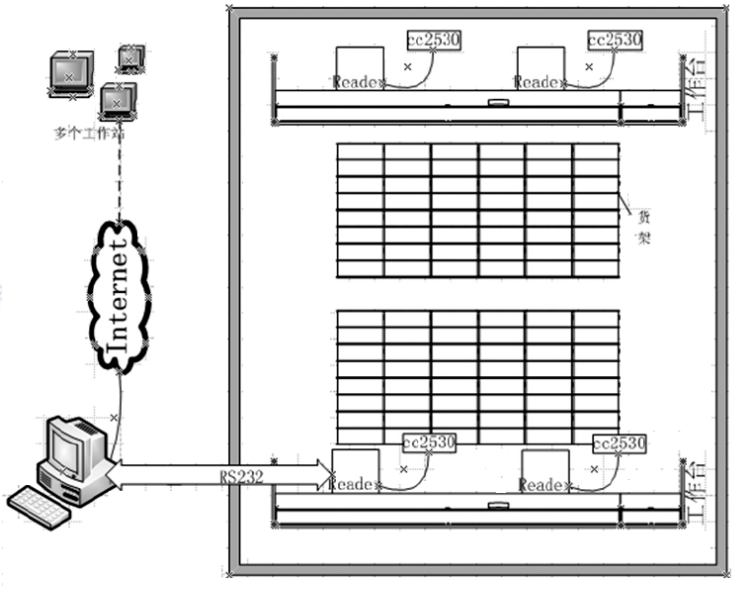


图6-26　系统总体结构示意图

整个系统由硬件系统和软件系统组成，硬件系统即所需要的硬件设备，包括PC机、射频模块和ZigBee模块都嵌入ARM11构成的新装置、仓库管理监控软件系统，通过融合射频识别和ZigBee组网，实现对仓库货物的无线实时监测。总体结构示意图见图6-27，通过将射频模块和ZigBee模块都嵌入ARM11平台构成一个新的装置，针对每个货架的货物位置寻找一个最优位置固定地安装一台或者多台此类设备，还可以将此设备置于小型机器人上移动式地监测，确保无线信号能够覆盖所需监测货物的范围，多个此设备之间组网无障碍，对每个货物进行数据采集后通过ZigBee无线网络传递给主控节点，再经由上层管理软件，实现仓库的集中监测和管理。

2. 硬件结构

硬件系统主要由S3c2440及外围必需设备组成的ARM11系统、远距离读写器模块、ZigBee模块等硬件设施构成，硬件结构见图6-27。

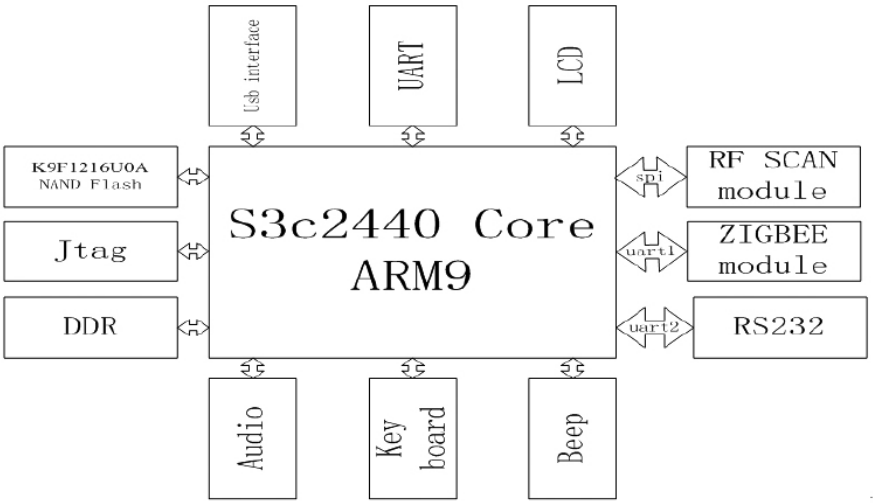


图6-27　系统硬件结构图

（1）该系统处理核心采用的是S3c2440处理器，实验用ARM9开发板，有完整的外围电路诸如Jtag调试接口、一个RS232串口、三个可扩展的TTL串口、USB 接口、RTC 备份电池、音频接口、RF45网络接口、LCD接口、SD卡座、系统总线接口、中断按键接口以及GPIO 接口等丰富的系统资源可供设计使用。

（2）ARM11没有可以直接使用的SPI口供给RF扫频模块，所以使用ARM11丰富的GPIO资源飞线连接RF扫频模块构成一个三线SPI通讯机制，来进行RF扫频模块和CPU之间的通信，此为半双工通信保证了信号的同步，具体的配置根据需求进行恰当的时钟频率等选择，RF射频模块配置433MHZ频率采集超级标签信息，远距离射频模块发射接收距离为5米，能够满足一般大小的货架需求；ARM11的RS232串口用于与外接的ZigBee模块通讯使用，ZigBee使用的是TI的CC2530，尽管现在流行的是MSP430处理器MCU配合CC2530是一种主流，但是由于系统的特殊和需要大屏幕LCD液晶屏显示功能，所以此处ARM9是最佳选择，CC2530是符合IEEE802.15.4 和ZigBee标准SOC解决方案的，2.4GHZ射频采用的是国际免费频道，能够实时满足ZigBee协议栈的处理需求。

（3）整个硬件系统功能全面，实际使用时可分为全功能设备和半功能设备，全功能设备就是包含上述硬件结构图所有外围电路，能够在后端软件处理蜂鸣器报警、液晶屏可以显示所有监控参数、可外接U盘、可与用户通过键盘进行直接的交互；半功能设备则是只嵌入射频模块和ZigBee数传模块和一些RAM的ARM11精简版系统。作为一个完整的仓储监测系统，此系统有一个全功能的设备作为主协调器，通过网络接口负责与上位机进行数据的远端传输，协议自定。

3.软件结构

该系统的信息管理单元主要由数据库和可视化界面组成，又可以分为嵌入式设备上的数据库和QT管理与上位机的后端软件两种。在功能上完成对货物的入库管理、出库管理、货位管理、数据库及报表管理等基本功能以及货品定位等扩展功能。数据的上传采用FTP 标准协议，通过ARM9网关实时上传数据并且通过心跳保活机制保证数据不丢失，即在断电的情况下

将数据存储到SD卡里面，而不会发生断网就丢失数据的危险。另外，ARM9的网卡驱动和串口驱动可以根据实际情况和电路进行设计，以使系统工作在低功耗模式下。最后根据ZigBee自组网协议设计组网算法，实时、快速、准确地采集并传输数据。

本系统的全功能设备兼无线通信和数据采集功能于一体，各节点数据信息汇聚到协调器节点进行数据的传输并统一处理，通过通讯接口传送到前端全功能设备(即协调器)Linux系统的数据库中和后端服务器的管理软件中。软件方面就ZigBee组网的改进算法在下面进行详细的说明。基于Tree+Z-AODV路由算法加入延时防抖和中端触发机制进行该系统ZigBee的自组网设计， 使用分等级的Tree路由算法和ZAODV相结合的混合路由算法网状拓扑网络结构。根据节点中传输数据的不同，通过设计数据帧帧头的DiscoverRouter域，分别使用抑制路由发现、使能路由发现、强制路由发现等路由方法，在各路处理中加入延时防抖和中断触发处理，能够有效减少ZZigBee无线数传的误传率和漏传率，提高系统的实时性和准确性。下面是路由新算法的流程图6-28。

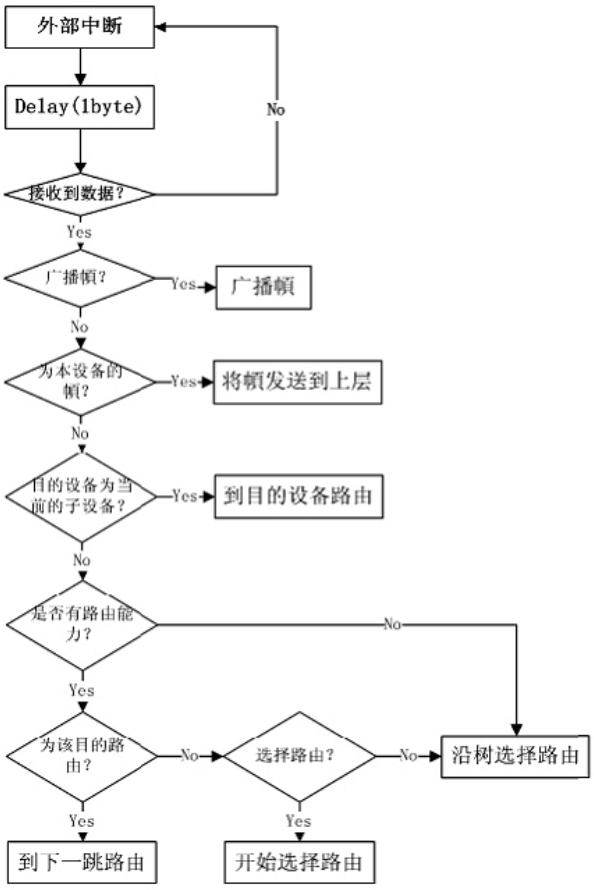


图6-28　改进的 ZigBee 组网流程图

9. 实验步骤

1.搭建仓储监测演示环境，运行系统软件。

2.操作系统软件实现对仓储信息的实时监测。

3.观察实验现象，记录实验结果。

10. 拓展思考

1.分析 RFID 传感器的可扩展应用。

2.总结 RFID 传感器的特性。

6.10 演示十 智能家居演示实验

一、实验目的

1.学习和理解传感器基本原理以及 IEEE802.15.4 通信标准。

2.掌握各种电器检测控制传感器模块的工作原理。

二、实验设备

- 各传感器模块
- 智能家居模型
- 智能家居实践实验平台

三、实验内容

通过实验使学生掌握基于物联网的智能家居系统工作原理，了解智能家居构建所设计到的通信协议，形成物联网智能家居系统应用的整体概念。

四、实验原理

1.系统介绍

基于物联网的智能家居系统内部组网需要控制的设备多，网络容量大；系统包含多种高速、中速及低速的数据设备，对信息传输的延时性、实时性要求严格；兼容广泛的连接技术；需要保证高级别的安全性。

本智能家居演示系统旨在运用ZigBee技术构建一个家居检测控制系统。系统拓扑结构图如图6-29所示，从图中可以看出，本系统大致有安防传感子网、家电控制子网、信息管理平台及远程终端等部分组成。其中，安防子网由温度传感器、烟雾传感器、温湿度传感器等各种传感器模块组成。家电控制子网内的设备基本为受控设备。通过检测环境温度、光照度、烟雾浓度以及红外入侵等信息，可直接联动控制相关受控设备或将信息发送至管理平台，由信息管理平台决策如何处理这些信息。

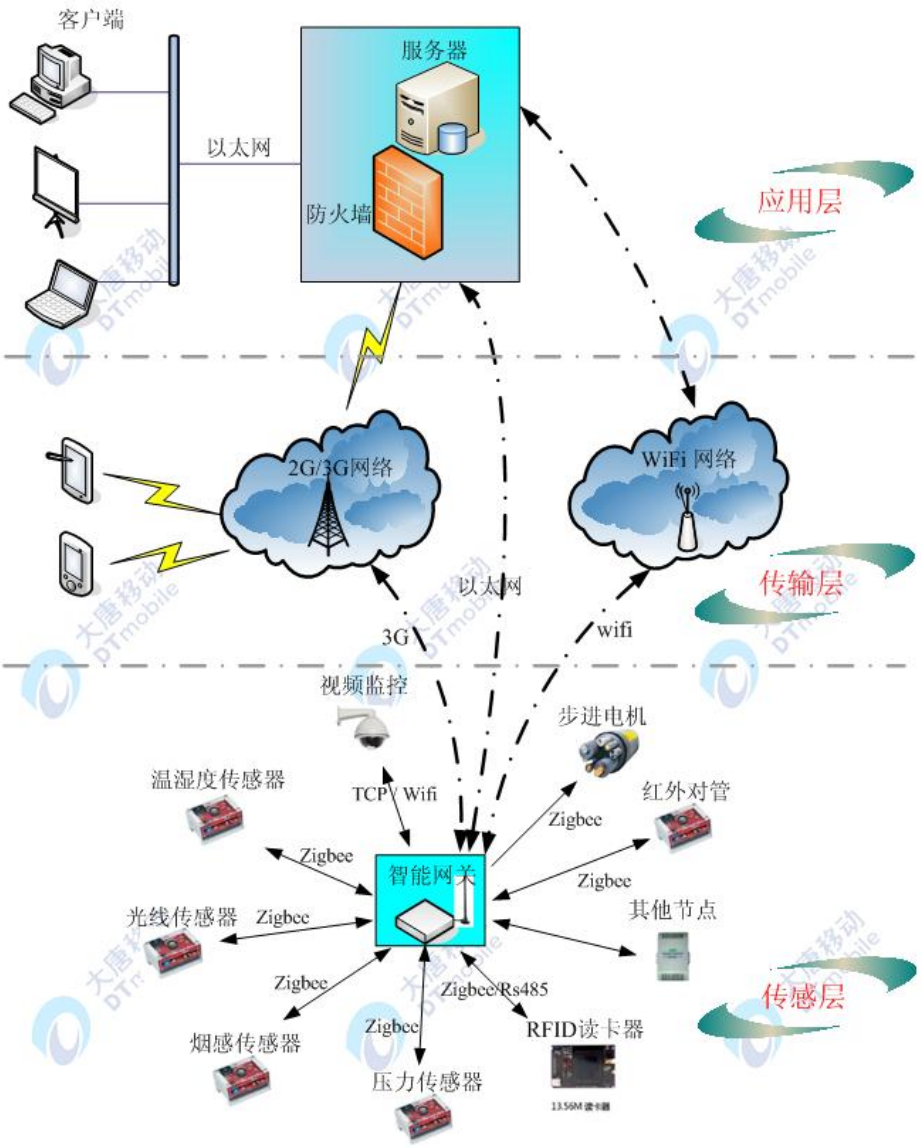


图6-29　　智能家居网络拓扑图

图6-29所示智能家居系统网络拓扑图中，家居设备、移动终端以及其他域设备通过网关实现信息互通、协议转换、信息共享和交互；借助高速有线网络与无线互联，在QoS、UPnP、DHCP等关键技术的支撑下，各种智能家居设备可以自动、无配置地接入系统，极大提供了系统的灵活性、易用性及可扩展性。

基于物联网的智能家居组网采用分层体系架构，可以划分为传感层、传输层、应用层 3 个层面。

智能家居传感层：包括了各种与家电、家居相关的传感器、控制器、执行器及识别装置等，以及有线网络结合无线泛在网络的物理连接。

智能家居传输层：主要包括家庭内部网络和骨干网络接入两部分。家庭内部组网支持的有线方式包括电子载波 X-10 和 CEBUS、电话线 HomePNA、以太网 IEEE802.3 和 802.3u、串行总线 USB1.1、USB2.0 和 IEEE1394 等；无线方式包括无线局域网、家庭射频技术、蓝

牙、红外、ZigBee 等。网络接入层通过家庭网关，实现不同应用协议规范的互联互通互操作，并与骨干网络实现无缝连接。

智能家居应用层：以用户为中心的融合业务层，提供用户接口和不同技术支持的智能家居服务。通过智能家居组网多层协作的自适应QoS，自适应匹配异构网络及终端设备，充分保证端到端的媒体、语音、安防等多种业务的服务。智能家居演示系统包括门锁控制单元、灯光亮度控制单元、家用电器控制单元和家庭模式选择设置控制单元。这些控制单元对家庭生活中常用的电器设备检测和控制，并通过无线传感网络与室内主控单元互联互通（主控）。

2. 传感器数据采集原理

传感器节点上电后，会自动加入到和其PANID相同的ZigBee网络中去，网络协调器会给节点分配唯一的网络地址，以便后期采集数据时以此网络地址作为搜索目标。通过设置软件界面的采集周期，软件平台会自动的向温度传感器节点发送读取数据的命令，首先软件平台通过网络通信发送命令到ARM网关，ARM网关解析后通过串口发送给网络协调器，协调器通过ZigBee协议将命令发送到温度传感器，温度传感器节点得到相应的请求，就会把测量的值逐层返回到软件平台。下面分别介绍各个传感器模块的工作原理：

2.1温湿度传感器

SHT10属于Sensirion温湿度传感器家族中的贴片封装系列。传感器将传感元件和信号处理电路集成在一块微型电路板上，输出完全标定的数字信号。传感器采用专利的CMOSens技术，确保芯片具有极高的可靠性与卓越的长期稳定性。传感器包括一个电容性聚合体测湿敏感元件、一个用能隙材料制成的测温元件，并在同一芯片上，与14位的A/D转换器以及串行接口电路实现无缝连接。

温湿度传感器 SHT10 有 4 个引脚：GND、DATA、SCK、VDD。下图 6-30 为传感器的典型应用电路，也是它与单片机的连接方式。

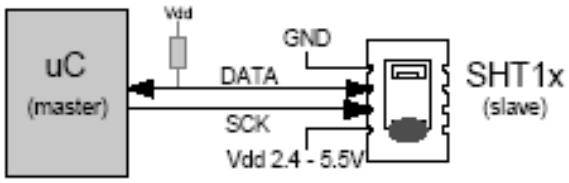


图6-30　典型应用电路

温湿度传感器芯片采用 SHT10；

- SHT10 的 DATA 口接 CC2531 的 P0_1　//定义通讯数据端口
- SHT10 的 SCK 口接 CC2531 的 P0_7　//定义通讯时钟端口
- SHT10 的 VDD 口接 CC2531 的 VCC　//2.4~3.3V 供电（SHT10 的“电量不足”功能可监测到 VDD 电压低于 2.47V 的状态。精度为 0.05V。）
- SHT10 的 GND 口接 CC2531 的 GND　//地

2.2烟雾传感器

我们实验用的典型型号 MQ-2 烟雾传感器属于半导体气敏式烟雾传感器。它由微型 AL2O3 陶瓷管、二氧化锡敏感层，测量电极和加热器构成的敏感元件固定在塑料或不锈钢制定的腔体内，加热器为气敏元件提供了必要的工作条件。封装好的气敏元件有 6 只针装管脚，其中 4 个用于信号取出，2 个用于提供加热电流。MQ-2 气体传感器所使用的气敏材料是在清洁空气中电导率较低的二氧化锡。当传感器所处环境中存在可燃气体时，传感器的电导率随空气中可燃气体浓度的增加而增大。使用简单的电路即可将电导率的变化转换为该气体浓度相对应的输出信号。该传感器常用于家庭和工厂的气体泄漏装置，适用于液化气、丁烷、丙烷、甲烷、酒精、氢气、烟雾等的探测，是一款适合多种应用的低成本传感器。

烟雾传感器在 A、B 处检测周围空气的烟雾浓度，电压供烟雾传感器的加热丝工作一段时间预热后，才能正常检测烟雾。当烟雾传感器所处的环境烟雾在允许的范围内时，两端输出电极的导电率很低，则加在电极两端的电压很低，这样，输出的模拟信号电压升高。我们对输出的模拟电压信号进行 ADC 采样，利用单片机进行 ADC 转换，并将转换的到的数据通过串口传输到电脑上，形成烟雾浓度曲线图。

2.3加速度传感器

我们实验用的MMA8452加速度传感器是一款具有12位分辨率的智能低功耗、三轴、电容式微机械加速度传感器。这款加速度传感器具有丰富的嵌入式功能，带有灵活的用户可编程选项，可以配置多达两个中断引脚。嵌入式中断功能可以节省整体功耗，解除主处理器不断轮询数据的负担。MMA8452Q 具有 ±2g/±4g/±8g的用户可选量程，可以实时输出高通滤波数据和非滤波数据。该器件可被配置成利用任意组合可配置嵌入式的功能生成惯性唤醒中断信号，这就使MMA8452Q 在监控事件同时，在静止状态保持低功耗模式。

加速度传感器能够检测到所有6个方向的加速度值。在这个基础上，我们应用倾斜感应，能够得到更多的详细信息。物体倾斜是一种静态测量，重力作为输入计算的对象，可以确定物体的倾斜度。加速度传感器能够测量出从1g到-1g这180°范围内的倾斜度。我们可以很容易从三个轴的加速度算出物体的倾斜度。设备检测的方向不再变化的角度称为“Z-锁定角”。我们检测的各个角度都能精确到±2°

2.4气压传感器

我们实验使用的MPL3115A2型传感器属于智能数字压力传感器。它基于微机电系统（MEMS）技术，可以在本地处理压力和温度数据，减少了分配给应用处理器的计算量。因此，与使用由主机处理器直接管理的基本传感器的系统相比，这种压力传感器所消耗的功耗更少。该压力传感器采用FIFO（先进/先出）内存缓冲、2微安的待机模式和8微安的低功率模式，减少电流消耗，实现了最优效率，具体取决于处理器条件和所选的输出数据速率选择。

MPL3115A2压力传感器结合了高精度、高采样频率和超低功耗特性，进一步提高了性能。该器件提供了气压和高度压力检测，支持高达30cm的分辨率，可根据用户偏好使用米或帕斯

卡为单位输出数据。MPL3115A2传感器还包含嵌入式功能和用户可编程选项，比如温度补偿，采样频率可高达128Hz。

综合这些传感器采集到的数据，我们就可以通过软件平台的显示，对房间的综合信息进行实时监控。还可以设定某些信息的阈值，当测量的数据不在阈值范围内时，可以做出相应的报警指示，同学们可以据此做相应的扩展应用。

3. 物体控制

控制实体是智能家居必不可少的一项功能，对于智能家居系统中的每个实体，通过我们的软件平台都可以读取和设置相关的参数，还可以在软件平台上对单个物体进行简单的控制操作，例如控制电视和DVD音响的开关及音量调节，控制风扇的开关和转速等。本实验通过操作软件平台使得学生实现对家居物体的智能化控制。

电视，DVD和风扇在真实家居环境中，都是通过红外遥控器按键来实现控制的，在实验环境中，我们用学习型红外遥控器来代替平时使用的遥控器。在软件平台点击相应的按钮，就相当于按下遥控器的某个按键，从而控制物体进行相应的操作。接下来我们分别介绍红外遥控器工作原理以及学习型红外遥控器的使用方法。

3.1红外遥控器工作原理

红外遥控是目前使用最广泛的一种通信和遥控手段。由于红外线遥控装置具有体积小、功耗低、功能强、成本低等特点，因而，继彩电、录像机之后，在录音机、音响设备、空调以及玩具等其他小型电器装置上也纷纷采用红外线遥控。

通用红外遥控系统由发射和接收两大部分组成，应用编/解码专用集成电路芯片来进行控制操作，如图6-31所示，发射部分包括键盘矩阵、编码调制、LED红外发送器；接收部分包括光、电转换放大器、解调、解码电路。

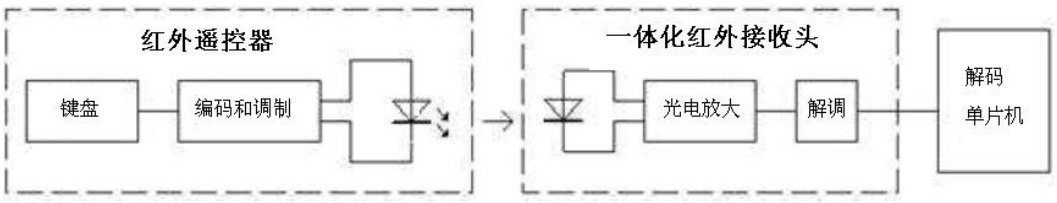


图6-31　红外线遥控系统框图

红外遥控的发射电路是采用红外发光二极管来发出经过调制的红外光波；红外接收电路由红外接收二极管、三极管或硅光电池组成，它们将红外发射器发射的红外光转换为相应的电信号，再送后置放大器。

发射机一般由指令键(或操作杆)、指令编码系统、调制电路、驱动电路、发射电路等几部分组成。当按下指令键或推动操作杆时，指令编码电路产生所需的指令编码信号，指令编码信号对载体进行调制，再由驱动电路进行功率放大后由发射电路向外发射经调制定指令编

码信号。

接收电路一般由接收电路、放大电路、调制电路、指令译码电路、驱动电路、执行电路(机构)等几部分组成。接收电路将发射器发出的已调制的编码指令信号接收下来，并进行放大后送解调电路，解调电路将已调制的指令编码信号解调出来，即还原为编码信号。指令译码器将编码指令信号进行译码，最后由驱动电路来驱动执行电路实现各种指令的操作控制（机构）。

3.2学习型红外遥控器

学习型遥控器也具有拷贝功能，可以拷贝任意一款固定码遥控器，学习型遥控器就相当于钥匙坯子，可以刻出任意形状的钥匙。只要将学习型遥控器出厂码清除，然后拷贝原遥控器，新配的遥控器就具有原遥控器的所有功能。

学习型红外遥控,可以分为两类：以固定码格式学习的遥控器和波形拷贝方式学习的遥控器。前者，需要收集各种不同种类的遥控器信号，然后进行识别比较，最后再记录。但是，要实现几乎所有的红外遥控器的成功复制就太难了。因为，红外遥控器的红外编码格式变化太多。不过这种学习型遥控器对硬件要求相对简单，处理器的工作频率可以不高，存储容量也较小，其缺点是对未知编码的遥控器无效。后者主要是把原始遥控器所发出的信号进行完全拷贝，而不管遥控器是什么格式，存储在EEPROM等存储器中。当发射时，只需将储存器中记录的波形长度还原成原始信号即可。这种学习型遥控器对MCU的主频要求高，RAM要求较大，其优点是对任何一种红外遥控器都可以进行学习。

4.实验过程：

计算机告知协调器，开始进行无线网络家庭组网。

协调器发起一次自组网网络，并持续监测是否有节点加入或离开本网络。网络容量为：1个ARM网关，1个协调，0-10个终端节点。

终端节点上电自动寻找并加入网络，黄色led灯不停闪烁表示寻找网络，红色led等亮表示已经加入到网络。

终端节点加入网络后，按照协调器的要求发送本地传感器信号和状态信号。

- 温度：模拟家庭温度计，实时监测室内温度值。
- 湿度：模拟家庭湿度计，实时监测室内湿度值。
- 压力：模拟家庭气压计，实时监测室内气压值。
- 关电：模拟家庭照度，判断室内光亮。
- 红外：模拟家庭遥控器。
- 步进电机：模拟窗帘打开关闭控制。
- 继电器：控制led灯，模拟开关。

计算机软件平台实时获得所有传感器上传的数据，并可模拟进行家庭物体的智能控制。

五、实验步骤

1.搭建智能家居环境，启动智能家居系统。

图6-32　智能家居软件

2.运行智能家居软件，获得实时监控的房间信息，并进行物体控制实验。

图6-33　智能家居房间监控

3.实验数据和实验过程，生成实验报告。

六、　拓展思考

1. 分析智能家居系统的特点，并思考还有哪些传感器可接入到该系统中。

2. 如何提高房间监控数据采集及显示的实时性。

第七章物联网综合实训

7.1 系统一 指纹考勤系统

一、实验目的

1.了解指纹提取以及识别原理。

2.掌握无线指纹考勤系统的工作原理并进行其他可扩展的应用。

二、实验设备

- 指纹考勤软件系统
- 无线传感网络
- 串口调试软件

三、实验内容

本实验通过指纹传感器模块进行指纹的提取以及识别，并通过无线传感网络在PC机显示出来。学生可以用串口调试软件得到指纹识别的码流信息，根据这些码流信息再做相应的数据分析和处理。

四、实验原理

1．指纹识别简介

指纹识别技术是依靠人体的特征来进行身份验证的生物识别技术。它的基本原理是通过取像设备读取指纹图像，然后用计算机识别软件建立指纹的特征资料，最后通过模糊匹配算法得到识别结果。要把人体的特征用于身份识别，这些特征必须具有唯一性和稳定性。研究和经验表明，人的指纹、掌纹、面孔、发音、视网膜、骨架等都具有唯一性和稳定性的特征，即每个人的这些特征都与别人不同、且终生不变，因此就可以根据此识别身份。基于这些特征，人们逐步对指纹识别、面部识别、发音识别等多种生物识别技术进行了探索和研究。目前，许多技术都已发展成熟并得以应用，其中，指纹识别技术更是生物识别技术的热点。

指纹识别技术的发展得益于现代电子集成制造技术和快速、可靠算法的研究。尽管指纹只是人体皮肤的一小部分，但用于识别的数据量相当大，对这些数据进行对比也不是简单的相等于与不相等的问题，而是需要使用进行大量运算的模糊匹配算法。现代电子集成制造技术使得我们可以制造相当小的指纹图像读取设备，同时，飞速发展的个人计算机运算速度提供了在微机、单片机上可以进行两个指纹比对运算的可能。另外，匹配算法可靠性也不断提

高，指纹识别技术已经非常实用。由于人体指纹的不变形和唯一性，指纹识别技术可以广泛应用于所有需要进行身份验证的场所，基于指纹识别技术的身份验证安全系统可以替代传统的基于密码和证件的安全系统。

2.指纹识别过程

指纹识别主要涉及四个步骤，如图 7-1 所示：

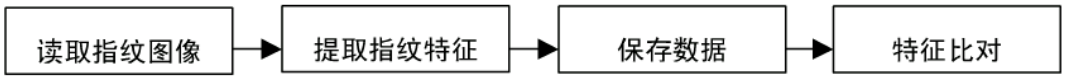


图7-1　　指纹识别步骤

通过指纹读取设备读取到人体指纹的图像，并对原始图像进行初步的处理，使其更清晰。获得的图像有很多噪音，这主要是由于平时的工作和环境所引起的，为了得到比较清晰的图像，要对读取到的指纹进行一个合理和匹配的滤镜，然后指纹识别软件建立指纹的数字表示--特征数据，一种单方向的转换，可以从指纹转换成特征数据，软件从指纹上找到被称为“节点”的数据点，也就是那些指纹纹路的分叉、终值或打圈出的坐标位置，这些点同时具有七种以上的唯一性特征。因为通常手指上平均具有 70 个节点，所以这种方法会产生大约 490 个数据。有的算法把节点和方向信息组合产生了更多的数据，这些方向信息表明了各个节点之间的关系，也有的算法还处理整幅指纹图像。总之，这些数据，通常称为模板，保存为 1K 大小的记录。最后运用计算机模糊比较的方法，对两个指纹的模板进行比较，计算出他们的相似程度，最终得到两个指纹的匹配结果。

3.验证和辨识

应用系统利用到的指纹识别技术可以分为两类：验证和辨识。验证就是通过把一个现场采集到的指纹与一个已经等级的指纹进行一对一的比对，来确认身份的过程。作为验证的前提条件，他或她的指纹必须在指纹库中已经注册。指纹以一定的压缩格式及加密算法存储，而不直接存储图像，并与其姓名或其表示联系起来。随后在比对现场，先验证其标识，然后利用系统的指纹与现场采集的指纹比对来证明其标识是合法的，这是应用系统中使用得较多的方法。

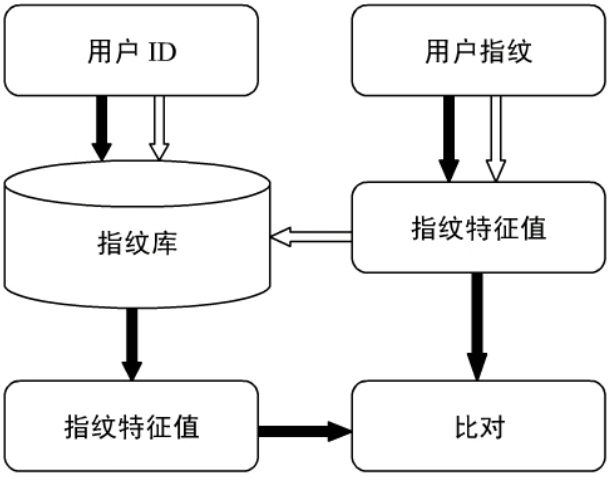


图7-2　　逻辑关系图

验证的逻辑关系如 7-2 所示。辨别则是把现场采集到的指纹同指纹数据库中的指纹逐一对比，从中找出与现场指纹相匹配的指纹。有效的指纹识别系统不仅仅依赖于识别算法，还有其他一些重要因素，这里称之为“系统问题”。包括注册和辨识过程，速度和工作学、用户信息的反馈、排斥欺骗和安全考虑等。

4.系统结构

无线指纹考勤系统通过指纹传感器采集考勤到的指纹，比对、识别后通过无线数据传输模块 ZigBee 将考勤记录数据及时、定时或相应地传送 PC 机，由安装在 PC 机上的考勤管理软件对考勤记录进行处理，实现数据的查询、统计、打印等功能。系统结构如图 7-3 所示：

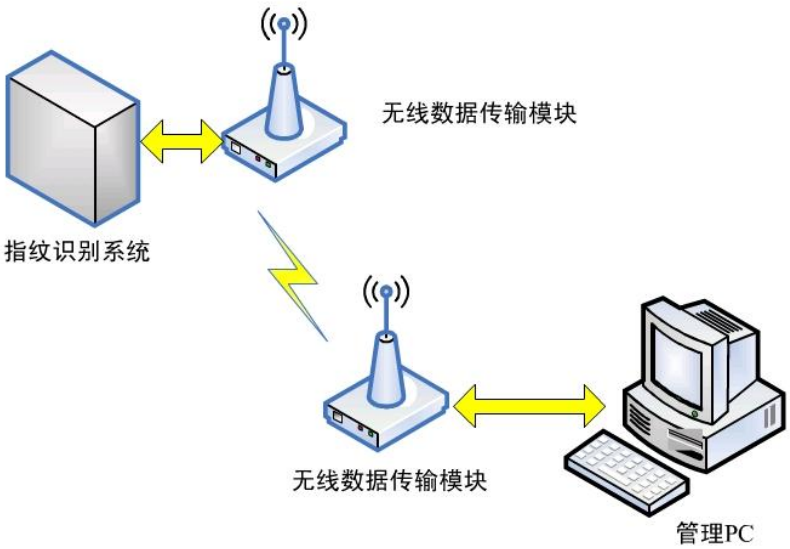


图7-3　　系统结构图

五、实验步骤

1.搭建指纹考勤系统环境。

2.录取指纹操作：对于系统中未保存的指纹，首先要进行录取指纹操作，将指纹的生物信息存于系统中。

3.读取指纹操作：将手指置于指纹仪上，指纹仪扫描指纹，观察上位机系统软件记录信息情况。

4.记录实验数据，生成实验报告。

六、拓展思考

1.分析影响指纹传感器模块工作效率的主要因素。

2.如何提供指纹识别的成功率。

7.2 系统二 基于三轴加速度的人体跌倒检测系统

一、实验目的

1.了解加速度传感器的工作原理。

2.掌握系统工作原理，测得实验数据并进行分析处理。

二、实验设备

- 人体跌倒检测软件系统
- 无线传感网络
- 串口调试软件

三、实验内容

本实验利用加速度传感器获得关于物体的相对位置信息，通过瞬间的位移变化情况得到相应的数据，进行分析处理，从而判断物体是否跌倒。学生可进行多次试验进行对比，掌握本系统的工作原理之后，可以根据测量得到的数据进行其他可扩展应用。

四、实验原理

1．加速度传感器

我们实验用的 MMA8452 加速度传感器是一款具有 12 位分辨率的智能低功耗、三轴、电容式微机械加速度传感器。这款加速度传感器具有丰富的嵌入式功能，带有灵活的用户可编程选项，可以配置多达两个中断引脚。嵌入式中断功能可以节省整体功耗，解除主处理器

不断轮询数据的负担。MMA8452Q 具有±2g/±4g/±8g 的用户可选量程，可以实时输出高通滤波数据和非滤波数据。该器件可被配置成利用任意组合，可配置嵌入式的功能生成惯性唤醒中断信号，这就使 MMA8452Q 在监控事件同时，在静止状态保持低功耗模式。MMA8452 加速度传感器的原理框图如下图 7-4 所示。

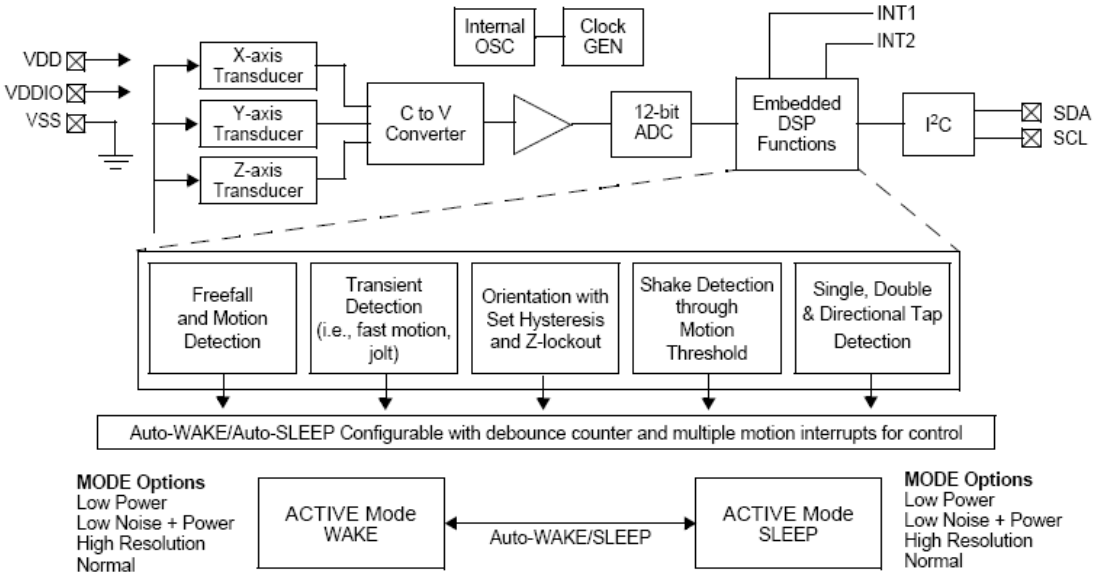


图7-4　　原理框图

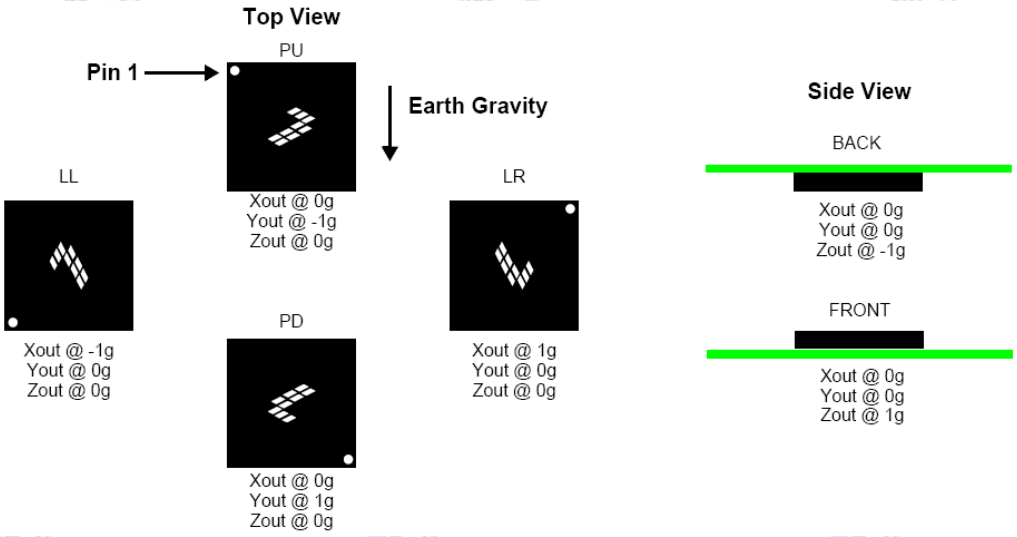


图7-5　　设备配置示意图

如上图显示了 6 个不同方向模式的设备配置，加速度传感器能够检测到所有 6 个方向的加速度值。在这个基础上，我们应用倾斜感应，能够得到更多的详细信息。物体倾斜是一种静态测量，重力作为输入计算的对象，可以确定物体的倾斜度。加速度传感器能够测量出从 1g 到-1g 这 180° 范围内的倾斜度。如下图 7-6 所示，我们可以很容易从三个轴的加速度算出物体的倾斜度。设备检测的方向不再变化的角度称为“Z-锁定角”。我们检测的各个角度都能精确到±2° 。

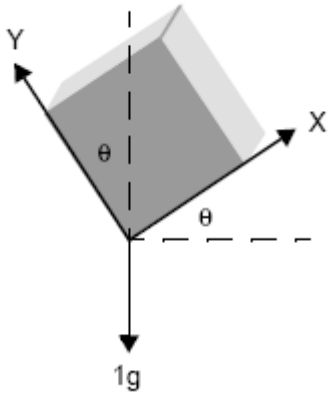


图7-6　　物体倾斜角度示意图

加速度传感器的应用示意图如图 7-7 所示：

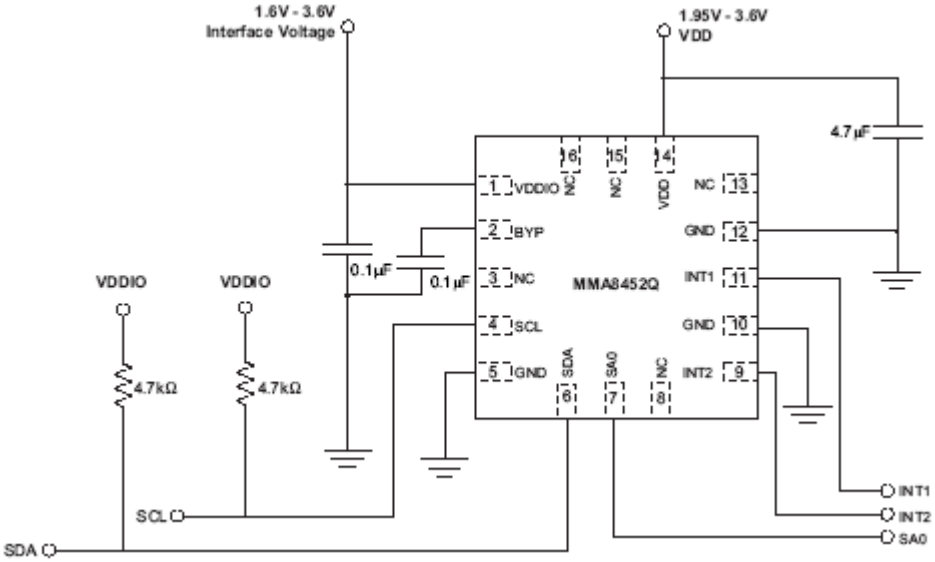


图7-7　　加速度传感器应用示意图

2.跌倒检测原理

人体发生跌倒是身体不自主失去平衡的行为，一般只发生在某一短暂的瞬间，在这段时间内，人体的重力、加速度和位移三种矢量均发生很大的变化，人体姿态也会发生相应变化，本系统即根据计算三轴加速度得到的人体姿态来判断跌倒情况。这种方法不但可以检测到跌倒，还可以判断出人体的具体姿态，如跌倒后是俯卧、仰卧或是侧卧等，使得判断结果更加准确具体。在三维空间，利用三轴加速度和重力之间的关系，可以得到桑格姿态角 pitch、roll 和 yaw 如下：

$$pitch = \arctan\left(a_x / \sqrt{a_y^2 + a_z^2}\right) \quad (1)$$

$$roll = \arctan\left(a_y / \sqrt{a_x^2 + a_z^2}\right) \quad (2)$$

$$yaw = \arctan\left(\sqrt{a_x^2 + a_y^2} / a_z\right) \quad (3)$$

式（1）~（3）中，a_x 是 X 轴方向测量到的加速度，a_y 是 Y 轴方向测量到的加速度，a_z 是 Z 轴方向测量到的加速度。Pitch 代表 Y 轴的旋转角，即人体向前向后的俯仰角；roll 代表绕 X 轴的旋转角，即人体向左向右的侧偏角；yaw 代表绕 Z 轴的旋转角，即人体向左向右的转角。

众所周知，人体在跌倒时大部分是前后或侧向跌倒，这两种跌倒将分别导致 pitch 和 roll 在短时间内发生大幅度抖动，而 yaw 没有明显区分，但是 yaw 可以在系统检测到跌倒后，辅助计算人体的静止姿势。首先利用采集到的加速度值计算三个姿态角，再分别判断 pitch 和 roll 是否超过 ADL 的正常范围，若超过则经行预报警，然后通过连续采集 1s 数据，判断姿态角超过正常范围的概率，若符合跌倒姿态，则进行报警，否则取消预报警。

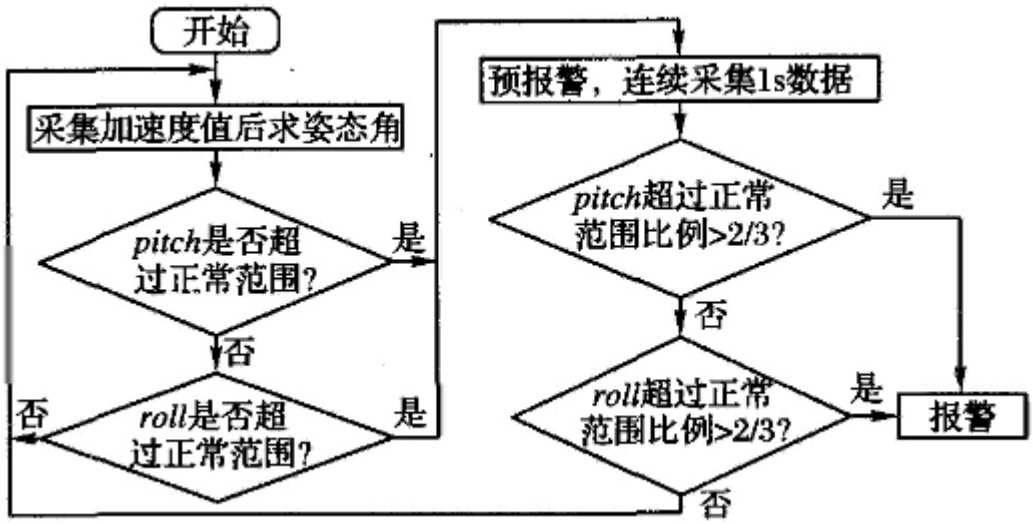


图7-8　　人体跌倒检测算法流程

3.系统架构

本系统主要包括两部分：一部分是人体跌到数据采集模块，包括一个三轴加速度传感器和一个无线发送模块，这部分可以由学生佩戴在身上，采集学生的动作信息；另一部分是网关，即无线接收模块，用来接收传感器的数据信息，网关利用网线与个人电脑相连，通过 socket 流将采集到的数据发送给 PC 进行分析和计算，检测是否发生跌倒。

五、实验步骤

1. 搭建人体跌倒检测系统环境。

2.实验测试：测试人处于不同运动情况时三个轴上的数据值。

3.记录实验数据，生成实验报告。

六、拓展思考

1.思考三轴加速度的创新与应用。

2.了解三轴加速度传感器在智能车路径识别中的应用。

7.3 系统三 楼宇集中温控节能系统

一、实验目的

1. 了解 ZigBee 无线传输网络机制。

2. 掌握系统工作原理，熟练使用软件操作。

二、实验设备

- 温控节能系统
- 无线传感网络
- 串口调试软件

三、实验内容

本实验所设计的楼宇集中温控节能系统，终端节点由CC2531芯片和数字温湿度传感器SHT10 构成，通过ZigBee 实现无线通信，数据经协调节点发送至上位机进行处理。该系统能实现温湿度数据采集和无线发送，并能在上位机显示。学生通过实验要了解ZigBee无线传输网络传输机制以及监测节点的硬件与软件设计原理。

四、实验原理

1．系统概述

系统由无线传感器网络和信息管理模块两部分组成。数据由无线传感器节点采集并发送给汇聚节点．汇聚节点接收运行状态数据并通过串口通讯将数据送给控制主机．每个工作区域有一个路由节点．负责处理和转发来自终端设备的温湿度及其光强信息。多个工作区域共有一个协调器节点。将来自路由器的各个监测区域的数据发往监控终端。主机显示运行状态并对其进行监测。该系统由远程管理中心和多个无线传感器节点组成。系统网络组成结构如

图7-9所示。放置在楼宇中的无线传感器节点以自组织的方式形成网络。采集的数据经多条传输方式传送到汇聚节点。数据最终由汇聚节点传送到管理中心。管理中心以图表、曲线等多种形式统计分析数据。

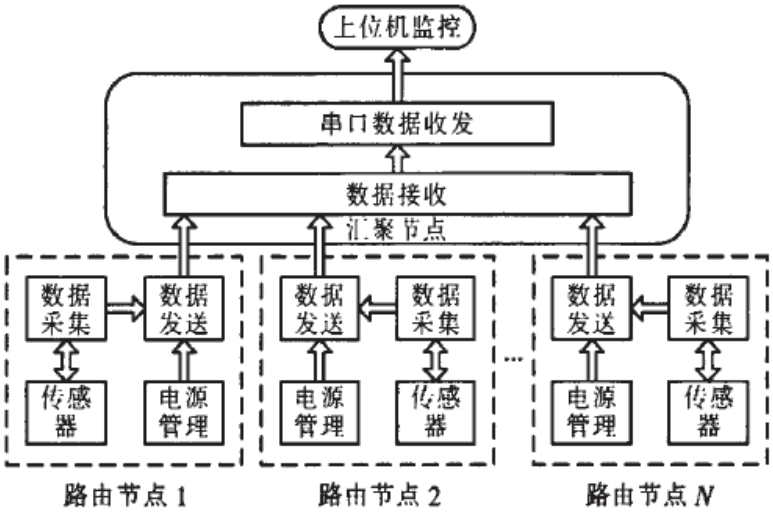


图7-9　　系统网络组成结构图

2.监测节点硬件设计

无线温湿度监测系统可分为协调节点和无线终端节点两大部分。每个节点都配CC2531芯片，CC2531是Chipcon公司(已在2006 年被美国德州仪器TI公司收购) 推出的用来实现嵌入式ZigBee应用的片上系统，它是世界上首个真正的单芯片ZigBee 解决方案，是世界上第一个真正意义上SoC。它支持2.4GHz IEEE 802.15.4 /ZigBee协议。协调节点的CC2531芯片经RS232串口线连接上位机，而无线终端节点通过CC2531的I/O口连接数字传感器采集信息。CC2531的应用电路原理图如图7-10所示。

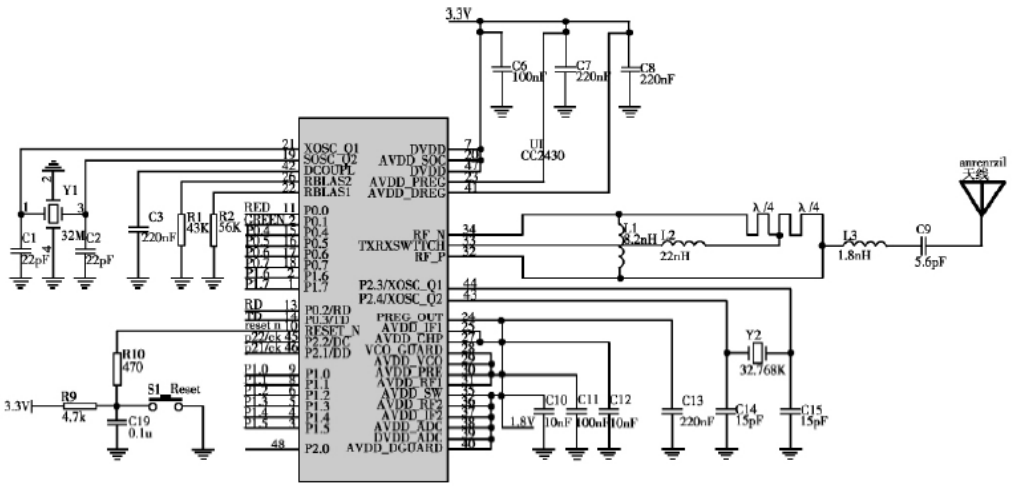


图7-10　　CC2531 的应用电路原理图

本系统的温湿度传感器采用集成温湿度传感器SHT10。封装格式采用DHT90，即将传感器器件SHT10和信号处理集成在一块微型电路板上，输出全标定的数字信号。传感器采用专利的CMOSens技术，确保产品具有极高的可靠性与卓越的长期稳定性。传感器包括一个电容性聚合体测湿敏感元件、一个用能隙材料制成的测温元件，并在同一芯片上与14 位的A/D转换器以及串行接口电路实现无缝连接。DHT90的供电电压范围为2.4-5.5V。DHT90 的串行接口，在传感器信号的读取及电源损耗方面，都做了优化处理；传感器不能按照I2C协议编址，但是，如果I2C总线上没有挂接别的元件，传感器基于ZigBee的智能家居温湿度监测系统设计可以连接到I2C总线上，但单片机必须按照传感器的协议工作。SCK 用于微处理器与DHT90之间的通讯同步，DATA 三态门用于数据的读取。

3.节点软件设计

无线温湿度检测系统的软件实现包括两部分：ZigBee 协调节点和温湿度传感器节点。ZigBee协调节点的主要作用是组建一个网络、接受终端节点入网和发送来的数据，并通过串口发送数据至PC机。首先对协调节点进行初始化（包括处理器、协议栈、中断、串口等）；之后新建一个网络并进入网络监听和等待状态，当收到子节点的入网请求后，协调节点先随机为子节点分配一个网络地址，然后向子节点发送入网确认信息，建立连接后等待接收终端节点发送温湿度数据; 数据接收成功后发送数据到PC机。

温湿度传感器节点作为终端节点，负责采集、处理和发送数据。首先温湿度传感器SHT10 用一组启动传输时序进行数据传输的初始化，然后发送一组测量命令(‘00000101’表示相对湿度，‘00000011’表示摄氏温度) ，释放DATA线，等SHT10下拉DATA至低电平，表示测量结束，同时接收数据，完成初始化过程；之后节点发送入网请求，加入网络成功后进入空闲状态待定时时间到进行数据采集并向其协调节点发送，如果发送失败继续尝试，直到发送成功为止。软件流程图如图7-11所示，图7-11(a)为ZigBee协调节点的软件流程图，图7-11(b)为温湿度传感器节点的软件流程图。

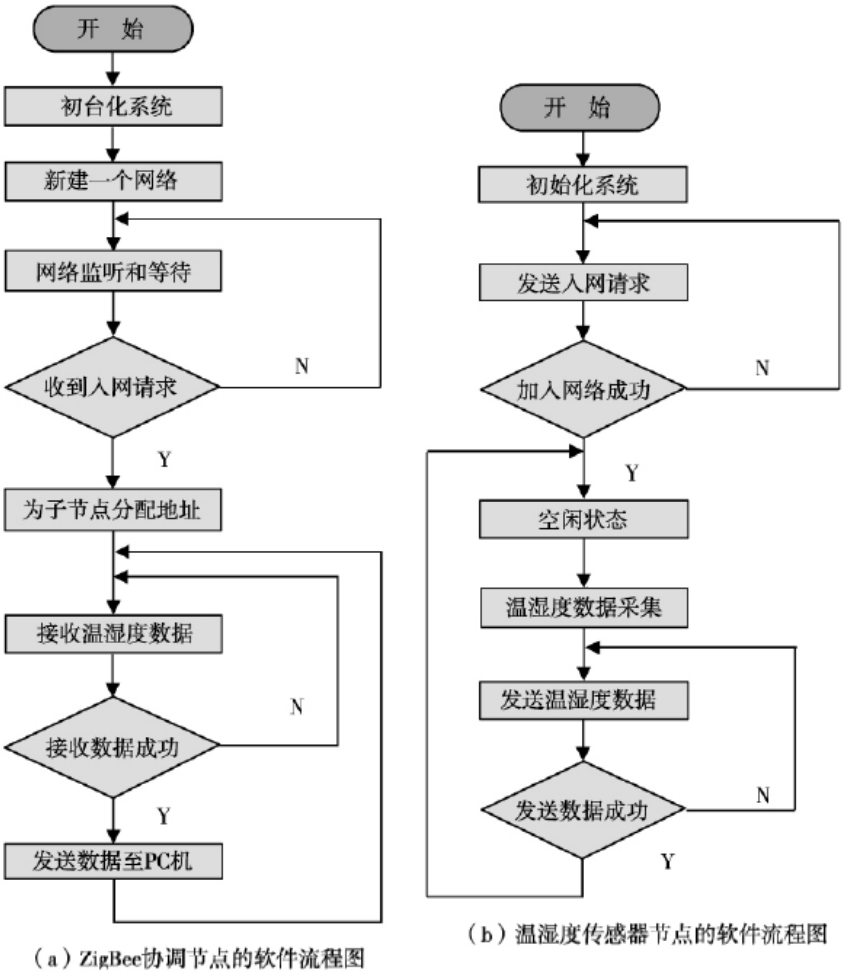


图7-11　　软件流程图

五、实验步骤

1. 搭建楼宇集中温控节能系统环境。

2. 实验测量：实时监控不同节点获取的数据值，改变室内温度，观察上位机系统提示的操作信息。

3. 记录实验数据，生成实验报告。

六、拓展思考

1. 总结温湿度传感器的工作原理。

2. 分析温湿度传感器在智慧农业中的应用。

7.4 系统四 无线智能家居照明系统

一、实验目的

1. 了解 ZigBee 无线传输网络机制。

2. 掌握系统工作原理，熟练使用软件操作。

二、实验设备

- 智能家居照明软件系统
- 无线传感网络
- 串口调试软件

三、实验内容

本实验系统可以完成对各室内温湿度光强数据的采集、传输、存储、分析，具有低功耗、自组织特点。管理节点利用ZigBee网络，根据管理中心的要求发送无线传感器网络各节点温湿度光强数据及报警信息。学生在采集到相应数据之后可做其他可扩展应用。

四、实验原理

1.系统总体方案

本系统设计主要用于中小型家居环境中，结构相对简单，采用星形网络拓扑结构。整个系统由无线网关、智能开关和无线摇控器组成。系统整体结构如图7-12所示。

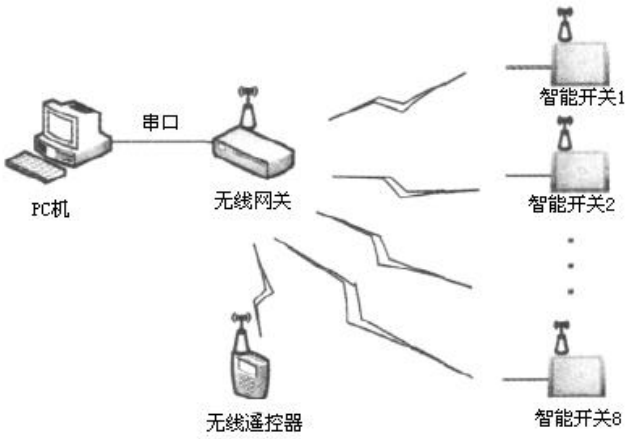


图7-12　　系统结构图

本系统采用ARM11作为无线网关的控制芯片，CC2531作为整个无线网络通讯的接口芯片。所有照明设备发出的指令要通过无线网关接收，经过处理后转发到相应的智能开关终端设备中。智能开关设备接收到指令后，就会按照指令进行开关或调光。遥控器为网络中所需要的无线手持设备，方便用户随意操纵照明设备。下面分述智能开关单元、无线网关单元和无线摇控单元三大系统硬件模块单元的设计。

2.人体感应电路设计

本设计中人体感应模块采用RE200B热释电人体感应传感器，它感应的红外波长和人体发射的红外波长接近，以非接触形式检测出人体辐射的红外线能量的变化，并将其转换结果以电压信号输出。人体感应电路如图7-13所示。U2为热释电人体感应传感器RE200B，当人体接近时，在2端感应出电压，经过U3 运算放大器放大后送给CC2531中的AD 转换模块，CC2531根据转化结果计算出人体位置，并将结果上传至CC2531的微控制模块8051，其中R7来调节电路放大倍数，即调节智能开关的灵敏度。

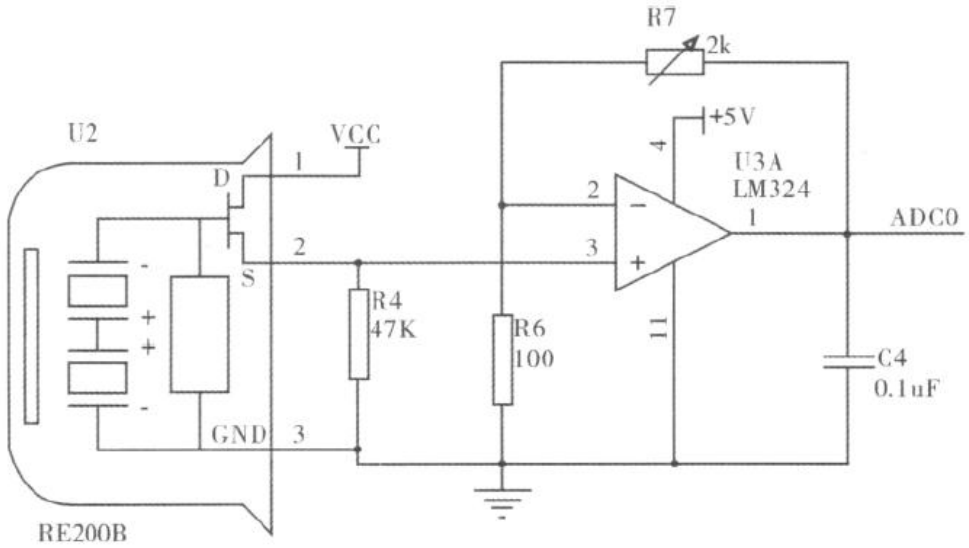


图7-13　　人体感应电路图

五、实验步骤

1. 搭建智能家居照明环境。

2. 通过遥控器控制智能开关状态，观察实验现象。

3. 记录实验数据，生成实验报告。

六、拓展思考

1. 总结人体感应模块的工作原理。

2，思考人体感应模块的扩展应用。

7.5 系统五 3G 远程监控实训系统

一、实验目的

1. 了解 3G 网络技术。

2. 掌握远程监控系统工作原理并作相应的可扩展应用。

二、实验设备

- 监控软件系统
- 3G 模块
- 串口调试软件

三、实验内容

本实验基于3G的手机远程监控系统，3G手机不仅可以用作采集实时数据信息，然后传送到中心平台进行处理，还能作为客户端，作为实时监控设备使用。学生要了解3G网络以及远程监控系统工作原理。

四、实验原理

1.3G 简介

3G是把无线通信技术和互联网等多媒体技术结合起来的一项全新的技术，3G 技术的改进主要是在传输图像和声音或视频等信息数据的速度上有了很大的提高，它同样能够处理图像和视频等多种媒体形式，不但能够提供第二代通信（2G）所能提供的所有信息服务并且在此基础上有了很大的改进，比如在传输速度上就有了很大的提高。受网络带宽、手机设备和终端设备的配置等因素的制约和影响，在2G 时代，大部分的手机使用者使用的网上功能都只是一些简单的应用，比如下载一些图片或音乐、上网浏览简单的新闻页面等等。在当前3G 时代，由于网速的大大提高，手机终端可以提供更多更加高级的应用或服务，如可以通过手机看电视，可以浏览更多的网上信息，还可以进行手机视频通话等等。

2. 远程监控系统

远程监控系统总体结构图如图7-14所示。主要包括前端采集模块，数据处理模块，无线传输模块，终端监控模块。

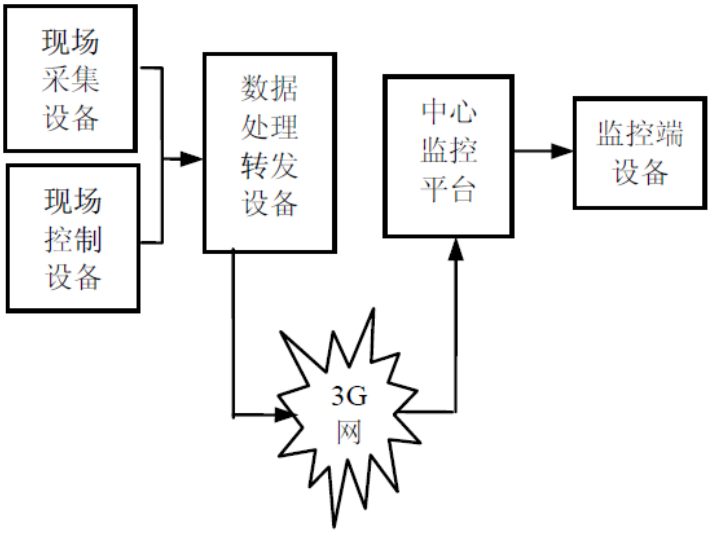


图7-14　　远程监控系统

（1）前端采集模块

前端采集模块实现的主要功能是从要监控的现场采集信息主要是视频信息，该模块中另外一部分是进行控制，这涉及到另外的控制设备，此模块支持不同的压缩比的图像大小。

（2）数据处理转发模块

该模块的功能是将前端设备采集来的信息进行相应地处理（如将信息转码处理），并储存处理后的信息，待终端需要时再进行传送。

（3）无线传输模块

视频采集压缩以后就可将压缩好的视频文件等传送到中心平台，传送的方式这里是通过3G网络。

（4）监控中心

监控中心的设备可以是普通的电脑或者域名服务器同样也可以使用具有相应功能的3G手机。待中心服务器需要时开启，随时等待各个监控终端的连接请求，对于始终在线模式的终端，可以按照需要调取查询当前的图像、视频信息，也可以自动实时地记录监控点的变化，以便实时监测。

（5）终端模块

该模块的主要功能是查看监控，如果监控现场的某些实物不需要处理，则该模块只实现监测查看现场的功能。

3. 3G 手机监控系统的体系结构

根据不同的应用需求，3G手机可作为视频或音频采集、图像获取等各种功能的采集终端或客户终端设备，在结合其它前端和客户端设备并由中心平台配置不同的功能后，便可达到

不同的应用目的。整体上基于3G 移动网络的各种应用均可在图7-15所示的体系结构下实现。

图7-15　　基于 3G 网络的远程监控系统

从上面的体系结构图可以看到，一个完整的手机远程监控系统从整体上可以简要的划分为三部分：前端，中心平台，客户端。 由于3G手机具有信息采集、视频浏览的功能并且具有可移动的特性，因此在整个体系结构中3G手机既能够作为前端设备，用于实时的图片拍摄和视频采集或是信息报警监测等，也可作为监控客户端进行实时视频浏览、报警查看、控制结果分析或者查看前端设备功能配置等等。

五、实验步骤

1. 搭建3G网络及中心平台环境。

2. 在中心平台中配置各前端、客户端功能。

3. 操作在手机终端，进行视频浏览、报警信息查看等。

4. 记录实验数据，生成实验报告。

六、拓展思考

1. 分析利用 3G 网络技术进行远程监控的优点。

2. 总结移动终端在远程监控系统中的作用

附录：缩略词

英文缩写	英文全称	汉语解释
IOT	Internet of Things	物联网
RFID	Radio Frequency Identification	射频识别
M2M	Machin to machin	机器与机器的对话
OSI	Open System interconnect	开放式系统互联
PHY	Physical	物理层，OSI 的最底层
MAC	Medium Access Control	媒体介入控制层，属于 OSI 模型中数据链路层下层子层
NWK	Network	网络层，
APL	Application	应用层
ZDO	ZigBee device object	ZigBee 设备对象
PAN	Personal area network	个人局域网络
LLC	Logical link control	逻辑链路控制
AODV	Ad hoc on-demand distance vector routing	源驱动路由协议
CDMA	Code Division Multiple Access	码分多址