目 录

第一章 嵌入式系统

嵌入式系统（Embedded System）在于结合微处理器或为控制器的系统电路与其专用的软件，来达到系统运作效率成本的最优化。今天凡移动电话，手表，电视机，电子游戏机、PDA、冰箱等家用电子和通信产品乃至于电动交通工具的控制核心无不与嵌入式系统息息相关。而在后 PC 时代，家电、玩具、汽车、新一代手机、数码产品、先进的医疗设备乃至于即将到来的智能型房屋、智能型办公室、与其它跟电相关的器材设备更是缺少不了嵌入式系统的这个核心技术。

1.1 嵌入式系统的概念

根据 IEEE 的定义，嵌入式系统是“控制、监视或者辅助装置、机器和设备运行的装置”。这主要是从应用上加以定义的，从中可以看出嵌入式系统是软件和硬件的综合体。不过上述定义并不能充分体现出嵌入式系统的精髓，目前国内一个普遍被认同的定义是：以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。简单地说，嵌入式系统集系统的应用软件与硬件于一体，类似于 PC 中 BIOS 的工作方式，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。嵌入式系统主要由嵌入式处理器、相关支撑硬件、嵌入式操作系统及应用软件系统等组成，它是可独立工作的“器件”。

在明确了嵌入式系统定义基础上，我们可从以下几方面来理解嵌入式系统：

1. 嵌入式系统是面向用户、面向产品、面向应用的，嵌入式系统是与应用紧密结合的，它具有很强的专用性，必须结合实际系统需求进行合理的裁减利用。嵌入式系统和具体应用有机地结合在一起，它的升级换代也是和具体产品同步进行，因此嵌入式系统产品一旦进入市场，具有较长的生命周期。

2. 嵌入式系统是将先进的计算机技术、半导体技术和电子技术和各个行业的具体应用相结合后的产物。这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

3. 嵌入式系统必须根据应用需求对软硬件进行裁剪，满足应用系统的功能、可靠性、成本、体积等要求。为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，而不是存贮于磁盘等载体中。

4. 嵌入式系统本身不具备自主开发能力，即使设计完成以后用户通常也是不能对其中的程序功能进行修改的，必须有一套开发工具和环境才能进行开发。

实际上，凡是与产品结合在一起的具有嵌入式特点的控制系统都可以叫嵌入式系统。现在人们讲嵌入式系统时，某种程度上指近些年比较热的具有操作系统的嵌入式系统。

1.2 嵌入式系统的组成

嵌入式系统是计算机软件和硬件的综合体，可涵盖机械或其他的附属装置。所以嵌入式系统可以笼统地分为硬件和软件两部分。嵌入式系统的构架可以分成四个部分：处理器、存储器、输入输出（I/O）和软件（由于多数嵌入式设备的应用软件和操作系统都是紧密结合的，在这里我们对其不加区分，这也是嵌入式系统和通用 PC 系统的最大区别）。如图 1.1.

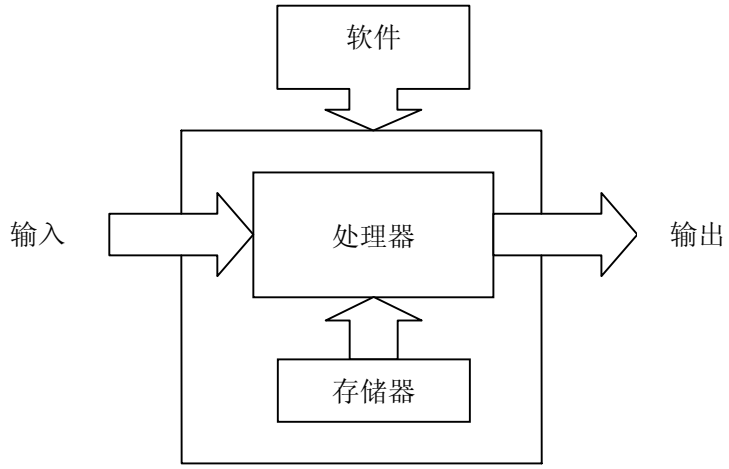


图 1.1 嵌入式系统组成

嵌入式系统的硬件部分，包括处理器/微处理器、存储器及外设器件和 I/O 端口、图形控制器等。嵌入式系统有别于一般的计算机处理系统，它不具备像硬盘那样大容量的存储介质，而大多使用 EPROM、EEPROM 或闪存（Flash Memory）作为存储介质。

从硬件方面来讲，各式各样的嵌入式处理器是嵌入式系统硬件中的最核心的部分。目前嵌入式处理器的寻址空间可以从 64kB 到 16MB，处理速度最快可以达到 2000MIPS，封装从 8 个引脚到 144 个引脚不等。

根据其现状，嵌入式处理器可以分成下面几类：

1．嵌入式微处理器（Embedded Microprocessor Unit，EMPU）

嵌入式微处理器采用“增强型”通用微处理器。根据实际要求，将嵌入式微处理器装配在专门设计的主板上，只保留和嵌入式应用有关的主板功能。由嵌入式微处理器及其存储器、总线、外设等安装在一块电路主板上构成一个通常所说的单板机系统。

2．嵌入式微控制器（Microcontroller Unit，MCU）

嵌入式微控制器又称单片机，它将整个计算机系统集成到一块芯片中。嵌入式微控制器一般以某种微处理器内核为核心，根据某些典型的应用，在芯片内部集成了 ROM/EPROM、RAM、总线、总线逻辑、定时/计数器、看门狗、I/O、串行口、脉宽调制输出、A/D、D/A、Flash RAM、EEPROM 等各种必要功能部件和外设。为适应不同的应用需求，对功能的设置和外设的配置进行必要的修改和裁减定制。

3．嵌入式 DSP 处理器（Embedded Digital Signal Processor，EDSP）

DSP 处理器是专门用于信号处理方面的处理器，其在系统结构和指令算法方面进行了特殊设计，具有很高的编译效率和指令的执行速度。在数字滤波、FFT、谱分析等各种仪器上 DSP 获得了大规模的应用。

4．嵌入式片上系统（System On Chip，SOC）

SOC 追求产品系统最大包容的集成器件，SOC 最大的特点是成功实现了软硬件无缝结合，直接在处理器片内嵌入操作系统的代码模块。绝大部分系统构件都是在系统内部，减小了系统的体积和功耗，提高了系统的可靠性和设计生产效率。

嵌入式处理器与通用型处理器最大的不同是嵌入式处理器大多工作在为特定用户群设计的系统中，它通常都具有功耗低、体积小、集成度高等特点，能够把通用处理器中许多由板卡完成的任务集成在芯片内部，从而有利于嵌入式系统设计趋于小型化，移动能力大大增强，和网络的耦合也越来越紧密。

嵌入式系统的软件部分包括操作系统和应用软件。应用软件决定了系统的运作和行为；而操作系统控制着应用软件与系统硬件的交互。多数嵌入式设备的应用软件和操作系统都是紧密结合的，这也是嵌入式系统和通用 PC 系统的主要区别之一。

1.3 嵌入式系统的发展

嵌入式系统的出现至今已经有 30 多年的历史了，嵌入式技术也历经了几个发展阶段。进入 90 年代后，以计算机和软件为核心的数字化技术取得了迅猛发展，不仅广泛渗透到社会经济、军事、交通、通信等相关行业，而且深入到家电、娱乐、艺术、社会文化等各个领域，掀起了一场数字化技术革命。多媒体技术与 Internet 的应用迅速普及，消费电子（Consumptive electron），计算机，通信（Communication），3C 一体化趋势日趋明显，嵌入式技术再度成为一个研究热点。综观嵌入式技术的发展，大致经历了 4 个阶段：

1．第一阶段是以单芯片为核心的可编程控制器形式的系统，同时具有与监测、伺服、指示设备相配合的功能。这种系统一般没有操作系统的支持，通过汇编语言编程对系统进行直接控制。这一阶段系统的主要特点是：系统结构和功能都相对单一，处理效率较低，存储容量较小，几乎没有用户接口。

2．第二阶段是以嵌入式 CPU 为基础、以简单操作系统为核心的嵌入式系统。这一阶段系统的主要特点是：CPU 种类繁多，通用性比较弱；系统开销小，效率高；操作系统具有一定的兼容性和扩展性；应用软件较专业，用户界面不够友好；系统主要用来控制系统负载以及监控应用程序运行。

3．第三阶段是以嵌入式操作系统为标志的嵌入式系统。这一阶段系统的主要特点是：嵌入式操作系统能运行于各种不同类型的微处理器上，兼容性好；操作系统内核精小、效率高，并且具有高度的模块化和扩展性；具备文件和目录管理、设备支持、多任务、网络支持、图形窗口以及用户界面等功能；具有大量的应用程序接口（API），开发应用程序简单；嵌入式应用软件丰富。

4．第四阶段是以基于 Internet 为标志的嵌入式系统，这是一个正在迅速发展的阶段。目前大多数嵌入式系统还孤立于 Internet 之外，但随着 Internet 的发展以及 Internet 技术与信息家电、工业控制技术等结合日益密切，嵌入式设备与 Internet 的结合将代表着嵌入式技术的真正未来。

早期的嵌入式系统只是为了实现某些特定功能，使用一个循环控制对外界的请求进行处理。不可否认，这对于简单的系统而言是足够的，但是当我们的系统变得日渐复杂庞大的时候，如果我要增添一项功能的时候，很可能不得不重新进行系统的设计，这无疑会增加开发的成本和系统复杂度。使用这种方式开发规模较大、功能复杂的嵌入式系统是不可想象的。20 世纪 80 年代初期嵌入式操作系统的出现使得快速便捷地开发规模较大的嵌入式系统成为现实，自这一时期开始就出现了各种各样的商用嵌入式操作系统，从而形成了目前多种形式的商用嵌入式操作系统百家争鸣的局面。基于嵌入式操作系统之上的系统才能够真正符合嵌入式系统定义中的软件可裁剪、功能的可扩展、高可靠性等特征。区别于

嵌入式系统，我们将不使用嵌入式操作系统的基于 MCU（微控制器）、MPU（微处理器）和 DSP 的循环控制系统称之为前后台系统。

1.4 嵌入式系统的应用前景

后 PC 时代是一个真实的阶段，而且是一个可以预测的时代。嵌入式系统就是与这一时代紧密相关的产物，它将拉近人与计算机的距离，形成一个人机和谐的工作与生活环境。从某一个角度来看，嵌入式系统可应用于人类工作与生活的各个领域，具有极其广阔的应用前景。

嵌入式系统在应用数量上远远超过了各种通用计算机系统，一台通用计算机的外部设备中就包含了 5-10 个微处理器，键盘、鼠标、软驱、硬盘、显示卡、显示器、Modem、网卡、声卡、打印机、扫描仪、数字相机、USB 集线器等均是由嵌入式系统控制的。在制造工业、过程控制、通讯、仪器、仪表、汽车、船舶、航空、航天、军事装备、消费类产品等方面均是嵌入式计算机的应用领域。嵌入式系统是将先进的计算机技术、半导体技术和电子技术和各个行业的具体应用相结合后的产物，这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

纵观嵌入式系统的发展过程，可以看出嵌入式系统由简单的无操作系统的循环控制程序向具有强大功能的基于操作系统的方向发展，从独立的系统向基于网络的系统发展。进入 90 年代后，以计算机和软件为核心的数字化技术取得了迅猛发展，不仅广泛渗透到社会经济、军事、交通、通信等相关行业，而且深入到家电、娱乐、艺术、社会文化等各个领域，掀起了一场数字化技术革命。多媒体技术与 Internet 的应用迅速普及，消费电子、计算机、通信（3C）一体化趋势日趋明显，嵌入式技术已再次成为一个研究热点。

未来几年，个人计算机将会延伸到各种大小型智能设备中，各种信息借助于网络形成连接为一个整体。要达到这样的目标非嵌入式系统不可，同时这也是我们对嵌入式系统的展望。

图 1.1 嵌入式系统应用

第二章 ARM9 体系结构

2.1 ARM 简介

2.1.1 ARM

ARM 的全称是 Advanced RISC Machine，这个公司不生产芯片也不销售芯片，它只是出售芯片技术授权。ARM 技术 IP 核的微处理器遍及汽车、消费电子、成像、工业控制、海量存储、网络、安保和无线等各类产品市场。目前世界上前 5 大半导体公司全部使用了 ARM 的技术授权，而前 10 大半导体公司有 9 家，前 25 大半导体公司中有 23 家采用了 ARM 的技术授权。它借助于拥有的强大技术优势，在世界范围内与许多业界领先的半导体厂商和芯片设计中心、实时操作系统软件开发商、应用软件公司、电子设计自动化（EDA）工具供应商和系统公司建立合作关系，其中包括世界顶级的厂商 Intel、NS、TI、Apple、Motorola、Mitsubishi、SUN、Lucent、Sanyo、Altera、Triscend、Agilent、Toshiba、Fujitsu、Sharp、ST、3COM、OKI、YAMAHA、Atmel、Rohm、Rockwell、Philips、Lucent、Samsung、Hyundai、Sony 和 Alcatel 等 56 家公司。由此可见 ARM 技术具有不可抗拒的魅力，1999 年采用 ARM IP 核的芯片出货量就达 1.8 亿片，占 RISC 市场份额 57.8%；2000 年采用 ARM IP 核的芯片出货量更是增长到 4 亿片，RISC 市场份额上升到 74.3%，同期 MIPS 芯片占 11.3%，Power PC 芯片占 3.4%。

到目前为止，ARM 微处理器及技术的应用几乎已经深入到各个领域：

1．工业控制领域：作为 32 的 RISC 架构，基于 ARM 核的微控制器芯片不但占据了高端微控制器市场的大部分市场份额，同时也逐渐向低端微控制器应用领域扩展，ARM 微控制器的低功耗、高性价比，向传统的 8 位/16 位微控制器提出了挑战。

2．无线通讯领域：目前已有超过 85%的无线通讯设备采用了 ARM 技术， ARM 以其高性能和低成本，在该领域的地位日益巩固。

3．网络应用：随着宽带技术的推广，采用 ARM 技术的 ADSL 芯片正逐步获得竞争优势。此外，ARM 在语音及视频处理上行了优化，并获得广泛支持，也对 DSP 的应用领域提出了挑战。

4．消费类电子产品：ARM 技术在目前流行的数字音频播放器、数字机顶盒和游戏机中得到广泛采用。

5．成像和安全产品：现在流行的数码相机和打印机中绝大部分采用 ARM 技术。手机中的 32 位 SIM 智能卡也采用了 ARM 技术。

除此以外，ARM 微处理器及技术还应用到许多不同的领域，并会在将来取得更加广泛的应用。

ARM 微处理器采用 RISC 架构，一般具有如下特点：

- ✧ 体积小、低功耗、低成本、高性能；
- ✧ 支持 Thumb（16 位）/ARM（32 位）双指令集，能很好的兼容 8 位/16 位器件；
- ✧ 大量使用寄存器，指令执行速度更快；
- ✧ 大多数数据操作都在寄存器中完成；
- ✧ 寻址方式灵活简单，执行效率高；
- ✧ 指令长度固定。

2.1.2 ARM 的内核简介

ARM 内核结构芯片具有 RISC 体系的一般特点：

- ✧ 具有大量寄存器
- ✧ 绝大多数操作都是在寄存器中进行，寄存器与内存打交道的唯一是通过 Load/Store 的体系结构在内存和寄存器之间传递数据
- ✧ 寻址方式简单
- ✧ 采用固定长度的指令

ARM 微处理器目前包括下面几个系列，以及其它厂商基于 ARM 体系结构的处理器，除了具有 ARM 体系结构的共同特点以外，每一个系列的 ARM 微处理器都有各自的特点和应用领域。

- ✧ ARM7 系列
- ✧ ARM9 系列
- ✧ ARM9E 系列
- ✧ ARM10E 系列
- ✧ SecurCore 系列
- ✧ Inter 的 Xscale
- ✧ Inter 的 StrongARM

其中，ARM7、ARM9、ARM9E 和 ARM10 为 4 个通用处理器系列，每一个系列提供一套相对独特的性能来满足不同应用领域的需求。SecurCore 系列专门为安全要求较高的应用而设计。

以下我们来详细了解一下各种处理器的特点及应用领域。

1．ARM7 微处理器系列

ARM7 系列微处理器为低功耗的 32 位 RISC 处理器，最适合用于对价位和功耗要求较高的消费类应用。ARM7 微处理器系列具有如下特点：

- ✧ 具有嵌入式 ICE－RT 逻辑，调试开发方便。
- ✧ 极低的功耗，适合对功耗要求较高的应用，如便携式产品。
- ✧ 能够提供 0.9MIPS/MHz 的三级流水线结构。
- ✧ 代码密度高并兼容 16 位的 Thumb 指令集。
- ✧ 对操作系统的支持广泛，包括 Windows CE、Linux、Palm OS 等。
- ✧ 指令系统与 ARM9 系列、ARM9E 系列和 ARM10E 系列兼容，便于用户的产品升级换代。
- ✧ 主频最高可达 130MIPS，高速的运算处理能力能胜任绝大多数的复杂应用。

ARM7 系列微处理器的主要应用领域为：工业控制、Internet 设备、网络和调制解调器设备、移动电话等多种多媒体和嵌入式应用。ARM7 系列微处理器包括如下几种类型的核：ARM7TDMI、ARM7TDMI-S、ARM720T、ARM7EJ。其中，ARM7TMDI 是目前使用最广泛的 32 位嵌入式 RISC 处理器，属低端 ARM 处理器核。

2．ARM9 微处理器系列

ARM9 系列微处理器在高性能和低功耗特性方面提供最佳的性能。具有以下特点：

- ✧ 5 级整数流水线，指令执行效率更高。
- ✧ 提供 1.1MIPS/MHz 的哈佛结构。
- ✧ 支持 32 位 ARM 指令集和 16 位 Thumb 指令集。
- ✧ 支持 32 位的高速 AMBA 总线接口。

- ✧ 全性能的 MMU，支持 Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统。MPU 支持实时操作系统。
- ✧ 支持数据 Cache 和指令 Cache，具有更高的指令和数据处理能力。

ARM9 系列微处理器主要应用于无线设备、仪器仪表、安全系统、机顶盒、高端打印机、数字照相机和数字摄像机等。ARM9 系列微处理器包含 ARM920T、ARM922T 和 ARM940T 三种类型，以适用于不同的应用场合。

3．ARM9E 微处理器系列

ARM9E 系列微处理器为可综合处理器，使用单一的处理器内核提供了微控制器、DSP、Java 应用系统的解决方案，极大的减少了芯片的面积和系统的复杂程度。ARM9E 系列微处理器提供了增强的 DSP 处理能力，很适合于那些需要同时使用 DSP 和微控制器的应用场合。ARM9E 系列微处理器的主要特点如下：

- ✧ 支持 DSP 指令集，适合于需要高速数字信号处理的场合。
- ✧ 5 级整数流水线，指令执行效率更高。
- ✧ 支持 32 位 ARM 指令集和 16 位 Thumb 指令集。
- ✧ 支持 32 位的高速 AMBA 总线接口。
- ✧ 支持 VFP9 浮点处理协处理器。
- ✧ 全性能的 MMU，支持 Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统。
- ✧ MPU 支持实时操作系统。
- ✧ 支持数据 Cache 和指令 Cache，具有更高的指令和数据处理能力。
- ✧ 主频最高可达 300MIPS。

ARM9 系列微处理器主要应用于下一代无线设备、数字消费品、成像设备、工业控制、存储设备和网络设备等领域。ARM9E 系列微处理器包含 ARM926EJ-S、ARM946E-S 和 ARM966E-S 三种类型，以适用于不同的应用场合。

4．ARM10E 微处理器系列

ARM10E 系列微处理器具有高性能、低功耗的特点，由于采用了新的体系结构，与同等的 ARM9 器件相比较，在同样的时钟频率下，性能提高了近 50%，同时，ARM10E 系列微处理器采用了两种先进的节能方式，使其功耗极低。ARM10E 系列微处理器的主要特点如下：

- ✧ 支持 DSP 指令集，适合于需要高速数字信号处理的场合。
- ✧ 6 级整数流水线，指令执行效率更高。
- ✧ 支持 32 位 ARM 指令集和 16 位 Thumb 指令集。
- ✧ 支持 32 位的高速 AMBA 总线接口。
- ✧ 支持 VFP10 浮点处理协处理器。
- ✧ 全性能的 MMU，支持 Windows CE、Linux、Palm OS 等主流嵌入式操作系统。
- ✧ 支持数据 Cache 和指令 Cache，具有更高的指令和数据处理能力。
- ✧ 主频最高可达 400MIPS。
- ✧ 内嵌并行读/写操作部件。

ARM10E 系列微处理器主要应用于下一代无线设备、数字消费品、成像设备、工业控制、通信和信息系统等领域。ARM10E 系列微处理器包含 ARM1020E、ARM1022E 和 ARM1026EJ-S 三种类型，以适用于不同的应用场合。

5．SecurCore 微处理器系列

SecurCore 系列微处理器专为安全需要而设计，提供了完善的 32 位 RISC 技术的安全解决方案，因此，SecurCore 系列微处理器除了具有 ARM 体系结构的低功耗、高性能的特

点外，还具有其独特的优势，即提供了对安全解决方案的支持。SecurCore 系列微处理器除了具有 ARM 体系结构各种主要特点外，还在系统安全方面具有如下的特点：

- ✧ 带有灵活的保护单元，以确保操作系统和应用数据的安全。
- ✧ 采用软内核技术，防止外部对其进行扫描探测。
- ✧ 可集成用户自己的安全特性和其他协处理器。

SecurCore 系列微处理器主要应用于一些对安全性要求较高的应用产品及应用系统，如电子商务、电子政务、电子银行业务、网络和认证系统等领域。SecurCore 系列微处理器包含 SecurCore SC100、SecurCore SC110、SecurCore SC200 和 SecurCore SC210 四种类型，以适用于不同的应用场合。

6．StrongARM 微处理器系列

Inter StrongARM SA-1100 处理器是采用 ARM 体系结构高度集成的 32 位 RISC 微处理器。它融合了 Inter 公司的设计和处理技术以及 ARM 体系结构的电源效率，采用在软件上兼容 ARMv4 体系结构、同时采用具有 Intel 技术优点的体系结构。Intel StrongARM 处理器是便携式通讯产品和消费类电子产品的理想选择，已成功应用于多家公司的掌上电脑系列产品。

7．Xscale 处理器

Xscale 处理器是基于 ARMv5TE 体系结构的解决方案，是一款全性能、高性价比、低功耗的处理器。它支持 16 位的 Thumb 指令和 DSP 指令集，已使用在数字移动电话、个人数字助理和网络产品等场合。Xscale 处理器是 Inter 目前主要推广的一款 ARM 微处理器。

各系列产品性能比较见下表2.1。

表 2.1　ARM 各系列产品性能比较

型号 \ 功能		Cache 大小（指令/数据）	紧密耦合存储器（TCM）	存储器管理	AHB 总线接口	Thumb	DSP	Jazelle
ARM7 性能特征	ARM7TDMI	无	无	无	有	有	无	无
	ARM7TDMI-S	无	无	无	有	有	无	无
	ARM7EJ-S	无	无	无	有	有	有	有
	ARM720T	8k	无	MMU	有	有	无	无
ARM9 性能特征	ARM920T	16k/16k	无	MMU	有	有	无	无
	ARM922T	8k/8k	无	MMU	有	有	有	无
	ARM940T	4k/4k	无	MMU	有	有	有	无
ARM9E 性能特征	ARM926EJ-S	4—128k/4—128k	有	MMU	双 AHB	有	有	有
	ARM946E-S	4—1MB/4—1MB	有	MPU	AHB	有	有	无
	ARM966E-S	无	有	无	AHB	有	有	无
ARM10E 性能特征	ARM1020E	32k/32k	无	MMU	双 AHB	有	有	无
	ARM1022E	16k/16k	无	MMU	双 AHB	有	有	无
	ARM1026EJ-S	可变	有	MMU+MMU	双 AHB	有	有	有
ARM11 性能特征	ARM1136J-S	4—64k	有	MMU	四个 64 位 AHB	有	有	无

	ARM1136JF-S	4—64k	有	MMU	四个 64 位 AHB	有	有	有
ARMSec--urCore	SC100	无	无	MPU	无	有	无	无
	SC110	无	无	MPU	无	有	无	无
	SC200	可选	无	MPU	无	有	有	有
	SC210	可选	无	MPU	无	有	有	有
ARMIntel	Strong ARM	16k/8k	无	MMU	N/A	无	无	无
	XScale	32k/32k	无	MMU	N/A	有	有	无

2.2 ARM 微处理器结构

2.2.1 RISC 体系结构

传统的 CISC（Complex Instruction Set Computer，复杂指令集计算机）结构有其固有的缺点，即随着计算机技术的发展而不断引入新的复杂的指令集，为支持这些新增的指令，计算机的体系结构会越来越复杂，然而，在 CISC 指令集的各种指令中，其使用频率却相差悬殊，大约有 20%的指令会被反复使用，占整个程序代码的 80%。而余下的 80%的指令却不经常使用，在程序设计中只占 20%，显然，这种结构是不太合理的。

基于以上的不合理性，1979 年美国加州大学伯克利分校提出了 RISC（Reduced Instruction Set Computer，精简指令集计算机）的概念，RISC 并非只是简单地去减少指令，而是把着眼点放在了如何使计算机的结构更加简单合理地提高运算速度上。RISC 结构优先选取使用频率最高的简单指令，避免复杂指令；将指令长度固定，指令格式和寻址方式种类减少；以控制逻辑为主，不用或少用微码控制等措施来达到上述目的。到目前为止，RISC 体系结构也还没有严格的定义，一般认为，RISC 体系结构应具有如下特点：

- ✧ 采用固定长度的指令格式，指令归整、简单、基本寻址方式有 2～3 种。
- ✧ 使用单周期指令，便于流水线操作执行。
- ✧ 大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载/存储指令可以访问存储器，以提高指令的执行效率。

除此以外，ARM 体系结构还采用了一些特别的技术，在保证高性能的前提下尽量缩小芯片的面积，并降低功耗：

- ✧ 所有的指令都可根据前面的执行结果决定是否被执行，从而提高指令的执行效率。
- ✧ 可用加载/存储指令批量传输数据，以提高数据的传输效率。
- ✧ 可在一条数据处理指令中同时完成逻辑处理和移位处理。
- ✧ 在循环处理中使用地址的自动增减来提高运行效率。

当然，和 CISC 架构相比较，尽管 RISC 架构有上述的优点，但决不能认为 RISC 架构就可以取代 CISC 架构，事实上，RISC 和 CISC 各有优势，而且界限并不那么明显。现代的 CPU 往往采用 CISC 的外围，内部加入了 RISC 的特性，如超长指令集 CPU 就是融合了 RISC 和 CISC 的优势，成为未来的 CPU 发展方向之一。

2.2.2 ARM 微处理器的寄存器结构

ARM 处理器共有 37 个寄存器，被分为若干个组（BANK），这些寄存器包括：

- ✧ 31 个通用寄存器，包括程序计数器（PC 指针），均为 32 位的寄存器。
- ✧ 6 个状态寄存器，用以标识 CPU 的工作状态及程序的运行状态，均为 32 位，目前只使用了其中的一部分。

同时，ARM 处理器又有 7 种不同的处理器模式，如下表 2.2 在每一种处理器模式下均有一组相应的寄存器与之对应。即在任意一种处理器模式下，可访问的寄存器包括 15 个通用寄存器（R0～R14）、一至二个状态寄存器和程序计数器。在所有的寄存器中，有些是在 7 种处理器模式下共用的同一个物理寄存器，而有些寄存器则是在不同的处理器模式下有不同的物理寄存器。

表 2.2 ARM 处理器的 7 种处理器模式

处理器模式	描述
用户模式（User，usr）	正常程序执行的模式
快速中断模式（FIQ，fiq）	用于高速数据传输和通道处理
外部中断模式（IRQ，irq）	用于通常的中断处理
特权模式（Supervisor, Sve）	供操作系统使用的一种保护模式
数据访问中止模式（Abort,abt）	用于虚拟存储及存储保护
未定义指令中止模式（Undefined，und）	用于支持通过软件仿真硬件的协处理器
系统模式（System，sys）	用于运行特权级的操作系统任务

除了用户模式之外其它的 6 种处理器模式称为特权模式。在这些模式下，程序可以访问所有的系统资源，也可以任意的进行处理器模式的切换，其中除了系统模式外，其它 5 种特权模式又称为异常模式。处理器模式可以通过软件控制来进行切换，也可以通过外部中断或异常处理过程进行切换。大多数的用户程序可以运行在用户模式下，这时，应用程序不能够访问一些受应用程序保护的系统资源。应用程序不能直接进行处理器模式的切换。当需要处理器的模式切换的时候，应用程序可以产生异常处理，在异常处理过程种进行处理器模式的切换，这种体系结构可以使整个操作系统控制整个系统的资源。当应用程序发生异常中断时，处理器进入相应的异常模式，在每一种异常模式中都有一组寄存器，供相应的异常处理程序使用，这样就保证在进入异常模式的时候，用户模式下的寄存器（保存了程序的运行的状态）不被破坏。

2.2.3 ARM 微处理器的指令结构

ARM 微处理器的在较新的体系结构中支持两种指令集：ARM 指令集和 Thumb 指令集。其中，ARM 指令为 32 位的长度，Thumb 指令为 16 位长度。Thumb 指令集为 ARM 指令集的功能子集，但与等价的 ARM 代码相比较，可节省 30%～40%以上的存储空间，同时具备 32 位代码的所有优点。

关于 ARM 处理器的指令结构，在后面的相关章节将会详细描述。

2.3 寄存器组织

ARM 微处理器共有 37 个 32 位寄存器，其中 31 个为通用寄存器，6 个为状态寄存器。但是这些寄存器不能被同时访问，具体哪些寄存器是可编程访问的，取决微处理器的工作状态及具体的运行模式。但在任何时候，通用寄存器 R14～R0、程序计数器 PC、一个或两个状态寄存器都是可访问的。

2.3.1 ARM 状态下的寄存器组织

通用寄存器：

通用寄存器包括 R0～R15，可以分为三类：

- ✧ 未分组寄存器 R0～R7；
- ✧ 分组寄存器 R8～R14；
- ✧ 程序计数器 PC（R15）。

未分组寄存器 R0～R7：

在所有的运行模式下，未分组寄存器都指向同一个物理寄存器，他们未被系统用作特殊的用途，因此，在中断或异常处理进行运行模式转换时，由于不同的处理器运行模式均使用相同的物理寄存器，可能会造成寄存器中数据的破坏，这一点在进行程序设计时应引起注意。

分组寄存器 R8～R14：

对于分组寄存器，他们每一次所访问的物理寄存器与处理器当前的运行模式有关。

对于 R8～R12 来说，每个寄存器对应两个不同的物理寄存器，当使用 fiq 模式时，访问寄存器 R8_fiq～R12_fiq；当使用除 fiq 模式以外的其他模式时，访问寄存器 R8_usr～R12_usr。

对于 R13、R14 来说，每个寄存器对应 6 个不同的物理寄存器，其中的一个是用户模式与系统模式共用，另外 5 个物理寄存器对应于其他 5 种不同的运行模式。

采用以下的记号来区分不同的物理寄存器：

R13_<mode>

R14_<mode>

其中，mode 为以下几种模式之一：usr、fiq、irq、svc、abt、und。

寄存器 R13 在 ARM 指令中常用作堆栈指针，但这只是一种习惯用法，用户也可使用其他的寄存器作为堆栈指针。而在 Thumb 指令集中，某些指令强制性的要求使用 R13 作为堆栈指针。

由于处理器的每种运行模式均有自己独立的物理寄存器 R13，在用户应用程序的初始化部分，一般都要初始化每种模式下的 R13，使其指向该运行模式的栈空间，这样，当程序的运行进入异常模式时，可以将需要保护的寄存器放入 R13 所指向的堆栈，而当程序从异常模式返回时，则从对应的堆栈中恢复，采用这种方式可以保证异常发生后程序的正常执行。

R14 也称作子程序连接寄存器（Subroutine Link Register）或连接寄存器 LR。当执行 BL 子程序调用指令时，R14 中得到 R15（程序计数器 PC）的备份。其他情况下，R14 用作通用寄存器。与之类似，当发生中断或异常时，对应的分组寄存器 R14_svc、R14_irq、R14_fiq、R14_abt 和 R14_und 用来保存 R15 的返回值。

寄存器 R14 常用在如下的情况：在每一种运行模式下，都可用 R14 保存子程序的返回

地址，当用 BL 或 BLX 指令调用子程序时，将 PC 的当前值拷贝给 R14，执行完子程序后，又将 R14 的值拷贝回 PC，即可完成子程序的调用返回。以上的描述可用指令完成：

1．执行以下任意一条指令：

MOV　PC，LR

BX　　LR

2．在子程序入口处使用以下指令将 R14 存入堆栈：

STMFD　SP！,{<Regs>,LR}

对应的，使用以下指令可以完成子程序返回：

LDMFD　SP！,{<Regs>,PC}

R14 也可作为通用寄存器。

程序计数器 PC（R15）

寄存器 R15 用作程序计数器（PC）。在 ARM 状态下，位[1：0]为 0，位[31：2]用于保存 PC；在 Thumb 状态下，位[0]为 0，位[31：1]用于保存 PC；虽然可以用作通用寄存器，但是有一些指令在使用 R15 时有一些特殊限制，若不注意，执行的结果将是不可预料的。在 ARM 状态下，PC 的 0 和 1 位是 0，在 Thumb 状态下，PC 的 0 位是 0。R15 虽然也可用作通用寄存器，但一般不这么使用，因为对 R15 的使用有一些特殊的限制，当违反了这些限制时，程序的执行结果是未知的。

由于 ARM 体系结构采用了多级流水线技术，对于 ARM 指令集而言，PC 总是指向当前指令的下两条指令的地址，即 PC 的值为当前指令的地址值加 8 个字节。

表 2.3 ARM 状态下的寄存器组织

在 ARM 状态下，任一时刻可以访问以上所讨论的 16 个通用寄存器和一到两个状态寄存器。在非用户模式（特权模式）下，则可访问到特定模式分组寄存器，表 2.3 说明在每一种运行模式下，哪一些寄存器是可以访问的。

寄存器 R16：

寄存器 R16 用作 CPSR（Current Program Status Register，当前程序状态寄存器），CPSR

可在任何运行模式下被访问，它包括条件标志位、中断禁止位、当前处理器模式标志位，以及其他一些相关的控制和状态位。每一种运行模式下又都有一个专用的物理状态寄存器，称为 SPSR（Saved　Program Status Register，备份的程序状态寄存器），当异常发生时，SPSR 用于保存 CPSR 的当前值，从异常退出时则可由 SPSR 来恢复 CPSR。由于用户模式和系统模式不属于异常模式，他们没有 SPSR，当在这两种模式下访问 SPSR，结果是未知的。

2.3.2 Thumb 状态下的寄存器组织

Thumb 状态下的寄存器集是 ARM 状态下寄存器集的一个子集，程序可以直接访问 8 个通用寄存器（R7～R0）、程序计数器（PC）、堆栈指针（SP）、连接寄存器（LR）和 CPSR。同时，在每一种特权模式下都有一组 SP、LR 和 SPSR。表 2.4 表明 Thumb 状态下的寄存器组织。

表 2.4　Thumb 状态下的寄存器组织

通用寄存器与程序计数器					
System & User	FIQ	Supervisor	About	IRG	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	SP_fiq	SP_svg	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC
程序状态寄存器					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und
= 分组寄存器					

Thumb 状态下的寄存器组织与 ARM 状态下的寄存器组织的关系：

- ✧ Thumb 状态下和 ARM 状态下的 R0～R7 是相同的。
- ✧ Thumb 状态下和 ARM 状态下的 CPSR 和所有的 SPSR 是相同的。
- ✧ Thumb 状态下的 SP 对应于 ARM 状态下的 R13。
- ✧ Thumb 状态下的 LR 对应于 ARM 状态下的 R14。
- ✧ Thumb 状态下的程序计数器对应于 ARM 状态下 R15。

以上的对应关系如图 2.1 所示：

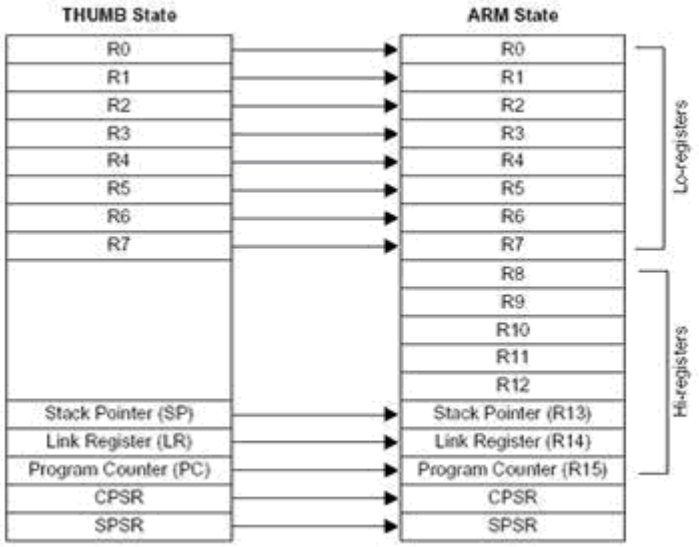


图 2.1　Thumb 状态下的寄存器组织

访问 THUMB 状态下的高位寄存器（Hi-registers）：

在 Thumb 状态下，高位寄存器 R8～R15 并不是标准寄存器集的一部分，但可使用汇编语言程序受限制的访问这些寄存器，将其用作快速的暂存器。使用带特殊变量的 MOV 指令，数据可以在低位寄存器和高位寄存器之间进行传送；高位寄存器的值可以使用 CMP 和 ADD 指令进行比较或加上低位寄存器中的值。

2.3.3 程序状态寄存器

ARM 体系结构包含一个当前程序状态寄存器（CPSR）和五个备份的程序状态寄存器（SPSRs）。备份的程序状态寄存器用来进行异常处理，其功能包括：

- ✧　保存 ALU 中的当前操作信息
- ✧　控制允许和禁止中断
- ✧　设置处理器的运行模式

程序状态寄存器的每一位的安排如图 2.2 所示：

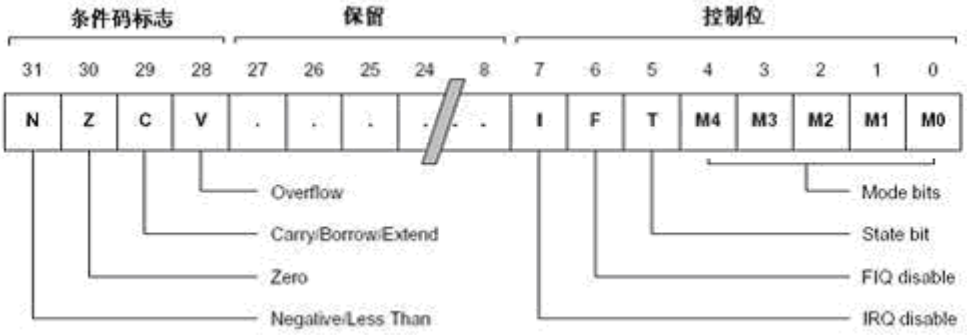


图 2.2　程序状态寄存器格式

条件码标志（Condition Code Flags）：

N、Z、C、V 均为条件码标志位。它们的内容可被算术或逻辑运算的结果所改变，并且可以决定某条指令是否被执行。在 ARM 状态下，绝大多数的指令都是有条件执行的；

在 Thumb 状态下，仅有分支指令是有条件执行的。

条件码标志各位的具体含义如表 2.5 所示：

表 2.5 条件码标志的具体含义

标志位	含义
N	当用两个补码表示的带符号数进行运算时，N=1 表示运算的结果为负数；N=0 表示运算的结果为正数或零；
Z	Z=1 表示运算的结果为零；Z=0 表示运算的结果为非零；
C	可以有 4 种方法设置 C 的值： — 加法运算（包括比较指令 CMN）：当运算结果产生了进位时（无符号数溢出），C=1，否则 C=0。 — 减法运算（包括比较指令 CMP）：当运算时产生了借位（无符号数溢出），C=0，否则 C=1。 — 对于包含移位操作的非加/减运算指令，C 为移出值的最后一位。 — 对于其他的非加/减运算指令，C 的值通常不改变。
V	可以有 2 种方法设置 V 的值： — 对于加/减法运算指令，当操作数和运算结果为二进制的补码表示的带符号数时，V=1 表示符号位溢出。 — 对于其他的非加/减运算指令，V 的值通常不改变。
Q	在 ARM v5 及以上版本的 E 系列处理器中，用 Q 标志位指示增强的 DSP 运算指令是否发生了溢出。在其他版本的处理器中，Q 标志位无定义。

控制位：

PSR 的低 8 位（包括 I、F、T 和 M[4：0]）称为控制位，当发生异常时这些位可以被改变。如果处理器运行特权模式，这些位也可以由程序修改。

中断禁止位 I、F：

I=1　禁止 IRQ 中断; F=1　禁止 FIQ 中断。

T 标志位：该位反映处理器的运行状态。对于 ARM 体系结构 v5 及以上的版本的 T 系列处理器，当该位为 1 时，程序运行于 Thumb 状态，否则运行于 ARM 状态。对于 ARM 体系结构 v5 及以上的版本的非 T 系列处理器，当该位为 1 时，执行下一条指令以引起为定义的指令异常；当该位为 0 时，表示运行于 ARM 状态。

运行模式位 M[4：0]：

M0、M1、M2、M3、M4 是模式位。这些位决定了处理器的运行模式。具体含义如表 2.6 所示：

表 2.6 运行模式位 M[4：0]的具体含义

M[4：0]	处理器模式	可访问的寄存器
0b10000	用户模式	PC，CPSR,R0-R14
0b10001	FIQ 模式	PC，CPSR, SPSR_fiq，R14_fiq-R8_fiq, R7～R0
0b10010	IRQ 模式	PC，CPSR, SPSR_irq，R14_irq,R13_irq,R12～R0
0b10011	管理模式	PC，CPSR, SPSR_svc，R14_svc,R13_svc,,R12～R0,
0b10111	中止模式	PC，CPSR, SPSR_abt，R14_abt,R13_abt, R12～R0,
0b11011	未定义模式	PC，CPSR, SPSR_und，R14_und,R13_und, R12～R0,
0b11111	系统模式	PC，CPSR（ARM v4 及以上版本）, R14～R0

由表 2.6 可知，并不是所有的运行模式位的组合都是有效的，其他的组合结果会导致处理器进入一个不可恢复的状态。

保留位：

PSR 中的其余位为保留位，当改变 PSR 中的条件码标志位或者控制位时，保留位不要被改变，在程序中也不要使用保留位来存储数据。保留位将用于 ARM 版本的扩展。

2.4 异常（Exceptions）

当正常的程序执行流程发生暂时的停止时，称之为异常，例如处理一个外部的中断请求。在处理异常之前，当前处理器的状态必须保留，这样当异常处理完成之后，当前程序可以继续执行。处理器允许多个异常同时发生，它们将会按固定的优先级进行处理。ARM 体系结构中的异常，与 8 位/16 位体系结构的中断有很大的相似之处，但异常与中断的概念并不完全等同。

2.4.1 ARM 体系结构所支持的异常类型

ARM 体系结构所支持的异常及具体含义如表 2.7 所示。

表 2.7 ARM 体系结构所支持的异常

异常类型	具体含义
复位	当处理器的复位电平有效时，产生复位异常，程序跳转到复位异常处理程序处执行。
未定义指令	当 ARM 处理器或协处理器遇到不能处理的指令时，产生未定义指令异常。可使用该异常机制进行软件仿真。
软件中断	该异常由执行 SWI 指令产生，可用于用户模式下的程序调用特权操作指令。可使用该异常机制实现系统功能调用。
指令预取中止	若处理器预取指令的地址不存在，或该地址不允许当前指令访问，存储器会向处理器发出中止信号，但当预取的指令被执行时，才会产生指令预取中止异常。
数据中止	若处理器数据访问指令的地址不存在，或该地址不允许当前指令访问时，产生数据中止异常。
IRQ（外部中断请求）	当处理器的外部中断请求引脚有效，且 CPSR 中的 I 位为 0 时，产生 IRQ 异常。系统的外设可通过该异常请求中断服务。
FIQ（快速中断请求）	当处理器的快速中断请求引脚有效，且 CPSR 中的 F 位为 0 时，产生 FIQ 异常。

2.4.2 对异常的响应

1．将下一条指令的地址存入相应连接寄存器 LR，以便程序在处理异常返回时能从正确的位置重新开始执行。若异常是从 ARM 状态进入，LR 寄存器中保存的是下一条指令的地址（当前 PC＋4 或 PC＋8，与异常的类型有关）；若异常是从 Thumb 状态进入，则在 LR 寄存器中保存当前 PC 的偏移量，这样，异常处理程序就不需要确定异常是从何种状态进入的。例如：在软件中断异常 SWI，指令 MOV PC，R14_svc 总是返回到下一条指令，不

管 SWI 是在 ARM 状态执行，还是在 Thumb 状态执行。

2．将 CPSR 复制到相应的 SPSR 中。

3．根据异常类型，强制设置 CPSR 的运行模式位。

4．强制 PC 从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

还可以设置中断禁止位，以禁止中断发生。

如果异常发生时，处理器处于 Thumb 状态，则当异常向量地址加载入 PC 时，处理器自动切换到 ARM 状态。ARM 微处理器对异常的响应过程用伪码可以描述为：

```
R14_<Exception_Mode> = Return Link
SPSR_<Exception_Mode> = CPSR
CPSR[4：0] = Exception Mode Number
CPSR[5] = 0                          ；当运行于 ARM 工作状态时
If <Exception_Mode> == Reset or FIQ then
                                     ；当响应 FIQ 异常时，禁止新的 FIQ 异常
    CPSR[6] = 1
CPSR[7] = 1
    PC = Exception Vector Address
```

2.4.3 从异常返回

异常处理完毕之后，ARM 微处理器会执行以下几步操作从异常返回：

1．将连接寄存器 LR 的值减去相应的偏移量后送到 PC 中。

2．将 SPSR 复制回 CPSR 中。

3．若在进入异常处理时设置了中断禁止位，要在此清除。

可以认为应用程序总是从复位异常处理程序开始执行的，因此复位异常处理程序不需要返回。

2.4.4 各类异常的具体描述

FIQ（Fast Interrupt Request）：

FIQ 异常是为了支持数据传输或者通道处理而设计的。在 ARM 状态下，系统有足够的私有寄存器，从而可以避免对寄存器保存的需求，并减小了系统上下文切换的开销。

若将 CPSR 的 F 位置为 1，则会禁止 FIQ 中断，若将 CPSR 的 F 位清零，处理器会在指令执行时检查 FIQ 的输入。注意只有在特权模式下才能改变 F 位的状态。

可由外部通过对处理器上的 nFIQ 引脚输入低电平产生 FIQ。不管是在 ARM 状态还是在 Thumb 状态下进入 FIQ 模式，FIQ 处理程序均会执行以下指令从 FIQ 模式返回：

```
SUBS    PC,R14_fiq ,#4
```

该指令将寄存器 R14_fiq 的值减去 4 后，复制到程序计数器 PC 中，从而实现从异常处理程序中的返回，同时将 SPSR_mode 寄存器的内容复制到当前程序状态寄存器 CPSR 中。

IRQ（Interrupt Request）：

IRQ 异常属于正常的中断请求，可通过对处理器的 nIRQ 引脚输入低电平产生，IRQ 的优先级低于 FIQ，当程序执行进入 FIQ 异常时，IRQ 可能被屏蔽。

若将 CPSR 的 I 位置为 1，则会禁止 IRQ 中断，若将 CPSR 的 I 位清零，处理器会在

指令执行完之前检查 IRQ 的输入。注意只有在特权模式下才能改变 I 位的状态。

不管是在 ARM 状态还是在 Thumb 状态下进入 IRQ 模式，IRQ 处理程序均会执行以下指令从 IRQ 模式返回：

SUBS　PC , R14_irq , #4

该指令将寄存器 R14_irq 的值减去 4 后，复制到程序计数器 PC 中，从而实现从异常处理程序中的返回，同时将 SPSR_mode 寄存器的内容复制到当前程序状态寄存器 CPSR 中。

ABORT（中止）：

产生中止异常意味着对存储器的访问失败。ARM 微处理器在存储器访问周期内检查是否发生中止异常。

中止异常包括两种类型：

✧　指令预取中止：发生在指令预取时。

✧　数据中止：发生在数据访问时。

当指令预取访问存储器失败时，存储器系统向 ARM 处理器发出存储器中止（Abort）信号，预取的指令被记为无效，但只有当处理器试图执行无效指令时，指令预取中止异常才会发生，如果指令未被执行，例如在指令流水线中发生了跳转，则预取指令中止不会发生。若数据中止发生，系统的响应与指令的类型有关。当确定了中止的原因后，Abort 处理程序均会执行以下指令从中止模式返回，无论是在 ARM 状态还是 Thumb 状态：

SUBS PC, R14_abt, #4　　；指令预取中止

SUBS PC, R14_abt, #8　　；数据中止

以上指令恢复 PC（从 R14_abt）和 CPSR（从 SPSR_abt）的值，并重新执行中止的指令。

Software Interruupt（软件中断）：

软件中断指令（SWI）用于进入管理模式，常用于请求执行特定的管理功能。软件中断处理程序执行以下指令从 SWI 模式返回，无论是在 ARM 状态还是 Thumb 状态：

MOV　PC , R14_svc

以上指令恢复 PC（从 R14_svc）和 CPSR（从 SPSR_svc）的值，并返回到 SWI 的下一条指令。

Undefined Instruction（未定义指令）：

当 ARM 处理器遇到不能处理的指令时，会产生未定义指令异常。采用这种机制，可以通过软件仿真扩展 ARM 或 Thumb 指令集。

在仿真未定义指令后，处理器执行以下程序返回，无论是在 ARM 状态还是 Thumb 状态：

MOVS PC, R14_und

以上指令恢复 PC（从 R14_und）和 CPSR（从 SPSR_und）的值，并返回到未定义指令后的下一条指令。

2.4.5 异常进入/退出小节

表 2.8 总结了进入异常处理时保存在相应 R14 中的 PC 值，及在退出异常处理时推荐使用的指令。

表 2.8 异常进入/退出

	返回指令	以前的状态		注意
		ARM R14_x	Thumb R14_x	
BL	MOV PC，R14	PC＋4	PC＋2	1
SWI	MOVS PC，R14_svc	PC＋4	PC＋2	1
UDEF	MOVS PC，R14_und	PC＋4	PC＋2	1
FIQ	SUBS PC，R14_fiq，＃4	PC＋4	PC＋4	2
IRQ	SUBS PC，R14_irq，＃4	PC＋4	PC＋4	2
PABT	SUBS PC，R14_abt，＃4	PC＋4	PC＋4	1
DABT	SUBS PC，R14_abt，＃8	PC＋8	PC＋8	3
RESET	NA	－	－	4

注意：

1．在此 PC 应是具有预取中止的 BL/SWI/未定义指令所取的地址。

2．在此 PC 是从 FIQ 或 IRQ 取得不能执行的指令的地址。

3．在此 PC 是产生数据中止的加载或存储指令的地址。

4．系统复位时，保存在 R14_svc 中的值是不可预知的。

2.4.6 异常向量（Exception Vectors）

异常向量地址显示如表 2.9。

表 2.9 异常向量表

地 址	异 常	进入模式
0x0000,0000	复位	管理模式
0x0000,0004	未定义指令	未定义模式
0x0000,0008	软件中断	管理模式
0x0000,000C	中止（预取指令）	中止模式
0x0000,0010	中止（数据）	中止模式
0x0000,0014	保留	保留
0x0000,0018	IRQ	IRQ
0x0000,001C	FIQ	FIQ

2.4.7 异常优先级（Exception Priorities）

当多个异常同时发生时，系统根据固定的优先级决定异常的处理次序。异常优先级由高到低的排列次序如表 2.10 所示。

表 2.10 异常优先级

优先级	异　常
1（最高）	复位
2	数据中止
3	FIQ
4	IRQ
5	预取指令中止
6（最低）	未定义指令、SWI

2.4.8 应用程序中的异常处理

当系统运行时，异常可能会随时发生，为保证在 ARM 处理器发生异常时不至于处于未知状态，在应用程序的设计中，首先要进行异常处理，采用的方式是在异常向量表中的特定位置放置一条跳转指令，跳转到异常处理程序，当 ARM 处理器发生异常时，程序计数器 PC 会被强制设置为对应的异常向量，从而跳转到异常处理程序，当异常处理完成以后，返回到主程序继续执行。

2.5 ARM 微处理器的应用选型

2.5.1 ARM 选型原则

鉴于 ARM 微处理器的众多优点，随着国内外嵌入式应用领域的逐步发展，ARM 微处理器必然会获得广泛的重视和应用。但是，由于 ARM 微处理器有多达十几种的内核结构，几十个芯片生产厂家，以及千变万化的内部功能配置组合，给开发人员在选择方案时带来一定的困难，所以，对 ARM 芯片做一些对比研究是十分必要的。以下从应用的角度出发，对在选择 ARM 微处理器时所应考虑的主要问题做一些简要的探讨。

1．ARM 微处理器内核的选择

从前面所介绍的内容可知，ARM 微处理器包含一系列的内核结构，以适应不同的应用领域，用户如果希望使用 WinCE 或标准 Linux 等操作系统以减少软件开发时间，就需要选择 ARM720T 以上带有 MMU（Memory Management Unit）功能的 ARM 芯片，ARM720T、ARM920T、ARM922T、ARM946T、Strong-ARM 都带有 MMU 功能。

本书所讨论的 S3C2410 基于 ARM920T 内核，它带有 MMU，因此支持 Windows CE 和标准 Linux，并且在稳定性和其他方面也都有上佳表现。

2．系统的工作频率

系统的工作频率在很大程度上决定了 ARM 微处理器的处理能力。ARM7 系列微处理器的典型处理速度为 0.9MIPS/MHz，ARM9 系列微处理器的典型处理速度为 1.1MIPS/MHz，常见的 ARM9 的系统主时钟频率为 100MHz-233MHz，ARM10 最高可以达到 700MHz。不同芯片对时钟的处理不同，有的芯片只需要一个主时钟频率，有的芯片内部时钟控制器可以分别为 ARM 核和 USB、UART、DSP、音频等功能部件提供不同频率的时钟。

本书所讨论的 S3C2410 时钟频率为 200MHz 以上，若更换成 S3C2440 时钟频率最高可达到 500MHz。

3．芯片内存储器的容量

大多数的 ARM 微处理器片内存储器的容量都不太大，需要用户在设计系统时外扩存储器，但也有部分芯片具有相对较大的片内存储空间，如 ATMEL 的 AT91F40162 就具有高达 2MB 的片内程序存储空间，用户在设计时可考虑选用这种类型，以简化系统的设计。

4．片内外围电路的选择

除 ARM 微处理器核以外，几乎所有的 ARM 芯片均根据各自不同的应用领域，扩展了相关功能模块，并集成在芯片之中，我们称之为片内外围电路，如 USB 接口、IIS 接口、LCD 控制器、键盘接口、RTC、ADC 和 DAC、DSP 协处理器等，设计者应分析系统的需求，尽可能采用片内外围电路完成所需的功能，这样既可简化系统的设计，同时提高系统的可靠性。

2.5.2 ARM920T

ARM9 内核采用了与 StrongARM 相同的五级流水线、提供 1.1MIPS/MHz 的哈佛结构、全性能的 MMU 单元、可配置的数据 Cache 和可分立和指令和数据高速 AHB 接口。

1．五级流水线

ARM920T 处理器使用流水线来增加处理器指令流的速度。这样可以使几个操作同时进行，并使处理和存储器系统连续操作，能提供 1.1MIPS/MHz 的指令执行速度。流水线使用 5 个阶段，因此指令分 5 个阶段执行：取址→译码→执行→存储→写。

2．ARM920T 处理器结构

ARM920T 处理器功能方框图如图 2.3。

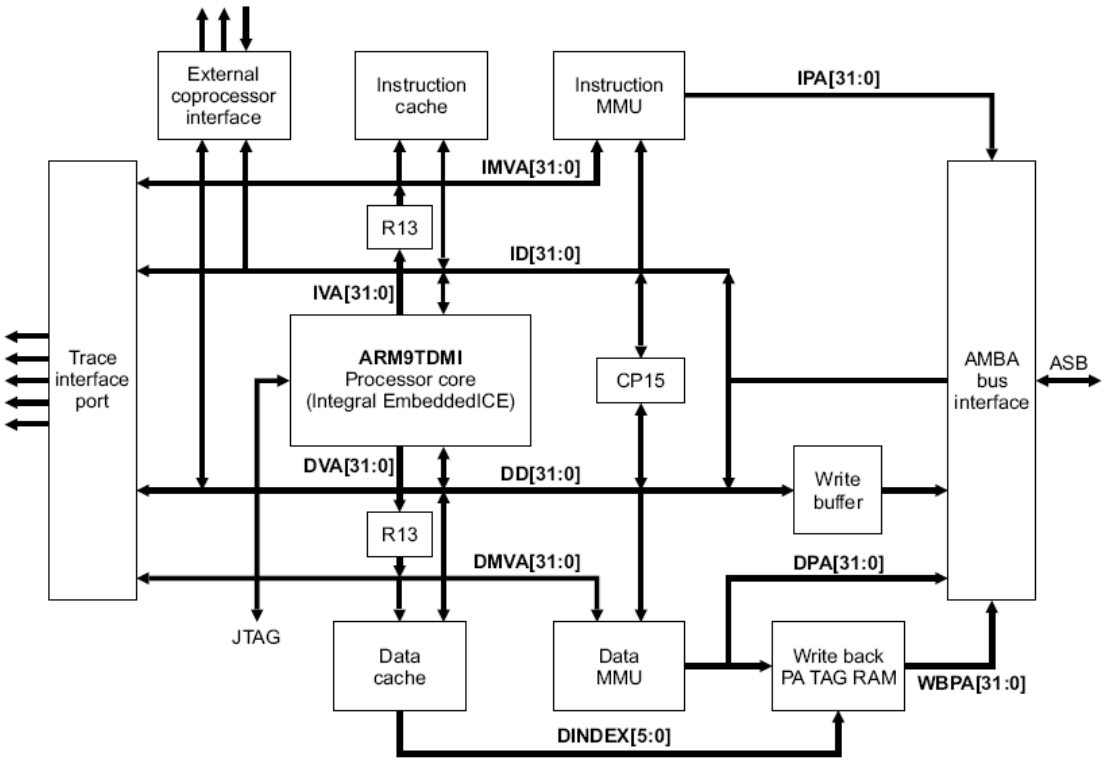


图 2.3 ARM920T 处理器功能方框图

第三章 OURS2410EDU 开发板说明

3.1 系统概述

OURS2410EDU 是一款基于三星 S3C2410X 16/32 位 RISC 处理器（ARM920T）的针对高校嵌入式教学和实验科研的平台。

S3C2410X 包含一个 16-/32-bit 的 RISC（ARM920T）的 CPU 内核，独立的 16KB 指令和 16KB 数据的缓存(cache)，用于虚拟内存管理的 MMU 单元，LCD 控制器(STN&TFT)，非线性(NAND)FLASH 的引导单元，系统管理器(包括片选逻辑控制和 SDRAM 控制器)，3 个通道的异步串口（UART）,4 个通道的 DMA，4 个通道的带脉宽调制（PWM）的定时器，输入输出端口，实时时钟单元（RTC），带有触摸屏接口的 8 个通道的 10-bit ADC，IIC 总线接口，IIS 总线接口，USB 的主机（host）单元，USB 的设备（device）接口，SD 卡和 MMC（Multimedia Card）卡接口，2 个通道的 SPI 接口和锁相环（PLL）时钟发生单元。

基于 S3C2410 处理器我们扩展了丰富的接口，系统功能框图如图 3.1 所示：

系统具有如下特性：

- S3C2410X：16-/32-bit RISC 微处理器，在系统结构上，我们把 CPU 及周围的最小系统部分作为一个“核心板”的模式通过 200P DIMM 插槽和主板相联，用户既可以采用 S3C2410X 的核心板，也可采用 S3C2440（500MHz）核心板，方便用户的升级。
- BOOT ROM：支持三种启动方式
  - AMD 2M-byte
  - Intel StrataFlash 16M-byte（可选）
  - SAMSUNG NAND flash 64M-byte/128M-byte （直接 smart media card 启动）
- SDRAM：64M-byte （32M-byte×2）
- TFT/STN 液晶和触摸屏接口 ：SHARP 3.5″真彩 TFT 液晶屏
- 3 个通道的异步串口，其中一个可以切换为红外（IrDA）接口
- 两个通道的 USB 接口，其中一个可以切换为 HOST 或 DEVICE 设备
- SD 卡主机（MMC）接口
- SMC 卡（Smart media card）接口
- JTAG 接口（MULTI-ICE 兼容）
- 实时时钟（RTC）单元
- IIC 总线接口（板载 IIC 接口的 EEPROM 24C16 和 IIC 接口的数码管驱动 ZLG7290）
- ADC 接口，芯片内部集成 8 通道 10bit ADC，板载输入有：
  - 电位器调节的电压输入
  - LM35 温度传感器输入
  - D/A 输出引脚经隔离后的输入
- SPI 接口，外接 SPI 接口的键盘控制芯片 ZLG7289
- 20 键的键盘矩阵输入
- 板载双以太网（Ethernet）接口
  - 通过 CS8900 扩展的 10M 以太网接口
  - 通过 DM9000 扩展的 10M/100M 以太网接口
- PCMCIA 接口，通过扩展卡也可以接为 CF 卡接口

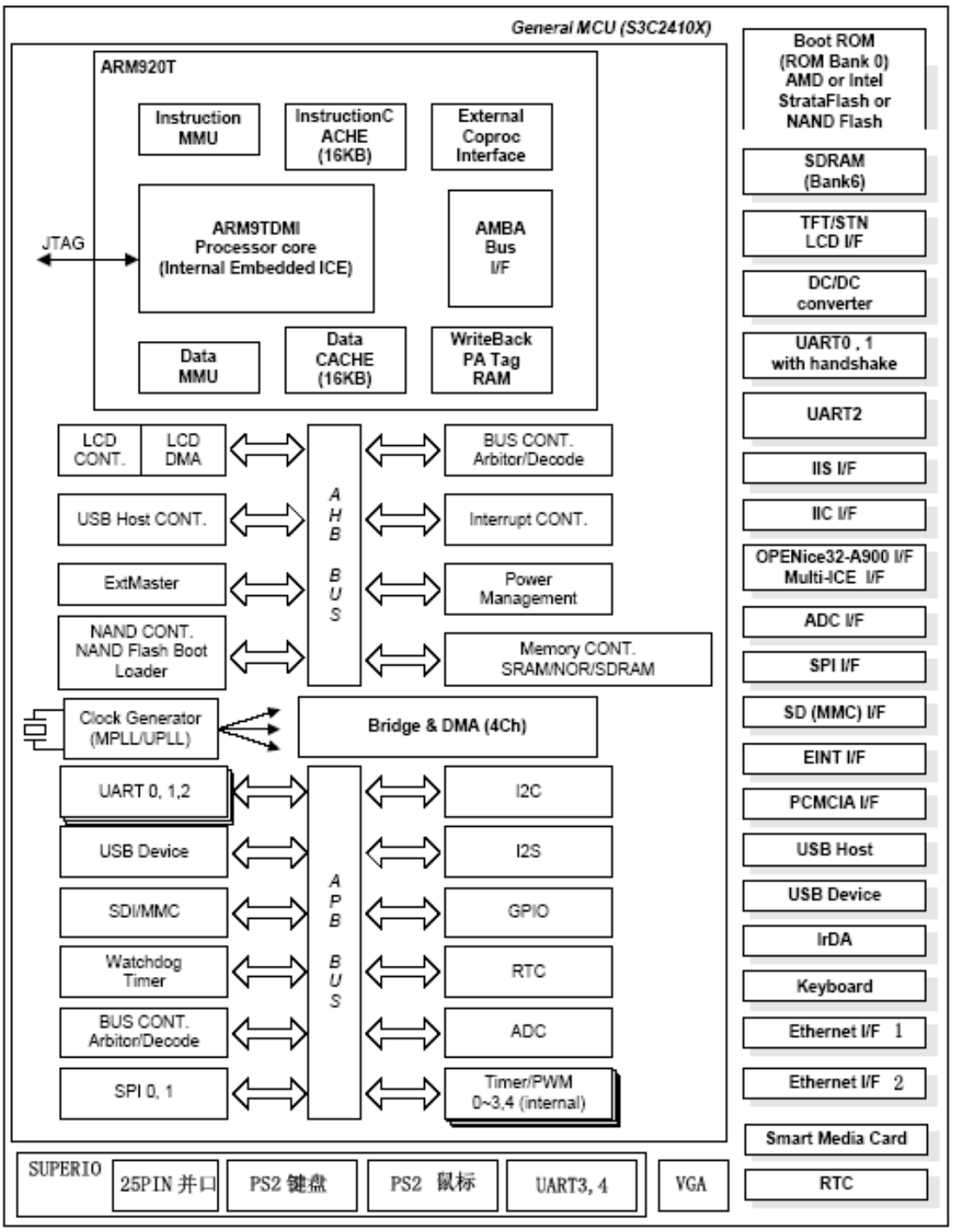


图 3.1（a） OURS2410EDU 功能框图

- ✧ LED 显示，系统带有三部分的 LED 显示部分：
  - ✧ 核心板上有 4 个发光二极管，可以方便显示调试信息
  - ✧ 通过 ZLG7290 驱动的 8 位 8 段数码管，方便显示数字信息
  - ✧ 通过总线驱动的 8×8 数码管点阵，方便显示点阵信息
- ✧ SUPERIO 接口，通过华邦 SUPERIO 芯片 W83977EF 扩展了如下资源：
  - ✧ 两个全功能的异步串口（UART）和一个红外串口，使板载的串口数达到了 6 个

- ✧ 25 针并口一个，方便扩展并口打印机等资源
- ✧ PS2 键盘接口一个
- ✧ PS2 鼠标接口一个

✧ VIDEO 扩展接口，通过专用视频编码芯片 CH7004 输出如下形式的视频信号：

- ✧ VGA 信号，可以直接连接显示器
- ✧ S 端子，直接连接电视机
- ✧ AV 端子，直接连接电视机

✧ IIS 音频扩展接口，通过音频扩展芯片支持 MIC 输入及耳机音频输出；为了方便视听，板载音频功放和两个 1W 的喇叭，左右声道经功放驱动后直接由板载喇叭发声，当然也可以通过板上跳线关闭这个在线播放的功能

✧ IDE 接口，通过板载的 IDE 接口可以直接连接笔记本的 IDE 硬盘，支持 DMA 方式

✧ 扩展总线接口，总线接口通过两个 96P×2 的欧式座扩展出来，不仅引出了总线的信号，而且通过板载的 CPLD 扩展了多个 I/O 口，方便了用户的使用；而且，结实耐用的欧式座适用用户的不断插拔；通过扩展总线可以扩展如下扩展模块：

- ✧ FPGA（Altera Cyclone）实验扩展模块，实现 MCU+FPGA 架构的综合实验
- ✧ DSP（TI 5402）扩展模块，实现 MCU+DSP 架构的综合实验
- ✧ GPS 扩展模块，实现地理信息相关的综合实验
- ✧ GPRS 扩展模块，实现远程无线通信的综合实验
- ✧ 电机扩展模块，包括步进、直流、交流电机模块，实现电机相关的综合控制实验
- ✧ 指纹传感器（ATMEL FCD4B14）扩展模块，实现信号处理的综合实验

扩展模块部分我们将针对不同的专业需要和不同应用不断扩充和完善；对用户而言，在基本平台不变的基础上，通过配置不同的扩展模块就可以满足各个不同专业的需要。

S3C2410EDU 实物图及实物说明图如图 3.1（b）（c）所示

图 3.1（b）S3C2410EDU 实物图

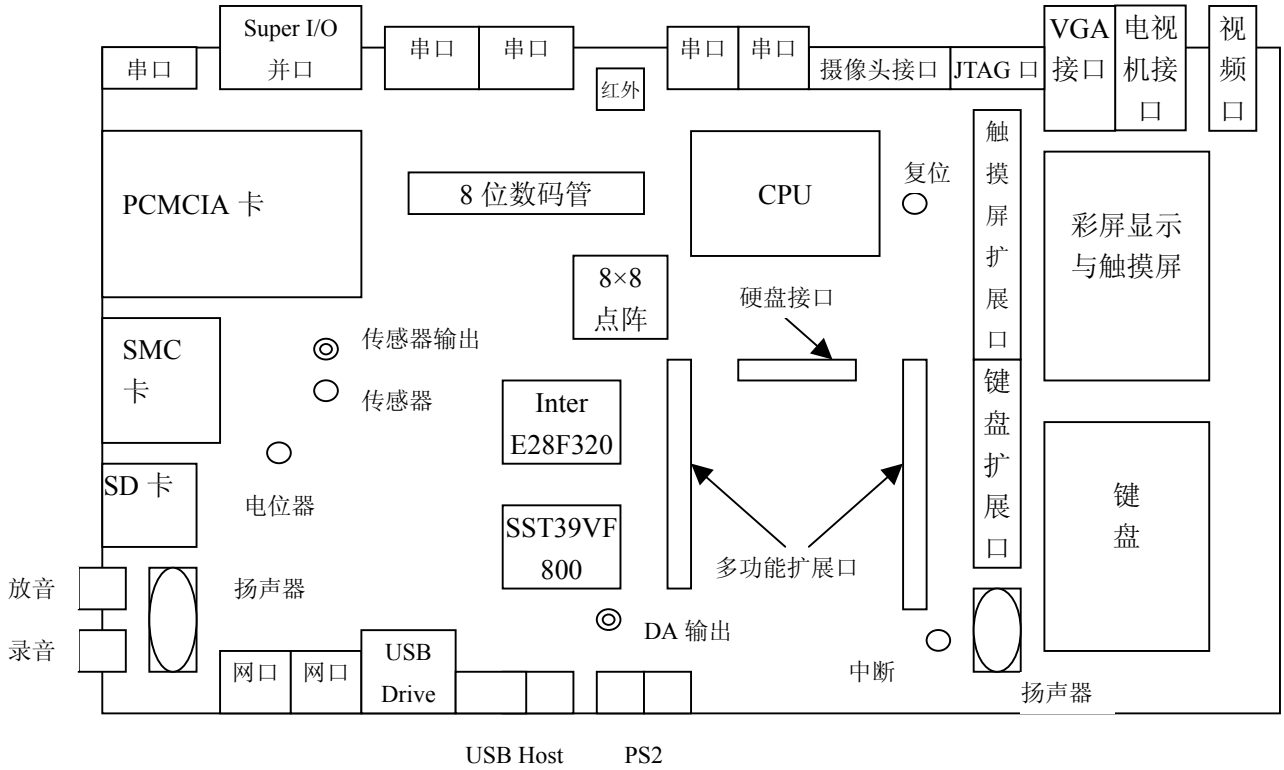


图 3.1（c）S3C2410EDU 实物说明图

3.2 电路说明

整个电路的框图如图 3.2 所示，

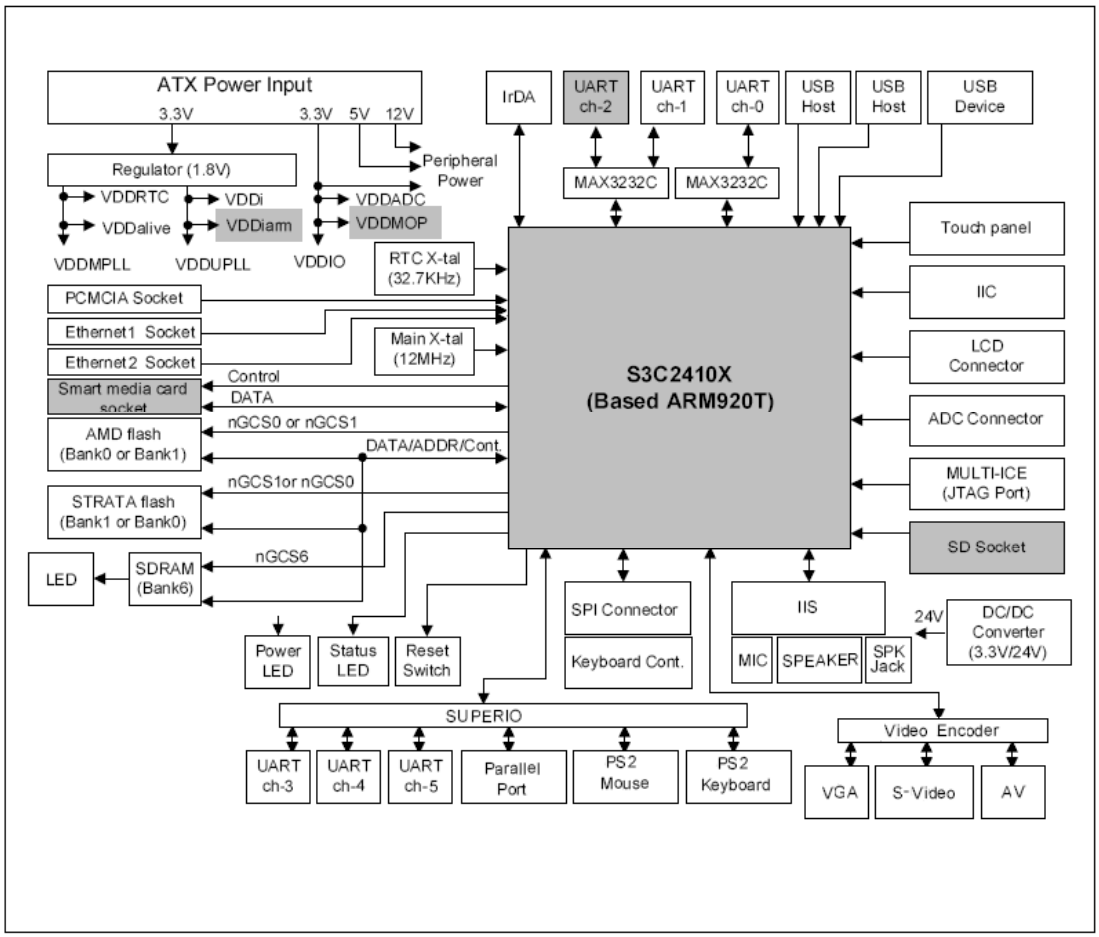


图 3.2 电路框图

为了让大家对整个系统有更全面的了解，下面我们对电路中的各个模块作比较详细的说明。

3.2.1 系统供电

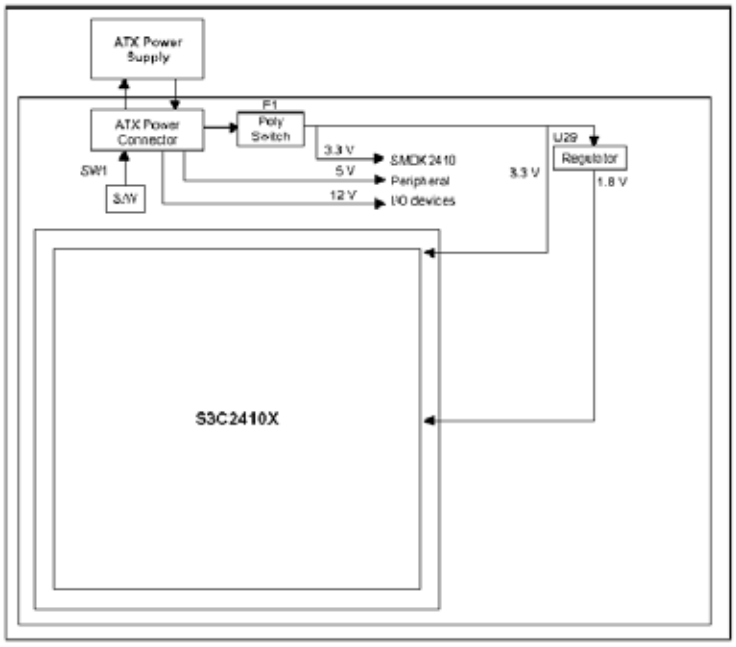


图 3.3 系统电源框图

开发板由开关电源供电，包括一个 5V/1A 和一对+12/100mA、-12V/100mA 的输入。5V 电源经线形稳压器（LDO）后得到 1.8V 电压供 CPU 的核心工作，另外 5V 电压再经线形稳压器后得到 3.3V 供 CPU I/O 口部分和其他接口器件工作。±12V 经 78L05 和 79L05 稳压后供 A/D 和 D/A 部分的运算放大器和 DAC 器件工作。在电源的保护方面，不仅在开关电源的 AC 输入端接有保险管；而且，开关电源的 5V 输出端串有可恢复的保险管，当系统连接出错而导致 5V 电流过大时，可恢复保险管会因过热而自动断开，当冷却了以后，又自动连接，实现系统的过流自动保护。

3.2.2 系统配置

1．时钟配置

板上时钟连接如图 3.4 所示：

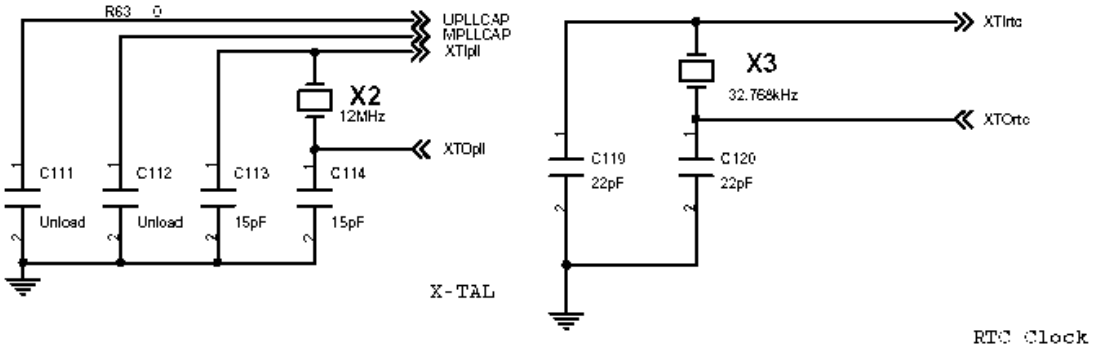


图 3.4 时钟连接

系统的时钟设置除了与外接的晶振有关系，还与芯片引脚 OM[3：2]有关，关系如表 3.1 所示：

表 3.1 时钟设置与芯片引脚 OM 关系

PIN FUNCTIONS	OM[3:2]		DESCRIPTIONS
Clock source selection	0	0	MPLL:XTAL. UPLL:XTAL
	0	1	MPLL:XTAL. UPLL:EXTCLK
	1	0	MPLL: EXTCLK. UPLL:XTAL
	1	1	MPLL: EXTCLK. UPLL: EXTCLKL

在我们的板上，OM[3：2]固定接为地，所以，CPU 的系统时钟和 USB 口的时钟都来自 12M 晶振，RTC 时钟来自外接的 32.768kHz 的晶振。

2．复位逻辑

复位逻辑电路如图 3.5 所示：

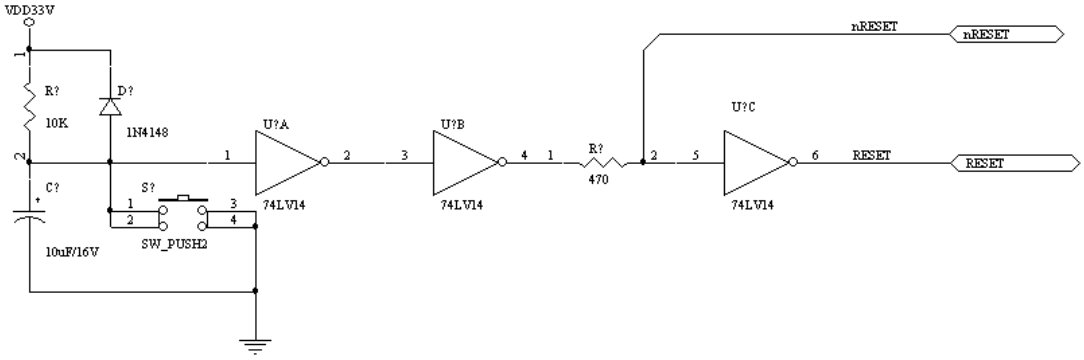


图 3.5 复位逻辑电路

引脚 nRESET 连接到 CPU 的复位端，上电或手动复位时，复位电路都可以在 nRESET 产生大于 10mS 的复位电平，保证系统可靠复位；JTAG 电路的复位引脚 nTRST 通过 4.7K 电阻和 nREST 连接，通过 JTAG 接口也可实现对系统的复位。

3．启动分区（BOOT ROM BANK0）

系统选择电路连接如图 3.6 所示：

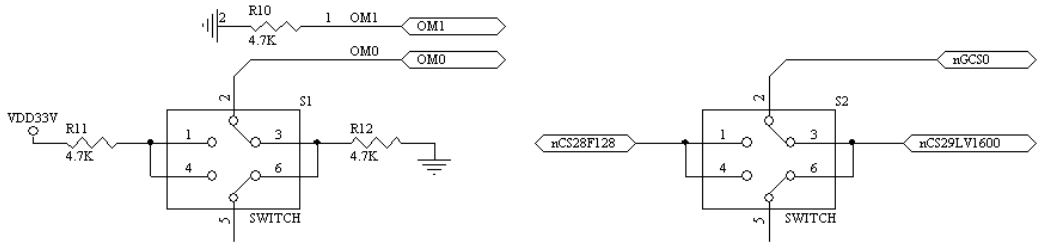


图 3.6 系统选择电路连接

通过双掷开关 S1、S2 可以设定系统的启动方式，关系如图 3.7 所示：

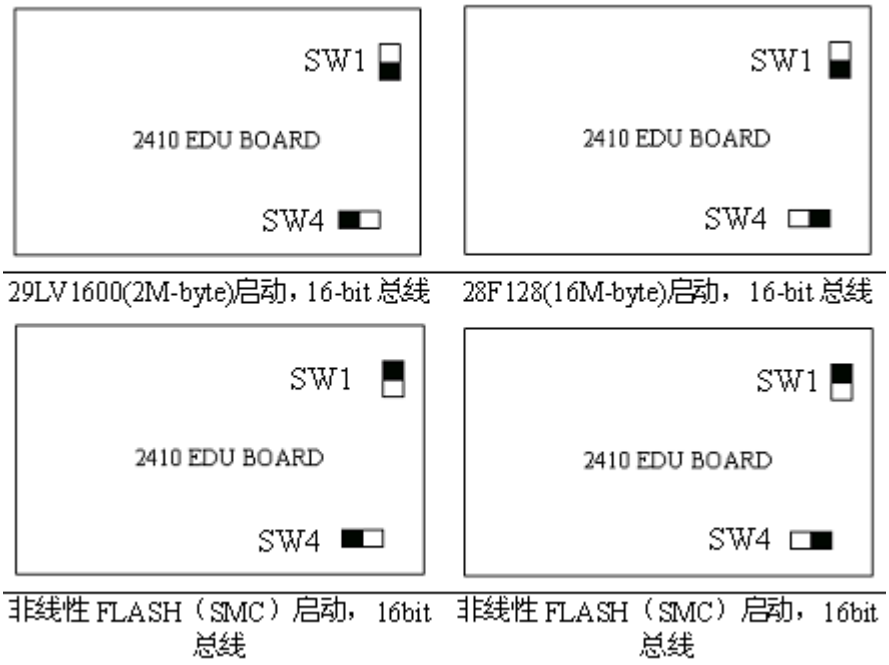


图 3.7 双掷开关设定系统的启动方式

4．LCD 和触摸屏接口

板载 SHARP 3.5″ TFT 液晶屏 LQ035Q7DB02，320×240，262,144 色，White LED 背光，带触摸屏。而且我们也留出了 LCD 的扩展接口，如图 3.8 所示，供用户扩展之用：

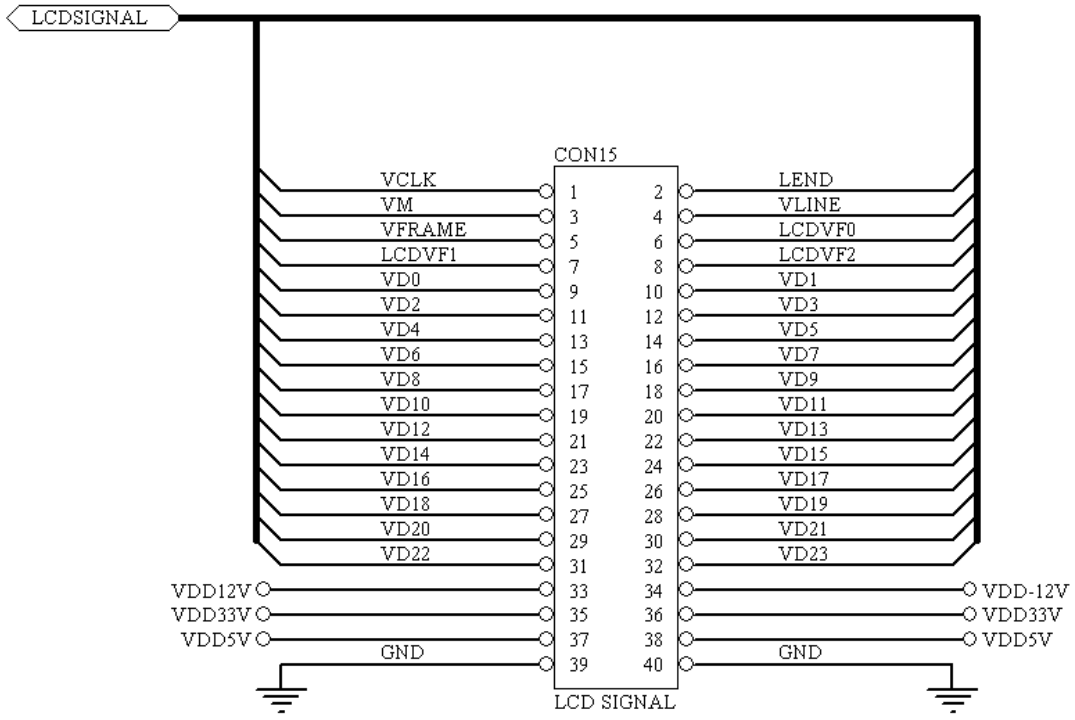


图 3.8 LCD 的扩展接口

SHARP 液晶自带四线电阻式触摸屏，可以直接和 2410 的触摸屏驱动电路连接，触摸位置直接用 CPU 内置的 ADC 电路采样而得。

板载触摸屏电路如图 3.9 所示：

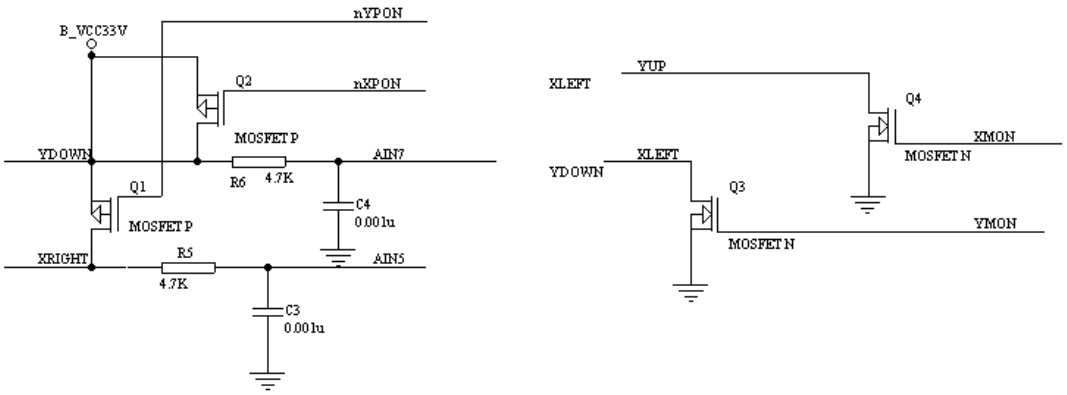


图 3.9 板载触摸屏电路

5．键盘和 SPI 接口

板载键盘扩展我们用的是 SPI 接口的键盘显示控制芯片 ZLG7289，电路连接关系如图 3.10 所示：

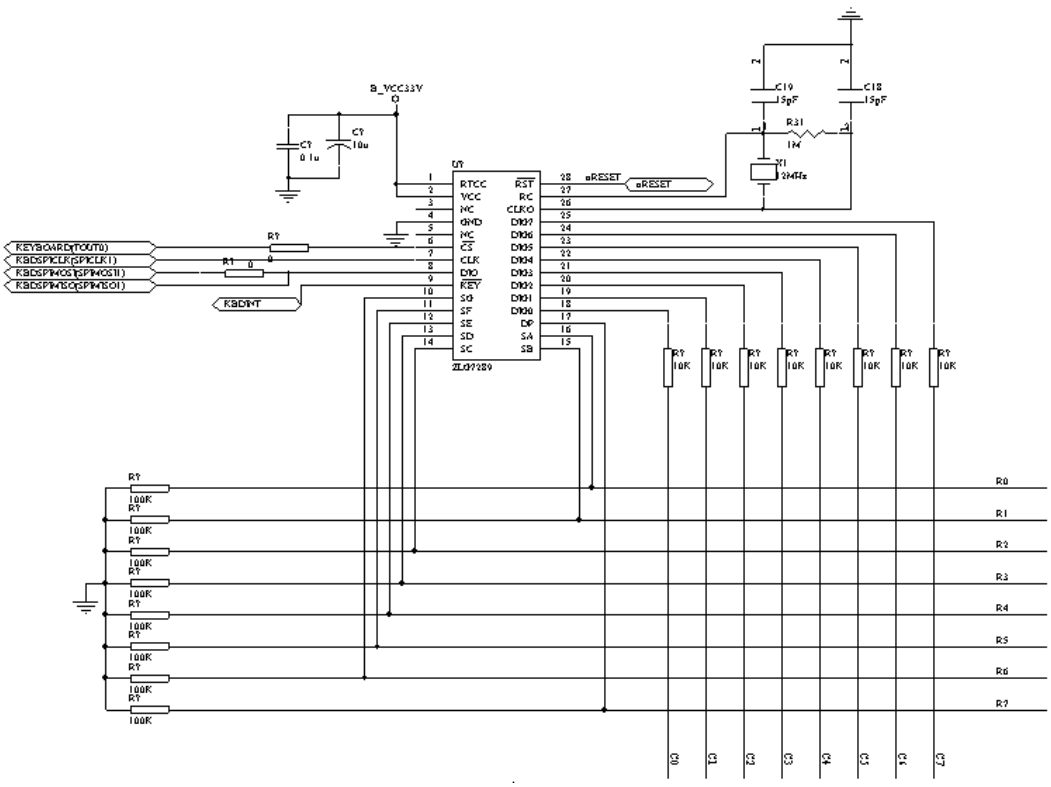


图 3.10 键盘和 SPI 接口电路连接关系

板载键盘的按键和芯片扫描的行线和列线之间的对应关系如表 3.2 所示：

表 3.2 按键和扫描的行列线间的对应

按键	键值	行线/列线	按键	键值	行线/列线
NumLock	32	R0/C0	5	42	R1/C1
/	40	R0/C1	6→	50	R1/C2
*	48	R0/C2	1/End	35	R1/C3
-	55	R0/C3	2/↓	43	R1/C4
7/Home	33	R0/C4	3/Pg Dn	51	R1/C5
8/↑	41	R0/C5	0/Ins	44	R1/C6
9/Pa Up	49	R0/C6	./Del	52	R1/C7
+	57	R0/C7	Enter	59	R2/C0
4/←	34	R1/C0			

6．A/D、D/A 转换接口

由于 CPU 内部已经内置了 8 个通道的 10-bit ADC 转换器,所以在我们的系统里面没有扩展另外的 ADC 转换芯片,而直接采用的是 CPU 内置的 ADC，AD 的参考电压为 3.3V;D/A 转换部分我们扩展了最常用的 8-bit DAC 转换芯片 DAC0832。为了方便用户测试和学生做实验，我们连接了多种信号作为 AD 的输入，可以是电位器调节的电压信号、温度传感器 LM35 输出的电压信号或者是 DA 的输出信号，这里 AD、DA 电路如图 3.11 所示：

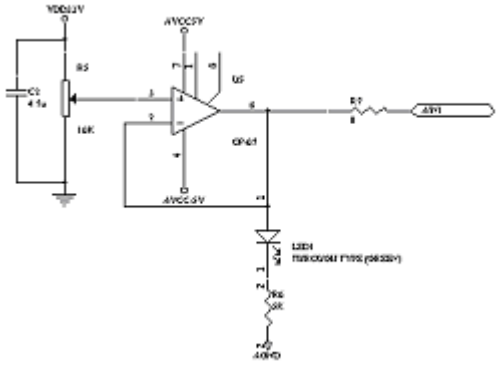


图 3.11（a）直接电压测量 AD 电路

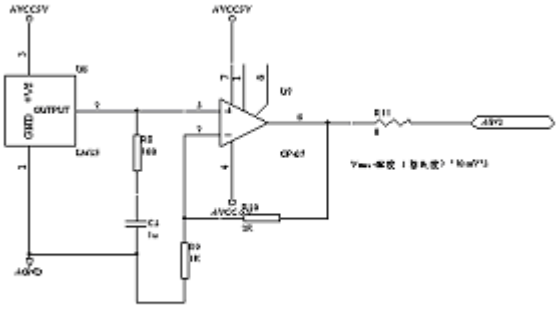


图 3.11（b）温度传感器 DA 电路

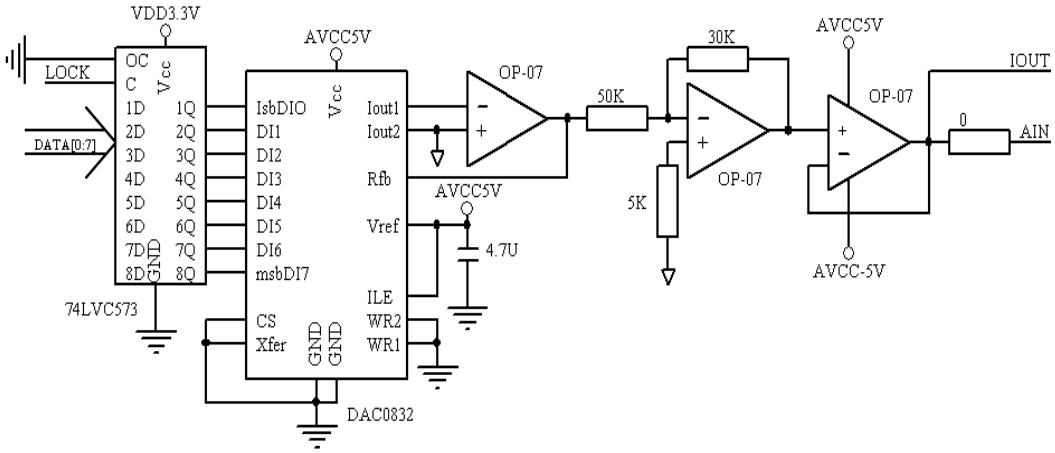


图 3.11（c）DA 电路

7．SmartMedia Card （NAND Flash memory ） 卡接口

在系统设计中，我们采用了 SmartMedia Card 接口，它是与 NAND Flash 兼容的接口，可以插入 SAMSUNG 的 SmartMedia Card 直接从卡里的 NAND Flash 启动系统，接口如图 3.12 所示：

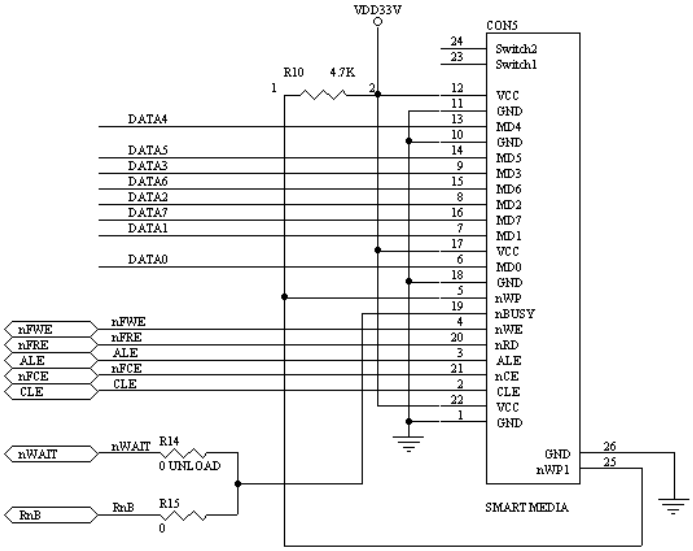


图 3.12 SmartMedia Card （NAND Flash memory ）卡接口

8．PCMCIA 接口

PCMCIA接口我们通过专用扩展芯片CL-PD6710扩展而得，芯片的片选读写连接到CPU的nGCS2引脚上，对应内存空间：0x10000000—0x17FFFFFF。接口如图3.13所示：

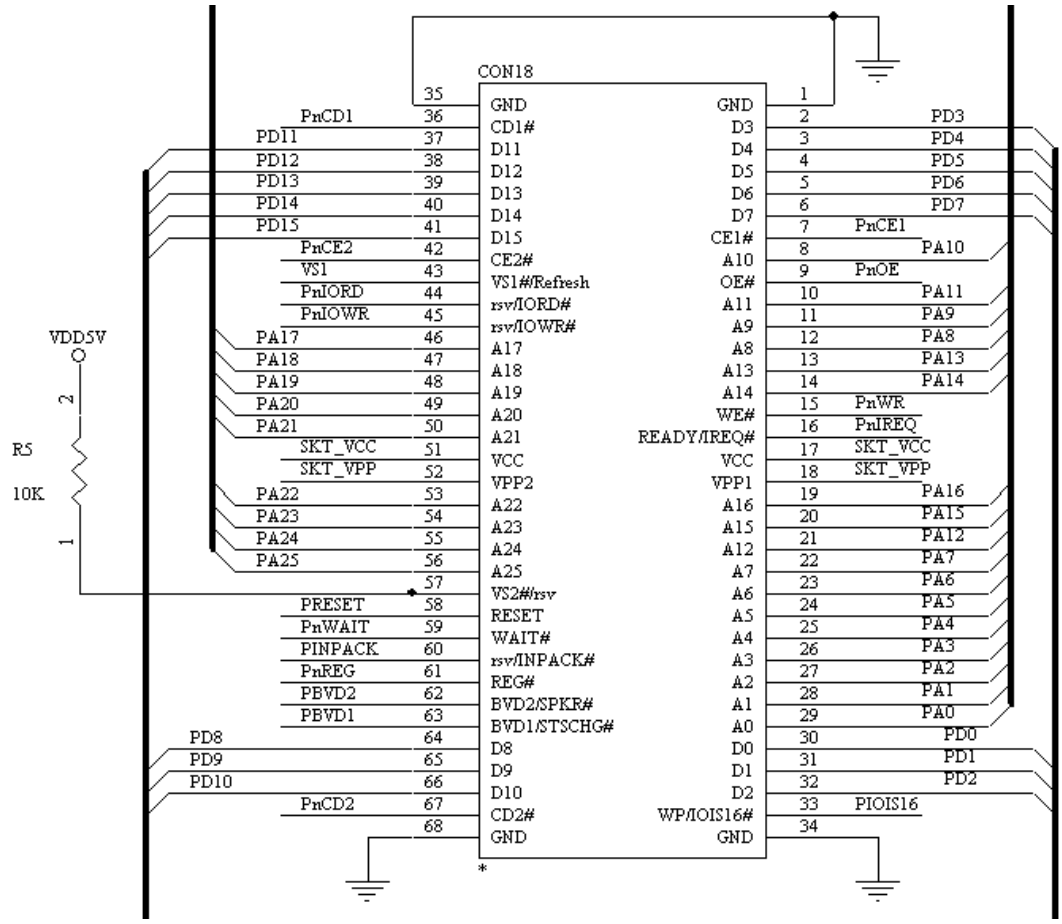


图 3.13　PCMCIA 接口

通过附带的 PCMCIA-CF 转接卡，可以直接连接 CF 卡设备，如无线网卡、存储设备等。

9．SD 卡主机（MMC）接口

接口电路如图 3.14 所示：

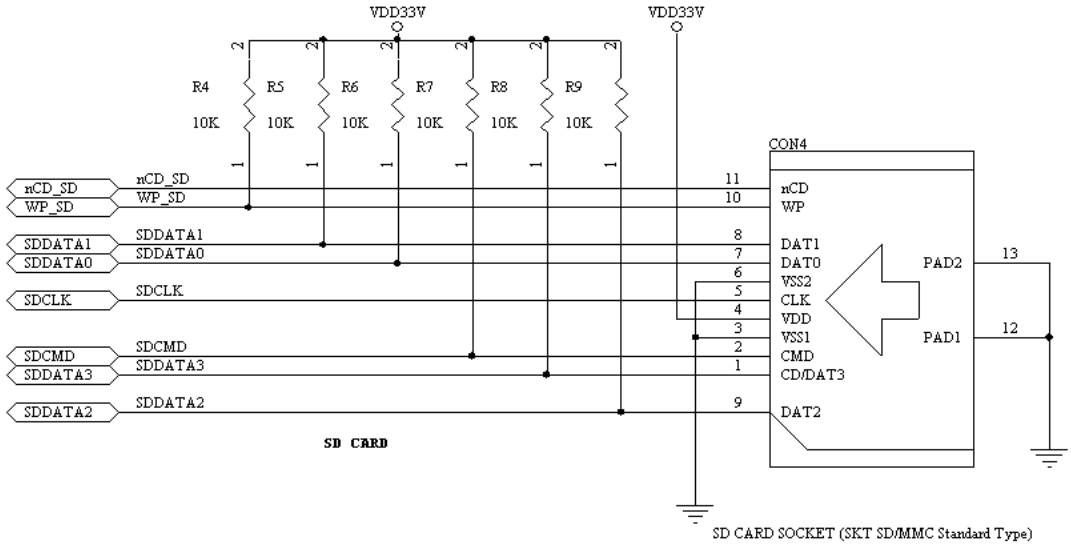


图 3.14　SD 卡主机（MMC）接口

10．IIC 接口

CPU 内置 IIC 总线控制器，为了方便用户测试 IIC 总线读写，板载两个 IIC 设备，一个是 IIC 接口的 EEPROM 24C16，为 16K-bit 的串行 EEPROM，方便用户存储一些小容量的数据，掉电不丢失；另一个 IIC 接口的 LED 数码管显示控制器 ZLG7290，通过控制器，控制 8 位 8 段数码管的动态扫描。

IIC EPROM 连接电路如图 3.15 所示：

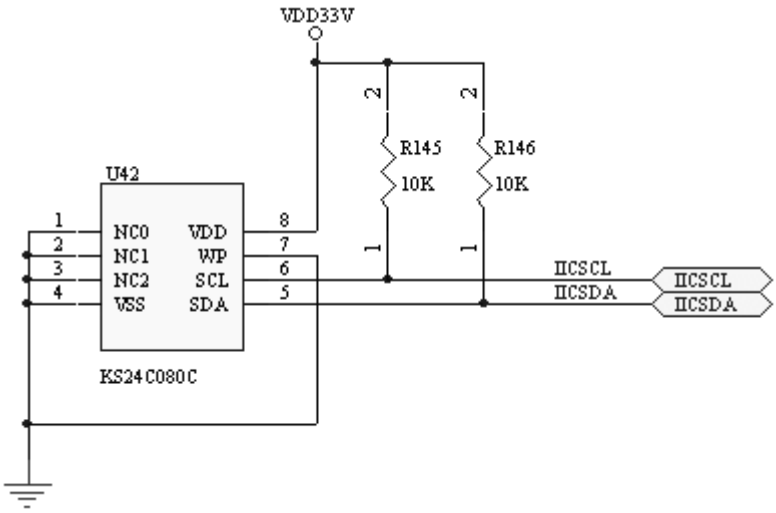


图 3.15 IIC EPROM 连接电路

IIC LED 控制器连接电路如图 3.16 所示：

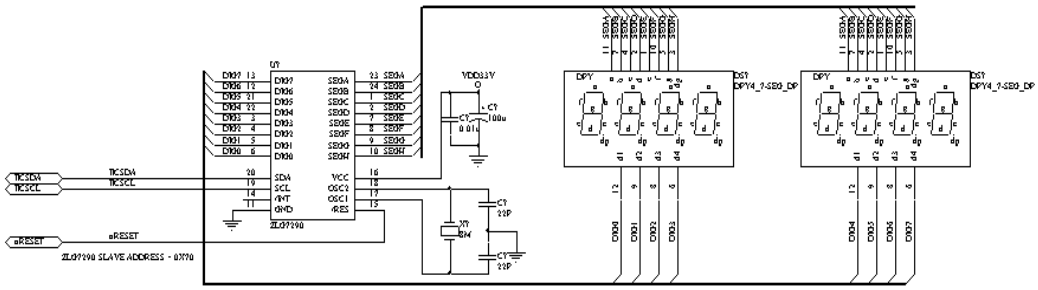


图 3.16　IIC LED 控制器连接电路

11．USB 接口

CPU 内置两个 USB 控制器，一个是 USB Host（主机）控制器，另外一个可以配置成 USB Host 或者 USB Device（设备）控制器。在板上，我们放置了 3 个 USB 的接口，两个是 HOST 的接口，一个是 DEVICE 的接口，第二个 USB 口的功能切换如果 USB 接口旁边的一个双掷开关来进行选择。电路连接如图 3.17 所示，当 S4 拨到 1 端时，第二个 USB 口置为 Device 的功能，拨到 3 端时，置为 Host 的功能。

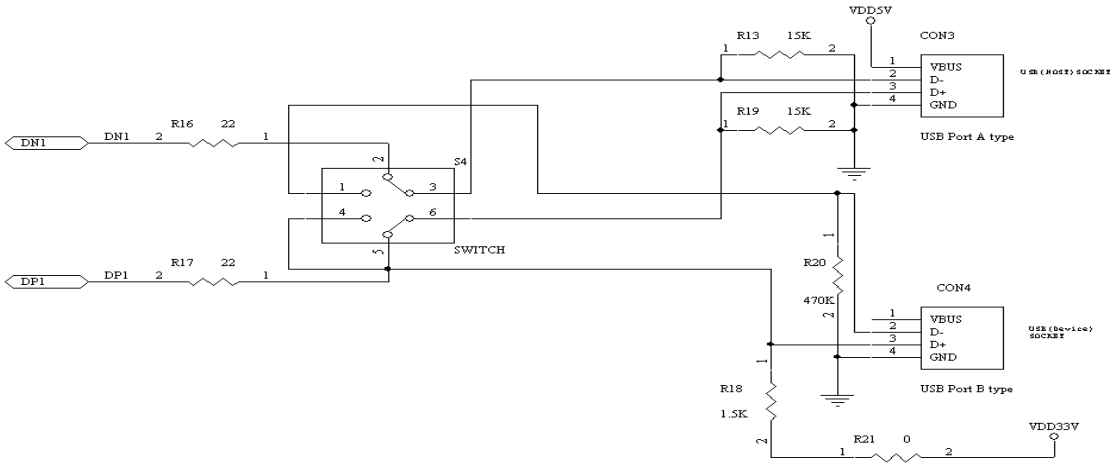


图 3.17　USB 接口

12．UART 接口和 irDA 接口

CPU 内置三个异步串口，第三个串口可以选为通用异步串口或红外接口，可以通过板上串口旁边的双掷开关 S6 来选择。串行接口如图 3.18 所示：

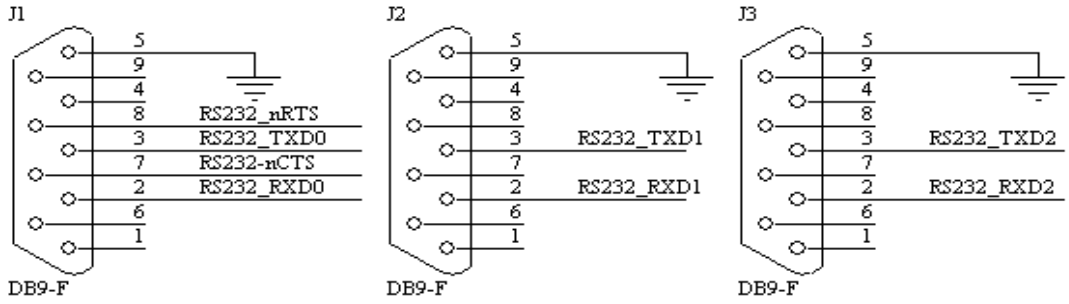


图 3.18 串行接口

13．音频接口

CPU 内置 IIS 总线，我们通过扩展音频芯片 UDA1341TS 实现对音频的录放功能，而且为了方便用户的使用，我们对双通道的音频信号增加了功放电路及 2W 的喇叭输出，增强现场效果，当然，板载喇叭输出也可以电路板左下角喇叭旁边的双掷开关 S5 关闭。

3.3 FLASH 芯片的烧录

平时，我们可以用仿真器（Multi-ICE 兼容）在 Windows 环境下的 ADS1.2 或 SDT2.5 软件下实现对系统软件、硬件的调试。当调试结束，我们想看到目标系统脱离 PC 机运行的情况时，我们便可以把我们的程序烧到目标系统的 Flash 芯片中，通过双掷开关 S1、S2 选择相应的启动方式，上电后，直接由相应的芯片启动系统程序。

3.3.1 建立硬件环境

当我们准备要往 FLASH 中烧录程序时，首先要先建立烧录的硬件环境，电路连接如图 3.19 所示：

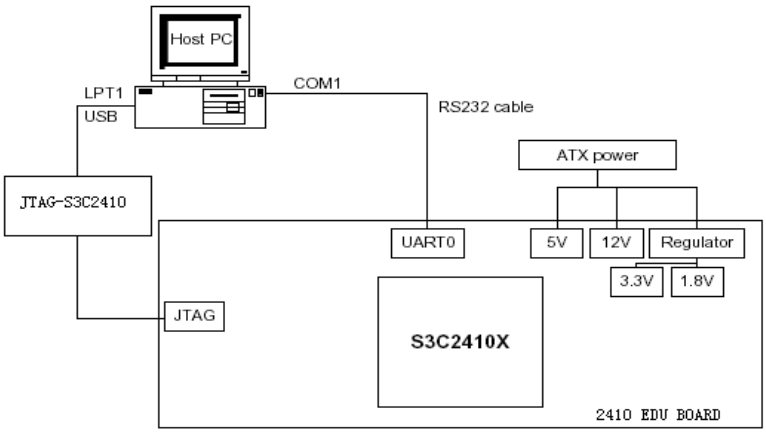


图 3.19 硬件环境连线

步骤：1．首先，关闭系统电源；

2．再用给客户的包装中带的一个“JTAG-S3C2410”的连接板，一头连接计算机的并口，一头连接开发板的 20-P JTAG 接口，FLASH 的烧录程序就通过 JTAG 接口与 CPU 通信，控制 CPU 通过总线实现对相应的 FLASH 烧写；

3．连接计算机的串口到开发板的串口，可以通过计算机的超级终端观察开发板串口输出信息，验证 FLASH 烧录是否正确。串口一般设置为 8 位数据位、1 位停止位、波特率 115200、无流控；

4．开启电源。

3.3.2 芯片的烧录

在附带光盘里有一个“SJF”的目录，里面有个可执行文件“SJF.exe”，我们烧录 FLASH 就是用这个程序。这个程序通过 PC 的并口模拟 JTAG 时序和 CPU 通信，从而完成烧录。烧录文件格式默认是 BIN 格式，可以通过在 ADS 或 SDT 软件环境中做些设置使生成的 AXF 格式的文件转化为 BIN 格式，具体操作过程可以参考软件的手册。另一个需要注意的地方是在生成 BIN 格式文件的时候，在编译器连接的时候，应该设程序的起始地址从 0 地址开始，而不是和我们在仿真调试的时候一样，把程序地址设置在内存区域。

1．烧录 AM29LV1600

1）确认板上双掷开关如图 3.20 所示，如不是，请先关电源在正确设置 S1、S2 开关的位置，再打开系统电源

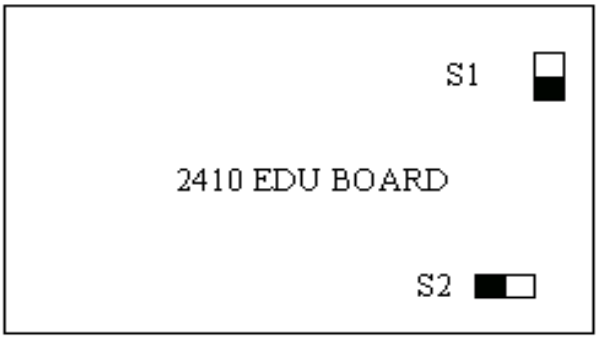


图 3.20 确认板上双掷开关

2）打开 Windows 的命令行窗口，把当前目录设置为 SJF 在硬盘上的对应目录，运行

sjf /f：2410mon.bin

其中 2410mon.bin 为目录下的例子目标代码，用户可以用自己的目标代码代替，按界面提示选择芯片型号为 AM29LV00，起始地址为 0，运行窗口如图 3.21 所示：

```
D:\SJF>sjf /f:2410mon.bin

+-------------------------------------+
|     SEC JTAG FLASH(SJF) v 0.4       |
|     (S3C2410X & SMDK2410 B/D)       |
+-------------------------------------+
Usage: SJF /f:<filename> /d=<delay>
> S3C2410X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog   2:AM29LV1600 Prog   3:Memory Rd/Wr
 4:Exit
Select the function to test:2

[AM29LV1600 Writing Program]
NOTE: AM29LV1600BB needs 4 step sequences for 1 half-word data.
      So,the program time is twice of Starata flash(2 step sequences).
[Check AM29LV1600]
Manufacture ID=   1(0x0001), Device ID(0x2249)=2249

Image Size:0h~5030h

Available Target Offset:
    0x0, 0x4000, 0x6000, 0x8000,0x10000,0x20000,0x30000,0x40000,
0x50000,0x60000,0x70000,0x80000,0x90000,0xa0000,0xb0000,0xc0000,
0xd0000,0xe0000,0xf0000
Input target offset:0

SectorOffset=0x0
SectorSize  =0x4000
Erase the sector:0x0.
Sector Erase is started!
Start of the sector data writing.
0 100 200 300 400 500 600 700 800 900 a00 b00 c00 d00 e00 f00 1000 1100 1200 13(
0 1400 1500 1600 1700 1800 1900 1a00 1b00 1c00 1d00 1e00 1f00 2000 2100 2200 23(
0 2400 2500 2600 2700 2800 2900 2a00 2b00 2c00 2d00 2e00 2f00 3000 3100 3200 33(
0 3400 3500 3600 3700 3800 3900 3a00 3b00 3c00 3d00 3e00 3f00
End of the sector data writing!!!
```

图 3.21 运行窗口

3）烧录完成以后，复位系统，便可以通过超级终端观察到系统输出的串口信息，如图

3.22 所示：

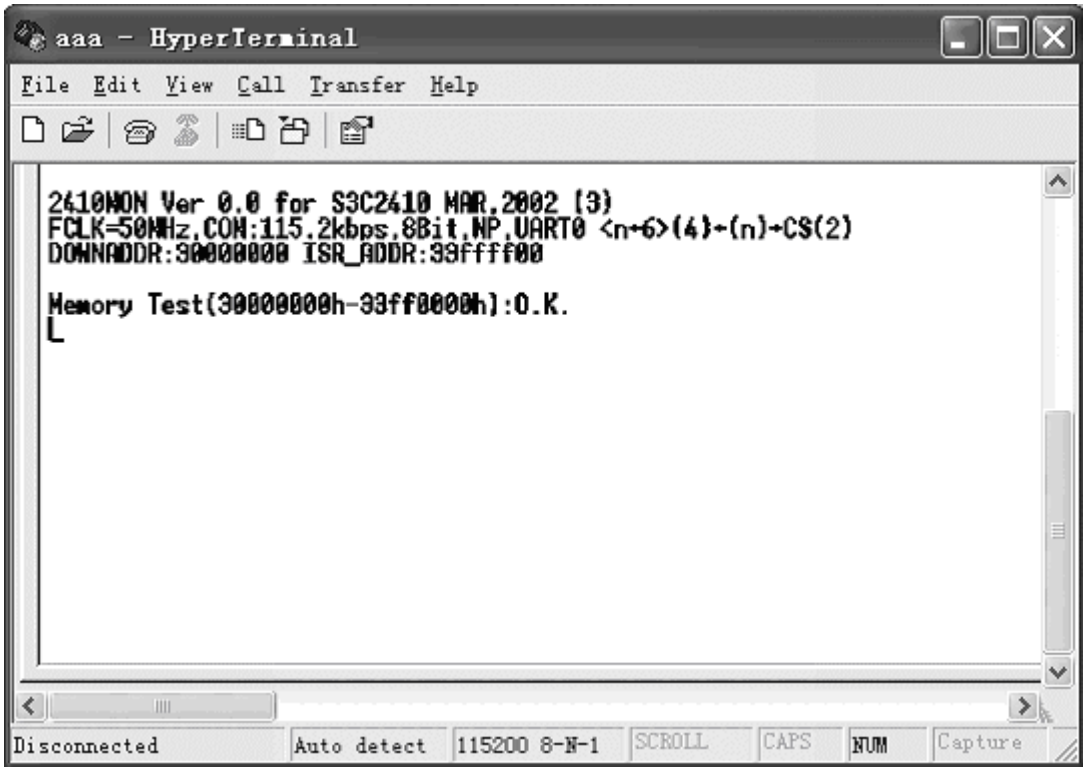


图 3.22 系统输出的串口信息

这个信息是对测试的目标代码而言，对用户自己的代码，信息自然不同。

2．烧录 28F128

1）确认板上双掷开关如图 3.23 所示，如不是，请先关电源在正确设置 S1、S2 开关的位置，再打开系统电源

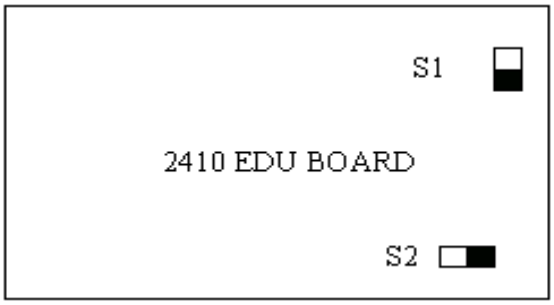


图 3.23 确认板上双掷开关

2）打开 Windows 的命令行窗口，把当前目录设置为 SJF 在硬盘上的对应目录，运行

sjf /f：2410mon.bin

其中 2410mon.bin 为目录下的例子目标代码，用户可以用自己的目标代码代替，按界面提示选择芯片型号为 28F128，起始地址为 0，运行窗口如图 3.24 所示：

```
D:\SJF>sjf /f:2410mon.bin

+--------------------------------------+
|     SEC JTAG FLASH(SJF) v 0.4        |
|     (S3C2410X & SMDK2410 B/D)        |
+--------------------------------------+
Usage: SJF /f:(filename) /d=(delay)
> S3C2410X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog   2:AM29LV1600 Prog   3:Memory Rd/Wr
 4:Exit
Select the function to test:1

[28F128J3A Flash JTAG Programmer]

Source size = 5030h

Available Target Offset Address:
0x0,0x20000,0x40000, ..., 0x1ce0000
Input target address offset [0x?] : 0
Target base address(0x00000000) = 0x0
Target offset      (0x0)        = 0x0
Target size        (0x20000*n)  = 0x5030
Manufacture ID=  89(0x0089), Device ID(0x0018)=  18

Erase the sector from 0x0.
Erase Sector:
clear block 0 ok
Block 00h Erase O.K.
Erase Sector 0 ok

Blank check is skipped.

Start of the data writing...
[100][200][300][400][500][600][700][800][900][a00][b00][c00][d00][e00][f00][1000
][1100][1200][1300][1400][1500][1600][1700][1800][1900][1a00][1b00][1c00][1d00][
1e00][1f00][2000][2100][2200][2300][2400][2500][2600][2700][2800][2900][2a00][2b
00][2c00][2d00][2e00][2f00][3000][3100][3200][3300][3400][3500][3600][3700][3800
][3900][3a00][3b00][3c00][3d00][3e00][3f00][4000][4100][4200][4300][4400][4500][
4600][4700][4800][4900][4a00][4b00][4c00][4d00][4e00][4f00][5000]
End of the data writing
```

图 3.24 运行窗口

3）烧录完成以后，复位系统，便可以通过超级终端观察到系统输出的串口信息，如图 3.25 所示：

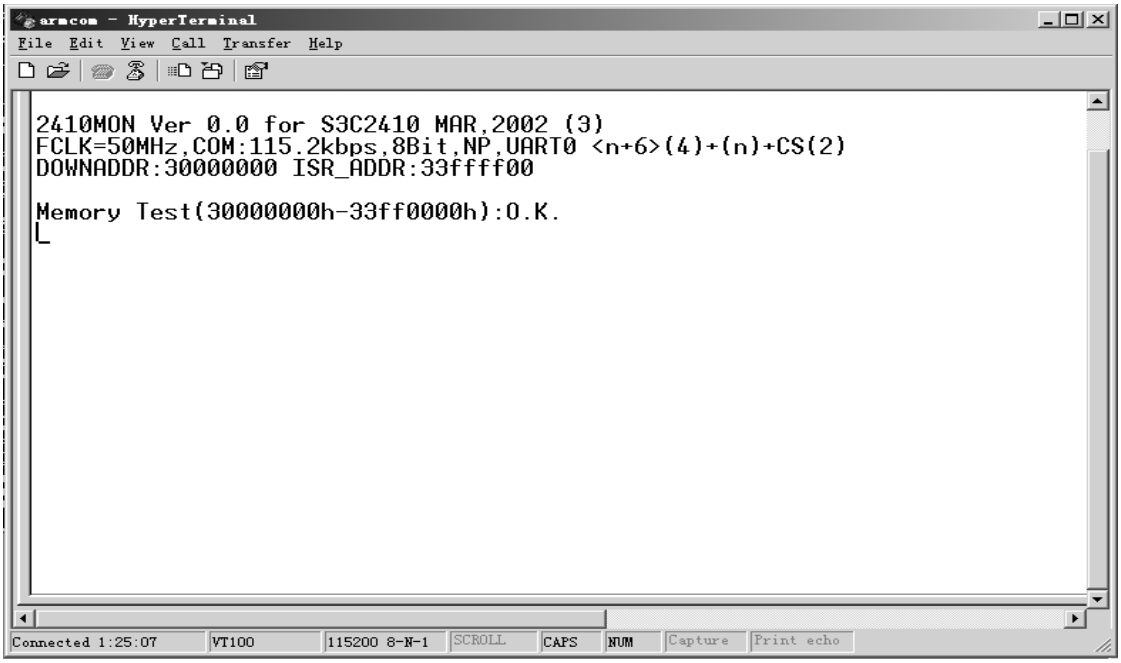


图 3.25 输出的串口信息

这个信息是对测试的目标代码而言，对用户自己的代码，信息自然不同。

3．烧录 K9F1208

1）确认板上双掷开关如图 3.26 所示，如不是，请先关电源在正确设置 S1、S2 开关的位置，再打开系统电源。

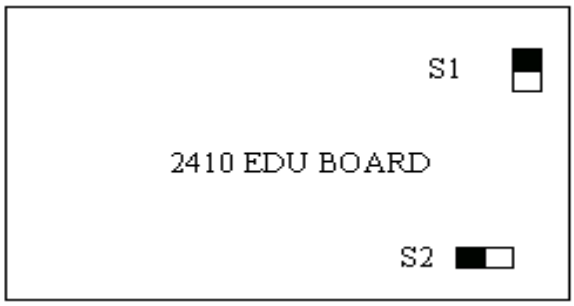


图 3.26 确认板上双掷开关

2）打开 Windows 的命令行窗口，把当前目录设置为 SJF 在硬盘上的对应目录，运行

sjf　/f：2410mon.bin

其中 2410mon.bin 为目录下的例子目标代码，用户可以用自己的目标代码代替，按界面提示选择芯片型号为 K9S1208，起始地址为 0，运行窗口如图 3.27 所示：

```
D:\SJF>sjf /f:2410mon.bin /d=8

+----------------------------------------+
!     SEC JTAG FLASH(SJF) v 0.4          !
!     (S3C2410X & SMDK2410 B/D)          !
+----------------------------------------+
Usage: SJF /f:<filename> /d=<delay>
> S3C2410X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog   2:AM29LV1600 Prog   3:Memory Rd/Wr
 4:Exit
Select the function to test:0

[K9S1208 NAND Flash JTAG Programmer]
K9S1208 is detected. ID=0xec76
 0:K9S1208 Program      1:K9S1208 Pr BlkPage   2:Exit
Select the function to test :0

[SMC(K9S1208V0M) NAND Flash Writing Program]

Source size:0h~502fh

Available target block number: 0~4095
Input target block number:0
target start block number      =0
target size         (0x4000*n) =0x8000
STATUS:Epppppppppppppppppppppppppppppppp
Epppppppppppppppppppppppppppppppp
 0:K9S1208 Program      1:K9S1208 Pr BlkPage   2:Exit
Select the function to test :2

D:\SJF>_
```

图 3.27 运行窗口

3）烧录完成以后，复位系统，便可以通过超级终端观察到系统输出的串口信息，图略。

第四章 ADS 开发及 Multi-ICE 仿真器应用

在这一章里，将介绍 ARM 开发软件 ADS（ARM Developer Suite）。通过学习如何在 CodeWarrior IDE 集成开发环境下编写，编译一个工程的例子，使读者能够掌握在 ADS 软件平台下开发用户应用程序。本章还描述了如何使用 AXD 调试工程，并通过 JTAG 仿真，最终在超级终端中查看结果。

本章主要内容有：

- ADS 软件组成介绍
- 使用 ADS 创建工程
- 用 AXD 进行代码调试
- 用 JTAG 仿真器
- 用超级终端查看结果

4.1 ADS 集成开发环境组成介绍

ARM ADS 全称为 ARM Developer Suite。是 ARM 公司推出的新一代 ARM 集成开发工具。现在 ADS 的最新版本是 1.2，它取代了早期的 ADS1.1 和 ADS1.0。它除了可以安装在 Windows NT4，Windows 2000，Windows 98 和 Windows 95 操作系统下，还支持 Windows XP 和 Windows Me 操作系统。

ADS 由命令行开发工具，ARM 时实库，GUI 开发环境（Code Warrior 和 AXD），实用程序和支持软件组成。有了这些部件，用户就可以为 ARM 系列的 RISC 处理器编写和调试自己的开发应用程序了。

下面就详细介绍一下 ADS 的各个组成部分。

4.1.1 命令行开发工具

命令行开发工具完成将源代码编译，链接成可执行代码的功能。ADS 提供下面的命令行开发工具：

armcc：armcc 是 ARM C 编译器。这个编译器通过了 Plum Hall C Validation Suite 为 ANSI C 的一致性测试。armcc 用于将用 ANSI C 编写的程序编译成 32 位 ARM 指令代码。因为 armcc 是我们最常用的编译器，所以对此作一个详细的介绍。

在命令控制台环境下，输入命令：

armcc –help

可以查看 armcc 的语法格式以及最常用的一些操作选项

armcc 最基本的用法为： armcc [options] file1 file2 ... filen

这里的 option 是编译器所需要的选项，fiel1,file2…filen 是相关的文件名。这里简单介绍一些最常用的操作选项。

-c：表示只进行编译不链接文件；

-C：（注意：这是大写的 C）禁止预编译器将注释行移走；

-D<symbol>：定义预处理宏，相当于在源程序开头使用了宏定义语句#define symbol ，这里 symbol 默认为 1；

-E：仅仅是对 C 源代码进行预处理就停止；

-g<options>：指定是否在生成的目标文件中包含调试信息表；

-I<directory>：将 directory 所指的路径添加到#include 的搜索路径列表中去；

-J<directory>：用 directory 所指的路径代替默认的对#include 的搜索路径；

-o<file>：指定编译器最终生成的输出文件名。

-O0：不优化；

-O1：这是控制代码优化的编译选项，大写字母 O 后面跟的数字不同，表示的优化级别就不同，-O1 关闭了影响调试结果的优化功能；

-O2：该优化级别提供了最大的优化功能；

-S：对源程序进行预处理和编译，自动生成汇编文件而不是目标文件；

-U<symbol>：取消预处理宏名，相当于在源文件开头，使用语句#undef symbol;

-W<options>：关闭所有的或被选择的警告信息；

有关更详细的选项说明，读者可查看 ADS 软件的在线帮助文件。

armcpp：armcpp 是 ARM C++编译器。它将 ISO C++ 或 EC++ 编译成 32 位 ARM 指令代码。

tcc：tcc 是 Thumb C 编译器。该编译器通过了 Plum Hall C Validation Suite 为 ANSI 一致性的测试。tcc 将 ANSI C 源代码编译成 16 位的 Thumb 指令代码。

tcpp：tcpp 是 Thumb C++ 编译器。 它将 ISO C++ 和 EC++ 源码编译成 16 位 Thumb 指令代码。

armasm：armasm 是 ARM 和 Thumb 的汇编器。它对用 ARM 汇编语言和 Thumb 汇编语言写的源代码进行汇编。

armlink：armlink 是 ARM 连接器。该命令既可以将编译得到的一个或多个目标文件和相关的一个或多个库文件进行链接，生成一个可执行文件，也可以将多个目标文件部分链接成一个目标文件，以供进一步的链接。ARM 链接器生成的是 ELF 格式的可执行映像文件。

armsd：armsd 是 ARM 和 Thumb 的符号调试器。它能够进行源码级的程序调试。用户可以在用 C 或汇编语言写的代码中进行单步调试，设置断点，查看变量值和内存单元的内容。

4.1.2 ARM 运行时库

1．运行时库类型和建立选项

ADS 提供以下的运行时库来支持被编译的 C 和 C++代码：

ANSI C 库函数：这个 C 函数库是由以下几部分组成：

- ✧ 在 ISO C 标准中定义的函数；
- ✧ 在 semihosted 环境下（semihosting 是针对 ARM 目标机的一种机制，它能够根据应用程序代码的输入/输出请求，与运行有调试功能的主机通讯。这种技术允许主机为通常没有输入和输出功能的目标硬件提供主机资源）用来实现 C 库函数的与目标相关的函数；
- ✧ 被 C 和 C++编译器所调用的支持函数。

ARM C 库提供了额外的一些部件支持 C++，并为不同的结构体系和处理器编译代码。

C++库函数：C++库函数包含由 ISO C++库标准定义的函数。C++库依赖于相应的 C 库实现与特定目标相关的部分，在 C++库的内部本身是不包含与目标相关的部分。这个库是由以下几部分组成的：

- ✧ 版本为 2.01.01 的 Rogue Wave Standard C++库；

- ✧ C++编译器使用的支持函数；
- ✧ Rogue Wave 库所不支持的其他的 C++函数。

正如上面所说，ANSI C 库使用标准的 ARM semihosted 环境提供例如，文件输入/输出的功能。Semihosting 是由已定义的软件中断（Software Interrupt）操作来实现的。在大多数的情况下，semihosting SWI 是被库函数内部的代码所触发，用于调试的代理程序处理 SWI 异常。调试代理程序为主机提供所需要的通信。Semihosted 被 ARMulator ，Angel 和 Multi-ICE 所支持。用户可以使用在 ADS 软件中的 ARM 开发工具去开发用户应用程序，然后在 ARMulator 或在一个开发板上运行和调试该程序。

用户可以把 C 库中的与目标相关的函数作为自己应用程序中的一部分，重新进行代码的实现。这就为用户带来了极大的方便，用户可以根据自己的执行环境，适当的裁剪 C 库函数。除此之外，用户还可以针对自己的应用程序的要求，对与目标无关的库函数进行适当的裁剪。

在 C 库中有很多函数是独立于其他函数的，并且与目标硬件没有任何依赖关系。对于这类函数，用户可以很容易地从汇编代码中使用它们。在建立自己的用户应用程序的时候，用户必须指定一些最基本的操作选项。例如：

- ✧ 字节顺序，是大端模式（big endian：字数据的高字节存放在低地址，低字节存放在高地址），还是小端模式（little endian：字数据的高字节存放在高地址，低字节存放在低地址）；
- ✧ 浮点支持：可能是 FPA，VFP，软件浮点处理或不支持浮点运算；
- ✧ 堆栈限制：是否检查堆栈溢出；
- ✧ 位置无关（PID）：数据是从与位置无关的代码还是从与位置相关的代码中读/写，代码是位置无关的只读代码还是位置相关的的只读代码。

当用户对汇编程序，C 程序或 C++程序进行链接的时候，链接器会根据在建立时所指定的选项，选择适当的 C 或 C++运行时库的类型。选项各种不同组合都有一个相应的 ANSI C 库类型。

2．库路径结构

库路径是在 ADS 软件安装路径的 lib 目录下的两个子目录。假设，ADS 软件安装在 D：\arm\adsv1_2 目录，则在 D：\arm\adsv1_2\lib 目录下的两个子目录 armlib 和 cpplib 是 ARM 的库所在的路径。

armlib：这个子目录包含了 ARM C 库，浮点代数运算库，数学库等各类库函数。与这些库相应的头文件在 D：\arm\adsv1_2\include 目录中。

cpplib：这个子目录包含了 Rogue Wave C++库和 C++支持函数库。Rogue Wave C++库和 C++支持函数库合在一起被称为 ARM C++库。与这些库相应的头文件安装在 D：\arm\adsv1_2\include 目录下。

环境变量 ARMLIB 必须被设置成指向库路径。另外一种指定 ARM C 和 ARM C++库路径的方法是，在链接的时候使用操作选项-libpath directory（directory 代表库所在的路径），来指明要装载的库的路径。无需对 armlib 和 cpplib 这两个库路径分开指明，链接器会自动从用户所指明的库路径中找出这两个子目录。

这里需要让读者特别注意的以下几点：

- ✧ ARM C 库函数是以二进制格式提供的；
- ✧ ARM 库函数禁止修改。如果读者想对库函数创建新的实现的话，可以把这个新的函数编译成目标文件，然后在链接的时候把它包含进来。这样在链接的时候，使用的是新的函数实现而不是原来的库函数。
- ✧ 通常情况下，为了创建依赖于目标的应用程序，在 ANSI C 库中只有很少的几个

函数需要实现重建。

- ✧ Rogue Wave Standard C++函数库的源代码不是免费发布的，可以从 Rogue Wave Software Inc.，或 ARM 公司通过支付许可证费用来获得源文件。

4.1.3 CodeWarrior 集成开发环境

CodeWarrior for ARM 是一套完整的集成开发工具，充分发挥了 ARM RISC 的优势，使产品开发人员能够很好的应用尖端的片上系统技术。该工具是专为基于 ARM RISC 的处理器而设计的，它可加速并简化嵌入式开发过程中的每一个环节，使得开发人员只需通过一个集成软件开发环境就能研制出 ARM 产品，在整个开发周期中,开发人员无需离开 CodeWarrior 开发环境，因此节省了在操作工具上花的时间,使得开发人员有更多的精力投入到代码编写上来。

CodeWarrior 集成开发环境（IDE）为管理和开发项目提供了简单多样化的图形用户界面。用户可以使用 ADS 的 CodeWarrior IDE 为 ARM 和 Thumb 处理器开发用 C，C++，或 ARM 汇编语言的程序代码。通过提供下面的功能，CodeWarrior IDE 缩短了用户开发项目代码的周期。

- ✧ 全面的项目管理功能；
- ✧ 子函数的代码导航功能，使得用户迅速找到程序中的子函数；
- ✧ 可以在 CodeWarrior IDE 为 ARM 配置在 4.1.1 中介绍的各种命令工具，实现对工程代码的编译，汇编和链接。

在 CodeWarrior IDE 中所涉及到的 target 有两种不同的语义。

目标系统（Target system）：是特指代码要运行的环境，是基于 ARM 的硬件。比如，要为 ARM 开发板上编写要运行在它上面的程序，这个开发板就是目标系统。

生成目标（Build target）：是指用于生成特定的目标文件的选项设置（包括汇编选项，编译选项，链接选项以及链接后的处理选项）和所用的文件的集合。CodeWarrior IDE 能够让用户将源代码文件，库文件还有其他相关的文件以及配置设置等放在一个工程中。每个工程可以创建和管理生成目标设置的多个配置。例如，要编译一个包含调试信息的生成目标和一个基于 ARM7TDMI 的硬件优化生成目标，生成目标可以在同一个工程中共享文件，同时使用各自的设置。

CodeWarrior IDE 为用户提供下面的功能：

- ✧ 源代码编辑器，它集成在 CodeWarrior IDE 的浏览器中，能够根据语法格式，使用不同的颜色显示代码；
- ✧ 源代码浏览器，它保存了在源码中定义的所有符号，能够使用户在源码中快速方便的跳转；
- ✧ 查找和替换功能，用户可以在多个文件中，利用字符串通配符，进行字符串的搜索和替换；
- ✧ 文件比较功能，可以使用户比较路径中的不同文本文件的内容。

ADS 的 CodeWarrior IDE 是基于 Metrowerks CodeWarrior IDE 4.2 版本的。它经过适当的裁剪以支持 ADS 工具链。针对 ARM 的配置面板为用户提供了在 CodeWarrior IDE 集成环境下配置各种 ARM 开发工具的能力，这样用户可以不用在命令控制台下就能够使用在 4.1.1 和将在 4.1.5 中介绍的各种命令。

以 ARM 为目标平台的工程创建向导，可以使用户以此为基础，快速创建 ARM 和 Thumb 工程。尽管大多数的 ARM 工具链已经集成在 CodeWarrior IDE，但是仍有许多功能在该集成环境中没有实现，这些功能大多数是和调试相关的，因为 ARM 的调试器没有集

成到 CodeWarrior IDE 中。由于 ARM 调试器（AXD）没有集成在 CodeWarrior IDE 中，这就意味着，用户不能在 CodeWarrior IDE 中进行断点调试和查看变量。对于熟悉 CodeWarrior IDE 的用户会发现，有许多的功能已经从 CodeWarrior IDE For ARM 中移走，比如快速应用程序开发模板等。

在 CodeWarrior IDE For ARM 中有很多的菜单或子菜单是不能使用的。下面介绍一下这些不能使用的选项。

1．View 菜单下不能使用的菜单选项有：

Processes，Expressions，Global Variable，Breakpoints，Registers。

2．Project 菜单不能使用的菜单选项：

Precompile 子菜单。因为 ARM 编译器不支持预编译的头文件。

3．Debug 菜单

该菜单中没有一个子菜单是可以使用的。

4．Browser 菜单中不能使用的菜单选项：

New Property，New Method 和 New Event Set。

5．Help menu 中不能用于 ADS 的菜单选项有：

CodeWarrior Help，Index，Search 和 Online Manuals。

有关 CodeWarrior IDE 中一些常用菜单的使用，将在后面的举例中具体说明的，在此，不在赘述。

4.1.4 ADS 调试器

调试器本身是一个软件，用户通过这个软件使用 debug agent 可以对包含有调试信息的，正在运行的可执行代码进行比如变量的查看，断点的控制等调试操作。

ADS 中包含有 3 个调试器：

- ✧ AXD（ARM eXtended Debugger）：ARM 扩展调试器；
- ✧ armsd（ARM Symbolic Debugger）：ARM 符号调试器；
- ✧ 与老版本兼容的 Windows 或 Unix 下的 ARM 调试工具，ADW/ADU（Application Debugger Windows/Unix）。

下面对在调试映像文件中所涉及到的一些术语做一个简单的介绍。

Debug target：在软件开发的最初阶段，可能还没有具体的硬件设备。如果要测试所开发的软件是否达到了预期的效果，这可以由软件仿真来完成。即使调试器和要测试的软件运行在同一台 PC 上，也可以把目标当作一个独立的硬件来看待。当然，也可以搭建一个 PCB 板，这个板上可以包含一个或多个处理器，在这个板上可以运行和调试应用软件。只有当通过硬件或者是软件仿真所得到的结果达到了预期的效果，才算是完成了应用程序的编写工作。

调试器能够发送以下指令：

1．装载映像文件到目标内存；

2．启动或停止程序的执行；

3．显示内存，寄存器或变量的值；

4．允许用户改变存储的变量值。

Debug agent：Debug agent 执行调试器发出的命令动作，比如：设置断点，从存储器中读数据，把数据写到存储器等。Debug agent 既不是被调试的程序，也不是调试器。在 ARM 体系中，它有这几种方式：Multi-ICE（Multi-processor in-circuit emulator），ARMulator 和 Angel。其中 Multi-ICE 是一个独立的产品，是 ARM 公司自己的 JTAG 在线仿真器，不

是由 ADS 提供的。

AXD 可以在 Windows 和 UNIX 下，进行程序的调试。它为用 C，C++，和汇编语言编写的源代码提供了一个全面的 Windows 和 UNIX 环境。

在后面的章节中，会结合具体实例为读者介绍如何使用 AXD 调试器。

4.1.5 实用程序

ADS 提供以下的实用工具来配合前面介绍的命令行开发工具的使用。

fromELF：这是 ARM 映像文件转换工具。该命令将 ELF 格式的文件作为输入文件，将该格式转换为各种输出格式的文件，包括 plain binary（BIN 格式映像文件），Motorola 32-bit S-record format（Motorola 32 位 S 格式映像文件）, Intel Hex 32 format（Intel 32 位格式映像文件），和 Verilog-like hex format（Verilog 16 进制文件）。FromELF 命令也能够为输入映像文件产生文本信息，例如代码和数据长度。

armar：ARM 库函数生成器将一系列 ELF 格式的目标文件以库函数的形式集合在一起，用户可以把一个库传递给一个链接器以代替几个 ELF 文件。

FLASH DOWNLOADER：用于把二进制映像文件下载到 ARM 开发板上的 FLASH 存储器的工具。

4.2 使用 ADS 创建工程

本节通过一个具体实例，为读者介绍如何使用该集成开发环境，利用 CodeWarrior 提供的建立工程的模板建立自己的工程，并学会如何进行编译链接，最终生成可执行文件。

4.2.1 建立一个工程

点击 WINDOWS 操作系统的“开始|程序|ARM Developer Suite v1.2 |Code Warrior for ARM Developer Suite”启动 Metrowerks Code Warrior，或双击“ADS 1.2”快捷方式启动。启动 ADS 1.2 如图 4.1 所示：

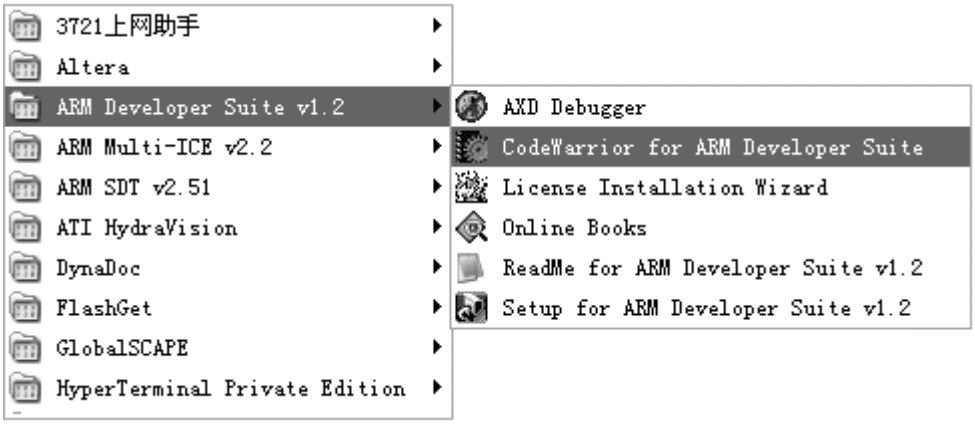


图 4.1　启动 ADS 1.2

在 CodeWarrior 中新建一个工程的方法有两种，可以在工具栏中单击“New”按钮，也可以在“File”菜单中选择“New…”菜单。这样就会打开一个如图 4.2 所示的对话框。

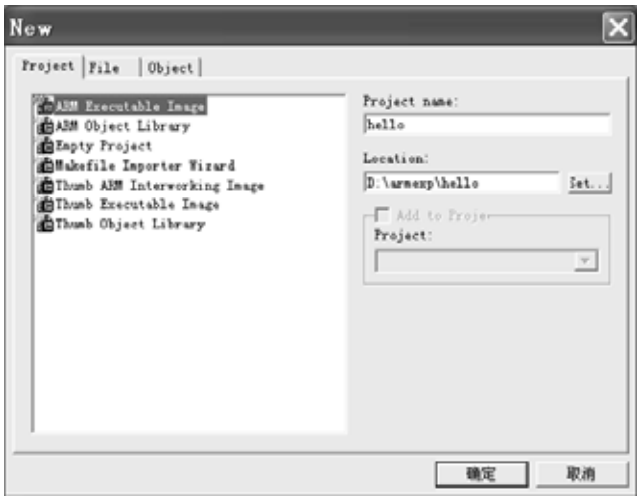


图 4.2　新建工程对话框

在这个对话框中为用户提供了 7 种可选择的工程类型。

ARM Executabl Image：用于由 ARM 指令的代码生成一个 ELF 格式的可执行映像文件；

ARM Object Library：用于由 ARM 指令的代码生成一个 armar 格式的目标文件库；

Empty Project：用于创建一个不包含任何库或源文件的工程；

Makefile Importer Wizard：用于将 Visual C 的 nmake 或 GNU make 文件转入到 CodeWarrior IDE 工程文件；

Thumb ARM Executable Image：用于由 ARM 指令和 Thumb 指令的混和代码生成一个可执行的 ELF 格式的映像文件；

Thumb Executable image：用于由 Thumb 指令创建一个可执行的 ELF 格式的映像文件；

Thumb Object Library：用于由 Thumb 指令的代码生成一个 armar 格式的目标文件库。

在这里选择 ARM Executable Image，在“Project name：”中输入工程文件名，本例为“HelloWorld”，点击“Location：”文本框的“Set…”按钮，浏览选择想要将该工程保存的路径，将这些设置好后，点击“确定”，即可建立一个新的名为 HelloWorld 的工程。这个时候会出现 HelloWorld.mcp 的窗口，如图 4.3 所示，有三个标签页，分别为 files,link order,target 默认的是显示第一个标签页 files。通过在该标签页点击鼠标右键，选中“Add Files…”可以把要用到的源程序添加到工程中。

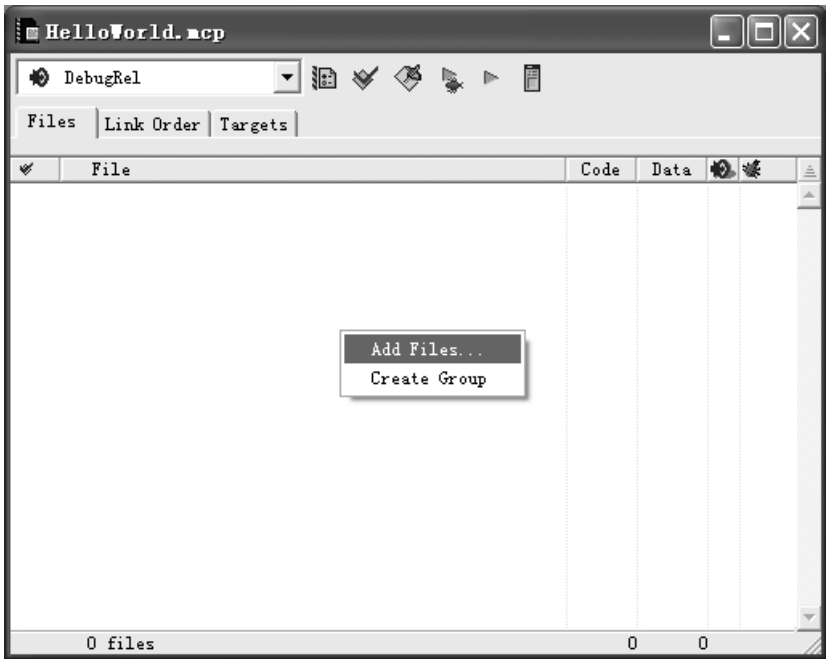


图 4.3　新建工程打开窗口

对于本例，由于所有的源文件都还没有建立，所以首先需要新建源文件。在“File”菜单中选择“New”，在打开的如图 4.2 所示的对话框中，选择标签页 File，在 File name 中输入要创建的文件名，输入“hello.s”，点击“确定”关闭窗口。在打开的文件编辑框中输入下面的汇编代码：

作为一个最简单的示例，hello.s 源文件如下所示。

```
        AREA    Example1,CODE,READONLY   ；声明代码段 Example1
        ENTRY                            ；标识程序入口
        CODE32                           ；声明 32 位 ARM 指令
START   MOV     R0,#15                       ；设置参数
        MOV     R1,#8
        ADDS    R0,R0,R1                 ；R0=R0+R1
        B       START
        END
```

在这里还有一个细节，希望读者注意。在建立好一个工程时，默认的 target 是 DebugRel，还有另外两个可用的 target，分别为 Realse 和 Debug，这三个 target 的含义分别为：

DebugRel：使用该目标，在生成目标的时候，会为每一个源文件生成调试信息；

Debug：使用该目标为每一个源文件生成最完全的调试信息；

Release：使用该目标不会生成任何调试信息。

在本例中，使用默认的 DebugRel 目标。现在已经新建了一个源文件，要把两个源文件添加到工程中去。

为工程添加源码常用的方法有两种，既可以使用如图 4.3 所示方法，也可以在“Project”菜单项中，选择“Add Files…”，这两种方法都会打开文件浏览框，用户可以把已经存在的文件添加到工程中来。当选中要添加的文件时，会出现一个对话框，如图 4.4 所示，询问用户把文件添加到何类目标中，在这里，我们选择 DebugRel 目标。把刚才创建的文件添加到工程中来。

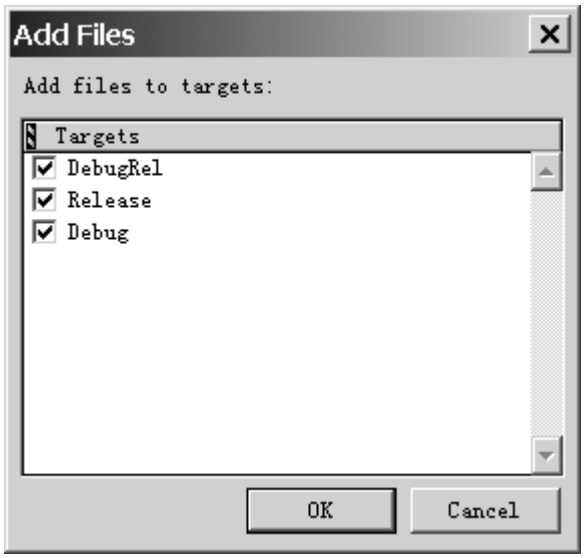


图 4.4　选择添加文件到指定目标

到目前为止，一个完整的工程已经建立，下面该对工程进行编译和链接工作。

4.2.2 配置生成目标

1．打开项目“hello.mcp”,在此项目窗口中，打开目标选择下拉表框，选择 Debug 生成目标，如图 4.5 所示；单击右侧的 Target Setting（此时已变为 Debug Setting）图标。如图 4.6 所示。

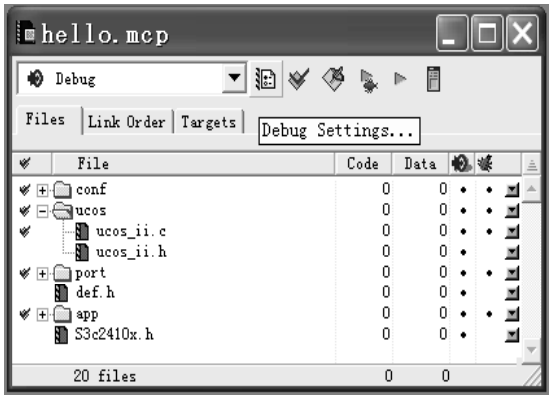


图 4.5 准备设置 Target Setting

2．在 Debug Setting 中包括 6 个面板，这里我们选择如下面板设置相关的生成选项：

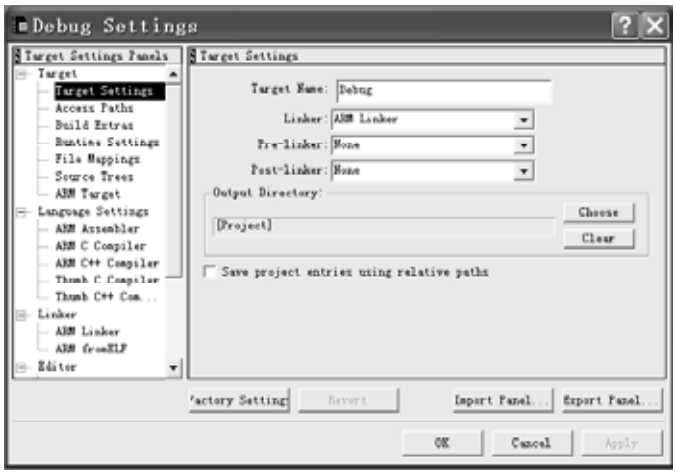


图 4.6 Target Setting 对话框

1）设置生成目标的基本选项（Target Settings），这里请按照图 4.6 对该选项进行相应的设置。

2）编译器的选项设置（Language Settings）。Language Settings 目录下选 ARM C Compiler，由于目标板采用的 S3C2410ARM 芯片属于 ARM9 系列，这里需要将该选项中的各个子选项对话框的“Target”或“Target and Source”选项卡下的“Architecture or Processor”对话框设定在“ARM920T”。如图 4.7 所示：

图 4.7 Language Settings 对话框

3）连接器的选项设置（Linker） 在 Target Settings Panels 列表框中选择 Linker 选项，再在其下选择 ARM Linker，即可得到连接器的选项设置对话框，如图 4.8 所示。

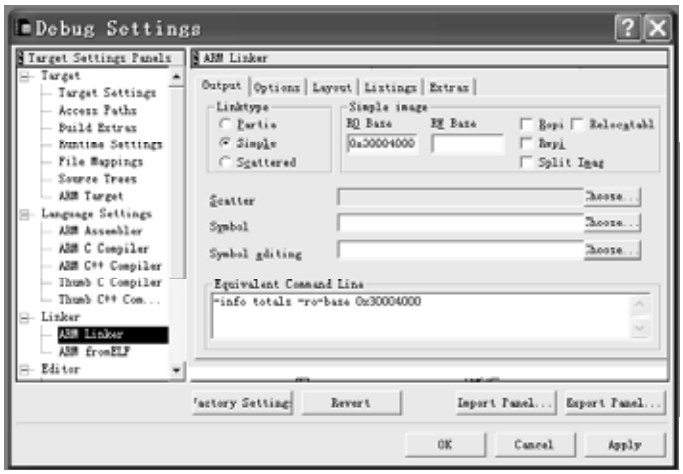


图 4.8 连接器选项设置

OUTPUT 选项卡：该选项卡用来控制连接器进行连接操作的类型。其中 Linktype 选项组中的单选按钮确定使用的连接方式。这里选择 Simple，连接器将根据连接器选项中指定的地址映射方式，生成简单的 ELF 格式的映像文件，所生成的映像文件中的地址映射关系比较简单.当选择 Simple 连接类型时，需要设置下列的连接器选项，如图 4.8 所示。

RO Base 文本框中填入 0x30000000。地址 0x30000000 是开发板上 SDRAM 的真实地址，是由系统的硬件决定的；RW Base 文本框中填入 0x31000000 指的是系统可读写内存的地址。也就是说，在 0x30000000－0x31000000 之间是只读区域，存放程序的代码段，从 0x31000000 开始是程序的数据段。

Layout 选项卡：该选项卡在连接方式位 Simple 时有效，它用来安排一些输入段在映像文件中的位置。Placeat beginning of 选项组用于指定将某个输入段放置在它所在的运行时域的开头。包含复位异常中断处理程序的输入段通常放置在运行时域的开头。

这里，在 Object/Symbol 文本框中指定目标文件的名称 init.o，在 Section 文本框中指定输入段的名称 init,从而确定了 init.s 源文件中的 init 输入段位指定的输入段。如图 4.9 所示。

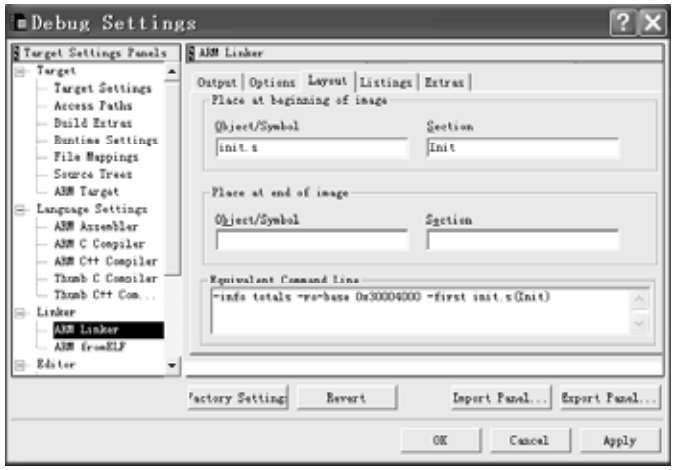


图 4.9　Layout 选项卡中连接器选项

到这里，对 Debug Setting 的设置基本完毕。点击图 4.9 中的 Apply 按钮后点击 OK 按钮，保存所有的设置。

4.2.3 编译连接工程

如图 4.10 所示为工程窗口中的图标按钮,通过这些图标按钮,可以快速的进行工程设置,编译连接,启动调试等等.它们从左到右分别为

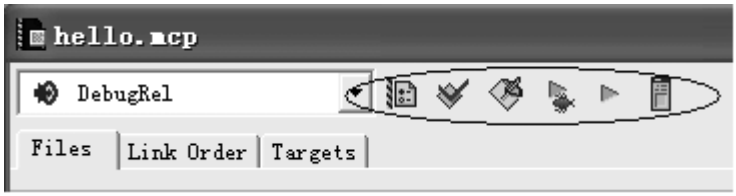


图 4.10 工程窗口中的图标按钮

DebugRel Settings…	工程设置
Synchronize Modification Dates	同步修改日期
Make	编译连接
Debug	启动 ADX 进行调试
Run	启动 ADX 调试,并直接运行
Project Inspector	工程检查,查看和配置工程中源文件的信息

对于简单的软件调试,直接点击工程窗口的”Make”图标按钮,即可完成编译.编译连接输出窗口如图 4.11 所示。

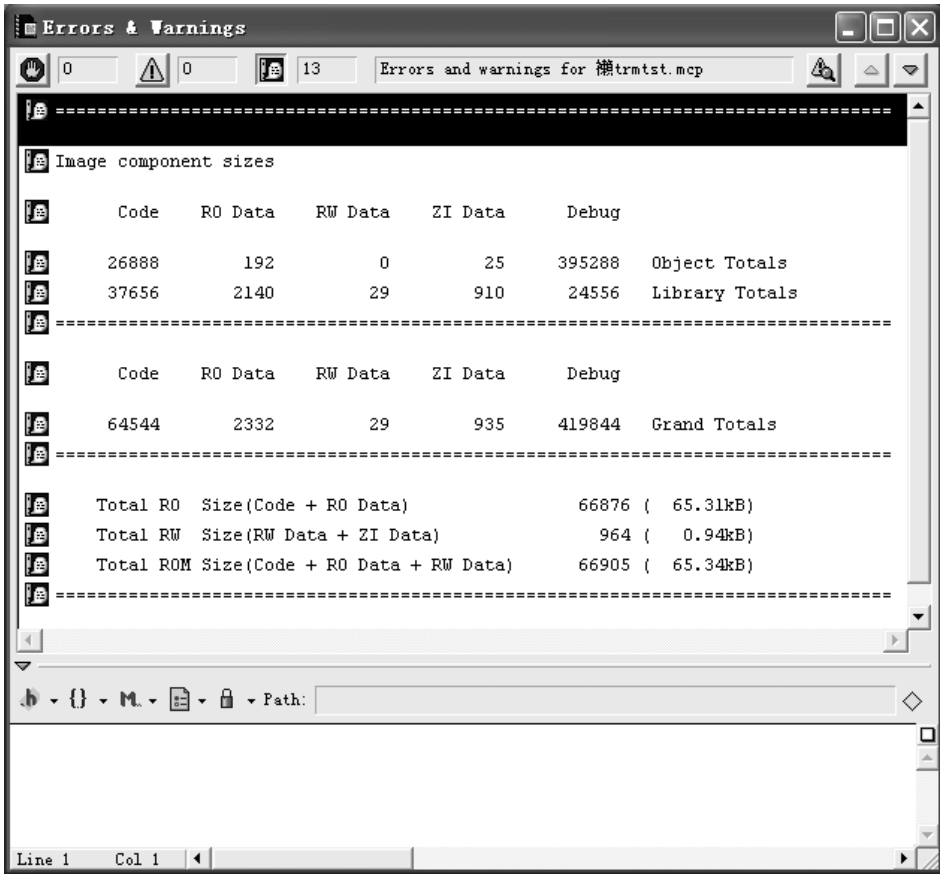


图 4.11 编译连接输出窗口

4.3 工程的调试

4.3.1 调试工具条

AXD 运行调试工具条如图 4.12 所示,调试观察窗口工具条如图 4.13 所示,文件操作工具条如图 4.14 所示.下面具体说明：

图 4.12 运行调试工具条

全速运行（Go）

停止运行（Stop）

单步运行

运行到光标

设置断点

图 4.13 调试观察窗口工具条

..打开寄存器窗口

打开观察窗口

打开变量观察窗口

打开存储器观察窗口

打开反汇编窗口

图 4.14 文件操作工具条

加载调试文件

重新加载文件

4.4 Multi-ICE 仿真器及超级终端

4.4.1 仿真器的连接

ARM JTAG 使用标准的 25 芯并口插座和 20 针的 JTAG 插座作为接口与 PC 的连接,线缆使用标准的 25 芯并口连接,线缆与目标板的连接线缆使用 20 芯的 IDC 宽带线缆.有的目标板可能使用 14 脚的 JTAG 插座信号.具体连接方法如图 4.15 所示：

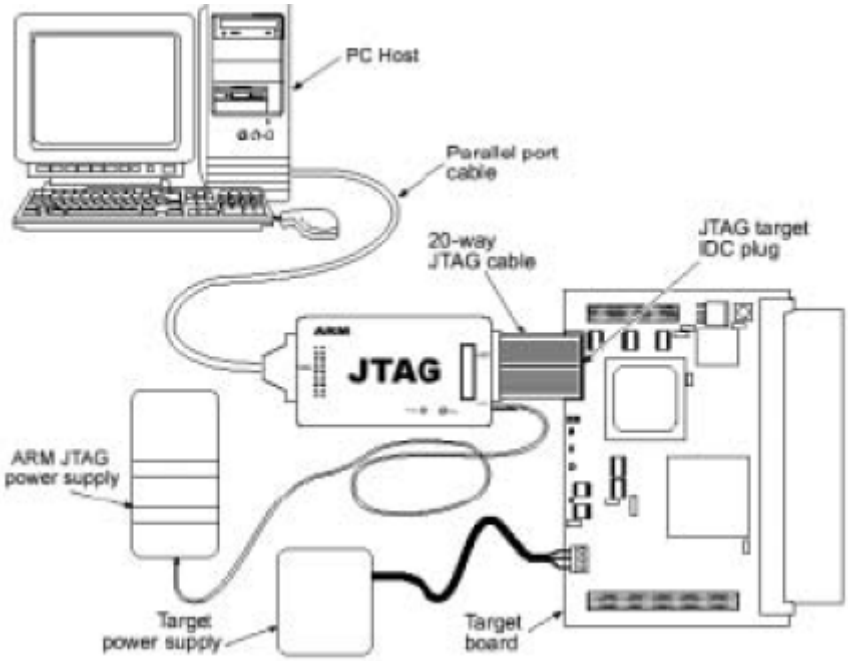


图 4.15 仿真器的连接

4.4.2 Multi-ICE 仿真器使用

Multi-ICE 是 ARM JTAG 的配置程序.通过它可以使 ARM JTAG 与目标板建立通讯连接并能够反馈目标板上 ARM 处理器的硬件信息 .在 PC 开发主机上选择开始|程序|ARM Multi-ICE v2.2|Multi-ICE Server 进入 Multi-ICE Server 主界面,如图 4.16 所示

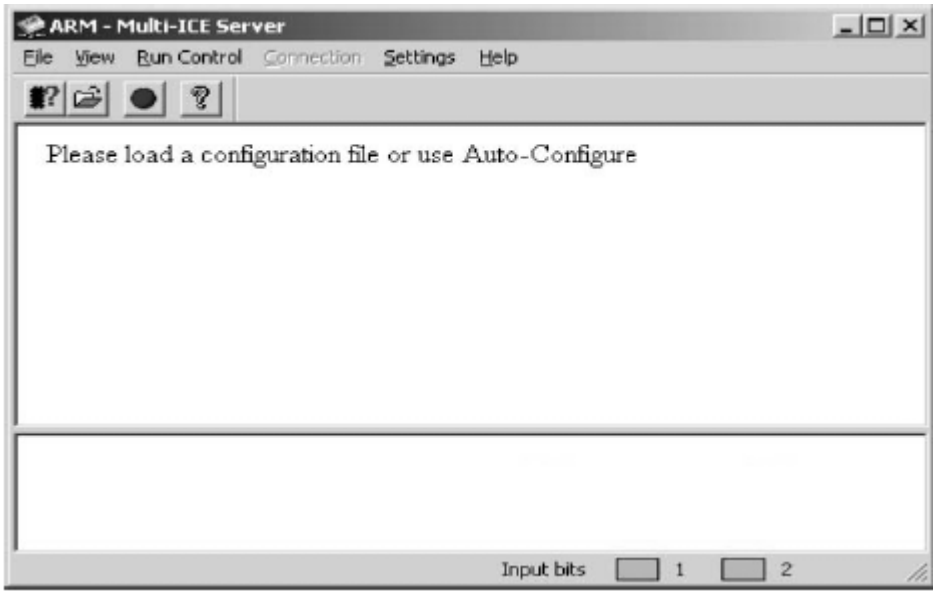


图 4.16　Multi-ICE Server 主界面

1．把 ARM 的 JTAG 仿真器连接到 PC 机的并行口和开发板上，串口线一端接计算机串口 1，一端接开发板 UART0，打开开发板和仿真器的电源，运行仿真器的驱动程序 Multi－Iceserver．exe，点击软件界面左上角的 Auto－Configure 按钮，显示如图 4.17 所示，仿真器连接成功。

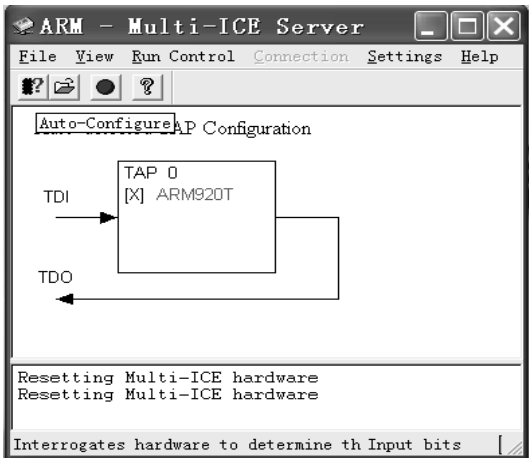


图 4.17 仿真器驱动程序

2．在 PC 开发主机上打开超级终端，波特率为 115200，数据位 8 位，奇偶校验选择无，停止位 1 位，数据流无，在设置好的窗口内观察从主机串口中传出的数据。

在后续的实验中我们还将用到超级终端调试运行实验程序, 观察程序运行的结果, 因此这里简单介绍一下超级终端的使用。

4.4.3 超级终端的使用

超级终端使用特别广泛，是每个 Windows 自带的内容。以下描述超级终端在 Windows 下的使用。超级终端一般包含在开始→程序→附件→通讯→超级终端文件夹中。如果找不

到超级终端，那可能是您在安装 Windows 时将它“删除”。按以下步骤安装超级终端到系统中。（注意：这里要使用到 Windows 启动盘！）

打开控制面板（开始→设置→控制面板）双击添加/删除程序按钮，添加/删除程序的对话框就会打开，如图 4.17 所示。

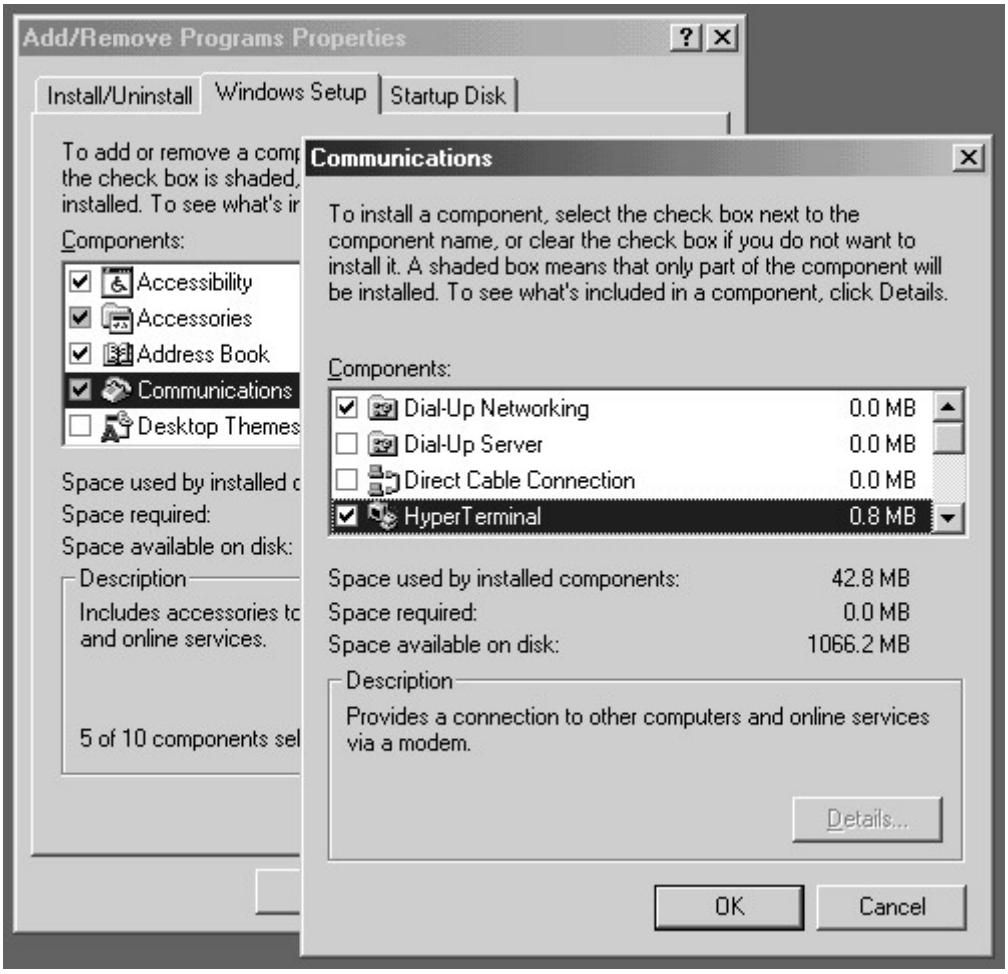


图 4.17　添加超级终端到 Windows 系统中

点击 Windows Setup 面板，浏览安装部件表；选择列表中的通讯并点击具体信息；在通讯窗口中选择超级终端（请划钩）；按 OK 关闭通讯窗口，再按 OK 关闭添加/删除程序。

这时系统可能会要求您插入 Windows CD。然后按屏幕显示的指示继续操作。

启动超级终端进行以下设置：

当连接设置对话框打开后，输入任意名称（例如 EM100），并选择一个图标，按 OK 确认，如图 4.18 所示：

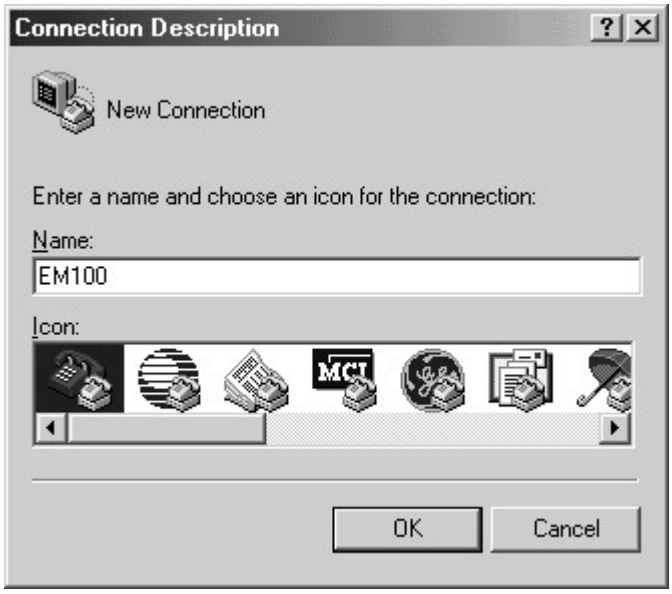


图 4.18　选择连接名称

当连接到对话框打开时，如图 4.19 所示，从 Connect Using 下拉框中选择一个合适的 COM port（适个人串口连接而定，这里假如连接到 PC 的 COM2 Port，那么请选择 COM2）并按 OK 确定。

图 4.19. 选择 COM port

当 COM 属性窗口打开后，设置通讯参数如下 Bits per second：115200；Data bits：8，Parity：None；Stop bits：1；Flow control：None，完成后点 OK 确定。超级终端主窗口就会打开，如图 4.20 所示：

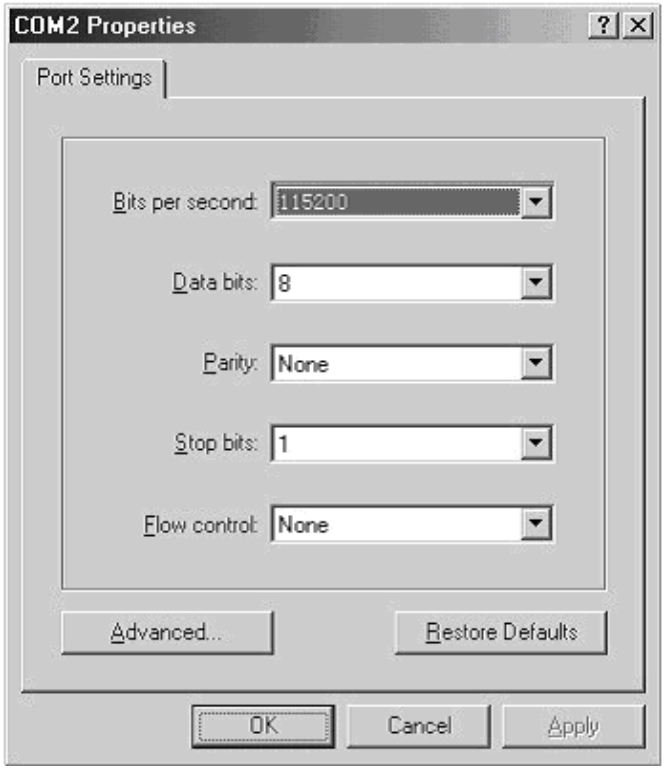


图 4.20　设置 COM port 参数

选择文档→属性，属性对话框打开后，选择设置，按 ASCII Setup 按钮，ASCII Setup 对话框就会打开，如图 4.21 所示。

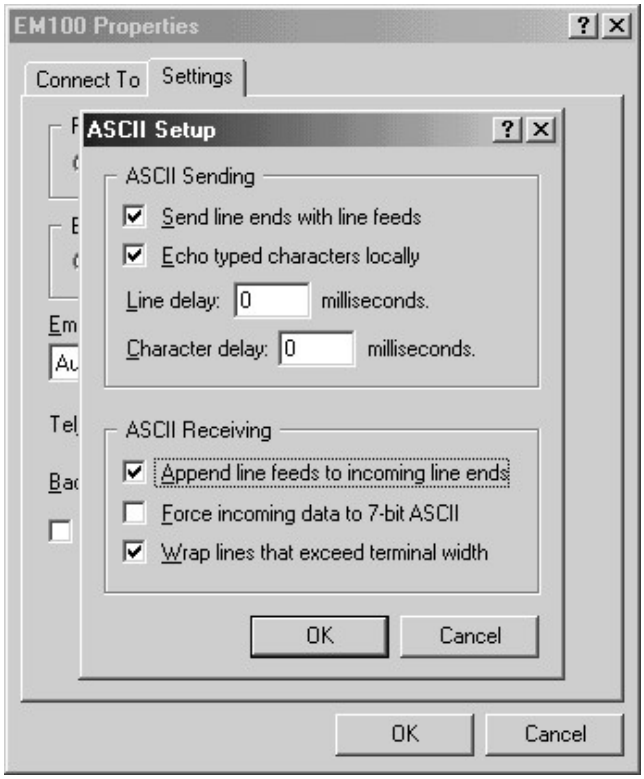


图 4.21　Setting options

对话框内包含以下几个选项，运行不同的实验可以选择不同的显示格式。

ASCII 码发送：

Send line ends with line feeds：以换行符作为发送行末尾

Echo typed characters locally：本地回显键入的字符

ASCII 码接收：

Append line feeds to incoming line ends：将换行符附加到传入行末尾

Force incoming data to 7-bit ASCII：将传入的数据转换为 7 位的 ASCII 码

Wrap lines that exceed terminal width：将超级终端宽度的行自动换行

点击OK关闭对话框。

此外，如果需要保存该超级终端设置备用，也就不需要再进行设置，从主菜单中选择文件→保存，即可命名保存设置结果。

第五章 嵌入式开发基础实验

实验一 汇编指令实验 1

1.1 实验目的

1．熟悉 ADS1.2 软件开发环境；
2．掌握 ARM7TDMI 汇编指令的用法，并能编写简单的汇编程序；
3．掌握指令的条件执行和使用 LDR/STR 指令完成存储器的访问。

1.2 实验内容

1．使用 LDR 指令读取 0x40003100 上的数据，将数据加 1，若结果小于 10，则使用 STR 指令把结果写回原地址，若结果大于等于 10，则把 0 写回原地址。

2. 使用 ADS1.2 软件仿真，单步，全速运行程序，设置断点，打开寄存器窗口（Processor Registers）监视 R0，R1 的值，打开存储器观察窗口（Memory）监视 0x40003100 上的值。

1.3 预备知识

1．ARM 指令系统内容；
2．ADS1.2 工程编辑和 AXD 调试的内容。

1.4 实验设备

硬件：PC 机一台。
软件：Windows98/XP/2000 系统，ADS1.2 集成开发环境。

1.5 实验步骤

1．启动 ADS1.2，使用 ARM Executable Image 工程模板建立一个工程 arm1.mcp。

2．建立汇编源文件 arm1.s，编写实验程序，然后添加到工程中。

3．设置工程连接地址 RO Base 为 0x40000000，RW Base 为 0x40003000，设置调试口地址 Image entry point 为 0x40000000。

4．编译连接工程，选择 Project|Debug，启动 AXD 进行软件仿真调试。

5．打开寄存器窗口（Processor Registers），选择 Current 项监视 R0，R1 的值。打开存储器观察窗口（Memory），设置观察地址为 0x40003100，显示方式 Size 为 32Bit，监视 0x40003100 地址上的值。

说明：在 Memory 窗口中点击鼠标右键，Size 项中选择显示格式为 8bit，16bit，32bit。如图 1.1 所示。

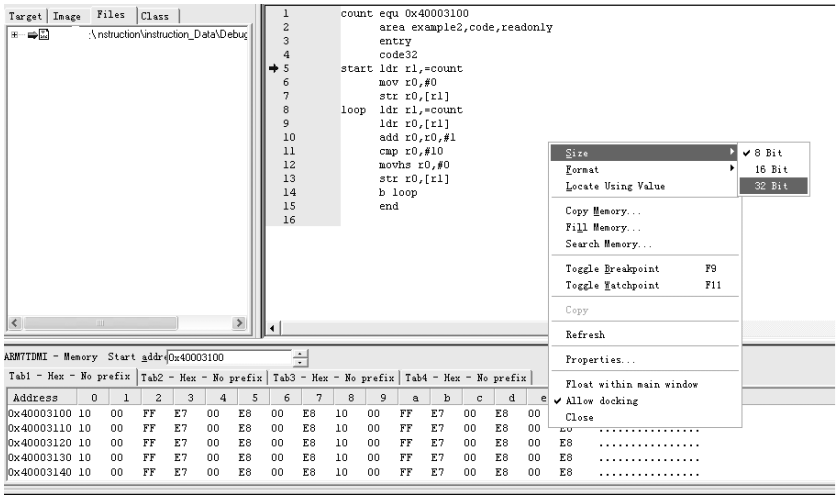


图 1.1 Memory 窗口显示格式

设置寄存器显示格式与之类似。使用鼠标左键选择某一个寄存器，然后点击鼠标右键，Format 项中选择显示格式 Hex，Decimal 等等。如图 1.2 所示。

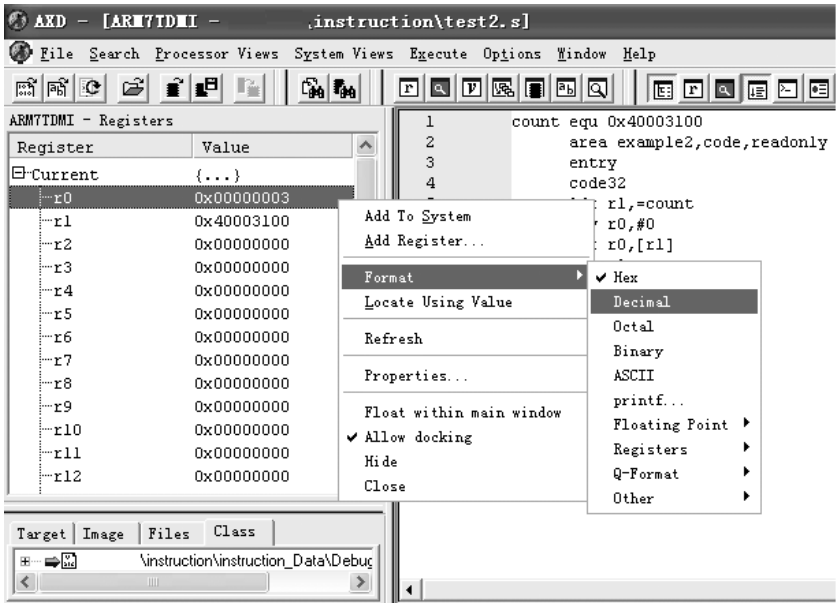


图 1.2 设置寄存器显示格式

6．可以单步运行程序，可以设置/取消断点，或者全速运行程序，停止程序运行。这时观察寄存器和 0x40003100 地址上的值。运行结果如图 1.3 所示。

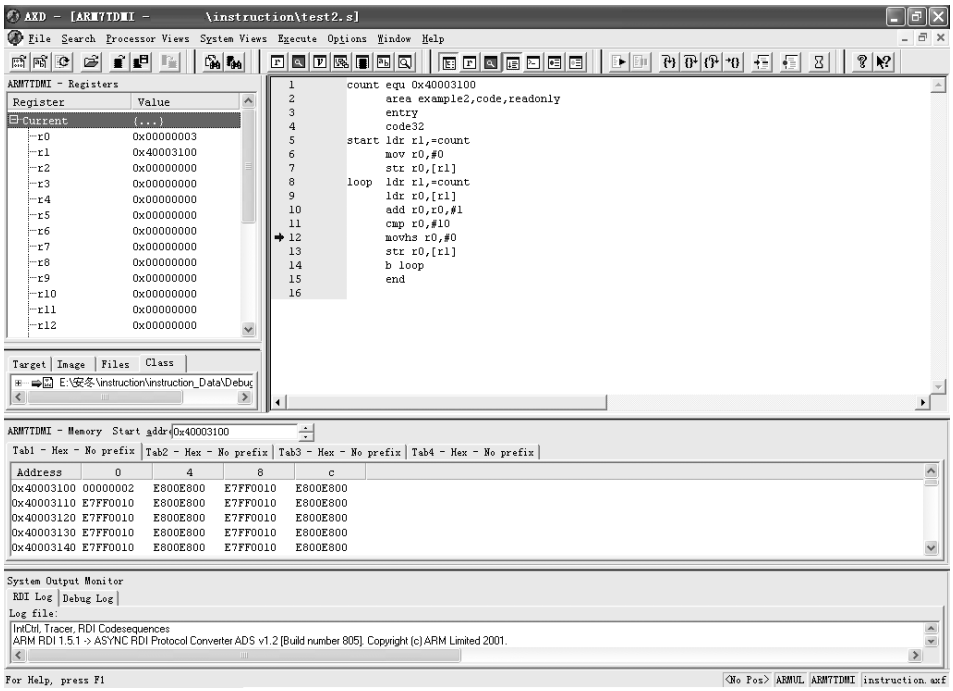


图 1.3 实验 1 结果

1.6 实验参考程序

汇编指令实验 1 的参考程序见以下程序清单。

```
COUNT EQU 0x40003100
      AREA Example1,CODE,READONLY
      ENTRY
      CODE32
START LDR R1,=COUNT
      MOV R0,#0
      STR R0,[R1]
LOOP  LDR R1,=COUNT
      LDR R0,[R1]
      ADD R0,R0,#1
      CMP R0,#10
      MOVHS R0,#0
      STR R0,[R1]
      B LOOP
      END
```

实验二 汇编指令实验 2

2.1 实验目的

1．掌握 ARM 乘法指令的使用方法；
2．了解子程序编写及调用。

2.2 实验内容

使用 STMFD/LDMFD，MUL 指令编写一个整数乘方的子程序，然后使用 BL 指令调用子程序计算 X^n 的值。

2.3 预备知识

1．ARM 指令系统内容；
2．ADS1.2 工程编辑和 AXD 调试的内容。

2.4 实验设备

硬件：PC 机一台。
软件：Windows98/XP/2000 系统，ADS1.2 集成开发环境。

2.5 实验原理

X^n= X*X*X …… *X，其中相乘的 X 的个数为 n 个。先将 X 的值装入 R0 和 R1，使用寄存器 R2 进行计数，循环 n-1 次 R0=R0*R1，运算结果就保存在 R0 中。（不考虑溢出问题）

注意：若 n=0，则运算结果直接赋 1；若 n=1，则运算结果直接赋 X。

2.6 实验步骤

1．启动 ADS1.2，使用 ARM Executable Image 工程模板建立一个工程 arm2.mcp。
2．建立汇编源文件 arm2.s，编写实验程序，然后添加到工程中。
3．设置工程连接地址 RO Base 为 0x40000000，RW Base 为 0x40003000，设置调试口地址 Image entry point 为 0x40000000。
4．编译连接工程，选择 Project|Debug，启动 AXD 进行软件仿真调试。
5．打开寄存器窗口（Processor Registers），选择 Current 项监视寄存器 R0，R1，R13（SP）和 R14（LR）的值。
6．打开存储器观察窗口（Memory），设置观察地址为 0x40003EA0，显示方式 Size 为

32Bit，监视从 0x40003F00 起始的满递减堆栈区。

7．单步运行程序，跟踪程序执行的流程，观察寄存器值的变化和堆栈区的数据变化，判断执行结果是否正确。

8．调试程序时，更改参数 X 和 n 来测试程序，观察是否得到正确的结果。例如：先复位程序，接着单步执行到“BL　POW”指令，在寄存器窗口中将 R0，R1 的值进行修改，然后继续运行程序。

说明：用鼠标双击寄存器窗口的寄存器，即可修改寄存器的值。输入数据可以是十进制数（如 136，198），也可以是十六进制数（如 0x123，0xF0），输入数据后回车确定。

2.7 实验参考程序

汇编指令实验 2 的参考程序见以下程序清单。

```
X           EQU         9
n           EQU         8
            AREA        Example3,CODE,READONLY
            ENTRY
            CODE32
START       LDR         SP,=0x40003F00
            LDR         R0,=X
            LDR         R1,=n
            BL          POW
HALT        B           HALT
POW         STMFD       SP!,{R1-R12,LR}
            MOVS        R2,R1
            MOVEQ       R0,#1
            BEQ         POW_END
            MOV         R1,R0
            SUB         R2,R2,#1
POW_L1      BL          DO_MUL
            SUBS        R2,R2,#1
            BNE         POW_L1
POW_END     LDMFD       SP!,{R1-R12,PC}
DO_MUL      MUL         R0,R1,R0
            MOV         PC,LR
            END
```

实验三 C 语言程序实验 1

3.1 实验目的

通过实验了解使用 ADS1.2 编写 C 语言程序,并进行调试。

3.2 实验内容

编写一个汇编程序文件和一个 C 程序文件,汇编程序的功能是初始化堆栈指针和初始化 C 程序的运行环境,然后跳转到 C 程序运行,这就是一个简单的启动程序.C 程序使用加法运算来计算 1+2+3+…+（N-1）+N 的值（N>0）。

3.3 预备知识

1．ARM 指令系统内容；
2．ADS1.2 工程编辑和 AXD 调试的内容。

3.4 实验设备

硬件：PC 机一台。
软件：Windows98/XP/2000 系统，ADS1.2 集成开发环境。

3.5 实验步骤

1．启动 ADS1.2，使用 ARM Executable Image 工程模板建立一个工程 c1.mcp。
2．建立汇编源文件 Startup.s 和 c1.c，编写实验程序，然后添加到工程中。
3．设置工程连接地址 RO Base 为 0x40000000，RW Base 为 0x40003000，设置调试口地址 Image entry point 为 0x40000000。
4．设置位于开始位置的起始代码段,如图 3.1 所示。

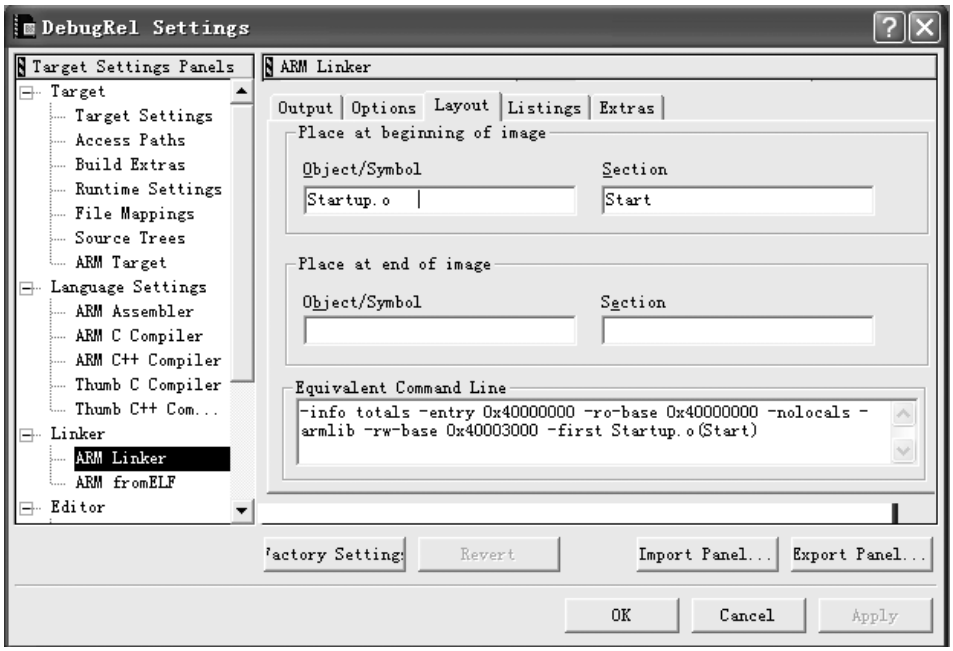


图 3.1 设置位于开始位置的起始代码段

5．编译连接工程，选择 Project|Debug，启动 AXD 进行软件仿真调试。

6．在 Startup.S 的”B Main”处设置断点，然后全速运行程序。

7．程序在断点处停止,单步运行程序,判断程序是否跳转到 C 程序中运行。

8．选择 Processor Views | Variables 打开变量观察窗口,观察全局变量的值,单步/全速运行程序，判断程序的运算结果是否正确。

3.6 实验参考程序

C 语言实验的参考程序见程序清单（一）；汇编启动代码见程序清单（二）。

程序清单（一） C 语言实验参考程序

```
#define     uint8       unsigned char
#define     uint32      unsigned int
#define     N           100
uint32      sum;
void        Main（void）
{uint32      i;
        sum=0;
        for（i=0;i<=N;i++）
              {sum+=i;}
             while（1）;
 }
```

程序清单（二） 简单的启动代码

```
        IMPORT   |Image$$RO$$Limit|
        IMPORT   |Image$$RW$$Base|
```

```
        IMPORT  |Image$$ZI$$Base|
        IMPORT  |Image$$ZI$$Limit|
        IMPORT  Main
            AREA    Start,CODE,READONLY
            ENTRY
            CODE32
Reset       LDR     SP,=0x40003f00
            LDR     R0,=|Image$$RO$$Limit|
            LDR     R1,=|Image$$RW$$Base|
            LDR     R3,=|Image$$ZI$$Base|
            CMP     R0,R1
            BEQ     LOOP1
LOOP0       CMP     R1,R3
            LDRCC   R2,[R0],#4
            STRCC   R2,[R1],#4
            BCC     LOOP0
LOOP1       LDR     R1,=|Image$$ZI$$Limit|
            MOV     R2,#0
LOOP2       CMP     R3,R1
            STRCC   R2,[R3],#4
            BCC     LOOP2
            B       Main
            END
```

实验四 C 语言程序实验 2

4.1 实验目的

掌握在 C 语言程序中调用汇编程序,了解 ATPCS 基本规则。

4.2 实验内容

在 C 程序调用汇编子程序,实现两个整数的加法运算。汇编子程序的原型为：

unit32　Add（unit32 x,unit32 y）

其中, unit32 已定义为 unsigned int.

4.3 预备知识

1．ARM 公司的 ATPCS 的相关文档,比如 ATPCS.PDF；

2．ADS1.2 工程编辑和 AXD 调试的内容。

4.4 实验设备

硬件：PC 机一台。

软件：Windows98/XP/2000 系统，ADS1.2 集成开发环境。

4.5 实验步骤

1．启动 ADS1.2，使用 ARM Executable Image 工程模板建立一个工程 c2.mcp。

2．建立汇编源文件 Startup.S , Add.S 和 c2.c，编写实验程序，然后添加到工程中。

3．设置工程连接地址 RO Base 为 0x40000000，RW Base 为 0x40003000，设置调试口地址 Image entry point 为 0x40000000。

4．设置工程连接选项,位于开始位置的起始代码段设置为 Startup.o 的 Start 段.

5．编译连接工程，选择 Project|Debug，启动 AXD 进行软件仿真调试。

6．在 c1 .c 文件中的调用 Add（）的代码处设置断点, 然后全速运行程序。

7．程序在断点处停止,使用 Step In 单步运行程序,观察程序是否跳转到汇编程序中 Add.s 运行。

8．选择 Processor Views | Variables 打开变量观察窗口,观察全局变量的值,单步/全速运行程序, 判断程序的运算结果是否正确。

4.6 实验参考程序

C 语言调用汇编程序实验的参考程序见程序清单（一）；
汇编加法函数代码见以下程序清单（二）。

程序清单（一） C 语言调用汇编的参考程序

```
#define     uint8       unsigned char
#define     uint32      unsigned int
extern      uint32      Add（uint32 x,uint32 y）;
uint32      sum;
void         Main（void）
{sum=Add（555,168）;
              while（1）;
 }
```

程序清单（二） 汇编加法函数程序

```
          EXPORT     Add
          AREA       Start,CODE,READONLY
          ENTRY
          CODE32
Add       ADD        R0,R0,R1
          MOV        PC,LR
          END
```

第六章 基本实验

实验一 I/O 接口实验

1.1 实验目的

1．熟悉 ARM 芯片 I/O 口配置方法；
2．通过实验掌握 ARM 芯片 I/O 控制 LED 显示的方法。

1.2 实验内容

1．学习如何在 ADS 集成开发环境中编辑编译和调试程序；
2．熟悉 S3C2410 上 I/O 配置寄存器，编程实现实验板上的发光二极管 LED1~LED4 依次点亮和熄灭。

1.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．熟悉 ARM 开发板的端口设置过程。

1.4 实验设备

1．ARM2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

1.5 基础知识

S3C2410X 芯片上有117个多功能I/O引脚.分别是：

- ✧ 端口 A（GPA）：23 个输出端口
- ✧ 端口 B（GPB）：11 个输入/输出端口
- ✧ 端口 C（GPC）：16 个输入/输出端口
- ✧ 端口 D（GPD）：16 个输入/输出端口
- ✧ 端口 E（GPE）：16 个输入/输出端口
- ✧ 端口 F（GPF）：8 个输入/输出端口
- ✧ 端口 G（GPG）：16 个输入/输出端口
- ✧ 端口 H（GPH）：11 个输入/输出端口

每个端口都可以通过软件配置寄存器来满足不同系统和设计的需要。在运行主程序之前，必须先对每一个用到的引脚的功能进行设置。如果某些引脚的复用功能没有使用，那么

可以先将该引脚设置为I/O口。

1．S3C2410X芯片与端口相关的寄存器

（1）端口控制寄存器（GPACON—GPHCON）：在S3C2410X芯片中，大部分引脚是多路复用的，所以要确定每个引脚的功能。PnCON（端口控制寄存器）能够定义引脚功能。如果GPF0—GPF7和GPG0—GPG7被用作掉电模式下的唤醒信号，那么这些端口必须配置成中断模式。

（2）端口数据寄存器（GPADAT—GPHDAT）：如果端口定义为输出口，那么输出数据可以写入PDATn中的相应位；如果端口定义为输入口，那么输入数据可以从PDATn相应的位中读入。

（3）端口上拉寄存器（GPBUP—GPHUP）：通过配置端口上拉寄存器，可以使该组端口与上拉电阻连接或断开。当寄存器中相应的位配置为0时，该引脚接上拉电阻；当寄存器中相应的位配置为1时，该引脚不接上拉电阻.。

（4）外部中断控制寄存器（EXTINTn）：通过不同的信号方式可以使24个外部中断被请求。EXTINTn寄存器可以根据外部中断的需要，将中断触发信号配置为低电平触发，高电平触发，下降沿触发，上升沿触发和边沿触发几种方式。

表1.1~表1.8为S3C2410 EDU实验板上各个端口的引脚定义。

表 1.1 端口 A 引脚定义

端口 A	引脚功能	端口 A	引脚功能	端口 A	引脚功能
GPA0	ADDR0	GPA8	ADDR23	GPA16	nGCS5
GPA1	ADDR16	GPA9	ADDR24	GPA17	CLE
GPA2	ADDR17	GPA10	ADDR25	GPA18	ALE
GPA3	ADDR18	GPA11	ADDR26	GPA19	nFWE
GPA4	ADDR19	GPA12	nGCS1	GPA20	nFRE
GPA5	ADDR20	GPA13	nGCS2	GPA21	nRSTOUT
GPA6	ADDR21	GPA14	nGCS3	GPA22	nFCE
GPA7	ADDR22	GPA15	nGCS4		

注：GPACON寄存器地址：0x56000000　　GPADAT寄存器地址：0x56000004

GPACON 复位默认值：0x7FFFFF

表 1.2 端口 B 引脚定义

端口 B	引脚功能	端口 B	引脚功能	端口 B	引脚功能
GPB0	TOUT0	GPB4	TCLK0	GPB8	nXDREQ1
GPB1	TOUT1	GPB5	nXBACK	GPB9	nXDACK0
GPB2	TOUT2	GPB6	nXBREQ	GPB10	nXDREQ0
GPB3	TOUT3	GPB7	nXDACK1		

注：GPBCON寄存器地址：0x56000010　　GPBDAT寄存器地址：0x56000014

GPBUP寄存器地址：0x56000018　　GPBCON复位默认值：0x0　　GPBUP复位默认值：0x0

表 1.3 端口 C 引脚定义

端口 C	引脚功能	端口 C	引脚功能	端口 C	引脚功能
GPC0	LEND	GPC6	LCDVF1	GPC12	VD4
GPC1	VCLK	GPC7	LCDVF2	GPC13	VD5
GPC2	VLINE	GPC8	VD0	GPC14	VD6

GPC3	VFRAME	GPC9	VD1	GPC15	VD7
GPC4	VM	GPC10	VD2		
GPC5	LCDVF0	GPC11	VD3		

注：GPCCON寄存器地址：0x56000020　　GPCDAT寄存器地址：0x56000024

GPCUP寄存器地址：0x56000028　　GPCCON复位默认值：0x0GPCUP　　复位默认值：0x0

表 1.4 端口 D 引脚定义

端口 D	引脚功能	端口 D	引脚功能	端口 D	引脚功能
GPD0	VD8	GPD6	VD14	GPD12	VD20
GPD1	VD9	GPD7	VD15	GPD13	VD21
GPD2	VD10	GPD8	VD16	GPD14	VD22（nSS1）
GPD3	VD11	GPD9	VD17	GPD15	VD23（nSS0）
GPD4	VD12	GPD10	VD18		
GPD5	VD13	GPD11	VD19		

注：GPDCON寄存器地址：0x56000030　　GPDDAT寄存器地址：0x56000034

GPDUP寄存器地址：0x56000038　　GPDCON复位默认值：0x0GPDUP　　复位默认值：0xF000

表 1.5 端口 E 引脚定义

端口 E	引脚功能	端口 E	引脚功能	端口 E	引脚功能
GPE0	I2SLRCK	GPE6	SDCMD	GPE12	SPIMOSI0
GPE1	I2SSCLK	GPE7	SDDAT0	GPE13	SPICLK0
GPE2	CDCLK	GPE8	SDDAT1	GPE14	IICSCL
GPE3	I2SSDI（nSS0）	GPE9	SDDAT2	GPE15	IICSDA
GPE4	I2SSDO（I2SSDI）	GPE10	SDDAT3		
GPE5	SDCLK	GPE11	SPIMISO0		

注：GPECON寄存器地址：0x56000040　　GPEDAT寄存器地址：0x56000044

GPEUP寄存器地址：0x56000048　　GPECON复位默认值：0x0GPEUP　　复位默认值：0x0

表 1.6 端口 F 引脚定义

端口 F	引脚功能	端口 F	引脚功能	端口 F	引脚功能
GPF0	EINT0	GPF3	EINT3	GPF6	EINT6
GPF1	EINT1	GPF4	EINT4	GPF7	EINT7
GPF2	EINT2	GPF5	EINT5		

注：GPFCON寄存器地址：0x56000050　　GPFDAT寄存器地址：0x56000054

GPFUP寄存器地址：0x56000058　　GPFCON复位默认值：0x0　　GPFUP复位默认值：0x0

表 1.7 端口 G 引脚定义

端口 G	引脚功能	端口 G	引脚功能	端口 G	引脚功能
GPG0	EINT8	GPG6	EINT14（SPIMOSI1）	GPG12	EINT20（XMON）
GPG1	EINT9	GPG7	EINT15（SPICLK1）	GPG13	EINT21（nXPON）
GPG2	EINT10（nSS0）	GPG8	EINT16	GPG14	EINT22（YMON）
GPG3	EINT11（nSS1）	GPG9	EINT17	GPG15	EINT23（nYPON）
GPG4	EINT12（LCD_PWREN）	GPG10	EINT18		

GPG5	EINT13（SPIMISO1）	GPG11	EINT19（TCLK1）		

注：GPGCON寄存器地址：0x56000060　　GPGDAT寄存器地址：0x56000064

GPGUP寄存器地址：0x56000068　　GPGCON复位默认值：0x0　　GPGUP复位默认值：0xF800

表 1.8 端口 G 引脚定义

端口 H	引脚功能	端口 H	引脚功能	端口 H	引脚功能
GPH0	nCTS0	GPH4	TXD1	GPH8	UCLK
GPH1	nRTS0	GPH5	RXD1	GPH9	CLKOUT0
GPH2	TXD0	GPH6	TXD2（nRTS1）	GPH10	CLKOUT1
GPH3	RXD0	GPH7	RXD2（nCTS1）		

注：GPHCON寄存器地址：0x56000070　　GPHDAT寄存器地址：0x56000074

GPHUP寄存器地址：0x56000078　　GPHCON复位默认值：0x0　　GPHUP复位默认值：0x0

2．电路原理

如图1.1所示，发光二极管LED1~LED4的正极接S2C2410板上的3.3V高电压，负极通过限流电阻分别与S2C2410的GPF4~GPF7引脚连接。

四盏灯的分配如下：LED1　红色LED　　EINT4/GPF4

LED2　红色LED　　EINT5/GPF5

LED3　绿色LED　　EINT6/GPF6

LED4　绿色LED　　EINT7/GPF7

这四个引脚属于端口F，已经配置为输出口。通过向GPFDAT寄存器中相应的位写入0或1，可以使引脚GPF4~GPF7输出低电平或高电平。当GPF4~GPF7输出低电平时，LED点亮；当GPF4~GPF7输出高电平时，LED熄灭。

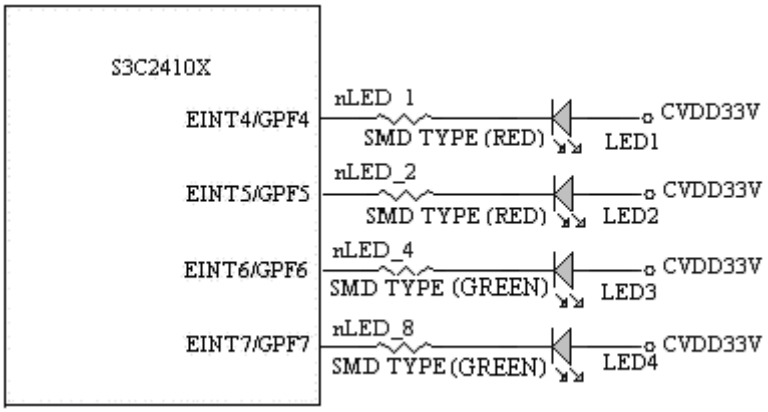


图1.1 发光二极管控制电路

1.6 实验步骤

1．参照模板,新建一个工程 GPIO.mcp，添加相应的文件，并修改 GPIO 的工程设置；

2．创建 Main.c 文件，并加入到工程文件 GPIO.mcp 中；

3．为 Main.c 文件的主任务 maintask 添加如下的语句：

```
void Main（void）
  { int i;
```

```
    Port_Init（）;                                    //初始化 I/O
    while （1）
     { Led_Display（0x0f）;                           //灯亮
       for （i=0; i<0xfffff; i++）;                   //延迟
       Led_Display（0x00）;                           //灯灭
       for （i=0; i<0xfffff; i++）;                   //延迟
     }
   }
```

4．编译 GPIO 工程；

5．下载程序并运行，观察结果。

1.7 实验报告要求

如何用两盏灯的状态组合循环显示 00—11.

实验二 串口通讯实验

2.1 实验目的

1．掌握 ARM 的串行口工作原理；
2．编程实现 ARM 的 UART 通讯；
3．掌握 S3C2410 寄存器配置方法。

2.2 实验内容

实现查询方式串口的收发功能。接收来自串口（通过超级终端）的字符并将接收到的字符发送到超级终端。

2.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．了解串口通讯的基本知识；
3．熟悉 S3C2410 串口有关寄存器。

2.4 实验设备

1．ARM2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

2.5 基础知识

串行通信接口电路一般由可编程的串行接口芯片、波特率发生器、EIA 与 TTL 电平转换器以及地址译码电路组成。采用的通讯协议有两类：异步协议和同步协议。随着大规模集成电路技术的发展，通用的同步（USRT）和异步（UART）接口芯片种类越来越多，它们的基本功能是类似的。采用这些芯片作为串行通信接口电路的核心芯片，会使电路结果比较简单。下面介绍了异步串行通信的基本原理、串行接口的物理层标准以及 S3C2410 串行口控制器。

1．异步串行通信

异步串行方式是将传输数据的每个字符一位接一位（例如先低位、后高位）地传送。数据的各不同位可以分时使用同一传输通道，因此串行 I/O 可以减少信号连线，最少用一对线即可进行。接收方对于同一根线上一连串的数字信号，首先接收完 1 个字符的各位，再按位组成字符。为了回复发送的信息，双方必须协调工作。在微型计算机中大量使用异步串行 I/O 方式，双方使用各自的时钟信号，而且允许时钟频率有一定误差，因此实现较

容易。但是由于每个字符都要独立确定起始和结束（即每个字符都要重新同步），字符和字符间还可能有长度不定的空闲时间，因此效率较低。

图 2.1 给出了异步串行通信中一个字符的传送格式。开始前，线路处于空闲状态，送出连续“1”。传送开始时首先发一个“0”作为起始位，然后出现在通信线上的是字符的二进制编码数据。每个字符的数据位长可以约定为 5 位、6 位、7 位或 8 位，一般采用 ASCII 编码。后面是奇偶校验位，根据约定，用奇偶校验位将所传字符中为“1”的位数凑成奇数个或偶数个。也可以约定不要奇偶校验，这样就取消奇偶校验位。最后是表示停止位的“1”信号，这个停止位可以约定持续 1 位、1.5 位或 2 位的时间宽度。至此一个字符传送完毕，线路又进入空闲，持续为“1”。经过一段随机的时间后，下一个字符开始传送又发出起始位。每一个数据位的宽度等于传送波特率的倒数。微机异步串行通信中，常用的波特率为 110，150，300，600，1200，2400，4800，9600 等。

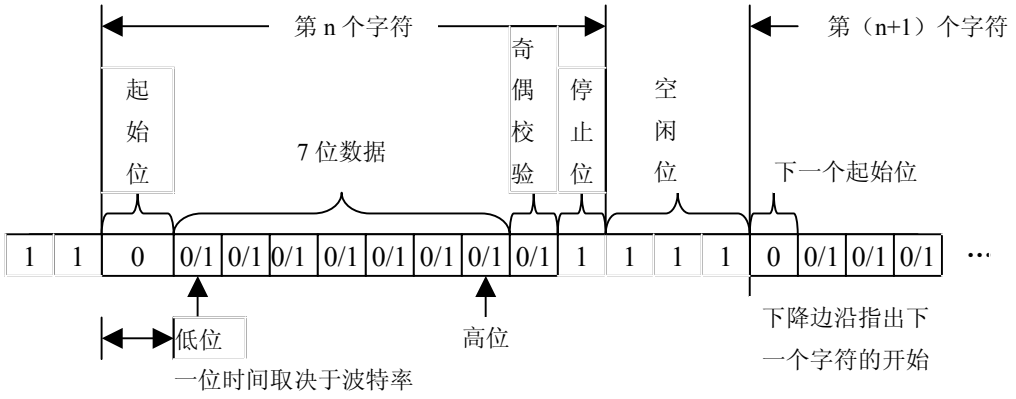


图 2.1 串行通信字符格式

2．串行接口的物理层标准

通用的串行 I/O 接口有许多种，现在就最常见的两种标准作简单介绍。

1）EIA RS232C

这是美国电子工业协会推荐的一种标准（Electronic industries Association Recoil-mended Standard）。它在一种 25 针插件（DB－25）上定义了串行通信的有关信号。这个标准后来被世界各国所接受并使用到计算机的 I/O 接口中。

在实际异步串行通信中，并不要求用全部的 RS-232C 信号，许多 PC/XT 兼容机仅用 15 针接插件（DB-15）来引出其异步串行 I/O 信号，而 PC 中更是大量采用 9 针接插件（DB-9）来担当此任。

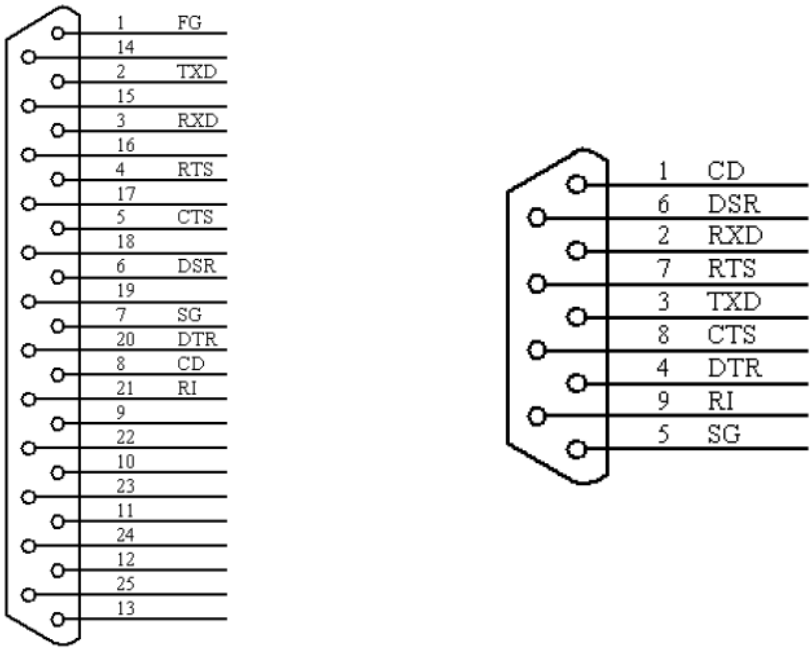


图 2.2　DB-25 和 DB-9 引脚定义

图 2.2 分别给出了 DB-25 和 DB-9 的引脚定义，表 2.1 列出了引脚的名称以及简要说明。

表 2.1 引脚说明

引脚序号	信号名称	符号	流向	功能
2	发送数据	TXD	DTE→DCE	DTE 发送串行数据
3	接收数据	RXD	DTE←DCE	DTE 接收串行数据
4	请求发送	RTS	DTE→DCE	DTE 请求 DCE 将线路切换到发送方式
5	允许发送	CTS	DTE←DCE	DCE 告诉 DTE 线路已接通可以发送数据
6	数据设备准备好	DSR	DTE←DCE	DCE 准备好
7	信号地			信号公共地
8	载波检测	DCD	DTE←DCE	表示 DCE 接收到远程载波
20	数据终端准备好	DTR	DTE→DCE	DTE 准备好
22	振铃指示	RI	DTE←DCE	表示 DCE 与线路接通,出现振铃

2）信号电平规定

RS-232C 规定了双极性的信号逻辑电平，它是一套负逻辑定义：-3V 到-25V 之间的电平表示逻辑“1”。+3V 到+25V 之间的电平表示逻辑“0”。以上标准称为 EIA 电平。PC/XT 系列使用的信号电平是−12V 和+12V，符合 EIA 标准，但在计算机内部流动的信号都是 TTL 电平，因此这中间需要用电平转换电路。常用专门的 RS-232 接口芯片，如 SP3232、SP3220 等，在 TTL 电平和 EIA 电平之间实现相互转换。PC/XT 系列以这种方式进行串行通信时，在波特率不高于 9600bps 的情况下，理论上通信线的长度限制为 15 米。

3．S3C2410 串行口控制器

S3C2410 自带 3 个异步串行口控制器，每个控制器有 16 字节的 FIFO（先入先出寄存器），最大波特率 115.2Kbps。每个 UART 有 7 种状态：溢出错误，校验错误，帧错误，暂停态，接收缓冲区准备好，发送缓冲区空，发送移位缓冲器空，这些状态可以由相应的 UTRSTATn 或 UERSTATn 寄存器表示，并且与发送接收缓冲区相对应的有错误缓冲区。串

行接口如图 2.3 所示：

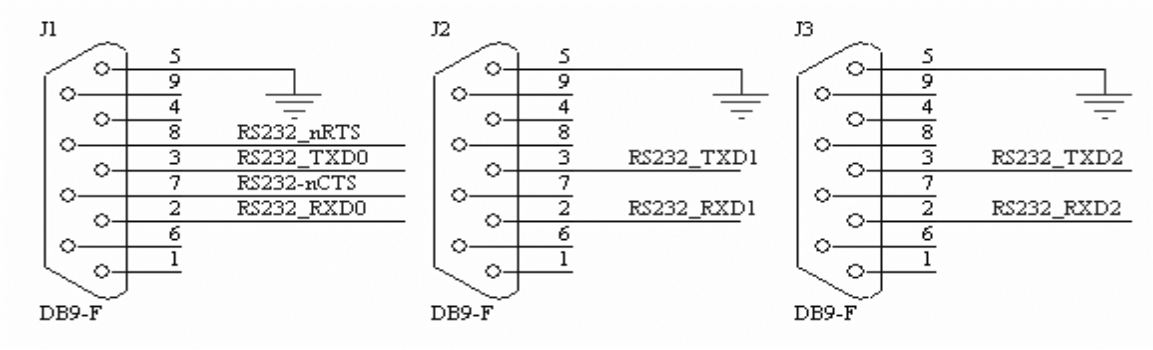


图 2.3 串行接口

波特率的大小由一个专用的 UART 波特率分频寄存器（UBRDIVn）控制，计算公式如下：

$$UBRDIVn = （int）[MCLK/（bps×16）]-1$$

其中：MCLK 是系统时钟，UBRDIVn 的值必须在 1~（2^{16}-1）之间.例如：在 40MHz 的情况下，当波特率取 115200 时，

$$UBRDIVn = （int）[40000000/（115200×16）] -1$$
$$= （int）（21.7）-1= 21 -1 = 20$$

与 UART 有关的寄存器主要有以下几个：

1）UART 行控制寄存器 ULCONn

下表列出了 ULCONn 各位的意义。该寄存器的位 6 决定是否使用红外模式，位 5~位 3 决定校验方式，位 2 决定停止位长度，位 1 和位 0 决定每帧的数据位数。

表 2.2 ULCONn

ULCONn	Bit	Description	Initial State
Reserved	[7]		0
Infra-Red Mode	[6]	Determine whether or not to use the Infra-Red mode. 0 = Normal mode operation 1 = Infra-Red Tx/Rx mode	0
Parity Mode	[5:3]	Specify the type of parity generation and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as 1 111 = Parity forced/checked as 0	000
Number of Stop Bit	[2]	Specify how many stop bits are to be used for end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0

Word Length	[1:0]	Indicate the number of data bits to be transmitted or received per frame. 00 = 5-bits 01 = 6-bits 10 = 7-bits 11 = 8-bits	00

表 2.2 ULCONn

ULCONn	Bit	Description	Initial State
Reserved	[7]		0
Infra-Red Mode	[6]	Determine whether or not to use the Infra-Red mode. 0 = Normal mode operation 1 = Infra-Red Tx/Rx mode	0
Parity Mode	[5:3]	Specify the type of parity generation and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as 1 111 = Parity forced/checked as 0	000
Number of Stop Bit	[2]	Specify how many stop bits are to be used for end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word Length	[1:0]	Indicate the number of data bits to be transmitted or received per frame. 00 = 5-bits 01 = 6-bits 10 = 7-bits 11 = 8-bits	00

2）UART 控制寄存器 UCONn

下表列出了 UCONn 各位的意义。该寄存器决定 UART 的各种模式。

表 2.3 UCONn

UCONn	Bit	Description	Initial State
Clock Selection	[10]	Select PCLK or UCLK for the UART baud rate. 0 = PCLK : UBRDIVn = (int)(PCLK/bps x 16)-1 1 = UCLK(@GPH8) : UBRDIVn= (int)(UCLK/ bps x 16)-1	0
Tx Interrupt Type	[9]	Interrupt request type. 0 = Pulse (Interrupt is requested as soon as the Tx buffer becomes empty in Non-FIFO mode or reaches Tx FIFO Trigger Level in FIFO mode.) 1 = Level (Interrupt is requested while Tx buffer is empty in Non-FIFO mode or reaches Tx FIFO Trigger Level in FIFO mode.)	0

Rx Interrupt Type	[8]	Interrupt request type. 0 = Pulse (Interrupt is requested the instant Rx buffer receive the data in Non-FIFO mode or reaches Rx FIFO Trigger Level in FIFO mode.) 1 = Level (Interrupt is requested while Rx buffer is receiving data in Non-FIFO mode or reaches Rx FIFO Trigger Level in FIFO mode.)	0
Rx Time Out Enable	[7]	Enable/Disable Rx time out interrupt when UART FIFO is enabled. The interrupt is a receive interrupt. 0 = Disable 1 = Enable	0
Rx Error Status Interrupt Enable	[6]	Enable the UART to generate an interrupt upon an exception, such as a break, frame error , parity error, or overrun error during a receive operation. 0 = Do not generate receive error status interrupt. 1 = Generate receive error status interrupt.	0
Loopback Mode	[5]	Setting loopback bit to 1 causes the UART to enter the loopback mode. This mode is provided for test purposes only. 0 = Normal operation 1 = Loopback mode	0
Send Break Signal	[4]	Setting this bit causes the UART to send a break during 1 frame time. This bit is automatically cleared after sending the break signal. 0 = Normal transmit 1 = Send break signal	0
Transmit Mode	[3:2]	Determine which function is currently able to write Tx data to the UART transmit buffer register. 00 = Disable 01 = Interrupt request or polling mode 10 = DMA0 request (Only for UART0), DMA3 request (Only for UART2) 11 = DMA1 request (Only for UART1)	00
Receive Mode	[1:0]	Determine which function is currently able to read data from UART receive buffer register. 00 = Disable 01 = Interrupt request or polling mode 10 = DMA0 request (Only for UART0), DMA3 request (Only for UART2) 11 = DMA1 request (Only for UART1)	00

NOTE: When the UART does not reach the FIFO trigger level and does not receive data during 3 word time in DMA receive mode with FIFO, the Rx interrupt will be generated (receive time out), and the users should check the FIFO ststus and read out the rest.

3）FIFO 控制寄存器 UFCONn

下表列出了 UFCONn 各位的意义。该寄存器用于收发缓冲的管理，包括缓冲的触发字

节数的设置、FIFO 的清除和使能。

表 2.4　UFCONn

UFCONn	Bit	Description	Initial State
Tx FIFO Trigger Level	[7:6]	Determine the trigger level of transmit FIFO. 00 = Empty　　01 = 4-byte 10 = 8-byte　　11 = 12-byte	00
Rx FIFO Trigger Level	[5:4]	Determine the trigger level of receive FIFO. 00 = 4-byte　　01 = 8-byte 10 = 12-byte　　11 = 16-byte	00
Reserved	[3]		0
Tx FIFO Reset	[2]	Auto-cleared after resetting FIFO 0 = Normal　　1 = Tx FIFO reset	0
Rx FIFO Reset	[1]	Auto-cleared after resetting FIFO 0 = Normal　　1 = Rx FIFO reset	0
FIFO Enable	[0]	0 = Disable　　1 = Enable	0

NOTE: When the UART does not reach the FIFO trigger level and does not receive data during 3 word time in DMA receive mode with FIFO, the Rx interrupt will be generated (receive time out), and the users should check the FIFO ststus and read out the rest.

4）MODEM 控制寄存器 UMCONn

下表列出了 UFCONn 各位的意义。该寄存器用于设置流控方式。在实验中没有使用流控。

表 2.5　UMCONn

UMCONn	Bit	Description	Initial State
Reserved	[7:5]	These bits must be 0’s	00
Auto Flow Control (AFC)	[4]	0 = Disable　　1 = Enable	0
Reserved	[3:1]	These bits must be 0’s	00
Request to Send	[0]	If AFC bit is enable, this value will be ignored. In this case the S3C2410X will control nRTS automatically. If AFC bit is disenable, nRTS must be controlled by software. 0 = ‘H’ level (Inactivate nRTS) 1 = ‘L’ level (Activate nRTS)	0

NOTE: UART 2 dose not support AFC function, because the S3C2410X has no nRTS2 and nCTS2.

5）状态寄存器 UTRSTATn

下表列出了 UTRSTATn 各位的意义。

表 2.6　UTRSTATn

UTRSTATn	Bit	Description	Initial State
Transmitter empty	[2]	Set to 1 automatically when the transmit	1

		buffer register has no valid data to transmit and the transmit shift register is empty. 0 = Not empty 1=Transmitter (transmit buffer & shifter register) empty	
Transmit buffer empty	[1]	Set to 1 automatically when the transmit buffer register is empty. 0 = The buffer register is not empty 1 = Empty (In Non-FIFO mode, Interrupt or DMA is requested. In FIFO mode, Interrupt or DMA is requested, when Tx FIFO Trigger Level is set to 00 (Empty)) If the UART uses the FIFO, users should check Tx FIFO Count bits and Tx FIFO Full bit in the UFSTAT register instead of this bit.	1
Receive buffer data ready	[0]	Set to 1 automatically whenever receive buffer register contains valid data, received over the RXDn port. 0 = Empty 1 = The buffer register has a received data (In Non-FIFO mode, interrupt or DMA is requested) If the UART uses the FIFO, users should check Rx FIFO Count bits and Rx FIFO full bit in the UFSTAT register instead of this bit.	0

6）错误状态寄存器 UERSTATn

下表列出了 UERSTATn 各位的意义。它可以反映芯片当前的错误类型。

表 2.7　UERSTATn

UERSTATn	Bit	Description	Initial State
Break Detect	[3]	Set to 1 automatically to indicate that a break signal has been received. 0 = No break receive 1 = Break receive (Interrupt is requested.)	0
Frame Error	[2]	Set to 1 automatically whenever a frame error occurs during receive operation. 0 = No frame error during receive 1 = Frame error (Interrupt is requested.)	0
Parity Error	[1]	Set to 1 automatically whenever a parity error occurs during receive operation.	0

		0 = No parity error during receive 1 = Parity error (Interrupt is requested.)	
Overrun Error	[0]	Set to 1 automatically whenever an overrun error occurs during receive operation. 0 = No overrun error during receive 1 = Overrun error (Interrupt is requested.)	0

NOTE: These bits (UERSATn[3:0]) are automatically cleared to 0 when the UART error status register is read.

7）FIFO 状态寄存器 UFSTATn

下表列出了 UFSTATn 各位的意义。通过它读出目前 FIFO 是否满以及其中的字节数。

表 2.8 UFSTATn

UFSTATn	Bit	Description	Initial State
Reserved	[15:10]		0
Tx FIFO Full	[9]	Set to 1 automatically whenever transmit FIFO is full during transmit operation 0 = 0-byte ≤ Tx FIFO data ≤ 15-byte 1 = Full	0
Rx FIFO Full	[8]	Set to 1 automatically whenever receive FIFO is full during receive operation 0 = 0-byte ≤ Tx FIFO data ≤ 15-byte 1 = Full	0
Tx FIFO Count	[7:4]	Number of data in Tx FIFO	0
Rx FIFO Count	[3:0]	Number of data in Rx FIFO	0

8）发送寄存器 UTXH 和接收寄存器 URXH

下表列出了 UTXH 和 URXH 各位的意义。这两个寄存器存放这发送和接收的数据，当然只有一个字节 8 位数据。需要注意的是,在发生溢出错误时，接收的数据必须被读出来,否则会引发下次溢出错误。

表 2.9 UTXH

UTXHn	Bit	Description	Initial State
TXDATAn	[7:0]	Transmit data for UARTn	-

表 2.10 URXH

URXHn	Bit	Description	Initial State
RXDATAn	[7:0]	Receive data for UARTn	-

9）波特率分频寄存器 UBRDIV

下表列出了 UBRDIV 各位的意义。该寄存器为十六位，算法参见上页的公式部分。

和 URXH 各位的意义一样,这两个寄存器存放着发送和接收的数据，当然只有一个字节 8 位数据。需要注意的是在发生溢出错误的时候，接收的数据必须要被读出来，否则会引发下次溢出错误。

表 2.11 UBRDIV

UBRDIVn	Bit	Description	Initial State

UBRDIV	[15:0]	Baud rate division value UBRDIVn >0	-

4．实验说明

串口在嵌入式系统中是一个重要的资源，常用来做输入输出设备，在后续的实验中也将使用串口的功能。串口的基本操作有三个：串口初始化、发送数据和接收数据，这些操作都是通过访问上节中描述的串口控制寄存器进行，下面将分别说明：

（1）串口初始化程序

```
MMU_Init（）;                          //初始化内存管理单元
                                       //设置系统时钟
    ChangeClockDivider（1,1）;         // 1：2：4
    ChangeMPllValue（0xa1,0x3,0x1）;   // FCLK=202.8MHz
    Port_Init（）;                     //初始化 I/O 口
    Uart_Init（0,115200）;             //初始化串口
    Uart_Select（0）;                  //选择串口 0
```

（2）发送数据

```
    while（!（rUTRSTAT0&0x2））;       //等待发送缓冲空
    rUTXH0=data;                       //将数据写到数据端口
```

（3）接收数据

```
    while（rUTRSTAT0&0x1==0x0）;       //等待数据
    data=rURXH0;                       //读取数据
```

2.6 实验步骤

1．参照模板工程，新建一个工程 UART，添加相应的文件，并修改 UART 的工程设置；

2．创建 Main.C 和 mmu.c 并加入到工程 UART 中；

3．编写串口操作函数实现如下功能：循环接收串口送来的数据，并将接收到的数据发送回去；

4．编译 UART；

5．将计算机的串口接到开发板的 UART0 上；

6．运行超级终端，选择正确的串口号，并将串口设置位：波特率（115200）、奇偶校验（None）、数据位数（8）和停止位数（1），无流控，打开串口；

7．运行程序，在超级终端中输入的数据将回显到超级终端上,如图 2.4

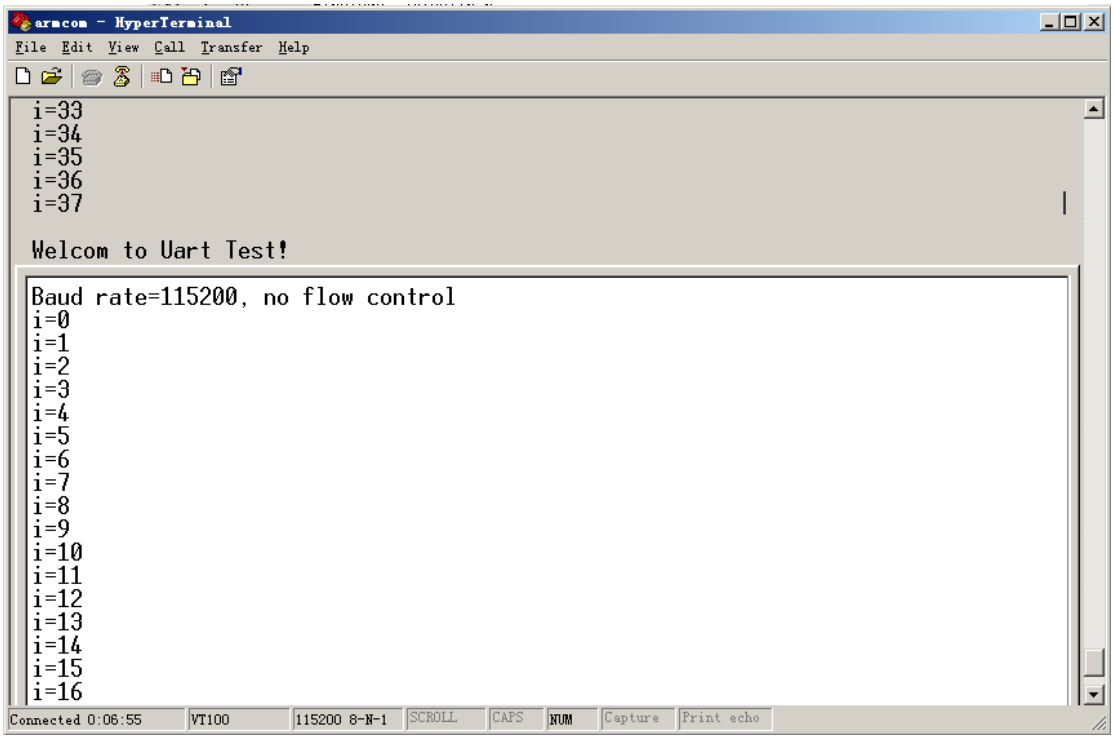


图 2.4　UART 显示结果

2.7 实验报告要求

1．简述串行接口的工作原理以及串行接口的优缺点；
2．RS-232C 的最基本数据传送引脚是哪几根？
3．简述串行接口通讯程序设计的基本步骤。

实验三 实时时钟实验

3.1 实验目的

1．了解实时时钟在嵌入式系统中的作用；
2．掌握实时时钟的使用。

3.2 实验内容

1．编程实现实时时钟功能，每秒显示实时时钟；
2．编程实现实时时钟告警功能。

3.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．了解 S3C2410 的实时时钟模块的使用。

3.4 实验设备

1．S3C2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

3.5 基础知识

1．实时时钟在嵌入式系统中的作用

在一个嵌入式系统中，实时时钟单元可以提供可靠的时钟，包括时分秒和年月日；即使在系统处于关机状态下，它也能正常工作（通常采用后备电池供电），它的外围也不需要太多的辅助电路，典型的就是只需要一个高精度的晶振。

2．S3C2410 的实时时钟单元

如图 3.1 为 S3C2410 的实时时钟框图。它具有以下特点：

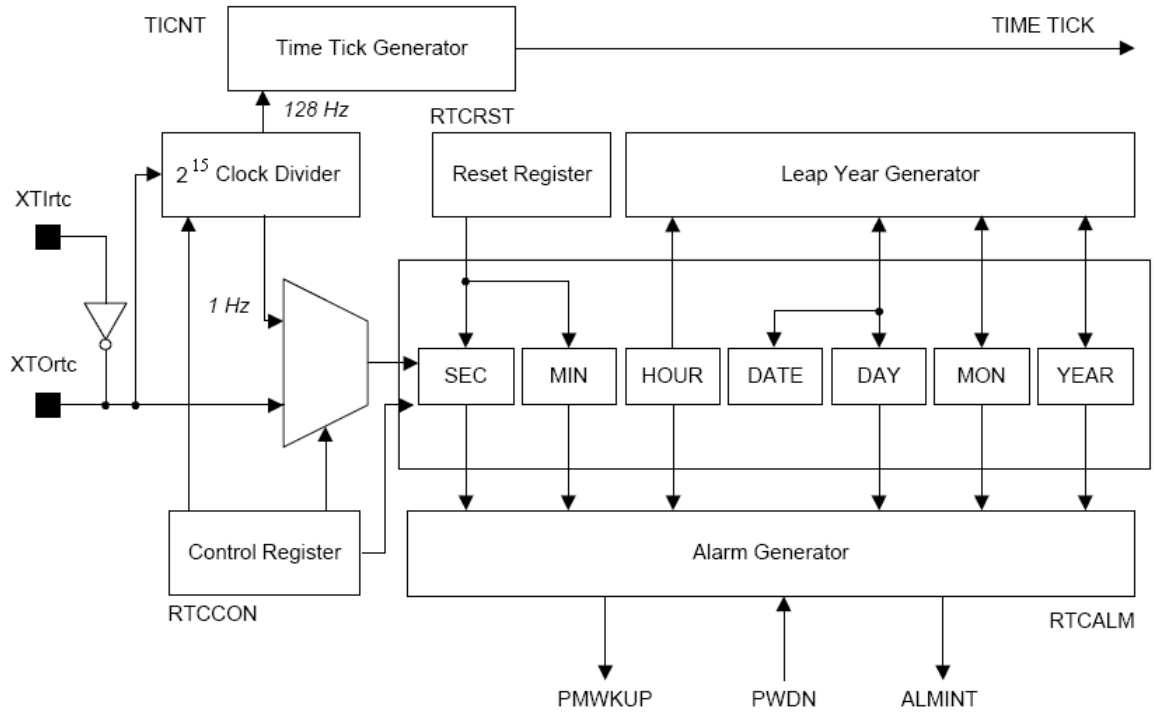


图 3.1　S3C2410 实时时钟框图

（1）时钟数据采用 BCD 编码；
（2）能够对闰年的年月日进行自动处理；
（3）具有告警功能，当系统处于关机状态时，能产生告警中断；
（4）无 2000 年问题；
（5）具有独立的电源输入；
（6）提供毫秒级时钟中断，该中断可用作嵌入式操作系统的内核时钟。

3．S3C2410 的实时时钟寄存器

（1）控制寄存器

表 3.1 控制寄存器

Register	Address	R/W	Description	Reset Value
RTCCON	0x57000040(L) 0x57000043(B)	R/W (by byte)	RTC control register	0x0

RTCCON	Bit	Description	Initial State
CLKRST	[3]	RTC clock count reset. 0 = No reset,　1 = Reset	0
CNTSEL	[2]	BCD count select. 0 = Merge BCD counters 1 = Reserved (Separate BCD counters)	0
CLKSEL	[1]	BCD clock select. 0 = XTAL 1/2^{15} divided clock 1 = Reserved (XTAL clock only for test)	0
RTCEN	[0]	RTC control enable.. 0 = Disable　1 = Enable	0

		NOTE: Only BCD time count and read operation can be performed.	

（2）告警控制寄存器

表 3.2 告警控制寄存器

Register	Address	R/W	Description	Reset Value
RTCALM	0x57000050(L) 0x57000053(B)	R/W (by byte)	RTC alarm control register	0x0

RTCALM	Bit	Description	Initial State
Reserved	[7]		0
ALMEN	[6]	Alarm global enable. 0 = Disable, 1 = Enable	0
YEAREN	[5]	Year alarm enable. 0 = Disable, 1 = Enable	0
MONREN	[4]	Month alarm enable. 0 = Disable, 1 = Enable	0
DAYEN	[3]	Day alarm enable. 0 = Disable, 1 = Enable	0
HOUREN	[2]	Hour alarm enable. 0 = Disable, 1 = Enable	0
MINEN	[1]	Minute alarm enable. 0 = Disable, 1 = Enable	0
SECEN	[0]	Second alarm enable. 0 = Disable, 1 = Enable	0

3）实时时钟计数器

表 3.3 实时时钟计数器

Register	Address	R/W	Description	Reset Value
TICNT	0x57000044(L) 0x57000047(B)	R/W (by byte)	Tick time count register	0x0

TICNT	Bit	Description	Initial State
TICK INT ENABLE	[7]	Tick time interrupt enable. 0 = Disable 1 = Enable	0
TICK TIME COUNT	[6:0]	Tick time count value (1~127). This counter value decreases internally, and users cannot read this counter value in working.	000000

4）告警时间寄存器

包括年月日时分秒。它们都以 BCD 的格式表示，地址见下表：

表 3.4 告警时间寄存器

Register	Address	R/W	Description	Reset Value
ALMSEC	0x57000054(L) 0x57000057(B)	R/W (by byte)	Alarm second data register	0x0

ALMMIN	0x57000058(L) 0x5700005B(B)	R/W (by byte)	Alarm minute data register	0x00
ALMHOUR	0x5700005C(L) 0x5700005F(B)	R/W (by byte)	Alarm hour data register	0x0
ALMDATE	0x57000060(L) 0x57000063(B)	R/W (by byte)	Alarm data data register	0x01
ALMMOM	0x57000064(L) 0x57000067(B)	R/W (by byte)	Alarm month data register	0x01
ALMYEAR	0x57000068(L) 0x5700006B(B)	R/W (by byte)	Alarm year data register	0x0

注：1、以上各寄存器都只能以字节模式进行读写操作；2．当系统使用 BIG-ENDIAN 模式时，各寄存器对应的地址位会有所改变，具体值请参考 S3C2410 的数据手册。

4．实验说明

本实验测试 S3C2410 的实时时钟功能，包括：时钟滴答功能测试，通过 LED 显示当前时刻；时间告警功能测试。

1）时钟滴答功能测试

a．首先需设置 TICK 的周期，在例程中设置的是 1 秒，并打开 TIME TICK 中断。

```
rTICNT=0x7F|0X80;
```

计算公式：T=（1+0x7f）/128 秒

b．注册中断服务例程

使用 install_isr_handler（HandleRTC,（void*）rtc_int_isr）将中断服务例程的地址填写到中断地址表中；

rtc_int_isr 函数为 RTC TIMETICK 的处理例程,在该例程中我们首先要清除中断标志,接着可以刷新 LED 的显示,在例程中我们使用了全局变量来实现 LED 显示的更新.代码如下：

```
rR_ISPC=BIT_TICK;
*（（U8*）0x0200006）=0x3E;
*（（U8*）0x0200004）=seg7table[led_index&0xF];
led_index++;
```

c．输入当前时间,并通过串口显示时间的运行情况

设置当前时间,即向各个寄存器赋初值：

```
rRTCON=0x01;
rBCDYEAR=p_date->year;
rBCDMON=p_date->mon;
rBCDDAY=p_date->day;
rBCDDATE=p_date->week_day;
rBCDHOUR=p_date->hour;
rBCDMIN=p_date->min;
rBCDSEC=p_date->sec;
rRTCCON=0x00;
```

d．每隔 1 秒读取一下时钟数据,并通过串口显示该数据

```
old_index=led_index;
Uart_Printf（“\r\n”）;
While（1）{
  /*每隔 1 秒更新一次数据*/
if（old_index!=led_index）{
  rtc_get_data（&m_data）;
  old_index=led_index;
  /*实时时钟数据为 BCD 码格式,以 16 进制显示*/
  Uart_Printf（“\r%02x： %02x： %02x”,m_date.hour,m_date.min,m_date.sec）;
  }
```

```
    };
```

2）时间告警功能测试

a．首先设置告警时间，如下例程设置每分钟的第 5 秒告警

```
m_date.sec=0x05;
rtc_alalm_set（&m_date.0x41）;
```

模式 0x41 表示使能 RTC 告警,以及使能秒时钟告警

b．注册中断例程，打开中断

```
install_isr_handler（HandleRTC,（void *）rtc_int_isr）;
rINTMSK=（rINTMSK&~（BIT_GLOBAL|BIT_RTC）;
```

c．中断服务例程中清除中断事件

```
rI_ISPC=BIT_RTC;
if（alarm_count&1）
    *（unsigned char*）0x20000000=0x0f;
else
    *（unsigned char*）0x20000000=0xff;
alarm_count++;
```

注意事项：

1. 程序最好使用中断方式处理 TIME TICK 以及告警；同时需要正确设置中断的工作方式，并使能相应的中断（详细程序请参考实验光盘下\ RTC）；

2. 读取的时钟数据格式是 BCD 码；在输入初始时间时也应该以 BCD 码的格式输入.

3.6 实验步骤

1．参照模板工程，新建一个工程 RTC，添加相应的文件，并修改 RTC 的工程设置；

2．创建 Main.c 并加入到工程 RTC 中；

3．编写程序每秒钟读取时钟滴答；

4．编写程序实现时间告警功能；

5．编译 RTC;

6．运行超级终端，选择正确的串口号，并将串口设置位：波特率（115200）、奇偶校验（None）、数据位数（8）和停止位数（1），无流控，打开串口；

7．装载程序并运行，如果运行正确，在超级终端中将会显示如图 3.2 所示内容：

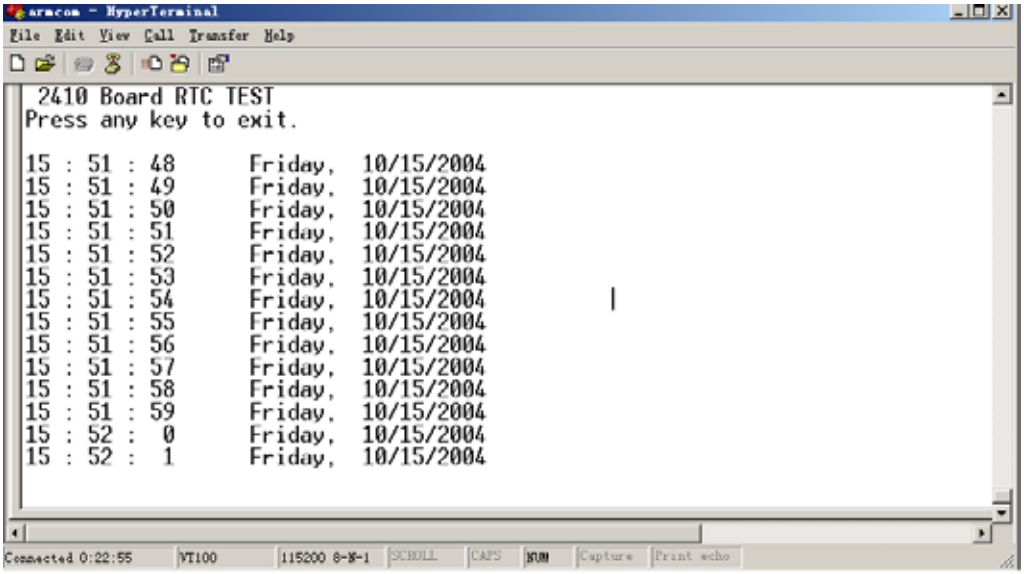


图 3.2　RTC 运行显示结果

3.7 实验报告要求

1．简述 RTC 的原理和应用；
2．说明 S3C2410 RTC 模块的使用方法。

实验四 I²C 实验 1—EEPROM 实验

4.1 实验目的

1．掌握硬件 I²C 接口的使用；
2．了解 AT24C16 的 E²PROM 的操作方法及注意事项。

4.2 实验内容

使用主模式 I²C 向 AT24C16 的 E²PROM 写入 0~FF。输入一个起始数字，校验正确则从起始数字开始依次写入数字一直到 FF，再写入 0~起始数字前的一个数字，停止写入；否则，如果校验错误则不断的蜂鸣报警。

4.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．熟悉 I²C 总线接口的用法；
3．了解 AT24C16 芯片的使用。

4.4 实验设备

1．ARM2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

4.5 基础知识

1．I²C 总线介绍

I²C 总线是一种用于 IC 器件之间连接的二进制总线。它通过 SDA（串行数据线）及 SCL（串行时钟线）两根线，在连到总线上的器件之间传送数据。它根据地址识别每个支持 I²C 接口的器件。

I²C 能替代不准的并行总线，能连接各种集成电路和功能模块。支持 I²C 的设备有微控制器、ADC、DAC、存储器、LCD 控制器、LED 驱动器以及实时时钟等。

（1）I²C 总线的基本结构

采用 I²C 总线标准的 IC 器件，其内部不仅有 I²C 接口电路，而且实现了将内部各单元电路按功能划分为若干相对独立的模块，通过软件寻址实现片选，减少了器件片选线的连接。CPU 不仅能通过指令将某个功能单元挂靠和摘离总线，还可对单元的工作状况进行检测，从而实现对硬件系统简单而灵活的扩展与控制。I²C 总线接口电路结构如图 4.1 所示。

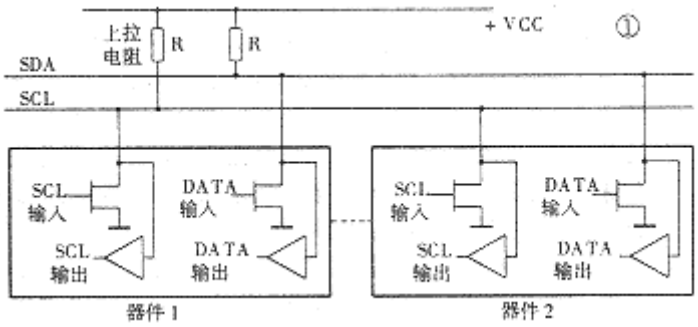


图 4.1　I²C 总线接口电路结构

（2）双向传输的接口特性

传统的单片机串行接口的发送和接收一般都各用一条线，如 MCS51 系列的 TXD 和 RXD，而 I²C 总线则根据器件的功能通过软件程序使其可工作于发送和接收方式。当某个器件向总线上发送信息时，它就是发送器（也称主器件），而当它从总线上接收信息时，又成为接收器（也称从器件）。主器件用于启动总线上传送数据并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。I²C 总线的控制完全由挂接在总线上的主器件送出的地址和数据决定。在总线上，既没有中心机，也没有优先机。

总线上主和从（即发送和接收）的关系不是一成不变的，而是取决于此时数据传送的方向。SDA 和 SCL 均为双向 I/O 线，通过上拉电阻接正电源。当总线空闲时，两根线都是高电平。连接总线的器件的输出级必须时集电极或漏极开路，以具有线“与”功能。I²C 总线的数据传送速率在标准工作方式下为 100kbit/s，快速方式下最高传送速率达 400kbit/s。

（3）I²C 总线上的时钟信号

在 I²C 总线上传送信息时的时钟同步信号是由挂接在 SCL 时钟线上的所有器件的连接“与”完成的。SCL 线上由高电平到低电平的跳变将影响到这些器件，一旦某个器件的时钟信号下跳为低电平，将使 SCL 线上一直保持低电平，使 SCL 线上的所有器件开始低电平期。此时，低电平周期短的器件的时钟由低至高的跳变并不能影响 SCL 线的状态，于是这些器件将进入高电平等待的状态。

当所有器件的时钟信号都上跳为高电平时，低电平期结束，SCL 线被释放返回高电平，即所有的器件都同时开始它们的高电平期。其后，第一个结束高电平期的器件又将 SCL 线拉成低电平。这样就在 SCL 线上产生一个同步时钟。可见，时钟低电平时间由时钟低电平期最长的器件确定，而时钟高电平时间由时钟高电平期最短的器件确定。

（4）数据的传送

在数据传送过程中，必须确认数据传送的开始和结束。在 I²C 总线技术规范中，开始和结束信号（也称启动和停止信号）的定义如图 4.2 所示。

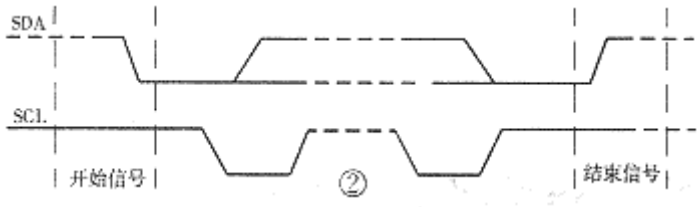


图 4.2 开始和结束信号的定义

当时钟线 SCL 为高电平时，数据线 SDA 由高电平跳变为低电平定义为“开始”信号；

当 SCL 线为高电平时，SDA 线发生低电平到高电平的跳变为“结束”信号。开始和结束信号都是由主器件产生。在开始信号以后，总线即被认为处于忙状态；在结束信号以后的一段时间内，总线被认为是空闲的。

I²C 总线的数据传送格式是：在 I²C 总线开始信号后，送出的第一个字节数据是用来选择从器件地址的，其中前 7 位是地址码，第 8 位是方向位（R/W）。方向位为“0”表示发送，即主器件把信息写到所选择的从器件；方向位为“1”表示主器件将从从器件读信息。开始信号后，系统中的各个器件将自己的地址和主器件送到总线上的地址进行比较，如果与主器件发送到总线上的地址一致，则该器件即为主器件寻址的器件，其接收信息还是发送信息则由第 8 位（R/W）确定。

在 I²C 总线上每次传送的数据字节数不限，但每一个字节必须为 8 位，而且每个传送的字节后面必须跟一个认可位（第 9 位），也叫应答位（ACK）。数据的传送过程如图 4.3 所示。每次都是先传最高位，通常从器件在接收到每个字节后都会作出响应，即释放 SCL 线返回高电平，准备接收下一个数据字节，主器件可继续传送。如果从器件正在处理一个实时事件而不能接收数据时，（例如正在处理一个内部中断，在这个中断处理完成之前就不能接收 I²C 总线上的数据字节）可以使时钟 SCL 线保持低电平，从器件必须使 SDA 保持高电平，此时主器件产生 1 个结束信号，使传送异常结束，迫使主器件处于等待状态。当从器件处理完毕时将释放 SCL 线，主器件继续传送。

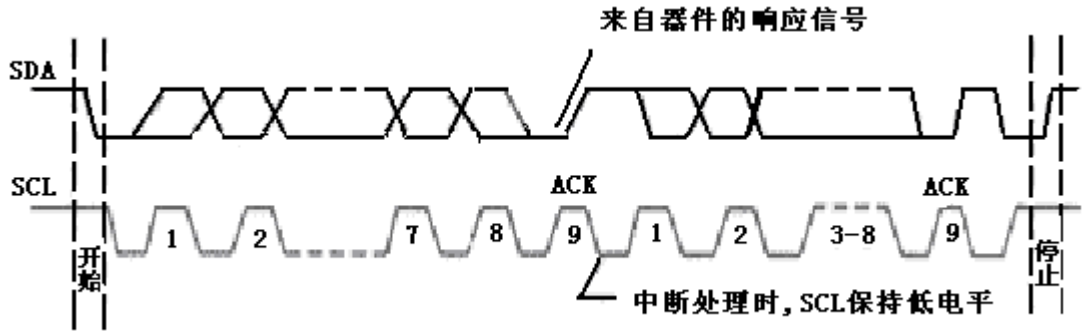


图 4.3 数据的传送过程

当主器件发送完一个字节的数据后，接着发出对应于 SCL 线上的一个时钟（ACK）认可位，在此时钟内主器件释放 SDA 线，一个字节传送结束，而从器件的响应信号结束后，SDA 线返回高电平，进入下一个传送周期。

I²C 总线还具有广播呼叫地址用于寻址总线上所有器件的功能。若一个器件不需要广播呼叫寻址中所提供的任何数据，则可以忽略该地址不作响应。如果该器件需要广播呼叫寻址中提供的数据，则应对地址作出响应，其表现为一个接收器。

（5）总线竞争的仲裁

总线上可能挂接有多个器件，有时会发生两个或多个主器件同时想占用总线的情况。例如，多单片机系统中，可能在某一时刻有两个单片机要同时向总线发送数据，这种情况叫做总线竞争。I²C 总线具有多主控能力，可以对发生在 SDA 线上的总线竞争进行仲裁，其仲裁原则是这样的：当多个主器件同时想占用总线时，如果某个主器件发送高电平，而另一个主器件发送低电平，则发送电平与此时 SDA 总线电平不符的那个器件将自动关闭其输出级。总线竞争的仲裁是在两个层次上进行的。首先是地址位的比较，如果主器件寻址同一个从器件，则进入数据位的比较，从而确保了竞争仲裁的可靠性。由于是利用 I²C 总线上的信息进行仲裁，因此不会造成信息的丢失。

（6）I²C 总线的一次典型工作流程

a.开始：信号表明传输开始。

b.地址：主设备发送地址信息，包含 7 位的从设备地址和 1 位的指示位（表明读或者写，即数据流的方向）。

c.数据：根据指示位，数据在主设备和从设备之间传输。数据一般以 8 位传输，具体能传输多少量的数据并没有限制。接收器上用一位的 ACK（回答信号）表明一个字节都收到了。传输可以被终止和重新开始。

d.停止：信号结束传输。

4．S3C2410 的 I²C 控制

S3C2410 处理器提供了一个 I²C 串行总线，其模块包括一个专门的串行数据线和串行时钟线。它的操作模式有四种：（1）主设备发送模式；（2）主设备接收模式；（3）从设备发送模式；（4）从设备接收模式。电路连接如图 4.4 所示

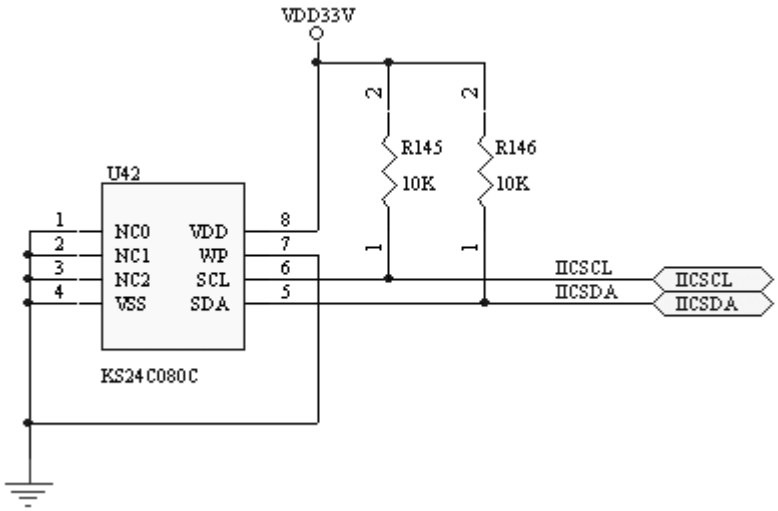


图 4.4　I²C 串行总线电路图

4.6 实验步骤

1．参照模板新建一个工程 EEPROM.mcp，添加相应的文件，并修改 EEPROM 的工程设置；

2．创建 main .c 文件，并加入到工程文件 EEPROM.mcp 中；

3．设计通过 I²C 向 AT24C16 的 E²PROM 写入 0~FF；

4．编译 EEPROM 工程；

5．下载程序并运行，在超级终端中观察结果，如果运行正确结果如图 4.5 所示：

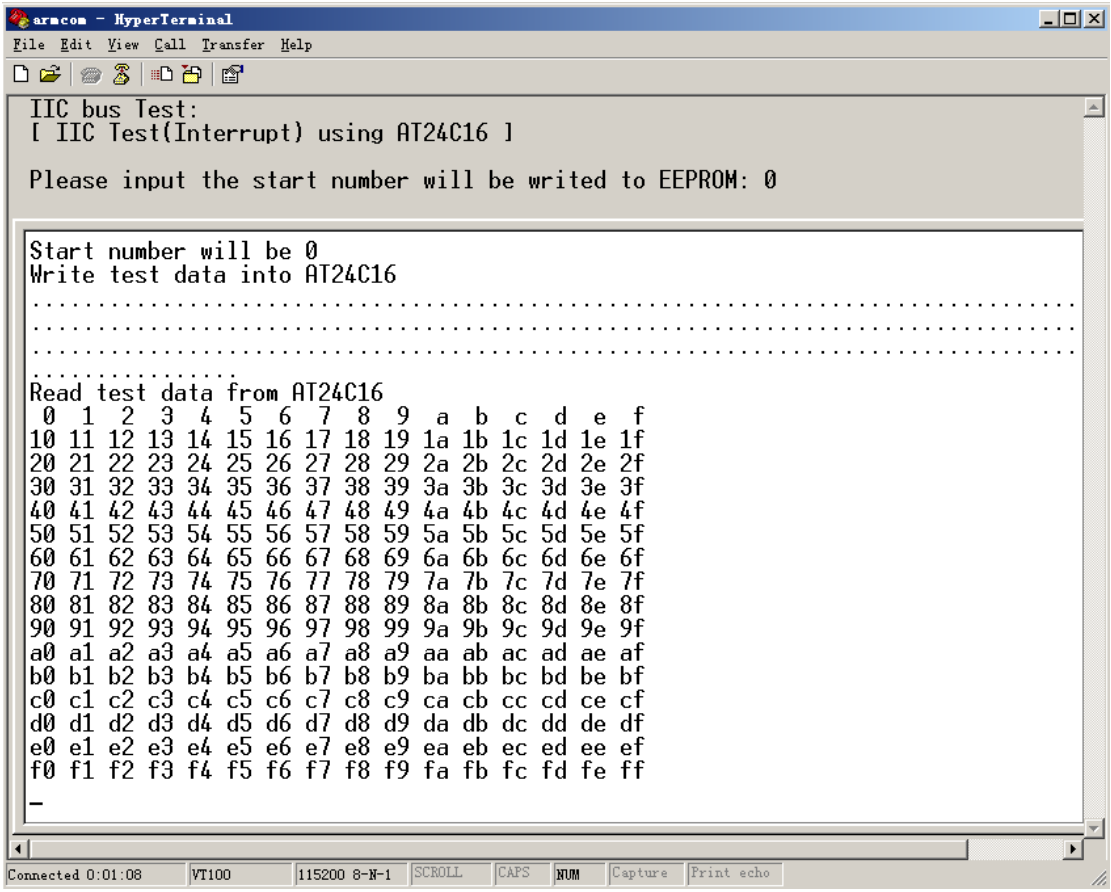


图 4.5　EEPROM 运行结果

4.7 实验报告要求

在实验参考程序中，若一次写入 20 字节数据，读出的数据是什么？（提示：EEPROM 页写问题）

实验五 I²C 实验 2—基于 I²C 数码管扫描电路

5.1 实验目的

1．了解数码管的显示原理；
2．了解 I²C 总线结构；
3．掌握基于 ARM 开发板实现对 I²C 器件的操作。

5.2 实验内容

使用 I²C 总线读写 ZLG7290LED 驱动器，在数码管上以“反显”的方式显示实时时钟中的“年.月.日”和“时.分.秒”。

5.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．熟悉 I²C 总线接口的用法；
3．了解 ZLG7290 器件的使用方法。

5.4 实验设备

1．ARM2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

5.5 基础知识

1．LED 显示原理

在嵌入式应用系统中，显示器是不可缺少的外部设备之一。为了便于人们观察和监视系统的运行情况，显示器常常用于显示系统运行的中间结果和状态信息等。显示器的种类很多，液晶、发光二极管以及 CRT 显示器等，都可以应用到嵌入式系统中。在一些小型应用系统中常常会用到发光二极管显示器，也就是通常说的 LED 显示器。LED 显示器具有耗电少、成本低、配置简单灵活、安装方便、耐振动和寿命常等优点。

7 段式 LED 由 7 个发光二极管按“日”字形排列，所有发光二极管的阳极连接在一起称为共阳极接法，阴极连接在一起称为共阴极接法。一般共阴极接法不需外接电阻，而共阳极接法中发光二极管必须外接电阻。LED 的结构及共阴、共阳接法如图 5.1 所示。

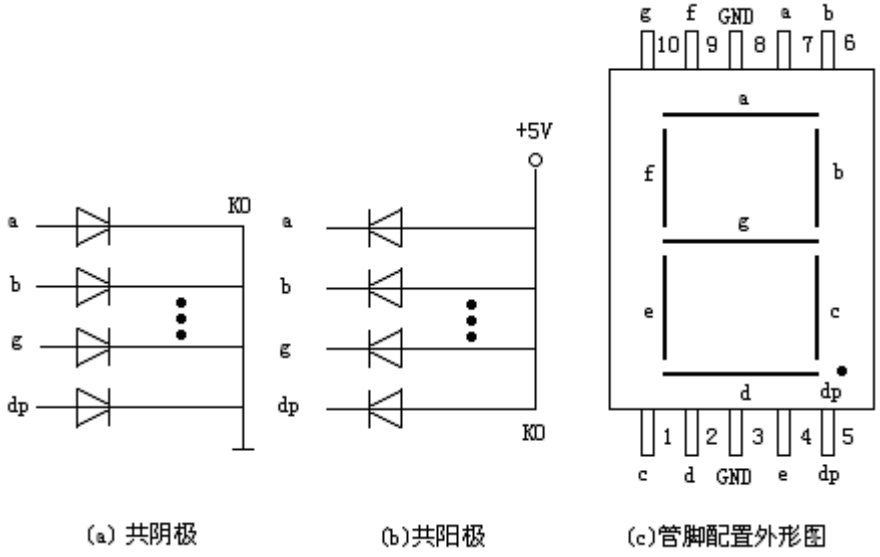


图 5.1　LED 的结构及共阴、共阳接法

当选用共阴极的 LED 显示器时，所有发光二极管的阴极连在一起接地，当某个发光二极管的阳极加上高电平时，对应的二极管点亮。因此要显示某字形就应使此字形的相应段的二极管点亮，也就时送一个用不同电平组合代表的数据字来控制 LED 的显示，此数据称为字符的段码。数字 0、1、2…9 以及字符 A、B、C、D、E、F 和 DP（小数点）的段码如表 5.1 所示。

表 5.1 共阴极 7 段 LED 显示字型编码表

显示字符	共阴极段选码	显示字符	共阴极段选码
0	3FH	9	6FH
1	06H	A	77H
2	5BH	B	7CH
3	4FH	C	39H
4	66H	D	5EH
5	6DH	E	79H
6	7DH	F	71H
7	07H	DP（小数点）	80H
8	7FH	熄灭	00H

说明：共阴的 LED，被选中时的段为高电平有效，熄灭的段码为 00H；
共阳的 LED，被选中时的段为低电平有效，熄灭的段码为 FFH。

2．LED 显示接口

LED 显示器的接口一般由静态显示和动态显示两种接口方式，分别介绍如下：

1）静态显示

LED 数码管采用静态接口时，共阴极或共阳极点连接在一起接地或接高电平。每个显示位的段选线于一个 8 位并行口线对应相连，只要在显示位上的段选线上保持段码电平不变，则该位就能保持相应的显示字符。这里的 8 位并行可可以直接采用并行 I/O 口，也可以采用串入/并出的移位寄存器或时其它具有三态功能的锁存器等。

2）动态显示

在多个 LED 同时需要显示时，为了简化电路，降低成本，将所有位的段选线并联在一

起，由一个 8 位 I/O 口控制。而共阴（或共阳）极公共端分别由不同的 I/O 线控制，实现各位的分时选通。由于各个数码管是共用同一个段码输出口，分时轮流通电的，因而硬件电路得到了简化，成本也随之降低。不过这种方式的数码管接口电路中不宜接太多的数码管，一般在 8 个以内，否则每个数码管所分配的实际导通时间太少，使得亮度显得不足。若 LED 个数较多，应采取措施增加驱动能力，从而提高显示亮度。

本实验系统中采用的是动态显示接口，共阴极接法，8 个数码管的位选通分别由 ZLG7290 芯片的 DIG0－DIG7 提供，而数码管的显示数据是通过控制芯片的 SegA－SegH 的组信号提供的。

这里提到的 ZLG7290 芯片是一款通过 I²C 总线实现接口键盘和 LED 驱动的专用芯片。（具体的芯片介绍，读者可参考 ZLG7290.PDF 文档）ARM 处理器是通过 I²C 总线与该芯片实现通信。接下来，就介绍一下 I²C 总线协议。

3．I²C 总线介绍

I²C 总线是一种用于 IC 器件之间连接的二进制总线。它通过 SDA（串行数据线）及 SCL（串行时钟线）两根线，在连到总线上的器件之间传送数据。它根据地址识别每个支持 I²C 接口的器件。

I²C 能替代不准的并行总线，能连接各种集成电路和功能模块。支持 I²C 的设备有微控制器、ADC、DAC、存储器、LCD 控制器、LED 驱动器以及实时时钟等。

1）I²C 总线的基本结构

采用 I²C 总线标准的 IC 器件，其内部不仅有 I²C 接口电路，而且实现了将内部各单元电路按功能划分为若干相对独立的模块，通过软件寻址实现片选，减少了器件片选线的连接。CPU 不仅能通过指令将某个功能单元挂靠和摘离总线，还可对单元的工作状况进行检测，从而实现对硬件系统简单而灵活的扩展与控制。

IIC LED 控制器连接电路如图 5.2 所示：

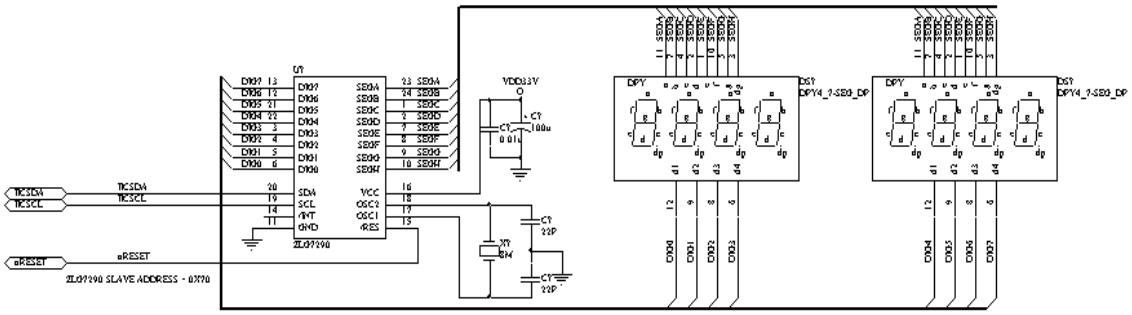


图 5.2　IIC LED 控制器连接电路

2）双向传输的接口特性

传统的单片机串行接口的发送和接收一般都各用一条线，如 MCS51 系列的 TXD 和 RXD，而 I²C 总线则根据器件的功能通过软件程序使其可工作于发送和接收方式。当某个器件向总线上发送信息时，它就是发送器（也称主器件），而当它从总线上接收信息时，又成为接收器(也称从器件)。主器件用于启动总线上传送数据并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。I²C 总线的控制完全由挂接在总线上的主器件送出的地址和数据决定。在总线上，既没有中心机，也没有优先机。

总线上主和从（即发送和接收）的关系不是一成不变的，而是取决于此时数据传送的方向。SDA 和 SCL 均为双向 I/O 线，通过上拉电阻接正电源。当总线空闲时，两根线都时高电平。连接总线的器件的输出级必须时集电极或漏极开路，即具有线“于”功能。I²C

总线的数据传送速率在标准工作方式下为 100kbit/s，快速方式下最高传送速率达 400kbit/s。

3）I²C 总线上的时钟信号

在 I²C 总线上传送信息时的时钟同步信号时由挂接在 SCL 时钟线上的所有器件的连接“与”完成的。SCL 线上由高电平到低电平的跳变将影响到这些器件，一旦某个器件的时钟信号下跳为低电平，将使 SCL 线上一直保持低电平，使 SCL 线上的所有器件开始低电平期。此时，低电平周期短的器件的时钟由低至高的跳变并不能影响 SCL 线的状态，于是这些器件将进入高电平等待的状态。

当所有器件的时钟信号都上跳为高电平时，低电平期结束，SCL 线被释放返回高电平，即所有的器件都同时开始它们阿高电平期。其后，第一个结束高电平期的器件又将 SCL 线拉成低电平。这样就在 SCL 线上产生一个同步时钟。可见，时钟低电平时间由时钟低电平期最长的器件确定，而时钟高电平时间由时钟高电平期最短的器件确定。

4）数据的传送

在数据传送过程中，必须确认数据传送的开始和结束。在 I²C 总线技术规范中，开始和结束信号（也称启动和停止信号）的定义如下图所示。当时钟线 SCL 为高电平时，数据线 SDA 由高电平跳变为低电平定义为“开始”信号；当 SCL 线为高电平时，SDA 线发生低电平电平到高电平的跳变为“结束”信号。开始和结束信号都时由主器件产生。在开始信号以后，总线即被认为处于忙状态；在结束信号以后的一段时间内，总线被认为时空闲的。

I²C 总线的数据传送格式时：在 I²C 总线开始信号后，送出的第一各字节数据时用来选择从器件地址的，其中前 7 位时地址码，第 8 位是方向位（R/W）。方向位为“0”表示发送，即主器件把信息写到所选择的从器件；方向位为“1”表示主器件将从从器件读信息。开始信号后，系统中的各个器件将自己的地址和主器件送到总线上的地址进行比较，如果与主器件发送到总线上的地址一致，则该器件即为主器件寻址的器件，其接收信息还是发送信息则由第 8 位（R/W）确定。

在 I²C 总线上每次传送的数据字节数不限，但每一个字节必须为 8 位，而且每个传送的字节后面必须跟一个认可位（第 9 位），也叫应答位（ACK）。数据的传送过程如下图所示。每次都是先传最高位，通常从器件在接收到每个字节后都会作出响应，即释放 SCL 线返回高电平，准备接收下一个数据字节，主器件可继续传送。如果从器件正在处理一个实时事件而不能接收数据时，（例如正在处理一个内部中断，在这个中断处理完成之前就不能接收 I²C 总线上的数据字节）可以使时钟 SCL 线保持低电平，从器件必须使 SDA 保持高电平，此时主器件产生 1 个结束信号，使传送异常结束，迫使主器件处于等待状态。当从器件处理完毕时将释放 SCL 线，主器件继续传送。

当主器件发送完一个字节的数据后，接着发出对应于 SCL 线上的一个时钟（ACK）认可位，在此时钟内主器件释放 SDA 线，一个字节传送结束，而从器件的响应信号结束后，SDA 线返回高电平，进入下一个传送周期。

I²C 总线还具有广播呼叫地址用于寻址总线上所有器件的功能。若一个器件不需要广播呼叫寻址中所提供的任何数据，则可以忽略该地址不作响应。如果该器件需要广播呼叫寻址中提供的数据，则应对地址作出响应，其表现位一个接收器。

5）总线竞争的仲裁

总线上可能挂接有多个器件，有时会发生两个或多个主器件同时想占用总线的情况。例如，多单片机系统中，可能在某一时刻有两个单片机要同时向总线发送数据，这种情况叫做总线竞争。I²C 总线具有多主控能力，可以对发生在 SDA 线上的总线竞争进行仲裁，其仲裁原则是这样的：当多个主器件同时想占用总线时，如果某个主器件发送高电平，而另一个主器件发送低电平，则发送电平与此时 SDA 总线电平不符的那个器件将自动关闭其

输出级。总线竞争的仲裁是在两个层次上进行的。首先是地址位的比较，如果主器件寻址同一个从器件，则进入数据位的比较，从而确保了竞争仲裁的可靠性。由于是利用 I²C 总线上的信息进行仲裁，因此不会造成信息的丢失。

6）I²C 总线的一次典型工作流程

开始：信号表明传输开始。

地址：主设备发送地址信息，包含 7 位的从设备地址和 1 位的指示位（表明读或者写，即数据流的方向）。

数据：根据指示位，数据在主设备和从设备之间传输。数据一般以 8 位传输，具体能传输多少量的数据并没有限制。接收器上用一位的 ACK（回答信号）表明一个字节都收到了。传输可以被终止和重新开始。

停止：信号结束传输。

2．S3C2410 的 I²C 控制

S3C2410 处理器提供了一个 I²C 串行总线，其模块包括一个专门的串行数据线和串行时钟线。它的操作模式有四种：

1）主设备发送模式；2）主设备接收模式；3）从设备发送模式； 4）从设备接收模式

5.6 实验步骤

1．熟悉参照附带工程 LED.mcp，了解实时时钟中数据的读取及通过 IIC 总线对 ZLG7290 芯片的读写。

2．计数据转换函数，把读到的数据转换为反显的数据

3．通过 IIC 总线把经过转换的数据写入到 ZLG7290 对应的数据寄存器中

4．通过数码管的显示验证自己的设计

5.7 实验报告要求

1．如何实现对编程实现对 ARM 的 I/O 控制；

2．用程序实现自己设计的点阵显示方式。

实验六 WATCHDOG 实验

6.1 实验目的

1．了解 WATCHDOG 的作用；
2．掌握 WATCHDOG 定时器的使用方法；

6.2 实验内容

1．编程添加看门狗功能，观察看门狗作用；
2．编程实现看门狗喂狗。

6.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．了解看门狗的原理和作用；
3．了解 S3C2410 看门狗的使用。

6.4 实验设备

1．S3C2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

6.5 基础知识

1．看门狗的功能及原理

嵌入式系统运行时受到外部干扰或者系统错误，程序有时会出现“跑飞”，导致整个系统瘫痪。为了防止这一现象的发生，在对系统稳定性要求较高的场合往往要加入看门狗（WATCHDOG）电路。看门狗的作用就是当系统“跑飞”而进入死循环时，恢复系统的运行。

其基本原理为：设本系统程序完整运行一周期的时间时 Tp，看门狗的定时周期为 Ti，Ti>Tp，在程序运行一周期后就修改定时器的计数值，只要程序正常运行，定时器就不会溢出，若由于干扰等原因使系统不能在 Tp 时刻修改时间的计数值，定时器将在 Ti 时刻溢出，引发系统复位，使系统得以重新运行，从而起到监控作用。

在一个完整的嵌入式系统和单片机最小系统中通常都有看门狗定时器，且一般集成在处理芯片中，看门狗实际上就是一个定时器，只是它在期满后将自动引起系统复位。

2．S3C2410 的看门狗控制

S3C2410 的看门狗定时器有两个功能：

1）作为常规定时器使用，并且可以产生中断；

2）作为看门狗定时器使用，期满时，它可以产生 128 个时钟周期的复位信号。

图 6.1 为 S3C2410 看门狗电路的示意图。输入时钟为 MCLK（该时钟频率对于系统的主频），它经过两级分频，最后将分频后的时钟作为该定时器的输入时钟，当计数器期满后可以产生中断或者复位信号。

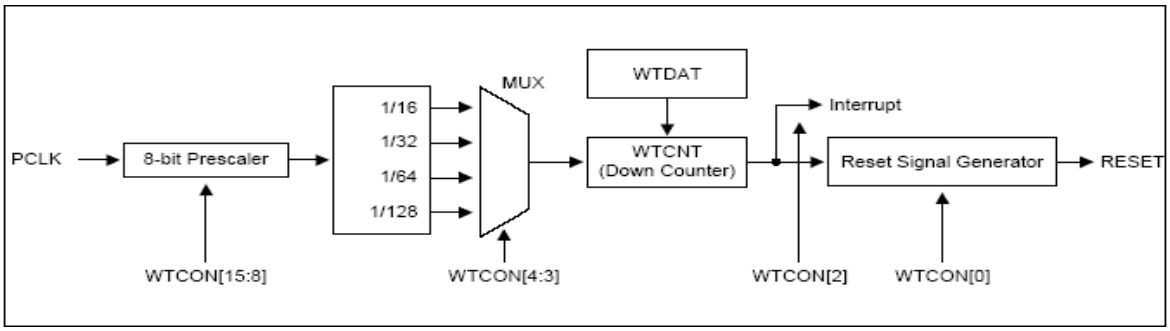


图 6.1　S3C2410 的看门狗

看门狗定时器计数值的计算公式如下：

输入到计数器的时钟周期：

t_watchdog = 1/（ PCLK /（Prescaler value + 1） / Division_factor ）

看门狗的定时周期：

T=WTCNT*t_watchdog

3．看门狗定时器寄存器

1）控制寄存器（WTCON）

通过该寄存器，可以使能/禁止看门狗、选择输入时钟源、使能/关闭中断、使能/关闭输出。该寄存器的地址以及各控制位的定义如下：

表 6.1 控制寄存器

Register	Address	R/W	Description	Reset Value
WTCON	0x53000000	R/W	Watchdog timer control register	0x8021

WTCON	Bit	Description	Initial State
Prescaler Value	[15:8]	Prescaler value. The valid range is from 0 to (28-1).	0x80
Reserved	[7:6]	Reserved. These two bits must be 00 in normal operation.	00
Watchdog Timer	[5]	Enable or disable bit of Watchdog timer. 0 = Disable 1 = Enable	1
Clock Select	[4:3]	Determine the clock division factor. 00:16　　01:32 10:64　　11:128	00
Interrupt Generation	[2]	Enable or disable bit of the interrupt. 0 = Disable 1 = Enable	0

Reserved	[1]	Reserved. This bit must be 0 in normal operation.	0
Reset Enable/Disable	[0]	Enable or disable bit of Watchdog timer output for reset signal. 1: Assert reset signal of the S3C2410X at watchdog time-out 0: Disable the reset function of the watchdog timer.	1

2）数据寄存器（WTDAT）

该时间寄存器用于设置看门狗定时器的超时值。在初始的操作中，该值不会自动加载到定时器中，首次定时器超时值为其初始值即 0x8000，以后该寄存器的值会被自动加载到 WTCNT 寄存器中。一般时该定时器工作在通用定时器模式下时使用。

表 6.2 数据寄存器

Register	Address	R/W	Description	Reset Value
WTDAT	0x53000004	R/W	Watchdog timer data register	0x8000

WTDAT	Bit	Description	Initial State
Count Reload Value	[15:0]	Watchdog timer count value for reload.	0x8000

3）计数器寄存器（WTCNT）

看门狗定时器的计数器，它的值表示该定时器的当前计数值，即到下一次期满还需要经历的时钟数。当该定时器工作在看门狗模式时使用寄存器，每次期满前需要重新设置其值，以防止发生复位。

表 6.3 计数器寄存器

Register	Address	R/W	Description	Reset Value
WTCNT	0x53000008	R/W	Watchdog timer count register	0x8000

WTCNT	Bit	Description	Initial State
Count Value	[15:0]	The current count value for the watchdog timer	0x8000

4．实验说明

本实验在“实时时钟实验”的基础上添加看门狗功能。

1）首先打开 WATCHDOG 定时器，并将其设置为 WATCHDOG 模式，让它引起复位

在上次实验的代码中增加使能 WATCHDOG 的代码，编译在运行，程序运行一段时间后应该被复位，现象就是不能正确的显示和执行串口打印；从超级终端上可以看到复位时的启动信息：

使能 WATCHDOG 的代码：

```
rWATCNT=0x1000;                                    /*设置看门狗初始值*/
rWATCON=BIT_WDT_ENABLE | BIT_WDT_RST_ENABLE | BIT_WDT_CLK_SEL|
BIT_WDT_PRE_SCALER;                                /*打开看门狗*/
```

其中 WDT_CLK_SEL 和 WDT_PRE_SCALER 的值如下：

```
#define WDT_CLK_SEL          （0x3<<3）      /*1/128*/
#define WDT_PRE_SCALER       （0xFF<<8）     /*255*/
```

此时看门狗定时器的时钟周期为（工作主频为 60MHz）

t_watchdog=1/（MCLK/（255+1）/128）;

将 WDTCNT 设置为 0x1000 ,其超时时间为：

t=0x1000/（60M/（255+1）/128）≈2 秒

根据计算, WATCHDOG 将会在 2 秒后引起复位;

2）使用程序定期的设置 WTCNT 寄存器（俗称“喂狗”），观察是否还出现复位情况，并记录复位的产生条件；

继续修改上一次实验的程序，在 TICK 中断中每秒重置一次 WDTCNT 的值（使用同一个值 0x1000），检查是否有复位发生：

```
rWATCNT=0x1000;                                    /*重新设置看门狗时间值*/
```

在 TICK 中断中每 3 秒重置一次 WDTCNT 的值（使用同一个值 0x1000）,检查是否有复位发生;

```
if（tick_index%3==0）
rWATCNT=0x1000;                                    /*重新设置看门狗时间值*/
```

6.6 实验步骤

1．参照模板工程，新建一个工程 watchdog，添加相应的文件，并修改 watchdog 的工程设置；创建 watchdog.c 并加入到工程 watchdog 中；

2．参照“定时器实验”的 timer.c，使定时器完成 2 秒后复位；

3．编译 watchdog，下载程序并运行，通过超级终端看是否复位；

4．添加 watchdog 定时器，并将其设置喂 watchdog 模式，重新编译运行，通过超级终端看是否复位,；

5．在 ICK 中断中每秒重置一次 WDTCNT 的值，重新编译运行，通过超级终端看是否复位,运行结果如图 6.2 所示。

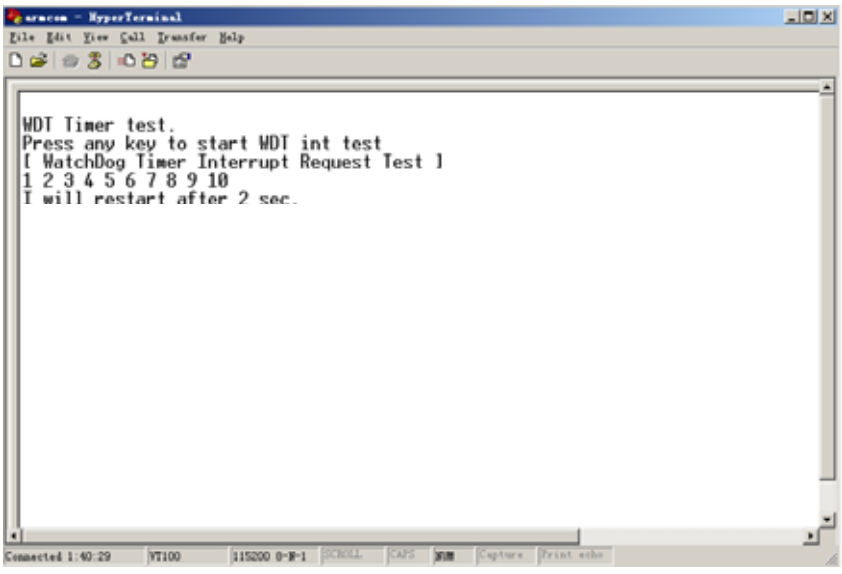


图 6.2　watchdog 运行结果

6.7 实验报告要求

简述 WATCHDOG 的原理和作用.

实验七 触摸屏控制实验

7.1 实验目的

1．通过实验掌握触摸屏的设计与控制方法；
2．熟练掌握 S3C2410 LCD 控制器的使用;
3．掌握 S3C2410 处理器的 A/D 转换功能.

7.2 实验内容

1．编程实现触摸屏坐标到 LCD 坐标的校准
2．编程实现触摸屏坐标采集以及 LCD 坐标的计算。

7.3 预备知识

1．学习触摸屏的原理；
2．A/D 采样的原理的显示原理和控制方法。
3．了解触摸屏与显示屏的坐标转换。

7.4 实验设备

1．S3C2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

7.5 基础知识

触摸屏的电路原理图如图 7.1 所示：

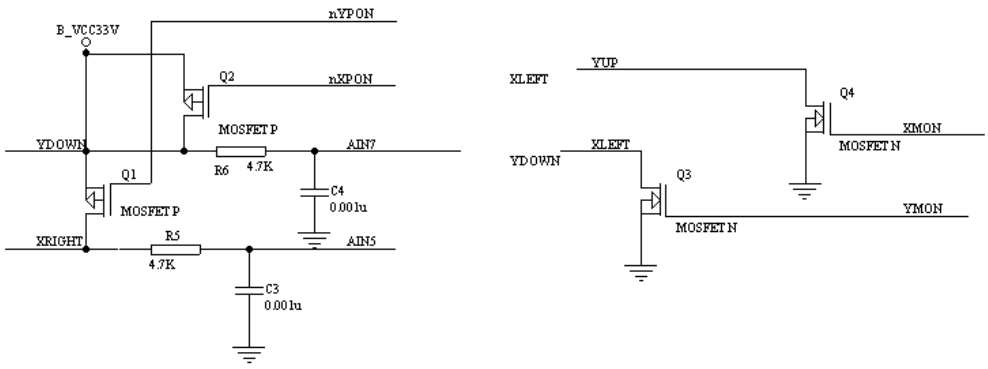


图 7.1　触摸屏的电路原理图

1．触摸屏的基本原理

触摸屏 （TSP, Touch Screen Panel ） 按其工作原理的不同分为避免声波屏、电容屏、电阻屏和红外屏几种。每一类触摸屏都有其各自的优缺点，下面简单介绍每一类触摸屏技术的工作原理和特点。

1）电阻式触摸屏

电阻式触摸屏的主要部分是一块与显示器表面非常配合的电阻薄膜屏，这是一种多层的复合薄膜，它以一层玻璃和硬塑料平板作为基层，表面涂有一层透明氧化金属（ITO 氧化铟，透明的导电电阻）导电层，上面在盖有一层外表面硬化处理、光滑防擦的塑料层、它的内表面也涂有一层 ITO 涂层、在他们之间有许多细小的（小于 1/1000 英寸）的透明隔离点隔开绝缘。当手指触摸屏幕时，两层导电层在触摸点位置就有了接触，如图 7.2 控制器侦测到这一接触并计算出（X，Y）的位置，再根据模拟鼠标的方式运作。这就是电阻式触摸屏的最基本的原理。

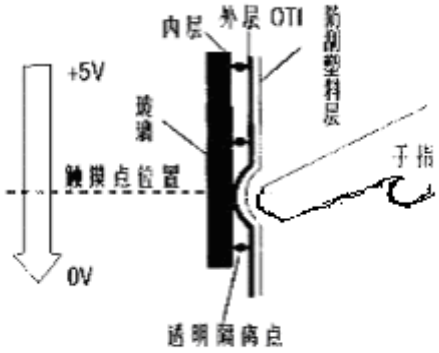


图 7.2　电阻式触摸屏按下状态

这种触摸屏的特点是：

✧　高解析度，高速传输反应；

✧　表面硬度处理，减少擦伤、刮伤及防化学处理。

✧　具有光面及雾面处理。

✧　一次校正，稳定性高，永不漂移。

2）表面声波触摸屏

表面声波触摸屏是利用声波在物体的表面进行传输，当有物体触摸到表面时，阻碍声波的传输，换能器侦测到这个变化，反映给计算机，进而进行鼠标的模拟。表面声波触摸屏的特点是：

✧　清晰度高，透光率好。

✧　高度耐久，抗刮伤性良好。

✧　一次校正不漂移。

✧　适合于办公室，机关单位及环境比较清洁的场所。

表面声波屏需要经常维护，因为灰尘，油污甚至饮料的液体玷污在屏的表面，都会阻塞触摸屏表面的导波槽，使声波不能正常反射，或使波形改变而控制器无法正常识别，从而影响触摸屏的正常使用，用户需严格注意环境卫生。必须经常擦抹屏的表面以保持屏面的光洁，并定期作一次全面彻底擦除。

3）电容式触摸屏

利用人体的电流感应进行工作。用户触摸屏幕时，由于人体电场，用户和触摸屏表面形成一个耦合电容，对应高频电流来说，电容是直接导体，于是手指从接触点吸走很小的电流。这个电流会从触摸屏的四角上的电极中流出，并且流经这四个电极的电流与手指到

四角的距离成正比，控制器通过对这四个电流比例的精确计算，得出触摸点的位置。电容触摸屏的特点是：

- ✧ 对大多数的环境污染物有抵抗力。
- ✧ 人体成为线路的一部分，因而漂移现象比较严重。
- ✧ 带手套不起作用。
- ✧ 需经常校准。
- ✧ 不适用于金属机柜。
- ✧ 当外界有电感和磁感的时候，会使触摸屏失灵。

2．触摸屏与显示器的配合

一般触摸屏将触摸时的 X、Y 方向的电压值送到 A/D 转换接口，经过 A/D 转换后的 X 与 Y 值仅是对当前触摸点的电压值的 A/D 转换值，它不具有实用价值。这个值的大小不但与触摸屏的分辨率有关，而且与触摸屏与 LCD 贴合的情况有关。

以四线电阻式触摸屏为例：每次按压后，将产生 4 个电压信号：X+、Y+、X−、Y−，它经过 A/D 得到相应的值，LCD 分辨率与触摸屏的分辨率一般是不一样的，坐标也不一样，因此，如果想得到体现 LCD 坐标的触摸屏位置，还需要在程序中进行转换。

3．实验说明

本实验主要目的是使学生了解触摸屏工作原理以及触摸屏数据采集编程方法。教学系统的触摸模块是由一个电阻式触摸屏和 S3C2410 的触摸屏控制电路组成。使用 S3C2410 的 GPIO 端口 E 来实现触摸屏的输入。触摸屏的采集步骤如下：

1）数据采集口初始化：

使用 S3C2410 的 GPIO 端口 E 来实现触摸屏的输入，在采集数据之前必须初始化端口。将 X 方向电压置低，具体指将 X 方向的某个脚对应的 I/O 端口设置为输出状态，并输出 0，其他脚对应的端口设置为输入状态，等待触摸事件发生，Ts_Sep 函数实现该初始化操作。

```
void Ts_Sep（void）
{
    Uart_Printf（"[Touch Screen Test.]\n"）;
    Uart_Printf（"Separate X/Y position conversion mode test\n"）;

    rADCDLY =  （50000）;   // ADC Start or Interval Delay

    rADCCON =  （1<<14）|（ADCPRS<<6）|（0<<3）|（0<<2）|（0<<1）|（0）;
      // Enable Prescaler,Prescaler,AIN7/5 fix,Normal,Disable read start,No operation
    rADCTSC =  （0<<8）|（1<<7）|（1<<6）|（0<<5）|（1<<4）|（0<<3）|（0<<2）|（3）;
      // Down,YM：GND,YP：AIN5,XM：Hi-z,XP：AIN7,XP pullup En,Normal,Waiting for
interrupt mode

    pISR_ADC     =  （unsigned）Adc_or_TsSep;
    rINTMSK      =~（BIT_ADC）;
    rINTSUBMSK =~（BIT_SUB_TC）;

    Uart_Printf（"\nType any key to exit!!!\n"）;
    Uart_Printf（"\nStylus Down, please..... \n"）;

    Uart_Getch（）;
```

```
    rINTSUBMSK |= BIT_SUB_TC;
    rINTMSK       |= BIT_ADC;
    Uart_Printf（"[Touch Screen Test.]\n"）;
}
```

2）等待触摸事件

如果此时有一个触摸动作，我们将在 Y 方向上得到低电平信号，由于这个信号同时被作为触摸屏中断信号的输入，所以会产生一个中断信号，以便进行采集。本实验通过 Adc_or_TsSep 函数等待触摸事件。程序如下：

```
void __irq Adc_or_TsSep（void）
{
    int i;
    U32 Pt[6];

    rINTSUBMSK |= （BIT_SUB_ADC|BIT_SUB_TC）; // Mask sub interrupt （ADC and TC）

      // TC（Touch screen Control） Interrupt
    if（rADCTSC & 0x100）
    {
        Uart_Printf（"\nStylus Up!!\n"）;
        rADCTSC &= 0xff;       // Set stylus down interrupt
    }
    else
    {
        Uart_Printf（"\nStylus Down!!\n"）;

          // <X-Position Read>
        rADCTSC=（0<<8）|（0<<7）|（1<<6）|（1<<5）|（0<<4）|（1<<3）|（0<<2）|（1）;
          // Down,Hi-Z,AIN5,GND,Ext vlt,Pullup Dis,Normal,X-position
        for（i=0;i<LOOP;i++）;                    //delay to set up the next channel
        for（i=0;i<5;i++）
        {
            rADCCON|=0x1;                        // Start X-position conversion
            while（rADCCON & 0x1）;              // Check if Enable_start is low
            while（!（0x8000&rADCCON））;     // Check ECFLG
            Pt[i]=（0x3ff&rADCDAT0）;
        }
        Pt[5]=（Pt[0]+Pt[1]+Pt[2]+Pt[3]+Pt[4]）/5;
        Uart_Printf（"X-Posion[AIN5] is %04d\n", Pt[5]）;

          // <Y-Position Read>
        rADCTSC=（0<<8）|（0<<7）|（1<<6）|（1<<5）|（0<<4）|（1<<3）|（0<<2）|（2）;
```

```
        // Down,GND,Ext vlt,Hi-Z,AIN7,Pullup Dis,Normal,Y-position
        for（i=0;i<LOOP;i++）;                    //delay to set up the next channel
        for（i=0;i<5;i++）
        {
            rADCCON|=0x1;                        // Start X-position conversion
            while（rADCCON & 0x1）;              // Check if Enable_start is low
            while（!（0x8000&rADCCON））;         // Check ECFLG
            Pt[i]=（0x3ff&rADCDAT1）;
        }
        Pt[5]=（Pt[0]+Pt[1]+Pt[2]+Pt[3]+Pt[4]）/5;
        Uart_Printf（"Y-Posion[AIN7] is %04d\n", Pt[5]）;
        rADCTSC=（1<<8）|（1<<7）|（1<<6）|（0<<5）|（1<<4）|（0<<3）|（0<<2）|
（3）;
          // Up,GND,AIN,Hi-z,AIN,Pullup En,Normal,Waiting mode
    }
```

3）采集数据

数据采集过程需要使用 A/D 来完成，为了保证正确性，通常我们采集 10 次以上并且去头去尾只取其中的部分数据再平均.。首先采集 Y 方向的数据，这时需要将 X+输出高电平，X−输出低电平；其次采集 X 方向的数据，这时需要将 Y+输出高电平，Y−输出低电平。

4）触摸屏的坐标

通过上述方式采集的坐标是相对于触摸屏的坐标，需要转换成为 LCD 坐标，这个过程之前需要进行两种坐标的校准工作，这里采用取平均值法。首先从触摸屏的 4 个顶角得到两个最大值和 2 个最小值，分别计为 x_min，y_min 和 x_max，y_max。X，Y 方向的确定如表 7.1 所示。

表 7.1　X，Y 方向的确定

方向	AD	N-MOS	P-MOS
X	AIN1	Q1（-）=0　Q2（+）=1	Q3（-）=1　Q4（+）=0
Y	AIN0	Q1（+）=1　Q2（-）=0	Q3（+）=0　Q4（-）=1

当系统处于休眠状态时，Q1，Q3 和 Q4 处于截止状态，Q2 导通。

当触摸屏被按下时，首先导通 MOS 管组 Q1 和 Q4，X+与 X-回路加上+3.3V 电源，同时将 MOS 管组 Q2 和 Q3 关闭，断开 Y+和 Y-，再启动处理器的 A/D 转换通道 1（AIN1），电路电阻与触摸屏按下产生的电阻输出分量电压，并由 A/D 转换器将电压值数字化，计算 X 轴的坐标。

接着先导通 MOS 管组 Q2 和 Q3，Y+与 Y-回路加上+3.3V 电源，同时将 MOS 管组 Q1 和 Q4 关闭，断开 X+和 X-，再启动处理器的 A/D 转换通道 0（AIN0），电路电阻与触摸屏按下产生的电阻输出分量电压，并由 A/D 转换器将电压值数字化，计算 Y 轴的坐标。

系统读到坐标值后，关闭 Q1、Q3 和 Q4，打开 Q2，回到初始状态，等待下一次笔触。

确定 X，Y 方向后，坐标值的计算公式如下：

$$X=（x\_max-Xa）\times 320 /（x\_max - x\_min）$$

$$Y=（y\_max- Ya）\times 240 /（y\_max- y\_min）$$

式中：

$$Xa=（X_1+X_2+...\ +Xn）/ n$$

$$Ya=（Y_1+Y_2+...\ +Yn）/ n$$

通过计算，触摸屏的坐标情况（n=5）如图 7.3 所示。

（320,240）

0	1	2	（240,180）
4	5	（160,120）	7
8	（80,60）	A	B
（0,0）	D	E	F

图 7.3　触摸屏的坐标范围

7.6 实验步骤

1．新建一个工程 TouchScr，添加相应的文件，并修改 TouchScr 的工程设置。

2．创建 TouchScr．c 并加入到工程 TouchScr 中

3．编写程序分别校正 LCD 左上角和右下角坐标；

4．编写程序采集触摸屏坐标屏将其转换到 LCD 坐标并通过串口打印出来；

5．编译 TouchScr，下载程序并运行，并观察超级终端上的输出结果 ，运行正确应该显示触摸点的横纵坐标位置，运行结果如图 7.4 所示．

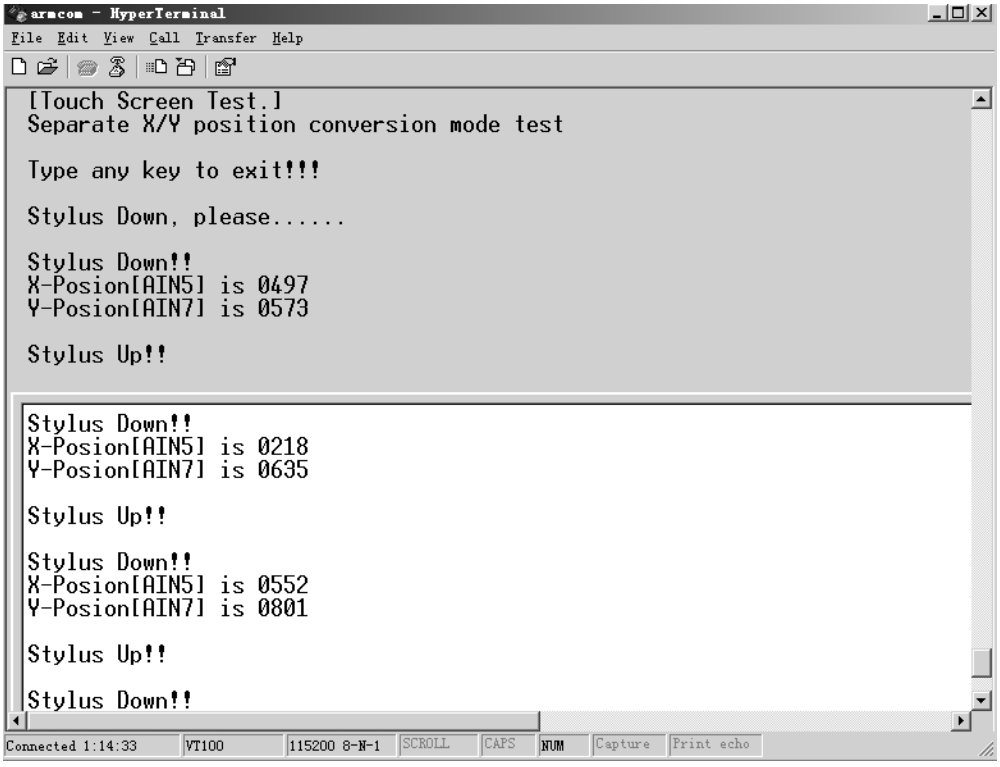


图 7.4 运行结果显示

7.7 实验报告要求

1．常见的触摸屏有哪几种，说明各自的优缺点；
2．四线电阻式触摸屏为例，说明电阻式触摸屏的工作原理；
3．举例说明触摸屏坐标与屏幕坐标之间的转换。

实验八 I/O 实验——8×8 发光二极管点阵试验

8.1 实验目的

1．学习并掌握 8×8 发光二极管点阵扫描显示的原理；
2．学习并掌握在发光二极管点阵上实现跑马灯试验的方法。

8.2 实验内容

1. 以动态扫描的方式在 8×8 的发光二极管点阵的中间位置显示一个 4×4 的方形点阵
2．在发光二极管点阵上实现跑马灯试验：
- ✧ 第 1 行、第 1 列的灯亮
- ✧ 第 1 行、第 2 列的灯亮
- ✧ ……
- ✧ 第 1 行、第 8 列的灯亮
- ✧ 第 2 行、第 1 列的灯亮
- ✧ ……
- ✧ 第 8 行、第 8 列的灯亮
- ✧ 重新返回第 1 行、第 1 列

3．在点阵上显示一个圆环，从中心开始，半径逐次增大，再重新开始（选做）

8.3 预备知识

1．熟悉 ADS 集成开发环境的仿真调试功能；
2．熟悉显示动态扫描的基本概念

8.4 实验设备

1．ARM2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序.

8.5 基础知识

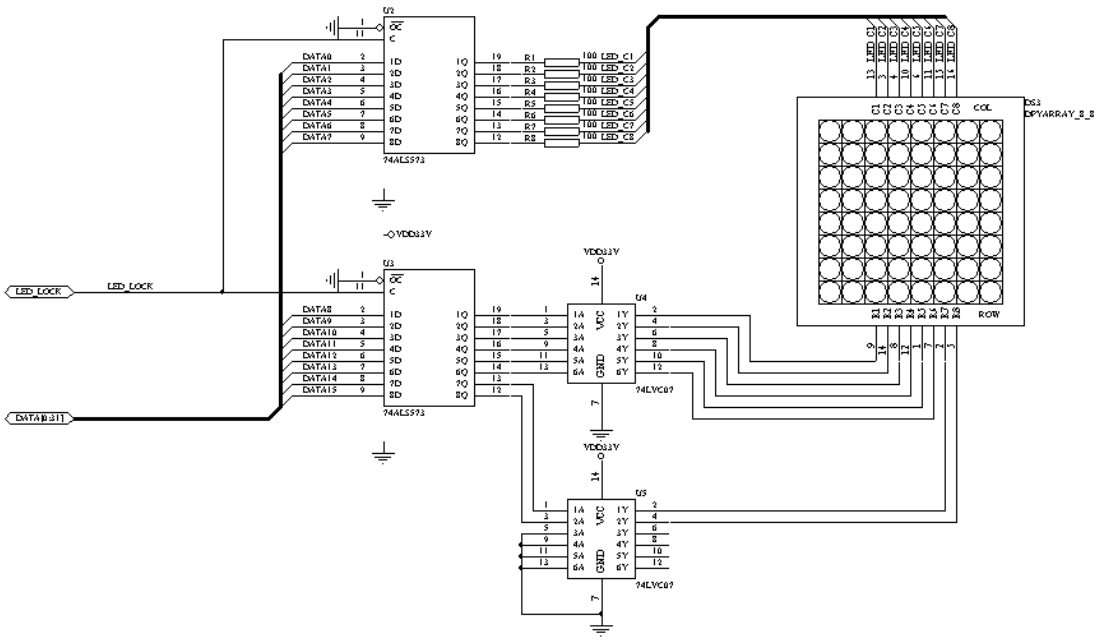


图 8.1　8×8 发光二极管点阵电路图

1．系统电路如图 8.1 所示。显示部分用的是一个 8×8 发光二极管点阵，我们常见的用于发布消息、显示汉字的点阵式 LED 显示屏通常由若干块 LED 点阵显示模块组成，8×8 显示点阵模块，每块有 64 个独立的发光二极管，为了减少引脚且便于封装，各种 LED 显示点阵模块都采用阵列形式排布，即在行列线的交点处接有显示 LED。（因此，LED 点阵显示模块的显示驱动只能采用动态驱动方式，每次最多只能点亮一行 LED（共阳形式 LED 显示点阵模块）或一列 LED（共阴形式 LED 显示点阵模块）。如上图所示的显示驱动原理图中，点阵为共阴，由总线驱动芯片 74244 为点阵显示模块提供列驱动电流，8 个行信号则由集电极开路门驱动器 7407 控制，行线和列线都挂在总线上，微处理器可以通过总线操作来完成对每一个 LED 点阵显示模块内每个 LED 显示点的亮、暗控制。

2．I/O 接口

在本开发板上，整个 LED 显示模块是作为一个 I/O 进行控制的。如电路原理图所示，DATA[0..7], DATA[8..15]分别对应系统数据线的低 16 位，LED_LOCK 信号是由系统总线的写信号和地址信号经简单的逻辑组合而得，在板载的 CPLD 内完成，控制该显示模块的 I/O 地址为 0x20000000。

8.6 实验步骤

1．参照例子工程 LedAry.mcp，新建一个用户自己的工程，编译下载运行，观察板上发光二极管点阵的显示

2．在主程序中定义一个表示点阵数据的数据，加入自己的动态扫描函数，实现对点阵数据的动态扫描

3．在扫描的基础上加入对点阵数据的改变的控制，实现走马灯的试验，在超级终端中仿真点阵的循环点亮，例如：Led row 0，col 0 ON 代表第一行第一列灯亮。如图 8.2 是超

级终端中显示的运行结果。

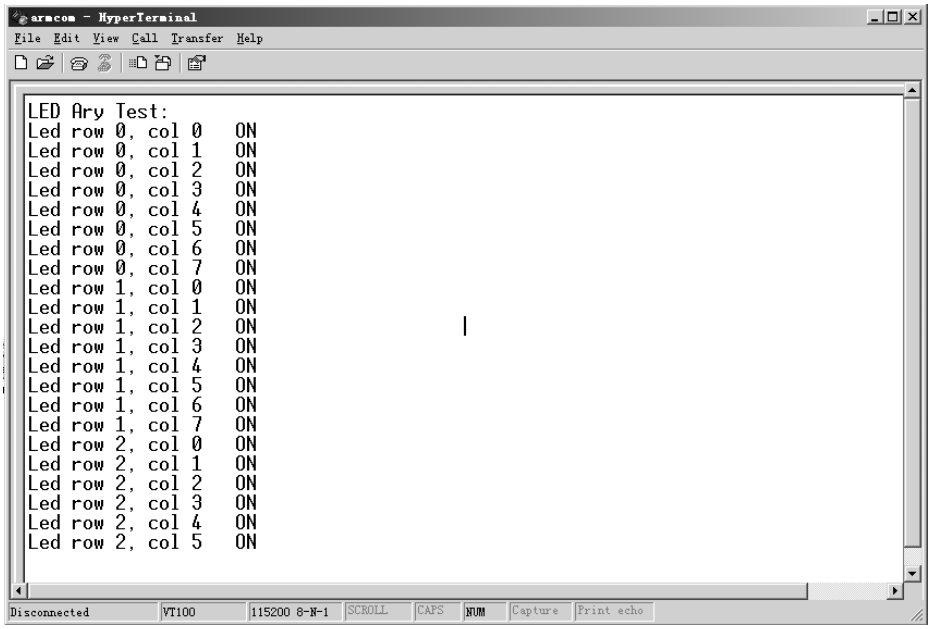


图 8.2　运行结果

8.7 实验报告要求

1．说明显示动态扫描的原理

2．说明在发光二极管点阵上实现走马灯试验的方法

实验九 A/D 转换实验

9.1 实验目的

1．了解模数转换的基本原理；
2．掌握模数转换的编程方法。

9.2 实验内容

1．编程对模拟量输入进行采集和转换，并将结果显示在超级终端上;
2．通过改变模拟量输入，观察显示结果。

9.3 预备知识

1．了解 A/D 采样的原理；
2．了解采样频率的设置。

9.4 实验设备

1．S3C2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

9.5 基础知识

1．A/D 转换的基本原理
1）采样和量化
作用：我们经常遇到的物理参数，如电流、电压、温度、压力、速度……电量和非电量都是模拟量。模拟量的大小是连续分布的，且经常也是时间上的连续函数。因此要将模拟量转换成数字信号需经采样——量化——编码三个基本过程（数字化过程）。
✧ 采样
按采样定理对模拟信号进行等时间间隔采样，将得到的一系列时域上的样值去代替 u＝f（t），即用 u0、u1、……un 代替 u＝f （t） .
这些样值在时间上是离散的值，但在幅度上仍然是连续模拟量。
✧ 量化
在幅值上采用离散值来表示。方法是用一个量化因子 Q 去度量：u1,u2,…,得到取整后的数字量.

u0=2.4Q =>2Q 010
u1=4.0Q =>4Q 100

u2=5.2Q =>5Q 101

u3=5.8Q =>5Q 101

✧ 编码

将整量化后的数字量进行编码，以便读入和识别：

编码仅是对数字量的一种处理方法。

例如：Q=0.5V/格,设用三位（二进制编码）

u0=2.4Q-------→2Q--------→（010）u0=（$0\times2^2+1\times2^1+0\times2^0$）×0.5V=1V

2）分类

按被转换的模拟量类型可分为时间/数字、电压/数字、机械变量/数字等。应用最多的是电压/数字转换器。电压/数字转换器又可分为多种类型：

按转换方式可分为：直接转换、间接转换。

按输出方式可分为：并行、串行、串并行。

按转换原理可分为：计数式、比较式。

按转换速度可分为：低速、中速、高速。

按转换精度和分辨率可分为：3 位、4 位、8 位、10 位、12 位、14 位、16 位等。

3）工作原理

类似于用天平称物体重量，设有一待测物为 4.42g；满度测量量程 R_{NFS}＝5.12g，砝码种类有四种：$0.5R_{NFS}$，$0.25\ R_{NFS}$，$0.125\ R_{NFS}$，$0.0625R_{NFS}$。

测量方法：先大砝码，后小砝码，依次比较（累计比较），要的记“1”，不要的记“0”。

实测物重 G：$1*0.5\ R_{NFS}+1*0.25\ R_{NFS}+0*0.125\ R_{NFS}+1*0.0625\ R_{NFS}$

一次为：2.56g<4.42g 留

二次为：2.56+1.28=3.84g<4.42g 留

三次为：3.84+0.64=4.44g>4.42g 去

四次为：3.84+0.32=4.16g <4.42g 留

误差= |4.16-4.42| = | -0.26 g | <0.32 g

误差<最小砝码（最小分辩砝码）

以上过程：通过 4 次比较后，得出结果；误差<最小砝码值。

2．S3C2410 的 A/D 转换器

1）A/D 转换有关的寄存器

表 9.1 A/D 转换有关的寄存器（一）

Register	Address	R/W	Description	Reset Value
ADCCON	0x58000000	R/W	ADC control register	0x3FC4

ADCCON	Bit	Description	Initial State
ECFLG	[15]	End of conversion flag (read only). 0 = A/D conversion in process 1 = End of A/D conversion	0
PRSCEN	[14]	A/D converter prescaler enable. 0 = Disable 1 = Enable	0
PRSCVL	[13:6]	A/D converter prescaler value. Data value: 1~255 Note that division factor is (N+1) when the prescaler value is N.	0xFF

SEL_MUX	[5:3]	Analog input channel select. 000 = AIN 0 001 = AIN 1 010 = AIN 2 011 = AIN 3 100 = AIN 4 101 = AIN 5 110 = AIN 6 111 = AIN 7 (XP)	0
STDBM	[2]	Standby mode select. 0 = Normal operation mode 1 = Standby mode	1
READ_START	[1]	A/D conversion start by read. 0 = Disable start by read operation 1 = Enable start by read operation	0
ENABLE_START	[0]	A/D conversion starts by setting this bit. If READ_START is enabled, this value is not valid. 0 = No operation 1 = A/D conversion starts and this bit is cleared after the start-up.	0

表 9.2　A/D 转换有关的寄存器（二）

Register	Address	R/W	Description	Reset Value
ADCDLY	0x58000008	R/W	ADC start or interval delay register	0x00ff

ADCDLY	Bit	Description	Initial State
DELAY	[15:0]	1)Normal Conversion Mode, Separate X/Y Position Conversion Mode, and Auto (Sequential) X/Y Position Conversion Mode. →X/Y Position Conversion Delay Value. 2)Waiting for Interrupt Mode. When Stylus down occurs in Waiting for Interrupt Mode, this register generates Interrupt signal (INT_TC) at intervals of several ms for Auto X/Y Position Conversion. **NOTE:** Do not use Zero value (0x0000)	00ff

NOTES:

1. Before ADC conversion, Touch screen uses X-tal clock or EXTCLK (Waiting for Interrupt Mode).
2. During ADC conversion, PCLK is used.

表 9.3　A/D 转换有关的寄存器（三）

Register	Address	R/W	Description	Reset Value
ADCDAT0	0x5800000C	R	ADC conversion data register	-

ADCDAT0	Bit	Description	Initial State
UPDOWN	[15]	Up or down state of Stylus at Waiting for Interrupt Mode. 0 = Stylus down state 1= Stylus up state	-
AUTO_PST	[14]	Automatic sequencing conversion of X-position and Y-position. 0 = Normal ADC conversion 1 = Sequencing measurement of X-position, Y-position	-
XY_PST	[13:12]	Manual measurement of X-position or Y-position. 00 = No operation mode 01 = X-position measurement 10 = Y-position measurement 11 = Waiting for Interrupt Mode	-
Reserved	[11:10]	Reserved	
XPDATA (Normal ADC)	[9:0]	X-position conversion data value. (include Normal ADC conversion data value) Data value: 0 – 3FF	-

表 9.4　A/D 转换有关的寄存器（四）

Register	Address	R/W	Description	Reset Value
ADCDAT1	0x58000010	R	ADC conversion data register	-

ADCDAT1	Bit	Description	Initial State
UPDOWN	[15]	Up or down state of Stylus at Waiting for Interrupt Mode. 0 = Stylus down state 1 = Stylus up state	-
AUTO_PST	[14]	Automatically sequencing conversion of X-position and Y-position. 0 = Normal ADC conversion 1 = Sequencing measurement of X-position, Y-position	-
XY_PST	[13:12]	Manual measurement of X-position or Y-position. 00 = No operation mode 01 = X-position measurement 10 = Y-position measurement 11 = Waiting for Interrupt Mode	-
Reserved	[11:10]	Reserved	
YPDATA	[9:0]	Y-position conversion data value Data value: 0 – 3FF	-

2）电路原理图

S3C2410 集成了一个 8 路 10 位的 A/D 转换器，电路图如图 9.1 所示，其分辨率为 10 比特，该转换器可以通过软件设置为 Sleep 模式，可以节电减少功率损失，最大转换速率为 500K，非线性度为正负 1 位。

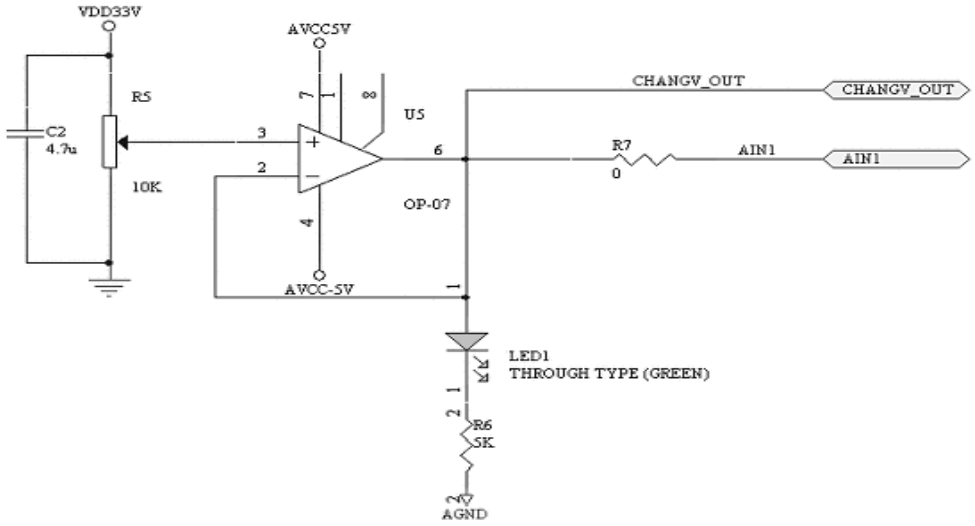


图 9.1　A/D 电路图

3．实验说明

1）设置 A/D 采样的时钟频率

A/D 采样频率取决于 ADCPSR 寄存器。假定 CPU 主时钟的频率为 66MHZ，并且将 A/D 采样预分频寄存器（ADCPSR）置为 20，而完成一次转换至少需要 16 个时钟周期，则采样频率可以采用下面公式计算：

f=66M /（2*（20+1））/16=98.2KHZ =10.2us

下面代码实现该功能：

rADCPSR=20;

值得注意的就是：尽管芯片的最大转换速率为 500KSPS，但由于 S3C2410 内部没有采样保持电路，所以要精确测量一个输入信号，输入信号的频率最好低于 100HZ。

2）启动采样

将 ADCCON 寄存器的 BIT0 置 1 可以启动转换，当启动转换后，该位会被自动清除。同时启动转换时还需要指定转换通道。

下面代码启动通道 2 的采样转换：

rADCCON=0x1 |（0x2<<2）;

3）获取转换结果

当 A/D 转换结束后，我们可以读取 ADCDAT 寄存器的内容。

下面代码等待通道 2 的 A/D 转换，完毕后读取数据。

While（!（rADCCON&0x40））;

Data=rADCDAT;

4．温度传感器

S3C2410EDU 实验平台中的温度传感器使用的是 LM35。LM35 是电压型集成温度传感器,其输出电压与摄氏温度成正比,无需外部校正,精度可达 0.5℃。

其电路符号如图 9.2 所示。特点有：

- ✧ 适用于 0℃～100℃的测量范围。
- ✧ 工作电压宽，4～30V。
- ✧ 输出阻抗低，1mA 时为 0.1，适用于远距离传送。
- ✧ 低功耗，吸入电流小于 60μA。
- ✧ 低自发热，在静止空气中小于 0.08℃。
- ✧ 直接以摄氏温度表示。
- ✧ 线性量程系数为＋10mA/℃。
- ✧ 精度可达 0.5℃。

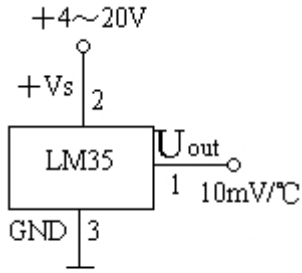


图9.2 LM35 电路符号

输出电压 Uout 与温度 t 的关系为： Uout＝10×t（mV）

例：25℃时，Uout＝250mA；100℃时，Uout＝1000Mv

温度传感器测量 AD 实验电路图如图 9.3 所示

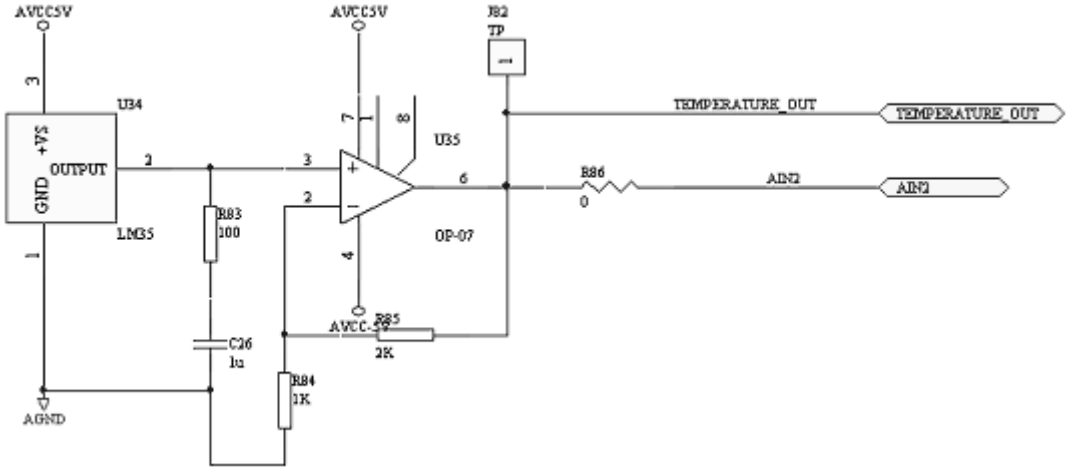


图 9.3 温度传感器测量 AD 实验电路图

9.6 实验步骤

1．新建一个工程 ad，添加相应的文件，并修改 ad 的工程设置；

2．创建 ad.c 并加入到工程 ad 中；

3．编写程序对通道 1 和通道 2 进行 A/D 转换，选择 0 或 1 （0：Display Count 100 ；1：Continued ）,我们这里选择 0，可以不间断的读取数据.

4．超级终端中显示的运行结果如图 9.2 所示。

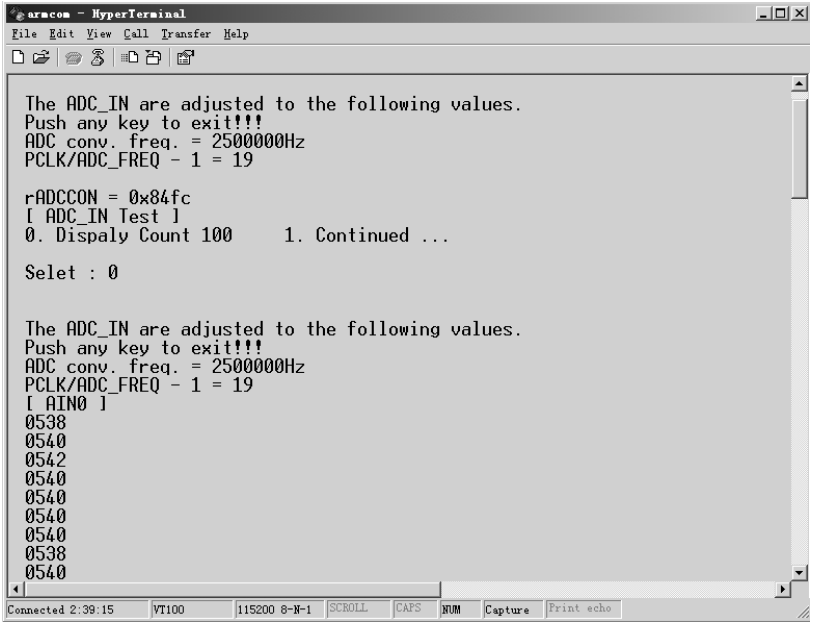


图 9.4 运行结果

9.7 实验报告要求

1．A/D 转换为什么要进行采样？采样频率应根据什么选定？

2．设输入模拟信号的最高有效频率位 5KHZ，应选用转换时间位多少的 A/D 转换器对它进行转换？

3．在本节实验基础上，参考温度传感器资料对 S3C2410 实验板上的温度传感器做 A/D 转换实验。

实验十 D/A 转换实验

10.1 实验目的

1．了解数模转换的基本原理；
2．掌握数模转换的编程方法。

10.2 实验内容

利用 D/A 转换器编程实现波形，并用示波器观察结果。

10.3 预备知识

1．了解数模转换的基本原理；
2．了解 DAC0832 功能及简单应用。

10.4 实验设备

1．S3C2410 嵌入式开发板，JTAG 仿真器。
2．软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，示波器。

10.5 基础知识

1．D/A 转换的基本原理

D/A 转换器（数/模转换器），它是把数字量转换成电模拟量。即把二进制数字量转换为与其数值成正比的电模拟量。

D/A 转换器的性能指标：

1）分辨率：是指 D/A 能转换的二进制位数，位数越多，分辨率越高；

例：转换 8 位，若电压满量程为 5V，则能分辨的最小电压为：5V/256≈20mV；

转换 10 位，若电压满量程为 5V，则能分辨的最小电压为：5V/1024≈5Mv。

2）转换时间：指数字量输入到转换输出稳定为止所需的时间；

3）精度：指 D/A 实际输出与理论值之间的误差，一般采用数字量的最低有效位作为衡量单位；

例：±1/2LSB，若是 8 位转换，则精度是±（1/2）×（1/256）满度=±1/512 满度。

4）线性度：当数字量变化时，D/A 输出的电模拟量按比例关系变化的程度。

模拟量输出偏离理想输出的最大值称为线性误差。

数据锁存问题：

有些 D/A 芯片本身不带数据锁存器，而 CPU 向 D/A 芯片输出一个数据只在 DB 上持

续很短时间，所以必须用外部芯片，如用 74LS273 或 8255A 作为 D/A 转换的数据锁存器，我们这里使用的是 74LS573 芯片。

2．8 位 D/A 转换芯片 0832 及其接口

0832 芯片采用 CMOS 工艺，电流输出型 D/A，8 位，转换时间约 1us。其内部结构框图如图 10.1 所示：

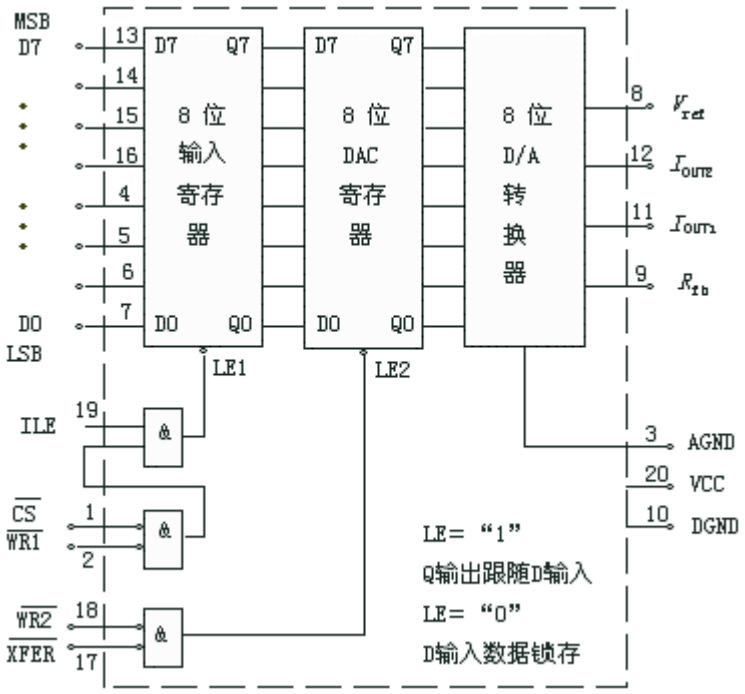


图 10.1　DAC0832 内部结构框图

a．8 位输入寄存器：可作为输入数据第一级缓冲；

b．8 位 DAC 寄存器：可作为输入数据第二级缓冲；

c．8 位 D/A 转换器：将 DAC 寄存器中的数据转换成具有一定比例的直流电流。

d．　逻辑控制部分：0832 芯片内部有两个数据缓冲器，分别由两组控制信号控制，当 ILE=1∩$\overline{CS}$=0∩$\overline{WR1}$=0 时，D_7～D_0 上的数据锁存到输入寄存器中；当$\overline{XFER}$=0∩$\overline{WR2}$=0 时，输入寄存器中的数据被锁存到 DAC 寄存器中。

外部引脚如图 10.2 所示：

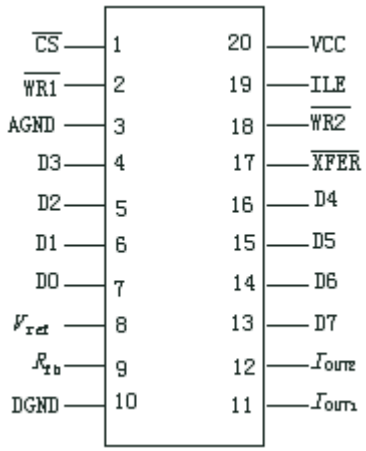


图 10.2　外部引脚图

其中：

D7～D0：8 位数据量输入；

ILE：数据输入锁存允许，高电平有效；

$\overline{CS}$：片选；

$\overline{WR1}$：输入寄存器写信号，当 ILE、 $\overline{CS}$、$\overline{WR1}$同时有效，数据装入输入寄存器，实现输入数据的第一级缓冲；

$\overline{XFER}$：数据传送控制信号，控制从输入寄存器到 DAC 寄存器的内部数据传送；

$\overline{WR2}$：DAC 寄存器写信号，当 $\overline{XFER}$和 $\overline{WR2}$均有效时，将输入寄存器中的数据装入 DAC 寄存器并开始 D/A 转换，实现输入数据的第二级缓冲；

V_{REF}：参考电压源，电压范围为-10V～+10V。

R_{fb}：内部反馈电阻接线端；

I_{OUT1}：DAC 电流输出 1。其值随输入数字量线性变化；

I_{OUT2}：DAC 电流输出 2。

当 DAC 寄存器内容全为 1 时，I_{OUT1} 最大，I_{OUT2}=0；

当 DAC 寄存器内容全为 0 时，I_{OUT1}=0，I_{OUT2}=最大；

当 DAC 寄存器内容为 N 时，$I_{OUT1}=V_{REF}\times N/(256\times R_{fb})$，$I_{OUT2}= V_{REF}/R_{fb}- I_{OUT1}$，无论 N 值多大：$I_{OUT1}+I_{OUT2}= V_{REF}/R_{fb}(1-2^8)$=常数≌$V_{REF}/R_{fb}$。

Vcc：工作电源，其值范围为+5V～15V，典型值为+15V；

AGND：模拟信号地线；

DGND：数字信号地线。

工作方式：

DAC0832 有双缓冲、单缓冲和直通三种方式。

- ✧ 双缓冲工作方式：进行两级缓冲；
- ✧ 单缓冲工作方式：只进行一级缓冲；
- ✧ 直通工作方式：不进行缓冲。适用于比较简单的场合。

接口电路

DAC0832 为电流输出型 D/A 转换芯片，使用时，R_{fb}、I_{OUT1}、I_{OUT2} 3 个引脚外接运算放大器，以便将转换后的电流变换成电压输出。若外接一个运算放大器为单极性输出，如图 11.　V_{OUT1} 输出；若使用了两个运算放大器为双极性输出，如图 10.3 V_{OUT2} 输出。

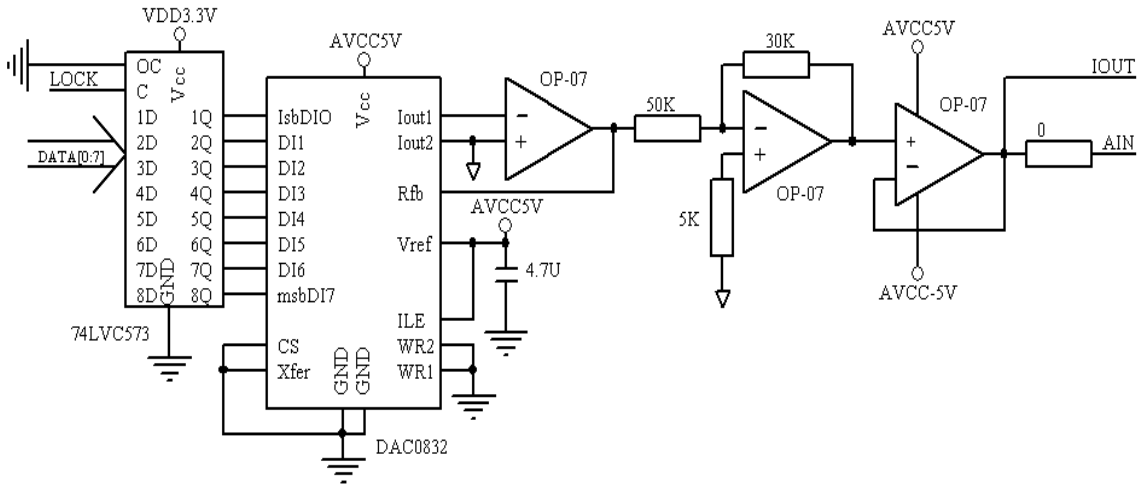


图 10.3　　DAC0832 接口电路

利用 D/A 转换器可以产生各种波形，如方波、三角波、锯齿波等，以及它们组合产生的复合波形和不规则波形。这些复合波形利用标准的测试设备是很难产生的。

下面是利用 DAC0832 在 V_{OUT1} 产生方波的程序段：

```
while （1）
{
    Write_DA（0xff）;              // 0xff 为高电平
    Delay（10）;                 // 持续 1ms
    Write_DA（0x00）;              // 0x00 为低电平
    Delay（10）;                 //持续 1ms
}
```

这样通过示波器就可以观察到一个方波了。

10.6 实验步骤

1．新建一个工程 DA．mcp，添加相应的文件，并修改 DA 的工程设置；

2．创建 DA．c 并加入到工程 DA 中；

3．编写 DAC0832 产生方波的程序。

4．编译运行程序，连接示波器，示波器一端接 CH1，开发板接板中央的 DA 口和地。

5．调节示波器的 TIME/DIV 和 VOLT/DIV 旋钮到波形最佳状态，打印波形如图 10.4 所示：

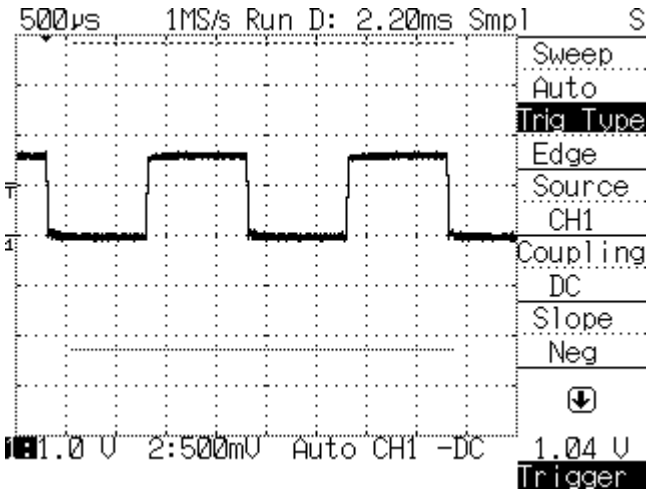


图 10.4　示波器打印结果

10.7 实验报告要求

1．试编写一个三角波或正弦波的程序，并用示波器打印结果；

2．简述 DA 转换的过程。

实验十一 音频接口 I²S 实验

11.1 实验目的

1．掌握有关音频处理的基础知识；
2．通过实验了解 I²S（Inter –IC Sound）音频接口的工作原理；
3．通过实验掌握对处理器 S3C2410 中 I²S 模块电路的控制方法；
4．通过实验掌握对常用 I²S 接口音频芯片的控制方法。

11.2 实验内容

编写程序播放一段 wav 文件保存的录音。

11.3 预备知识

1．熟悉 ADS 集成开发环境的基本功能；
2．了解 S3C2410 的音频模块的使用。

11.4 实验设备

硬件：S3C2410 嵌入式开发板，JTAG 仿真器。

软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

11.5 实验原理

1．数字音频基础

（1）采样频率和采样精度

在数字音频系统中，通过将声波波形转换成一连串二进制数据再现原始声音。这个过程中使用的设备是 A/D 转换器，即 ADC。ADC 以上万次每秒的速率对声波进行采样，每次采样都记录下了始声波在某一时刻的状态，称之为样本。

每秒采样的数目称为采样频率，单位为 Hz。采样频率越高，所能描述的声波频率就越高。系统对于每个样本均会分配一定的存储位（Bit 数）来表达声波的声波振幅状态，称之为采样精度。采样频率和精度共同决定声音还原的质量。

人耳的听觉范围通常是 20Hz～20kHz。根据奈奎斯特采样定理，用两倍于一个正弦波的频率进行采样能够真实的还原该波形；因此，当采样频率高于 40kHz 时，可以保证不产生失真。CD 音频的采样规格为 16 位、44kHz，就是根据以上原理制定的。

（2）音频编码

脉冲编码调制 PCM（Pulse Code Modulation）编码的方法是对语言信号进行采样，

然后对每个样值进行量化编码。对语音量化和编码就是一个 PCM 编码过程。ITU-T 的 64kb/s 语音编码标准 G．711 采用 PCM 编码方式，采样频率为 8kHz。每个样值用 8 为非线性的 μ 律或 A 律进行编码，总速率为 64kb/s。

CD 音频即是使用 PCM 编码格式，采样频率为 8kHz，对采样值采用 16 位编码。

使用 PCM 编码的文件在 Windows 系统中保存的文件格式为大家熟悉的 wav 格式，实验中用到的就是一个采样频率为 44.100kHz、16 位的立体声文件 t．wav。

在 PCM 基础上发展起来的还有自适应差分脉冲编码调制 ADPCM（Adaptive Differential Pulse Code Modulation）。ADPCM 编码的方法是对输入样值进行自适应预测，然后对预测误差进行量化编码。CCITT 的 32kb/s 语音编码标准 G．721 采用 ADPCM 编码方式，对每个语音采样值相当于使用 4 位进行编码。

其他编码方式还有线性预测编码 LPC（Linear Predictive Coding）、低时延码激励线性预测编码 LD-CELP（Low Delay-Code Excited Linear Prediction）等。

目前流行的一些音频编码格式还有 MP3（MPEG Audio Layer-3）、WMA（Windows Media Audio）和 RA（Real Audio）。它们有一个共同特点就是，压缩比高，主要针对网络传输，支持边读、边放。

2．I^2S 音频接口

I^2S 是一种串行总线设计技术，是 SONY 和 PHILIPS 公司等电子巨头共同推出的接口标准，主要针对数字音频处理技术和设备，例如便携 CD 机、数字音频处理器等。I^2S 将音频数据与时钟信号分离，避免由时钟带来的抖动问题，因此系统中不再需要消除抖动的器件。

I^2S 总线仅处理音频数据，对其他信号（如控制信号等）单独传送。基于减少引脚数目和布线的目的，I^2S 总线只由 3 根串行线组成；即分时复用的数据通道线（Serial Data，SD）、字选择线（Word Select，WS）和时钟线（Continuous Serial Clock，CSK）。

使用 I^2S 技术设计的系统连接配置参见图 11.1

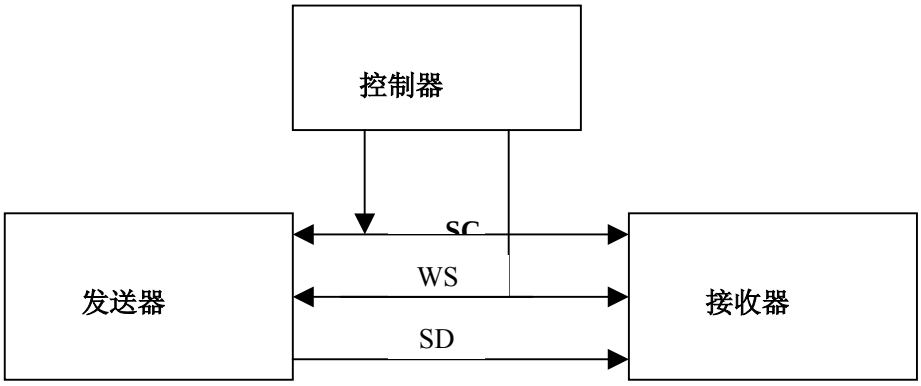


图 11.1　I^2S 系统连接配置图

I^2S 总线接口的基本时序参见图 11.2

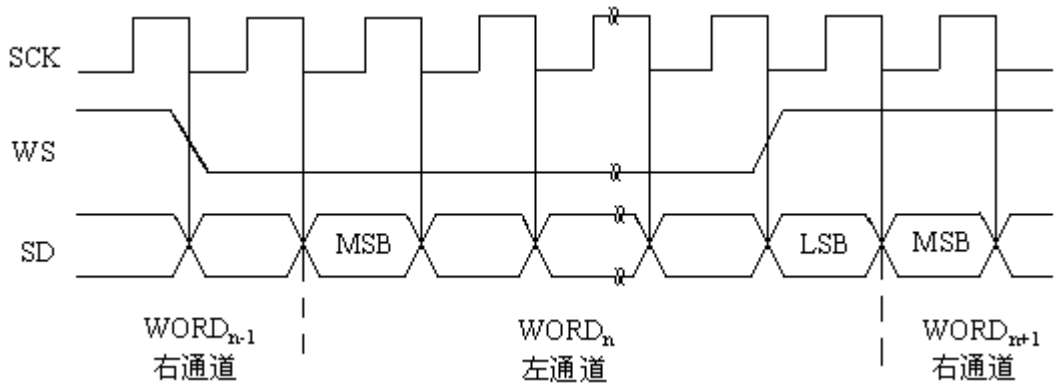


图 11.2　I²S 总线接口的基本时序

WS 信号线指示左通道或右通道的数据将被传输，SD 信号线按高有效位 MSB 到低有效位 LSB 的顺序传送字长的音频数据。MSB 总在 WS 切换后的第一个时钟发送。如果数据长度不匹配，那么接收器和发送器将对其自动截取或填充。关于 I²S 总线的其它细节可参见《I²S bus specification》。

在实验中，I²S 总线接口由处理器 S3C2410 的 I²S 模块和音频芯片 UDA1341 硬件来实现。需要关注的是对 I²S 模块和 UDA1341 芯片正确的配置，音频数据的传输相对来说比较简单。

3．WAV 声音格式

WAV 声音格式文件是 Windows 环境下的一种常用音频文件格式，它依循着一种称为“资源互换文件格式”（Resources Interchange File Format）的格式，简称 RIEF。RIEF 可以看做是一种树状结构，其基本构成单位为 chunk，犹如树状结构中的节点，每个 chunk 由“辨别码”、“数据大小”及“数据”所组成。

WAV 为 WAVEFORM（波形）的缩写。“RIEF”的格式辨别码为“WAVE”。整个文件由两个 chunk 所组成：辨别码“fmt”（注意，最后一个是空白字符!）及“data”。

在“fmt”的 chunk 下包含了一个 PCMWAVEFORMAT 数据结构，在“fmt”chunk 之后是原始声音的采样数据，该这些数据是可以直接送到 I²S 总线的数字音频符号。

一个典型的 WAV 格式文件结构如图 11.3 所示：

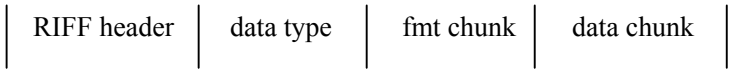


图 11.3　典型的 WAV 格式文件结构图

它包含 8 字节 RIFF 头、4 字节数据类型“WAVE”、“fmt”chunk（共 0x18 字节）和“data”chunk。因此，WAV 文件中从下式中的 sizeoff 开始的四字节表示声音数据的大小，dataoff 开始的位置为具体的声音数据。

sizeoff=0x8+0x4+0x18+0x4

dataoff=0x8+0x4+0x18+0x8

4．I²S 调用

（1）void uda1341_init（void）：UDA1341 音频 CODEC 初始化函数

（2）void dma_init（unsigned char * Buf, int size,int bplay）：BDMAO 控制寄存器设置，用于 I²S 录放音。其中 Buf 为音频数据缓冲区指针，size 为音频数据字节数，bplay 为 1 时初始化 BDMAO 为放音方式，bplay 为 0 时初始化 BDMAO 为录音方式。

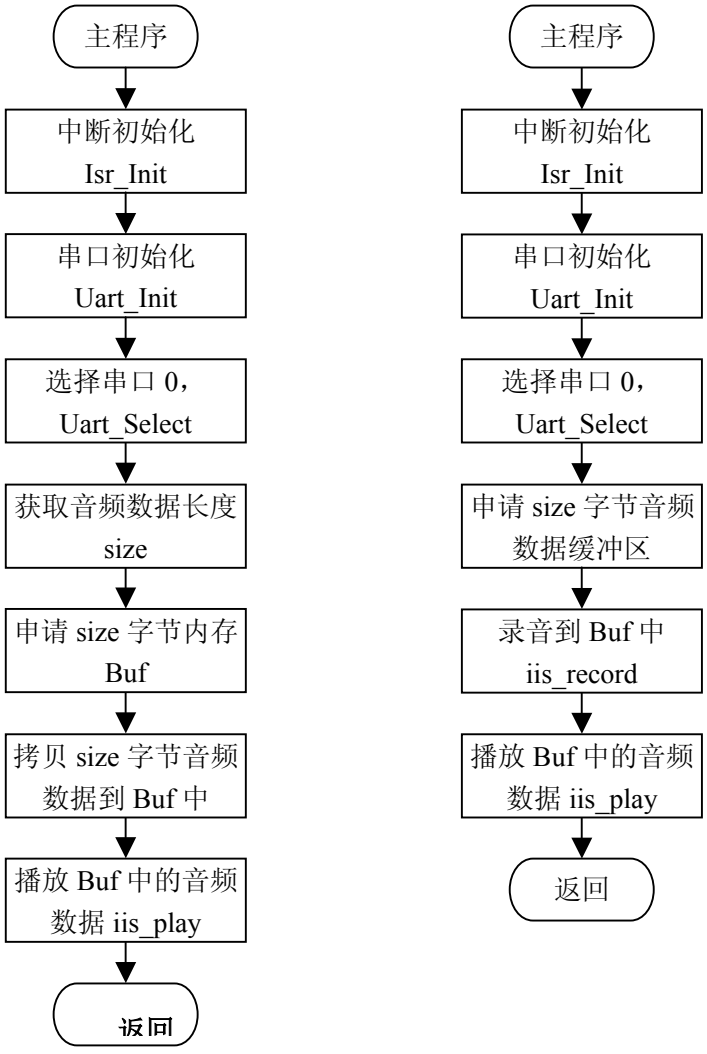
（3）void iis_init（int bplay）： I^2S 控制器初始化函数，bplay 为 1 时初始化 I^2S 为放音方式，bplay 为 0 时初始化 I^2S 为录音方式，本实验采用 16bit 双声道采样，采样频率为 22KHz。

（4）void iis_play（unsigned char * Buf, int size）：放音函数，Buf 为音频数据缓冲区指针，size 为音频数据字节数。

（5）void iis_record（unsigned char * Buf, int size）：录音函数，Buf 为音频数据缓冲区指针，size 为音频数据字节数。

5．录放音流程图

播放 WAV 文件流程图 A 如下所示，录音流程图如下图 B 所示：



A 放音流程图　　B 录音流程图

图 11.4　　录放音流程图

6．电路设计原理

（1）S3C2410 外围模块 I^2S 说明

S3C2410 数字音频接口模块由 I^2S（Inter-IC Sound）音频总线接口和 UDA1341 音频编解码器（CODEC）组成。

S3C2410 内置一个 I^2S 总线控制器，该总线专责于音频设备之间的数据传输，为数字立体声提供一个序列连接至标准编码解码器，它实现了到一个外部 8/16 位立体声音频

CODEC IC 的接口。支持 I²S 总线数据格式和 MSB-justified 数据格式。此控制器包含 FIFO，支持 DMA 传输模式。

I²S 总线控制器结构如图 11.5：

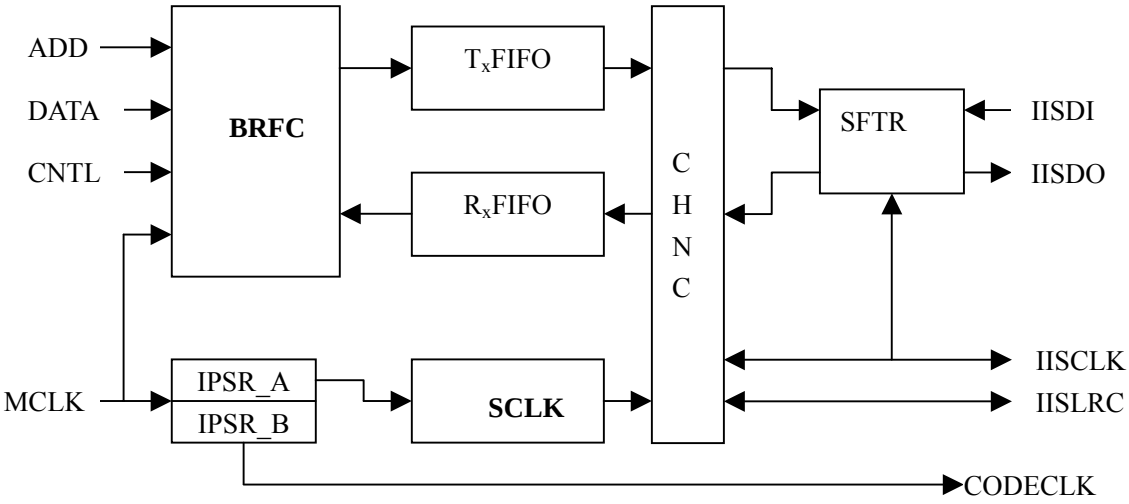


图 11.5　I²S 总线控制器结构

两个 3 比特预除器（IPSR）：一个（IPSR_A）用于产生 I²S 总线接口的主时钟，另外一个（IPSR_B）用作外部 CODEC 时钟产生器。

16 字节 FIFO：在发送数据时，数据被写进 TXFIFO，在接收数据时，数据从 RXFIFO 中读取。

主 I²SCLK 产生器（SCLKG）：在主模式，由主时钟产生串行位时钟。

通道产生器和状态机（CHNC）：I²SCLK 和 I²SLRCK 由通道状态机产生并控制。

16 比特移位寄存器（SFTR）：在发送数据时，并行数据经由 SFTR 变成串行数据输出，在接收数据时，串行数据由 SFTR 转变成并行数据。

（2）音频芯片 UDA1341TS 说明

电路中使用的音频芯片是 PHILIPS 公司的 UDA1341TS 音频数字信号编译码器。UDA1341TS 可将立体声模拟信号转化为数字信号，同样也能把数字信号转换成模拟信号，并可用 PGA（可编程增益控制）和 AGC（自动增益控制）对模拟信号进行处理。对于数字信号，该芯片提供了 DSP（数字音频处理）功能。实际使用中，UDA1341TS 广泛应用于 MD、CD、Notebook、PC 和数码摄象机等。

UDA1341TS 提供 2 组音频信号输入线、1 组音频信号输出线、1 组 I²S 总线接口信号线和 1 组 L3 总线。

I²S 总线接口信号线包括位时钟输入 BCK、字选择输入 WS、数据输入 DATAI，数据输出 DATAO 和音频系统时钟 SYSCLK 信号线。

UDA1341TS 的 L3 总线包括微处理器接口数据 L3DATA、微处理器接口模式 L3MODE、微处理器接口时钟 L3CLOCK 三根信号线。当该芯片工作于微处理器输入模式时，微处理器通过 L3 总线对 UDA1341TS 中的数字音频处理参数和系统控制参数进行配置。处理器 S3C2410 中没有 L3 总线专用接口，电路中使用 I/O 口连接 L3 总线。

（3）电路连接

S3C2410 数字音频接口模块如图 11.6 所示。

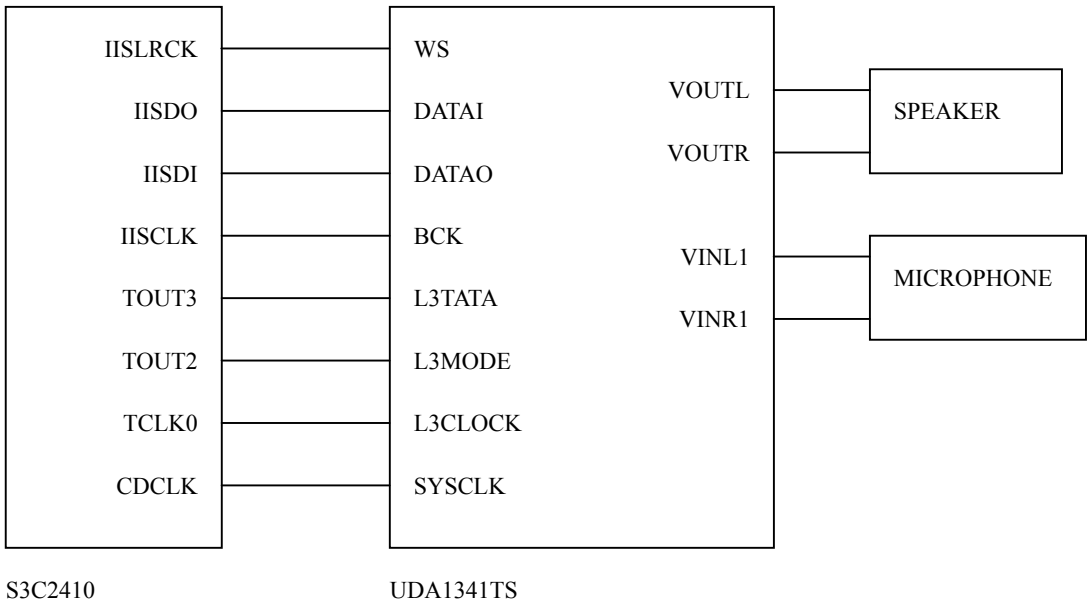


图 11.6　数字音频接口模块

11.6 实验步骤:

1．参照模板工程，新建一个工程 IIS，添加相应的文件，并修改 IIS 的工程设置；

2．创建 IIS．c 并加入到工程 IIS 中；

3．编写 IIS 主函数实现如下功能：

4．编译装载 IIS；

5．运行本实验附带的 DNW 应用程序（类似于超级终端），完成 WAV 文件的录放。具体操作步骤如图 11.7~11.14 所示：

图 11.7 打开 DNW

图 11.8 打开 DNW 配置文件

因个人 PC 机连接设置正确的串口号，并将串口设置为：波特率（115200）；USB 没有连接，暂时不需要设置。

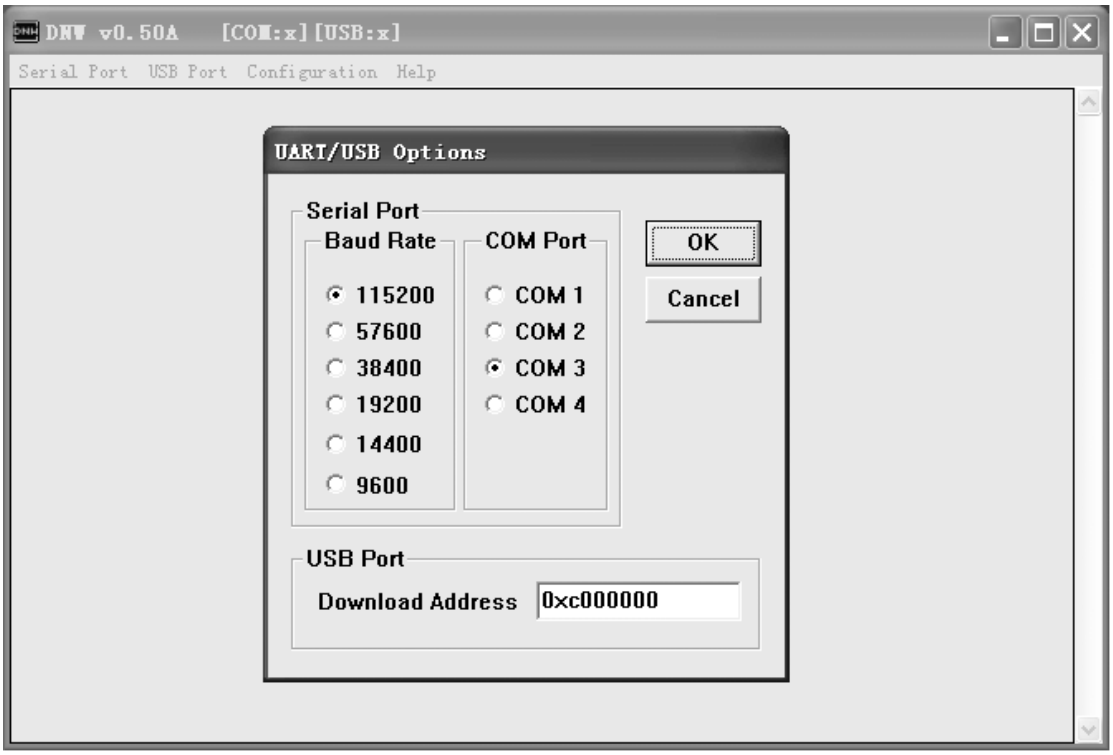


图 11.9 配置 DNW

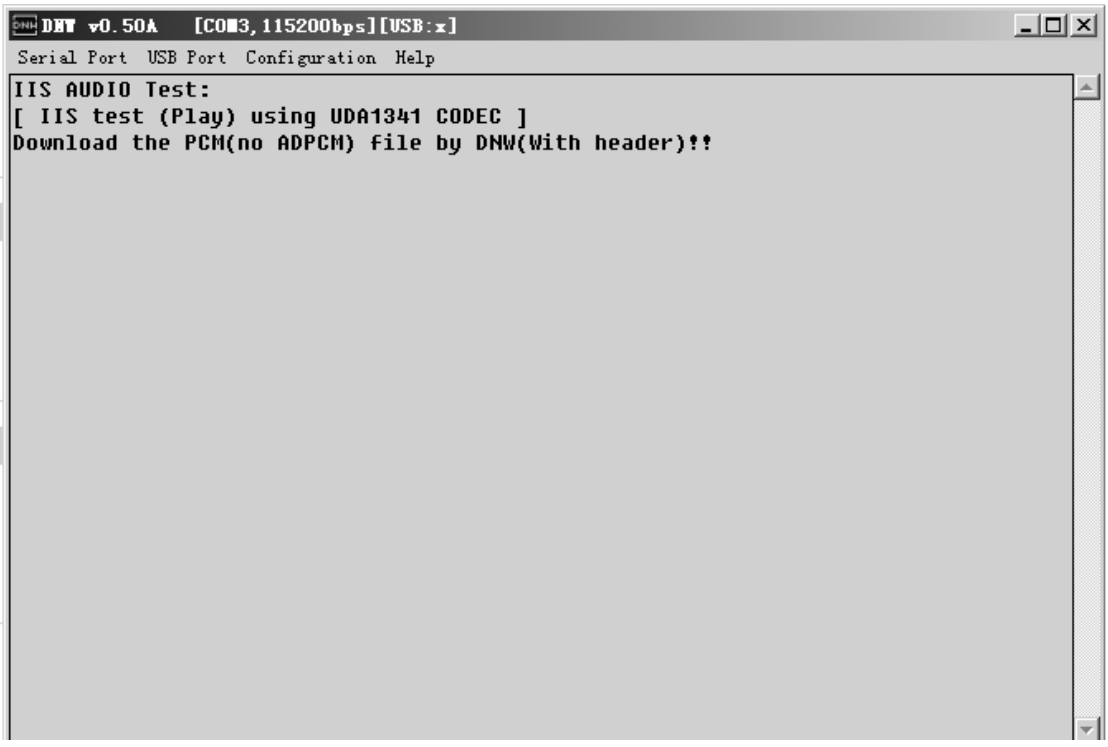


图 11.10


DNW v0.50A [COM3,115200bps][USB:x]
Serial Port
USB Port
Configuration
Help
Connect
Transmit
est:
(Play) using UDA1341 CODEC]
Download the PCM(no ADPCM) file by DNW(With header)!!

图 11.11

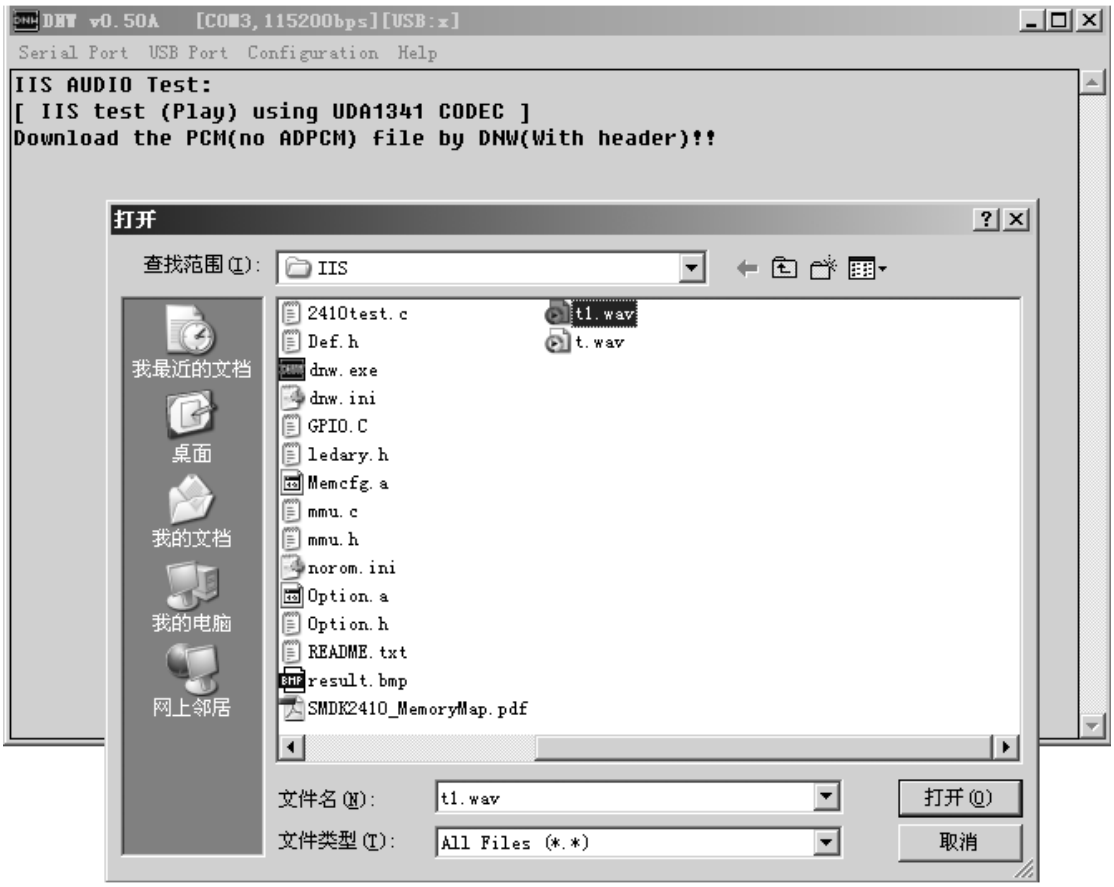

DNW v0.50A [COM3,115200bps][USB:x]
Serial Port
USB Port
Configuration
Help
IIS AUDIO Test:
[IIS test (Play) using UDA1341 CODEC]
Download the PCM(no ADPCM) file by DNW(With header)!!
打开
查找范围(I):
IIS
我最近的文档
桌面
我的文档
我的电脑
网上邻居
2410test.c
Def.h
dnw.exe
dnw.ini
GPIO.C
ledary.h
Memcfg.a
mmu.c
mmu.h
norom.ini
Option.a
Option.h
README.txt
result.bmp
SMDK2410_MemoryMap.pdf
t1.wav
t.wav
文件名(N):
t1.wav
文件类型(T):
All Files (*.*)
打开(O)
取消

图 11.12

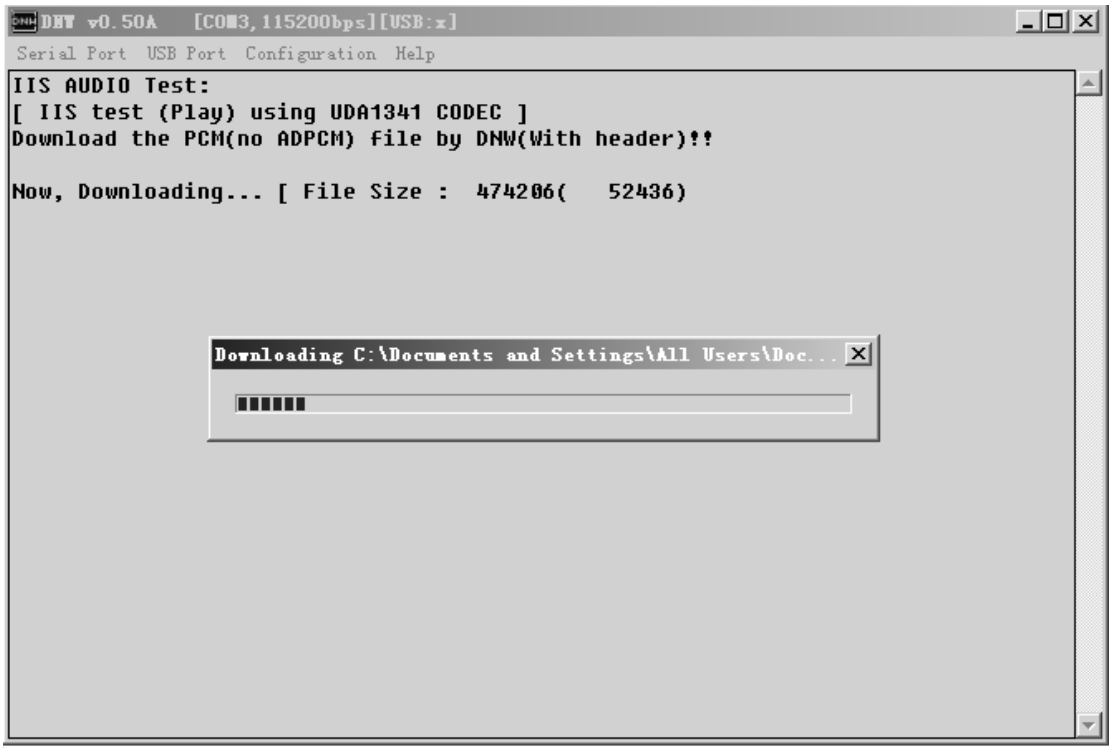


图 11.13

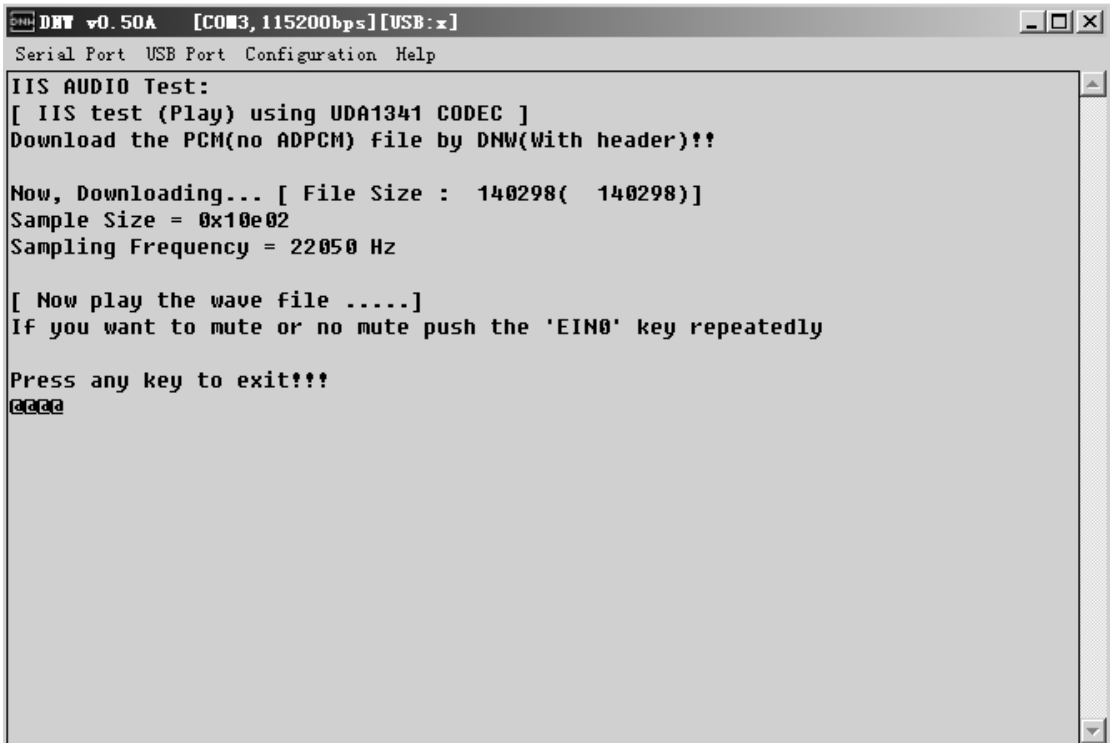


图 11.14

11.7 实验要求

简述音频处理的步骤。

实验十二 键盘中断实验

12.1 实验目的

1．通过实验掌握中断式键盘控制与设计方法；
2．熟练编写 S3C2410 中断服务程序。

12.2 实验内容

编写中断处理程序，处理一个键盘中断，并在串口打印中断及按键显示信息。

12.3 预备知识

1．了解键盘电路的构成以及原理；
2．了解中断的作用以及处理过程。

12.4 实验设备

硬件：S3C2410 嵌入式开发板，JTAG 仿真器。

软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

12.5 基础知识

用户设计行列键盘接口，一般常采用 3 种方法读取键值。一种是中断式，另外两种是扫描法和反转法。

✧ 中断式：在键盘按下时产生一个外部中断通知 CPU，并由中断处理程序通过不同的地址读取数据线上的状态，判断哪个按键被按下。本实验采用中断式实现用户键盘接口。

中断方式的原理示意图如图 12.1 所示。

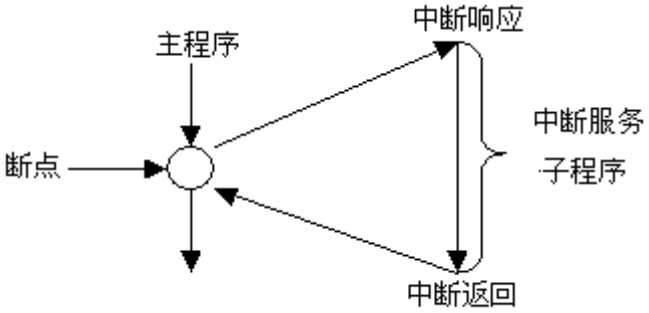


图 12.1 中断处理示意图

（1）中断响应

中断源向 CPU 发出中断请求，若优先级别最高，CPU 在满足一定地条件下，可以中断当前程序的运行，保护好被中断主程序的断点及现场信息。然后，根据中断源提供的信息，找到中断服务子程序的入口地址，转去执行新的程序段，这就是中断响应。

CPU 响应中断是有条件的，如内部允许中断、中断未被屏蔽、当前指令执行完等。

（2）中断服务子程序

CPU 响应中断以后，就会终止当前的程序，转去执行一个中断服务子程序，以完成为相应设备的服务。中断服务子程序的一般结构如下图 12.2 所示。

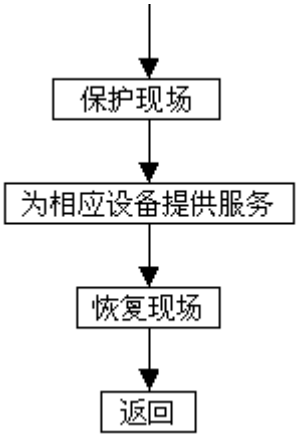


图 12.2　中断服务子程序处理流程

a.保护现场（由一系列的压栈指令完成）。目的是为了保护那些与主程序中有冲突的寄存器，（如 R0，R1，R2 等），如果中断服务子程序中所使用的寄存器与主程序中所使用的寄存器等没有冲突的话，这一步可以省略。

b.中断处理，中断处理程序在检查到相应的中断源后，调用对应的中断处理程序完成。

c.回复现场并返回（由一系列的出栈指令完成）。是与保护现场对应的，但要注意数据恢复的次序，以免混乱。

由于中断服务子程序需要打断主程序的执行，因此其处理应该及时完成，较长时间的延时将导致系统性能严重下降。

✧ 扫描法：对键盘上的某一行送低电平，其它行为高电平，然后读取列值。若列值中有一位是低，则表明该行与低电平对应列的键被按下；否则扫描下一行。

✧ 反转法：先将所有行扫描线输出低电平，读列值。若列值有一位是低，则表明有键按下；然后所有列扫描线输出低电平，再读行值。根据读到的值组合就可以得到相应的键码。

1．S3C2410 教学系统的键盘模块

键盘扩展我们用的是 SPI 接口的键盘显示控制芯片 ZLG7289，电路连接关系如图 12.3 所示：

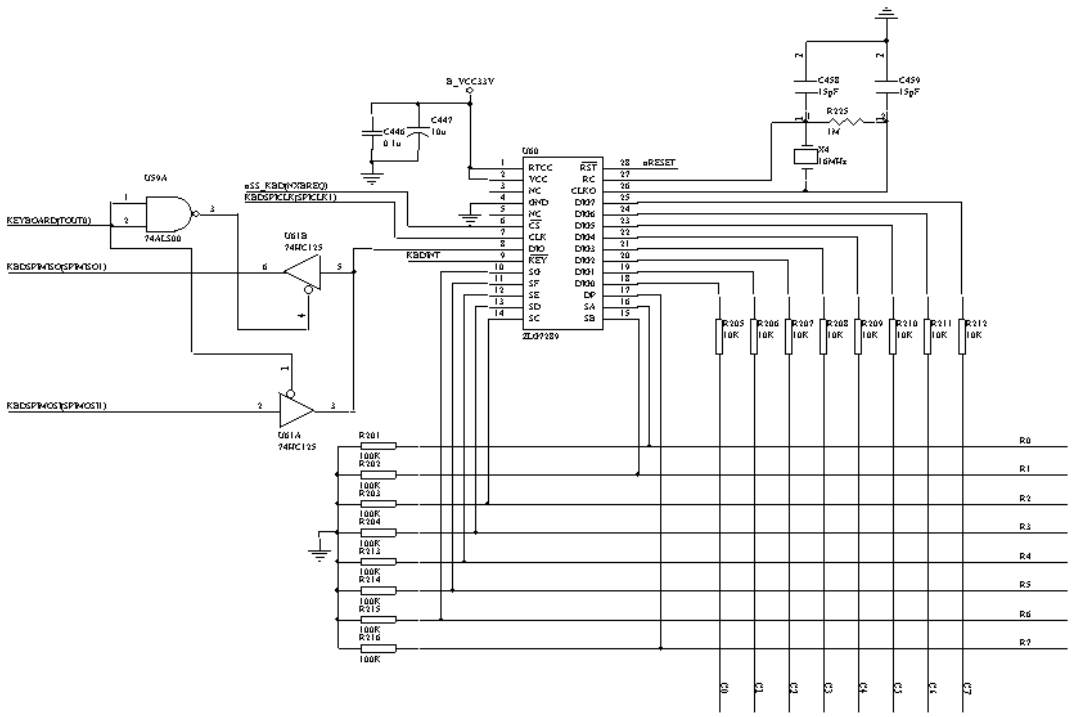


图 12.3　键盘模块原理图

板载键盘的按键和芯片扫描的行线和列线之间的对应关系如表 12.1 所示：

表 12.1 按键和扫描的行列线间的对应关系

按键	键值	行线/列线	按键	键值	行线/列线
NumLock	32	R0/C0	5	42	R1/C1
/	40	R0/C1	6→	50	R1/C2
*	48	R0/C2	1/End	35	R1/C3
-	55	R0/C3	2/↓	43	R1/C4
7/Home	33	R0/C4	3/Pg Dn	51	R1/C5
8/↑	41	R0/C5	0/Ins	44	R1/C6
9/Pa Up	49	R0/C6	./Del	52	R1/C7
+	57	R0/C7	Enter	59	R2/C0
4/←	34	R1/C0			

2．工作原理

ZLG7289 可用行线 R0～R2 和列线 C0～C7 构成矩阵键盘。同时在芯片内部可自动完成扫描、译码、去抖动处理等任务。当 ZLG7289 检测到有效的按键时，按键有效指示“KEY”引脚将从低电平变为高电平，并一直保持到按键代码被读取为止。在“KEY”为高电平期间，如果 ZLG7289 接收到“读键盘数据”命令，(即“CS”管脚变低)，则输出当前按键的键盘代码，ZLG7289 键盘代码的范围为 00H-0FH。如果在接收到“读键盘数据”时没有按键按下，ZLG7289 将输出 0xFFH。在一次读键盘过程完成后，按键有效指示“KEY”将变为低电平。利用按键有效指示“KEY”与单片机的外部中断端相连，可完成具有中断的键盘监控功能，从而提高 CPU 的工作效率，减少按键响应时间。

ZLG7289 工作时需要外接 RC 振荡电路以供系统工作，RC 元件的典型值为 R=3.3kΩ,C=20pF，此时的振荡频率约为 4MHz，由于此振荡频率较高，故在印制电路板布线时，所有元件尤其是振荡电路的元件应尽量靠近芯片，并尽量使电路连线最短。

ZLG7289 的 RESET 复位端在一般应用情况下，可以直接与正电源连接，在需要较高可靠性的情况下，可以连接外部 RC 复位电路，在上电或接收到 RESET 端的复位信号后，ZLG7289 大约需要经过 25ms 的复位时间才会进入到正常工作状态。程序中应尽可能地减少 CPU 对 ZLG7289 的访问次数，以提高程序的效率。

值得注意的是，如果有 2 个键同时被按下，则 ZLG7289 只能给出其中一个按键的代码，因此 ZLG7289 不适合应用于需要 2 个或 2 个以上按键同时被按下的应用场合。如确实需要双键组合使用或组合增加键盘数量，可在单片机的某 I/O 脚接入一键与 ZLG7289 共同组双键键盘监控电路。

3．串行接口及时序

ZLG7289 采用串行方式与单片机或微处理器接口，串行数据从“DIO”引脚输出，并由“CLK”端发出同步时钟脉冲。当 ZLG7289 检测到有键按下时，按键有效指示“KEY”变高，单片机检测到“KEY”信号变高后，便将片选端“CS”拉低，从而使得 ZLG7289 将取得的键盘数据在“CLK”引脚的上升沿从“DIO”脚依次送出。在单片机发出 8 个时钟脉冲后，即可从“DIO”端读取 8 位键值编码，该编码值的 D7 为最高位，D0 为最低位，然后单片机再使片选“CS”变高，并使“KEY”端重新输出低电平，至此，读键值过程结束。ZLG7289 的串行接口时序如图 12.4 所示。图中，T1 表示从“CS”下降沿至第一个 CLK 上升沿的延时，典型值为 15μs；T2 为 CLK 脉冲宽度，典型值为 10μs；T3 为 CLK 脉冲时间间隔，典型值为 10μs。

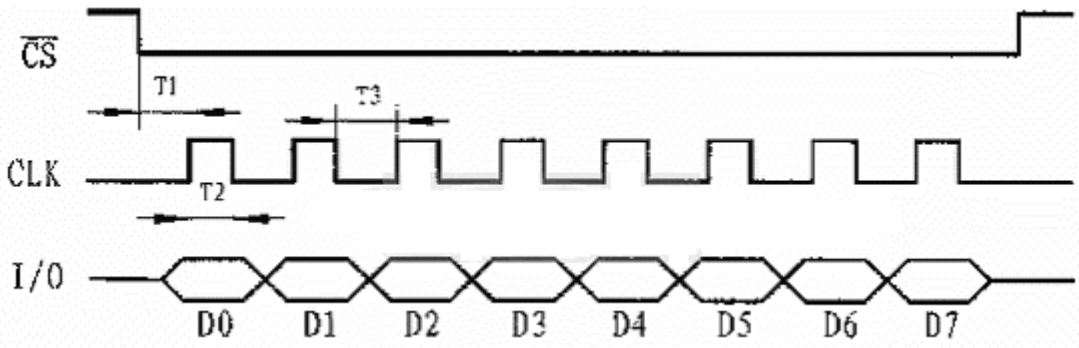


图 12.4　ZLG7289 串行接口时序

4．实验说明

结合实验系统中的用户键盘硬件控制电路，编写相应的程序，包括：硬件中断程序、按键识别程序及键值显示程序，以下是参考程序。

（1）键盘控制初始化程序

```
int KeyPadInit（）
{
    int   t;
    char dummy =  （char）0xff;
    // Setup IO port for SPI interface & Keyboard
    // Setup EINT1  （KBDINT）
    rGPFCON &= ~（0x3 << 2）;          // Clear GPF1
```

```
        rGPFCON |= （0x2 << 2）;            // Set GPF1 to EINT1 for Keyboard interrupt
        rEXTINT0 &= ~（0x7 << 4）;          // Clear EINT1
        rEXTINT0 |= （0x2 << 4）;           // fallig edge triggered for EINT1
        // setup SPI interface
        // GPG5 ： SPIMISO （KBDSPIMISO）
        // GPG6 ： SPIMOSI （KBDSPIMOSI）
        // GPG7 ： SPICLK （KBDSPICLK）
        rGPGCON &= ~（（0x3 << 10）|（0x3 << 12）|（0x3 << 14））;    // Clear GPG5,6,7
        rGPGCON |= （（0x3 << 10） | （0x3 << 12） | （0x3 << 14））;
        // setup _SS signal（nSS_KBD）
        rGPBCON &= ~（0x3 << 12）;              // Clear GPB6
        rGPBCON |= （ONEBIT << 12）;            // Set Port GPB6 to output for nSS signal
        KEYPADCSDIS;                            // Set /SS high
        // setup Dir signal （KEYBOARD） CPU->7289
        rGPBCON &= ~（0x3 << 0）;               // Clear GPB0
        rGPBCON |= （ONEBIT << 0）;             // Set Port GPB0 to output for _PWR_OK
signal
        //      rGPDDAT &=~（ONEBIT << 0）;             // set _PWR_OK to 0
        KEYPADDIRMO;
        // Setup SPI registers
        // Interrupt mode, prescaler enable, master mode, active high clock, format B, normal
mode
        // rSPCON1 = （ONEBIT<<5）|（ONEBIT<<4）|（ONEBIT<<3）|（0x0<<2）|
（ONEBIT<<1）;
        // Poll mode, prescaler enable, master mode, active high clock, format A, normal mode
        rSPCON1 = （0<<5）|（ONEBIT<<4）|（ONEBIT<<3）|（0x0<<2）|（0<<1）;
        // Developer MUST change the value of prescaler properly whenever value of PCLK is
changed.
        rSPPRE1 = 255;                          //  99.121K  =  203M/4/2/ （ 255+1 ）
PCLK=50.75Mhz FCLK=203Mhz SPICLK=99.121Khz
        putcToKBCTL（0xa4）;                    //send init command
        Uart_Printf（"Key Pad Init complete：\n"）;
        rSRCPND = BIT_EINT1; //to clear the previous pending states
        rINTPND = BIT_EINT1;
        //Define the keypad int entry
        pISR_EINT1 = （unsigned）KeyPad_Int;
        rINTMSK=~（BIT_EINT1）;                 //enable Eint
        return（TRUE）;
    }
```

（2）中断服务程序

```
    int   putcToKBCTL（U8 c）
    {
        U32       i;
```

```
//    UINT     rxbuf[10];
//    UINT     x;
      KEYPADCSEN;
      KEYPADDIRMO;
      Delay60us（）;
      while（（rSPSTA1 & ONEBIT）==0）;     // wait while busy
      rSPTDAT1 = c;                         // write left justified data
      while（（rSPSTA1 & ONEBIT）==0）;     // wait while busy
      KEYPADCSDIS;
      i = rSPRDAT1;
      return（i）;
}

U8 readKBValue（void）
{
      unsigned char i;
      KEYPADCSEN;
      KEYPADDIRMO;
      Delay60us（）;
      while（（rSPSTA1 & ONEBIT）==0）;     // wait while busy
      rSPTDAT1 = 0x15;                      // write read key value command
      while（（rSPSTA1 & ONEBIT）==0）;     // wait while busy
      Delay30us（）;//delay 30us
      KEYPADDIRMI;
      rSPTDAT1 = 0xff;                      // write read key value command
      while（（rSPSTA1 & ONEBIT）==0）;     // wait while busy
      i = rSPRDAT1;
      KEYPADCSDIS;
      KEYPADDIRMO;
      return（i）;
}
```

（3）串口打印程序

```
void __irq KeyPad_Int（void）
{
      U8     ui8ScanCode;
      ui8ScanCode=readKBValue（）;
      Uart_Printf（"中断产生  key %d pressed!\n",ui8ScanCode）;
      ClearPending（BIT_EINT1）;
}
```

程序流程图如图 12.5 所示：

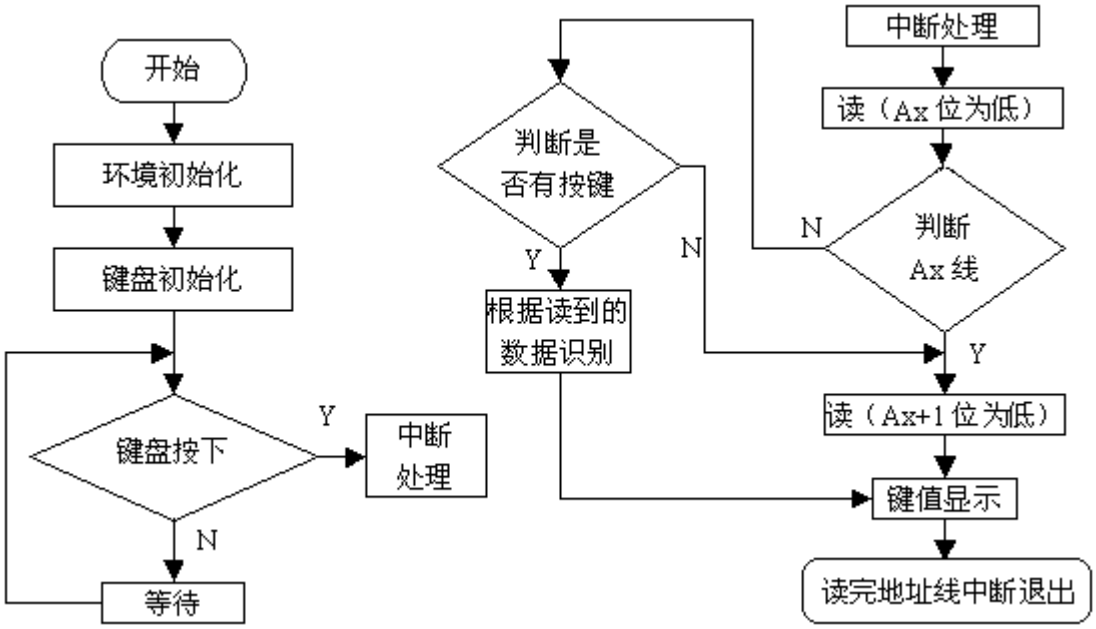


图 12.5　程序流程图

12.6 实验步骤

1．参照模板工程，新建一个工程 keypad，添加相应的文件，并修改 keypad 的工程设置；

2．创建 keypad．c 并加入到工程中；

3．按照上节流程图及参考程序编写键盘中断程序；

4．编译 keypad；

5．打开超级终端。

6．下载程序并运行，在超级终端中观察按键是否输出相应键值，结果如图 12.6 所示：

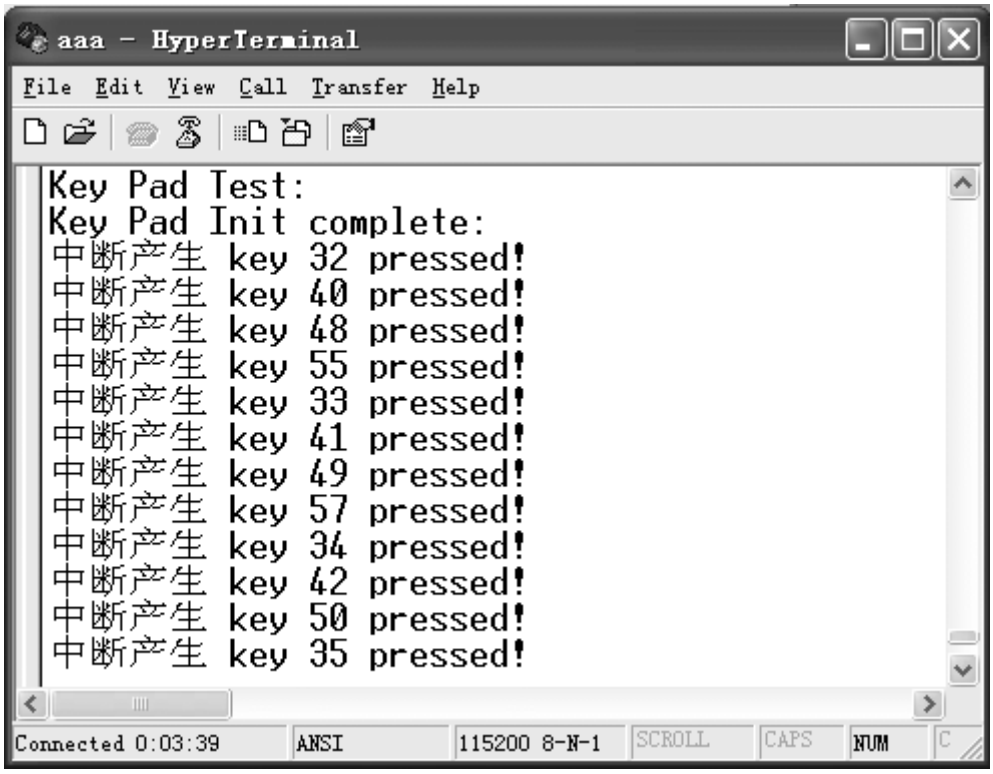


图 12.6　超级终端运行结果

12.7 实验报告要求

1．键盘扫描有哪几种方式，分别说明其基本原理；
2．思考使用扫描法如何处理键盘输入，试画出流程图。

第七章 扩展实验

实验一 GPS 实验

1.1 实验目的

1．掌握 GPS 全球定位系统的基本知识；
2．掌握 GPS 卫星定位信息的接收以及定位参数的提取方法。

1.2 实验内容

1．接收 GPS 卫星定位信息；
2．提取定位参数。

1.3 预备知识

1．了解 ADS 集成开发环境；
2．了解 GPS 协议格式。

1.4 实验设备

硬件：S3C2410 嵌入式开发板，GPS/GPRS 扩展模块，JTAG 仿真器。

软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

1.5 实验原理

1．GPS 简介

GPS 即全球定位系统（Global Positioning System）的简称。GPS 是美国国防部(U.S.Department Of Deefense--DOD)从本世纪 70 年代开始研制，历时 20 年，耗资一百多亿美元，于 1994 年全面建成。具有在海、陆、空进行全方位实时三维导航与定位能力的新一代卫星导航与定位系统。我国测绘等部门在近 10 年的使用表明，GPS 是全天候、高精度、高效率，多功能等显著特点，赢得广大测绘工作者的信赖，并成功地应用于大地测量、工程测量、航空摄影测量、运载工具导航和管制、地壳运动监测、工程变形监测、水纹测试、资源勘察、地球动力学等多种学科，从而给测绘领域带来一声场深刻的技术革命。

GPS 系统的特点：

高精度、全天候、高效率、多功能、操作简便、应用广泛等。

GPS 的应用：

GPS 是目前技术最成熟且真正实用的一种卫星导航和定位系统。它具有定位快、全时

域、全天候、高动态等特点，现在交通、测量、勘探、航空、航海等诸多民用领域获得了广泛应用。

由于 GPS 定位技术具有精度高、速度快、成本低的显著优势，因而以成为目前世界上应用最广泛、应用最强的全球精密授时、测距、导航的定位系统。

2．GPS 原理：

GPS 由三个独立的部分组成：

空间部分——GPS 卫星星座；

地面控制部分——地面监控系统；

用户设备部分——GPS 信号接收机。

GPS 接收机硬件一般由主机、天线和电源组成，我们提供的 GPS/GPRS 扩展模块就是一个简单的 GPS 接收机，其硬件结构如图 1.1 所示。

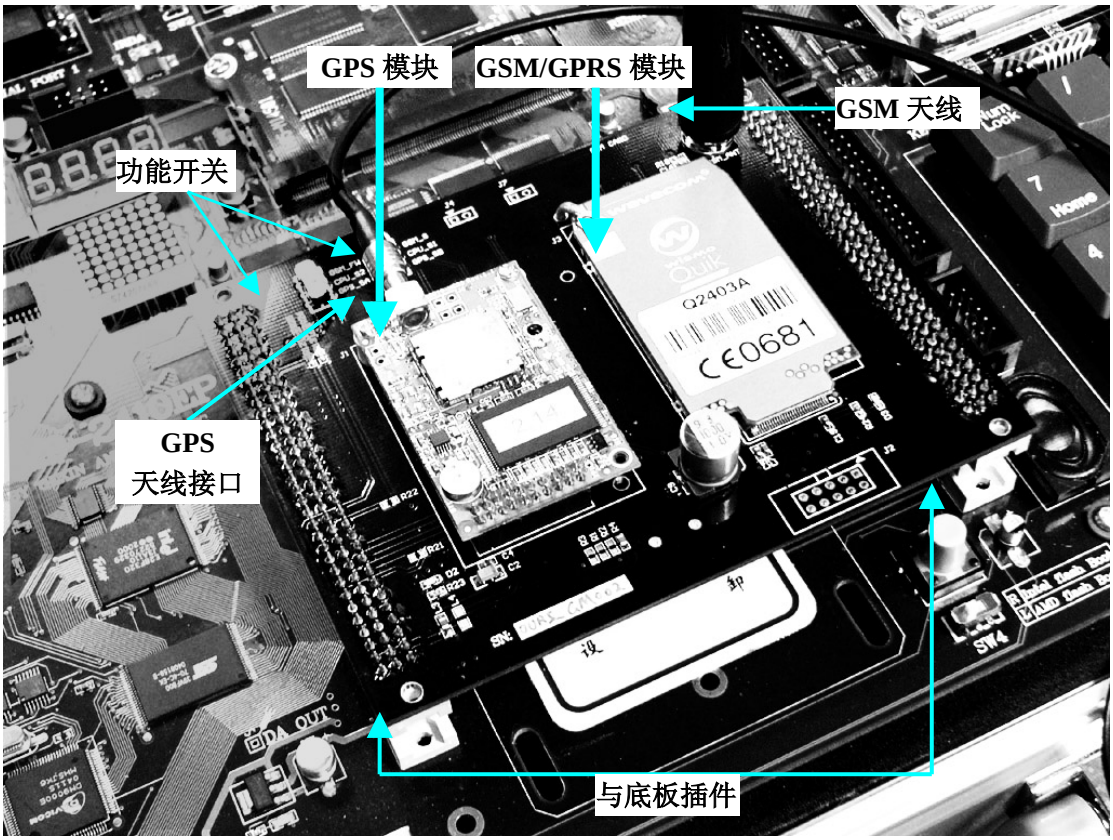


图 1.1（b）GPS/GPRS 扩展模块结构图

3．GPS 模块使用说明：

（1）将模块插到 S3C2410 实验板的多功能扩展口上，方向如上图所示。

（2）将功能开关 S1、S2 扳到下面以选择 GPS 模块（注：功能开关扳上选择 GSM/GPRS 模块，扳下选择 GPS 模块）。

（3）接 GPS 天线。GPS 天线如图 1.2 所示。将小头插入 GPS 天线接口，GPS Active Antenna 端放到一个可以接收到信号的地方，如果在室内实验，建议将 GPS Active Antenna 端伸出窗外，其平滑的一面有磁性，可附着在其他铁制的长物体上。这是能否正确接收信息的最关键一步。

（4）连接好实验箱上的串口、仿真器（如果程序烧入实验板则不需要连接仿真器），用 ADS 1.2 运行程序，通过超级终端接收到的数据查看 GPS 定位信息。

4．NMEA 0183 格式

GPS 的通讯接口协议采用美国的 NMEA（National Marine Electronics Association）0183 ASCII 码协议，NMEA 0183 是一种航海、海运方面有关于数字信号传递的标准，此标准定义了电子信号所需要的传输协议，传输数据时间。下列描述了其数据帧的格式定义，包括波特率选择，秒脉冲输出，RTCM 定义输出。我们这里只介绍我们要用的位置信息〈GGA〉语句。

$GPGGA,031736.594,3957.1451,N,11618.8424,E,1,05,1.7,111.4,M,-8.8,M,,*48

GGA 讯息格式如表 1.1 所示。

表 1.1 GGA 讯息格式说明

名　称	数　值	单　位	说　　明
讯息代码	$GPGGA		GGA 讯息前引
标准定位时间 UTC Time	031736.594		时时分分秒.秒秒秒 (Hhmmss.sss)
纬度	3957.1451		度度秒秒.秒秒秒秒(ddmm.mmmm)
南 / 北纬	N		N：北纬　S：南纬
经度	11618.8424		度度度秒秒.秒秒秒秒 (dddmm.mmmm)
东 / 西经	E		E：东经　W：西经
定位代码	1		详 5-3 表
使用中的卫星数	05		范围：0～12
水平稀释精度	1.7		水平稀释精度，0.5 至 99.9 米。
海拔高度	111.4	公尺	
单为	M	公尺	
地表平均高度		公尺	
单位	M	公尺	
偏差修正使用期间		秒	0 表未使用偏差修正
偏差修正，基地台代码			
总合检查码	*48		
<CR> <LF>			结束

我们主要查看时间、经度、纬度数据。

GPS 显示的时间是格林威治时间，北京时间=格林威治时间+8 小时。读上例中的数据 031736.594，其中 03 代表时，17 代表分，36.594 代表秒，那么北京时间为 11 点 37 分 36 秒。

经度、纬度读出为北纬 3957.1451，东经 11618.8424，通过电子地图我们可以查到具体位置。

1.6 实验步骤

1．参照模板新建一个工程 gps，添加相应文件，并修改 gps 的工程设置；
2．创建 gps.c 并加入到工程 gps 中；
3．编写 gps 初始化函数，并将结果打印到串口中；
4．编译 gps；
5．安装 gps 模块，并将计算机的串口连接到 S3C2410 实验箱的串口上；

6．运行超级终端，选择正确的串口号，并将串口设置为：波特率（115200）、奇偶校验（无）、数据位数（8）和停止位数（1），无流控，并打开串口；

7．下载程序并运行，在超级终端中选择 gps，如图 1.2 所示。

```
 GPRS/GPS TEST

 Press any key to continue...
======== Menu ========
[1] Run GPRS test
[2] Run GPS test
Select:
```

图 1.2 超级终端菜单

正确接收到信息后，在超级终端中显示如图 1.3 所示的信息。按照上文的说明可以查看到 gps 的定位结果。

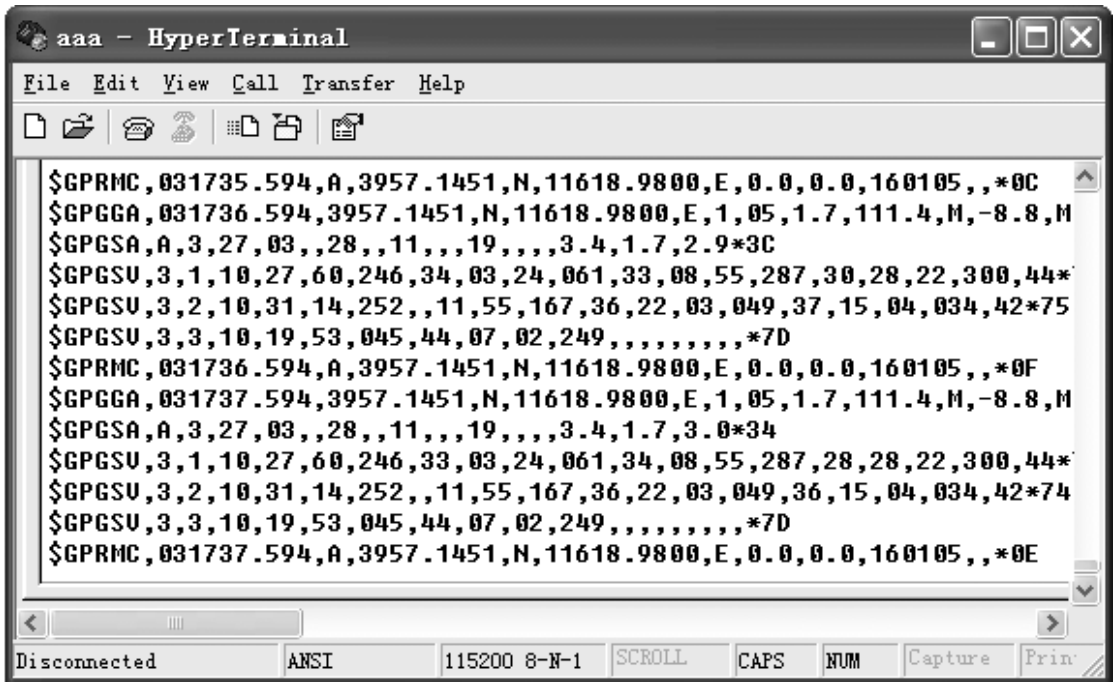


图 1.3 超级终端运行结果

1.7 实验报告要求

1．简述 GPS 信息提取的基本过程。

实验二 GPRS 实验

2.1 实验目的

1．了解 GPRS 无线通讯模块的基本知识；
2．掌握无线通讯模块的使用。

2.2 实验内容

1．掌握 AT 命令集
2．通过计算机串口控制 GPRS 模块

2.3 预备知识

1．了解 GPRS 网络体系结构；
2．了解 S3C2410 的 GPRS 模块的使用。

2.4 实验设备

硬件：S3C2410 嵌入式开发板，JTAG 仿真器。

软件：PC 机操作系统 Win98、Win2000 或 WinXP，ADS1.2 集成开发环境，仿真器驱动程序，超级终端通讯程序。

2.5 实验原理

1．GPRS 简介

业界通常将移动通信分为三代。第一代是模拟的无线网络，第二代是数字通信包括 GSM、CDMA 等，第三代是分组型的移动业务，称为 3G。GPRS 是通用无线分组业务的缩写（General Packet Radio System）,是介于第二代和第三代之间的一种技术，通常称为 2.5G，目前通过升级 GSM 网络实现。称之为 2.5G 是比较恰当的，因为它是一个混合体，采用 TDMA 方式传输语音，采用分组的方式传输数据。

GPRS 是欧洲电信协会 GSM 系统中有关分组数据所规定的标准。它可以提供高达 115Kbps 的空中接口传输速率。GPRS 使若干移动用户能够同时共享一个无线信道，一个移动用户也可以使用多个无线信道。实际不发送或接收数据包的用户仅占很小一部分网络资源。有了 GPRS，用户的呼叫建立时间大为缩短，几乎可以做到“永远在线”（always online）。此外，GPRS 是营运商能够以传输的数据量而不是连接时间为基准来计费，从而另每个用户的服务成本更低。

GPRS 采用信道捆绑和增强数据速率改进实现高速接入，目前 GPRS 的设计可以在一个载频或 8 个信道中实现捆绑，将每个信道的传输速率提高到 14.4Kbps,因此 GPRS 方式最大速率是 8×14.4=115.2Kbps。GPRS 发展的第二步是通过增强数据速率改进（EDGE）将每

个信道的速率提高到 48Kbps，因此第二代的 GPRS 设计速率为 384Kbps。

GPRS 即“通用分组无线业务”（General Packet Radio Service 的英文简称）GPRS 是在现有 GSM 网络上开通的一种新型的分组数据传输技术。相对于原来 GSM 以拨号接入的电路交换数据传送方式。GPRS 是一项高速数据处理的科技，方法是以“分组”的形式传送资料到用户手上。虽然 GPRS 是作为现有 GSM 网络向第三代移动通信演变的过渡技术，但是它在许多方面都具有显著的优势。（如：永远在线、自如切换、高速传输等）。

GPRS 的主要特点：

（1）GPRS 采用分组交换技术，高效传输高速或低速数据和信令，优化了对网络资源和无线资源的利用。

（2）定义了新的 GPRS 无线信道，且分配方式十分灵活：每个 TDMA 帧可分配 1 到 8 个无线接口时隙。时隙能为活动用户所共享，且向上链路和向下链路的分配是独立的。

（3）支持中、高速率数据传输，可提供 9.05 ---171.2kbit/s 的数据传输速率（每用户）。GPRS 采用了与 GSM 不同的信道编码方案，定义了 CS-1、CS-2、CS-3 和 CS-4 四种编码方案。

（4）GPRS 网络接入速度快，提供了与现有数据网的无缝连接。

（5）GPRS 支持基于标准数据通信协议的应用，可以和 IP 网、X.25 网互联互通。支持特定的点到点和点到多点服务，以实现一些特殊应用如远程信息处理。GPRS 也允许短消息业务（SMS）经 GPRS 无线信道传输。

（6）GPRS 的设计使得它既能支持间歇的爆发式数据传输，又能支持偶尔的大量数据的传输。它支持四种不同的 QoS 级别。GPRS 能在 0.5 ---1 秒之内恢复数据的重新传输。GPRS 的计费一般以数据传输量为依据。

（7）在 GSM PLMN 中，GPRS 引入两个新的网络节点：一个是 GPRS 服务支持节点（SGSN），它和 MSC 在同一等级水平，并跟踪单个 MS 的存储单元，实现安全功能和接入控制。节点 SGSN 通过帧中继连接到基站系统。另一个是 GPRS 网关支持节点 GGSN，GGSN 支持与外部分组交换网的互通，并经由基于 IP 的 GPRS 骨干网和 SGSN 连通。

（8）GPRS 的安全功能同现有的 GSM 安全功能一样。身份认证和加密功能由 SGSN 来执行。其中的密码设置程序的算法、密钥和标准与目前 GSM 中的一样，不过 GPRS 使用的密码算法是专为分组数据传输所优化过的。GPRS 移动设备（ME）可通过 SIM 访问 GPRS 业务，不管这个 SIM 是否具备 GPRS 功能。

（9）蜂窝选择可由一个 MS 自动进行，或者基站系统指示 MS 选择某一特定的蜂窝。MS 在重选择另一个蜂窝或蜂窝组（即一个路由区）时会通知网络。

（10）为了访问 GPRS 业务，MS 会首先执行 GPRS 接入过程，以将它的存在告知网络。在 MS 和 SGSN 之间建立一个逻辑链路，使得 MS 可进行如下操作：接收基于 GPRS 的的 SMS 服务、经由 SGSN 的寻呼、GPRS 数据到来通知。

（11）为了收发 GPRS 数据，MS 会激活它所想用的分组数据地址。这个操作使 MS 可被相应的 GGGSN 所识别，从而能开始与外部数据网络的互通。

（12）用户数据在 MS 和外部数据网络之间透明地传输，它使用的方法是封装和隧道技术：数据包用特定的 GPRS 协议信息打包并在 MS 和 GGSN 之间传输。这种透明的传输方法缩减了 GPRS PLMN 对外部数据协议解释的需求，而且易于在将来引入新的互通协议。用户数据能够压缩，并有重传协议保护，因此数据传输高效且可靠。

（13）GPRS 可以实现基于数据流量、业务类型及服务质量等级（QoS）的计费功能，计费方式更加合理，用户使用更加方便。

（14）GPRS 的核心网络层采用 IP 技术，底层款可使用多种传输技术，很方便地实现与高速发展的 IP 网无缝连接。

GPRS 的应用优势：

- ✧ 永远在线——方便：只要激活 GPRS 应用后，将一直保持在线，类似于无线专线网络服务。
- ✧ 按量计费——实惠：GPRS 服务虽然保持一直在结，但您不必担心费用问题。因为只有产生通信流量时才计费。
- ✧ 自如切换——灵活：语音和数据业务可以切换，交替使用。
- ✧ 高速传输——快捷：目前 GPRS 可支持 53.6Kbps 的峰值传输速率，理论峰值传输可达 100 余 Kbps 。

GPRS 与 GSM 的对比：

我国卫星定位技术应用于车辆联网服务可划分为两个阶段：（1）专用网络服务阶段.其技术特点是采用专用频段的无线专网,如自组无线数据网 CDPD 网、集群网、短波网等作为信息传输平台，实时性高是该阶段服务的特点，但建设投资大、无线盲点多、用户容量小等不利因素限制了其公共车辆服务的应用；（2）公众网络服务阶段：随着公共移动通信网络的发展，特别是 GSM 网络服务质量高及短数据业务的发展，为车辆信息服务系统的发展提供了适宜的通信平台，GSM 系统虽有实时性差（5 秒），但建设投资少、无线盲点极少、用户容量大等特点使其主导了车辆服务方面的应用。

各种类型的卫星车辆信息服务系统已应用多年了，由于受短信息服务费、服务中心接入限制、数据传输的延时及服务质量较差等因素困扰，阻碍了业务的开展及用户要求的满足，各厂家所提供的服务大同小异，存在的问题也极为相似。GSM/GPRS 和 CDMA/1X 无线数据通信技术给车辆信息服务系统带来了希望，解决了上述两个技术发展阶段存在的问题，为系统服务质量的提高奠定了基础。

因此，GSM/GPRS 和 CDMA/1X 无线数据通信技术业务、改善了系统性能指标、提高终端稳定性，对全国性的服务运营商、公安、公交、出租、物流、私家车辆等管理调度、定位的应用，基于 GPRS 或 CDMA/ 1X 车辆信息服务系统应用及发展是必然的。

2．GPRS 模块的使用

S3C2410 使用集成 GPRS 无线通信模块，它提供一个支持 RS232 的接口，可直接由计算机串口通过 S3C2410 的 UART 1 接口驱动该模块。此时，计算机作为 DTE（数字终端设备），GPRS 模块作为 DCE（数字电路设备）。在 DTE 和 DCE 之间，用一套 AT 命令实现各种功能，GSM/GPRS 的各种功能都有依赖于 DTE 向 DCE 发送的命令实现，所以 AT 命令可以视为 DTE 和 DCE 之间的软件接口。

3．AT 命令的语法

AT 命令语法集如下表 1.1 所示

表 2.1 AT 命令语法集

定义	命令	内容
厂家认证	AT+CGMI	获得厂家的标识
模式认证	AT+CGMM	查询支持频段
修订认证	AT+CGMR	查询软件版本
生产序号	AT+CGSN	查询 IMEI NO.
TE 设置	AT+CSCS	选择支持网络
查询 IMSI	AT+CIMI	查询国际移动电话支持认证
卡的认证	AT+CCID	查询 SIM 卡的序列号
功能列表	AT+GCAP	查询可供使用的功能列表
重复操作	A/	重复最后一次操作

关闭电源	AT+CPOF	暂停模块软件运行
设置状态	AT+CFUN	设置模块软件的状态
活动状态	AT+CPAS	查询模块当前活动状态
报告错误	AT+CMEE	报告模块设备错误
键盘控制	AT+CKPD	用字符模拟键盘操作
拨号命令	ATD	拨打电话号码
挂机命令	ATH	挂机
回应呼叫	ATA	当模块被呼叫时回应呼叫
详细错误	AT+CEER	查询错误的详细原因
DTMF 信号	AT+VTD，+VTS	+VTD 设置长度，+VTS 发送信号
重复呼叫	ATDL	重复拨叫最后一次号码
自动拨号	AT%Dn	设备自动拨叫号码
自动接应	ATS0	模块自动接听呼叫
呼入载体	AT+CICB	查询呼入的模式，DATA or FAX or SPEECH
增益控制	AT+VGR，+VGT	+VGR 调整听筒增益，+VGT 调整话筒增益
静音控制	AT+CMUT	设置话筒静音
声道选择	AT+SPEAKER	选择不同声道（2 对听筒和话筒）
回声取消	AT+ECHO	根据场所选择不同回声程度
单音修改	AT+SIDET	选择不同回声程度
初始声音参数	AT+VIP	恢复到厂家对声音参数的默认设置
信号质量	AT+CSQ	查询信号质量
网络选择	AT+COPS	设置选择网络方式（自动/手动）
网络注册	AT+CREG	当前网络注册情况
网络名称	AT+WOPN	查询当前使用网络提供者
网络列表	AT+CPOL	查询可供使用的网络
输入 PIN	AT+CPIN	输入 PIN 码
输入 PIN2	AT+CPIN2	输入第二个 PIN 码
保存尝试	AT+CPINC	显示可能的各个 PIN 码
简单上锁	AT+CLCK	用户可以锁住状态
改变密码	AT+CPWD	改变各个 PIN 码
选择电话簿	AT+CPBS	选择不同的记忆体上存储的电话簿
读取电话簿	AT+CPBR	读取电话簿目录
查找电话簿	AT+CPBF	查找所需电话目录
写入电话簿	AT+CPBW	增加电话簿条目
电话号码查找	AT+CPBP	查找所需电话号码
动态查找	AT+CPBN	查找电话号码的一种方式
用户号码	AT+CNUM	选择不同的本机号码（因网络服务支持不同）
避免电话簿初始化	AT+WAIP	选择是否防止电话簿初始化
选择短消息服务	AT+CSMS	选择是否打开短消息服务以及广播服务
短消息存储	AT+CPMS	选择短消息优先存储区域
短消息格式	AT+CMGF	选择短消息支持格式（TEXT or PDU）
保存设置	AT+CSAS	保存+CSCA and +CSMP 参数设置
恢复设置	AT+CRES	恢复+CSCA and +CSMP 参数设置

显示 TEXT 参数	AT+CSDH	显示当前 TEXT 模式下结果代码
新消息提示	AT+CNMI	选择当有新的短消息来时系统提示方式
读短消息	AT+CMGR	读取短消息
列短消息	AT+CMGL	将存储的短消息列表
发送短消息	AT+CMGS	发送短消息
写短消息	AT+CMGW	写短消息并保存在存储器中
从内存中发短消息	AT+CMSS	发送在存储器中保存的短消息
设置 TEXT 参数	AT+CSMP	设置在 TEXT 模式下条件参数
删除短消息	AT+CMGD	删除保存的短消息
服务中心地址	AT+CSCA	提供短消息服务中心的号码
选择广播类型	AT+CSCB	选择系统广播短消息的类型
广播标识符	AT+WCBM	读取 SIM 卡中系统广播标识符
短消息位置修改	AT+WMSC	修改短消息位置
短消息覆盖	AT+WMGO	写一条短消息放在第一个空位
呼叫转移	AT+CCFC	设置呼叫转移
呼入载体	AT+CLCK	锁定呼入载体以及限制呼入或呼出
修改 SS 密码	AT+CPWD	修改提供服务密码
呼叫等待	AT+CCWA	控制呼叫等待服务
呼叫线路限定	AT+CLIR	控制呼叫线路认证
呼叫线路显示	AT+CLIP	显示当前呼叫线路认证
已连接线路认证	AT+COLP	显示当前已连接线路认证
计费显示	AT+CAOC	报告当前费用
累计呼叫	AT+CACM	累计呼叫费用
累计最大值	AT+CAMM	设置累计最大值
单位计费	AT+CPUC	设置单位费用以及通话计时
多方通话	AT+CHLD	保持或挂断某一通话线路（支持多方通话）
当前呼叫	AT+CLCC	列出当前呼叫
补充服务	AT+CSSN	设置呼叫增值服务
非正式补充服务	AT+CUSD	非正式的增值服务
保密用户	AT+CCUG	选择是否在保密状态
载体选择	AT+CBST	选择数据传输的类型
选择模式	AT+FCLASS	选择发送数据 or 传真
服务报告控制	AT+CR	是否报告提供服务
结果代码	AT+CRC	报告不同的结果代码
设备速率报告	AT+ILRR	是否报告当前传输速率
协议参数	AT+CRLP	设置无线连接协议参数
其他参数	AT+DOPT	设置其他的无线连接协议参数
传输速度	AT+FTM	设置传真发送的速度
接收速度	AT+FRM	设置传真接收的速度
HDLC 传输速度	AT+FTH	设置传真发送的速度（使用 HDLC 协议）
HDLC 接收速度	AT+FRH	设置传真接收的速度（使用 HDLC 协议）
停止传输并等待	AT+FTS	停止传真的发送并等待
静音接收	AT+FRS	保持一段静音等待

固定终端速率	AT+IPR	设置数据终端设备速率
其他位符	AT+ICF	设置停止位、奇偶校验位
流量控制	AT+IFC	设置本地数据流量
设置 DCD 信号	AT&C	控制数据载体探测信号
设置 DTR 信号	AT&D	控制数据终端设备准备信号
设置 DSR 信号	AT&S	控制数据设备准备信号
返回在线模式	ATO	返回到数据在线模式
结果代码抑制	ATQ	是否模块回复结果代码
DCE 回应格式	ATV	决定数据通信设备回应格式
默认设置	ATZ	恢复到默认设置
保存设置	AT&W	保存所有对模块的软件修改
自动测试	AT&T	自动测试软件
回应	ATE	是否可见输入字符
回复厂家设置	AT&F	软件恢复到厂家设置
显示设置	AT&V	显示当前的一些参数的设置
认证信息	ATI	显示多种模块认证信息
区域环境描述	AT+CCED	用户获取区域参数
自动接收电平显示	AT+CCED	扩展到显示接收信号强度
一般显示	AT+WIND	
数据计算模式	AT+CRYPT	
键盘管理	AT+EXPKEY	
PLMN 上的信息	AT+CPLMN	
模拟数字转换测量	AT+ADC	
模块事件报告	AT+CMER	
选择语言	AT+WLPR	选择可支持的语言
增加语言	AT+WLPW	增加可支持的语言
读 GPIO 值	AT+WIOR	
写 GPIO 值	AT+WIOW	
放弃命令	AT+WAC	用于放弃 SMS、SS and PLMN
设置单音	AT+WTONE	设置音频信号（WMOi3）
设置 DTMF 音	AT+WDTMF	设置 DTMF 音（WMOi3）

接下来我们以几个实例来熟悉一下 AT 命令的使用。

（1）查询 SIM 卡 AT 命令

格式为 AT+CCID

命令发送成功后，我们可以看到超级终端上显示了 SIM 卡的序列号。

（2）电话主叫 AT 命令呼叫电话 stringnum：

格式为 ATD<stringnum>；

比如：要拨打电话 61234567 则操作为 ATD61234567；

注意打电话命令要以分号结尾。如果成功地和被叫方取得联络，可以听到回铃音，否则模块返回 BUSY（网络忙）或 NO CARRIER（脱网或网络拒绝服务）；如果对方摘机，模块将返回 OK，表明建立了正确的话音通路；通话结束后，如果对方挂机，模块返回 NO CARRIER，如果主叫方想挂机，发送 ATH 命令，模块返回 OK 后，语音通路被解除。

（3）电话被叫 AT 命令

拨打模块的号码超级终端的屏幕上会显示“RING”，为了使被叫具有来电显示功能我们要发送如下命令：

AT+CR=1

AT+CLIP=1

两条命令分开发送，每条命令正确发送后都应该返回 OK 提示。这时再拨模块号码，超级终端上会显示被叫号码。

（4）短消息有关的 AT 命令

在收发短信方面，按时间产生先后，共产生了三种模式：Block Mode、基于 AT 指令的 Text Mode、基于 AT 指令的 PDU Modem, Text Mode 比较简单，多款诺基亚手机均支持该模式。西门子的手机大多只支持 PDU 模式，PDU 模式是发送或接收手机 SMS 信息的一种方法，短信息正文经过十六进制编码后被传送。

与短信相关指令如表 1.2 所示：

表 2.2 短信相关指令

AT 指令	功 能
AT+CMGC	Send an SMS command（发出一条短消息命令）
AT+CMGD	Delete SMS message（删除 SIM 卡内存的短消息）
AT+CMGF	Select SMS message formate（选择短消息信息格式：0-PDU;1-文本）
AT+CMGL	List SMS message from preferred store(列出 SIM 卡中的短消息 PDU/text：0/"REC UNREAD"-未读，1/"REC READ"-已读，2/"STO UNSENT"-待发，3/"STO SENT"-已发，4/"ALL"-全部的）
AT+CMGR	Read SMS message（读短消息）
AT+CMGS	Send SMS message（发送短消息）
AT+CMGW	Write SMS message to memory（向 SIM 内存中写入待发的短消息）
AT+CMSS	Send SMS message from storage（从 SIN|M 内存中发送短消息）
AT+CNMI	New SMS message indications（显示新收到的短消息）
AT+CPMS	Preferred SMS message storage（选择短消息内存）
AT+CSCA	SMS service center address（短消息中心地址）
AT+CSCB	Select cell broadcast messages（选择蜂窝广播消息）
AT+CSMP	Set SMS text mode parameters（设置短消息文本模式参数）
AT+CSMS	Select Message Service（选择短消息服务）

下面我们具体说明如何发送接收查看短消息：

a．发送短消息

我们使用最简单的文本方式（1 是选择文本方式），步骤为：

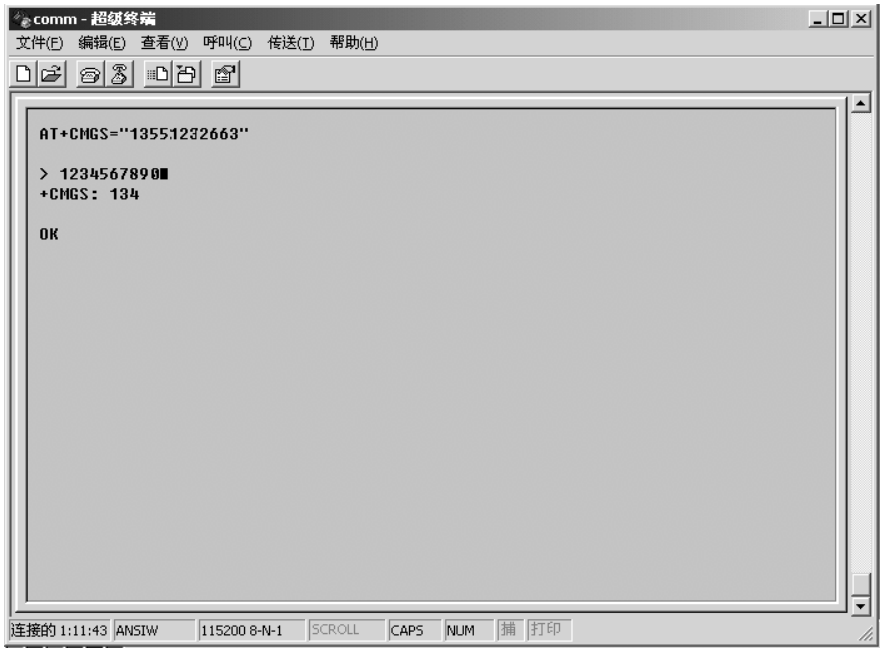
AT+CMGF=1　　执行命令后，模块返回 OK；

AT+CSCA=“+8613800100500（北京局）”　　执行命令后，模块返回 OK；

AT+CMGS=“1355XXXX663”　　等模块返回“>”符号；

>0123456789（以^Z 结束）　　发送成功后，返回 OK；失败返回 ERROR。

我们可以看到超级终端中的显示结果如图 2.1



2.1 发送短消息结果

b．接收查看短消息：

同样我们可以选择 PDU 格式或文本格式，这里使用 PDU 格式（0 是选择 PDU 方式）。

AT+CMGF=0　　执行命令后，模块返回 OK；

AT+CMGL=“ALL”　　执行命令后，模块返回 OK。

这时我们可以看到短消息列表，如图 2.2 所示。

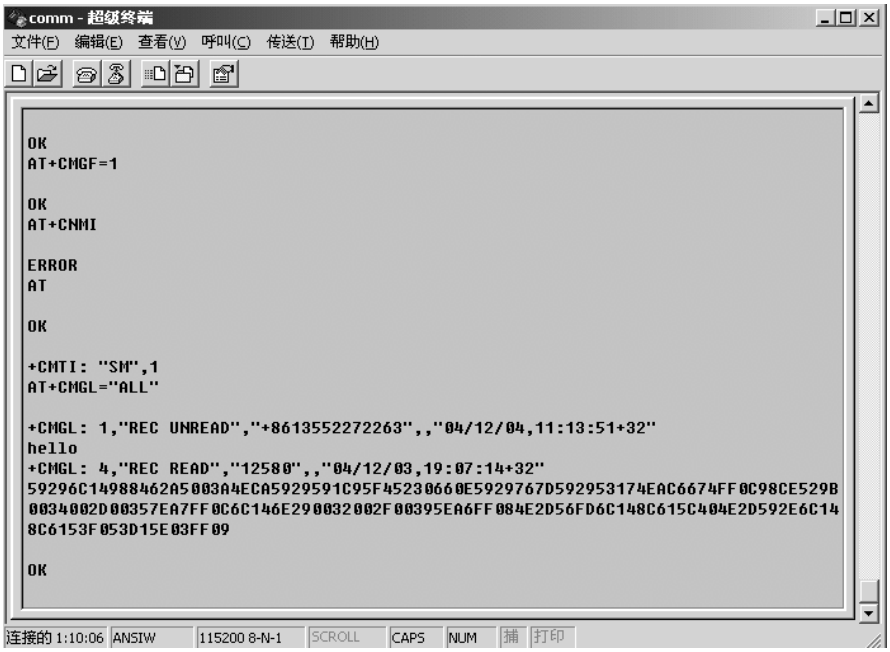


图 2.2 接收查看短消息结果

其中：

“REC　UNREAD”　　未读短信

“REC　READ”　　已读短信

“ST0　UNSENT”　　存储但未发送短信

“ST0　SENT”　　存储且已发送短信

“ALL”　　全部短信

按照表 1.2 所列命令我们可以对短信进行操作。

2.6 实验步骤

1．新建一个工程 GPRS．mcp，添加相应的文件，并修改 GPRS 的工程设置；

2．创建 GPRS．C 并加入到工程 GPRS．Mcp 中；

3．编写一段分支程序，使每个入口指向一个 AT 命令函数，函数完成在串口中打印 AT 命令行的任务；

4.将计算机的串口连接到 S3C2410 的串口上,在 GPRS 模块的 SIM 卡卡座上插入 SIM 卡，如图 2.3 所示，将模块上功能开关 S1、S2 扳到上面以选择 GPRS 模块；

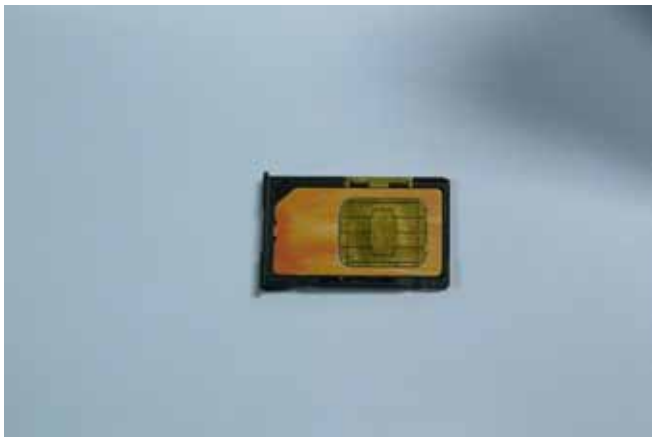


图 2.3（a） 安装 SIM 卡步骤一

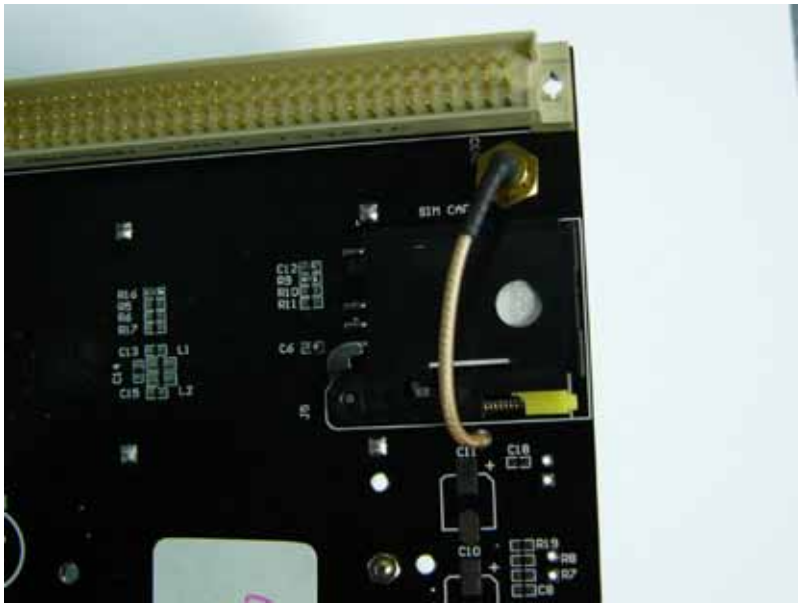


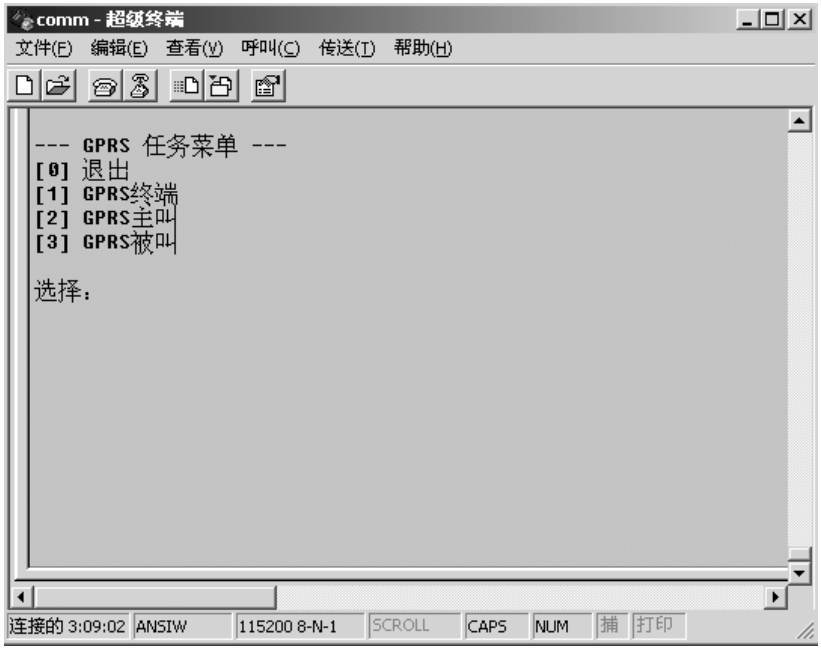
图 2.3（b） 安装 SIM 卡步骤二

5．接通电源，编译 GPRS。

6．运行超级终端，选择正确的串口，并将串口设置为：波特率（115200）、奇偶校验（None）、数据位数（8）和停止位数（1），无流控，并打开串口；

7．下载程序并运行。

超级终端打开界面如图 2.4 所示



2.4 超级终端打开界面

选择（1）手动输入 AT 命令。

选择（2）主叫。拨打一个号码，这时可以听到对方电话振铃,按 ESC 挂机；

选择（3）被叫。当外部一个号码打入时超级终端上会有来电显示，如图 2.5 所示，按回车接听，按 ESC 挂机。

图 2.5 超级终端来电接听界面

2.7 实验报告要求

1．简述接打电话的 AT 命令；
2．简述收发短信过程；
3．按照 AT 指令集试运行其他的 AT 命令。