

EduKitIII S3C44B0 MDK 实验教程

基于 Embest EduKitIII
S3C44B0 嵌入式开发与应用实验平台



深圳市英蓓特信息技术有限公司

Embest Info &Tech Co., Ltd.

地址:深圳市罗湖区太宁路 85 号罗湖科技大厦 509 室 (518020)

Tel: 86-755-25635656 25638952 25638953

Fax:86-755-25616057

E-mail: sales@embedinfo.com support@embedinfo.com

<http://www.embedinfo.com>

<http://www.embed.com.cn>

目 录

| | |
|----------------------------|-----|
| 第一章 嵌入式系统开发与应用概述 | 3 |
| 1.1 嵌入式系统开发与应用 | 3 |
| 1.2 基于ARM的嵌入式开发环境概述 | 5 |
| 1.3 各种ARM开发工具简介 | 7 |
| 1.4 如何学习基于ARM嵌入式系统开发 | 15 |
| 第二章 EMBEST ARM实验教学系统 | 17 |
| 2.1 教学系统介绍 | 17 |
| 2.2 教学系统安装 | 22 |
| 2.3 集成开发环境使用说明 | 27 |
| 第三章 嵌入式软件开发基础实验 | 54 |
| 3.1 ARM汇编指令实验一 | 54 |
| 3.2 ARM汇编指令实验二 | 63 |
| 3.3 Thumb 汇编指令实验 | 69 |
| 3.4 ARM处理器工作模式实验 | 73 |
| 3.5 C语言实例一 | 80 |
| 3.6 C语言实验程序二 | 83 |
| 3.7 汇编与C语言相互调用实例 | 90 |
| 3.8 综合实验 | 94 |
| 第四章 基本接口实验 | 101 |
| 4.1 存储器实验 | 101 |
| 4.2 I/O 接口实验 | 111 |
| 4.3 中断实验 | 120 |
| 4.4 串口通信实验 | 131 |
| 4.5 实时时钟实验 | 141 |
| 4.6 数码管显示实验 | 150 |
| 4.7 看门狗控制实验 | 155 |
| 第五章 人机接口实验 | 163 |
| 5.1 液晶显示实验 | 163 |
| 5.2 5x4 键盘控制实验 | 176 |
| 5.3 触摸屏控制实验 | 180 |
| 第六章 通信与音频接口实验 | 194 |
| 6.1 IIC 串行通信实验 | 194 |
| 6.2 以太网通讯实验 | 205 |
| 6.3 音频接口IIS实验 | 225 |
| 6.4 USB接口实验 | 236 |
| 第七章 基础应用实验 | 246 |
| 7.1 A/D转换实验 | 246 |
| 附录 售后服务与技术支持 | 255 |

深圳市英蓓特信息技术有限公司©2005

版权所有，保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本书的部分或全部，并不得以任何形式传播。

Embest[®] 为深圳市英蓓特信息技术有限公司的商标，不得仿冒。

Copywrite©2005 by Shenzhen Embest Info&Tech Co.,LTD.

All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Embest Info&Tech Co.,LTD.

Embest[®] is registered trademarks of Embest Info&Tech Co.,LTD.

第一章 嵌入式系统开发与应用概述

1.1 嵌入式系统开发与应用

以嵌入式计算机为技术核心的嵌入式系统是继网络技术之后，又一个 IT 领域新的技术发展方向。由于嵌入式系统具有体积小、性能强、功耗低、可靠性高以及面向行业具体应用等突出特征，目前已经广泛地应用于军事国防、消费电子、信息家电、网络通信、工业控制等各个领域。嵌入式的广泛应用可以说是无所不在。就我们周围的日常生活用品而言，各种电子手表、电话、手机、PDA、洗衣机、电视机、电饭锅、微波炉、空调器都有嵌入式系统的存在，如果说我们生活在一个充满嵌入式的世界，是毫不夸张的。据统计，一般家用汽车的嵌入式计算机在 24 个以上，豪华汽车的在 60 个以上。难怪美国汽车大亨福特公司的高级经理也曾宣称，“福特出售的‘计算能力’已超过了 IBM”，由此可见嵌入式计算机工业的应用规模、应用深度和应用广度。

嵌入式系统组成的核心部件是各种类型的嵌入式处理器/DSP。随着嵌入式系统不断深入到人们生活中的各个领域，嵌入式处理器也进而得到前所未有的飞速发展。目前据不完全统计，全世界嵌入式处理器/DSP 的品种总量已经超过 1500 多种，流行体系结构也有近百个系列，现在几乎每个半导体制造商都生产嵌入式处理器/DSP，越来越多的公司有自己的处理器/DSP 设计部门。

嵌入式微处理器技术的基础是通用计算机技术。现在许多嵌入式处理器也是从早期的 PC 机的应用发展演化过来的，如早期 PC 诸如 TRS-80、Apple II 和所用的 Z80 和 6502 处理器，至今仍为低端的嵌入式应用。在应用中，嵌入式微处理器具有体积小、重量轻、成本低、可靠性高的优点。嵌入式处理器目前主要有 Am186/88、386EX、SC-400、Power PC、68000、MIPS、ARM 等系列。

在早期实际的嵌入式应用中，芯片选择时往往以某一种微处理器内核为核心，在芯片内部集成必要的 ROM/EPROM/Flash/EEPROM、SRAM、接口总线及总线控制逻辑、定时/计数器、WatchDog、I/O、串行口、脉宽调制输出、A/D、D/A 等各种必要的功能和外设。为了适应不同的应用需求，一般一个系列具有多种衍生产品，每种衍生产品的处理器内核几乎都是一样的，不同之处在于存储器的种类、容量和外设接口模块的配置及芯片封装。这样可以最大限度地和实际的应用需求相匹配，满足实际产品的开发需求，使得芯片功能不多不少，从而降低功耗、减少成本。随着嵌入式系统应用普及的日益广泛，嵌入式系统的复杂度提高，控制算法也更加冗繁，尤其是嵌入式 Internet 的广泛应用、嵌入式操作系统的引入、触摸屏等复杂人机接口的广泛使用以及芯片设计及加工工艺的提高，以 32 位处理器为核的 SOC 系统芯片的大范围使用，极大的推动了嵌入式 IT 技术的发展速度。计算机应用的普及、互联网技术的使用以及纳米微电子技术的突破，也正有力地推动着未来的工业生产、商务活动、科学实验和家庭生活等领域的自动化和信息化进程。全生产过程自动化产品制造、大范围电子商务活动、高度协同科学实验以及现代化家庭起居，也为嵌入式产品造就了崭新而巨大的商机。

除了信息高速公路的交换机、路由器和 Modem，构建 CIMS 所需的 DCS 和机器人以及规模较大的家用汽车电子系统。最有量产效益和时代特征的嵌入式产品应数因特网上的信息家电，如 Web 可视电话、Web 游戏机、Web PDA(俗称电子商务、商务通)、WAP 电话手机、以及多媒体产品，如 STB(电视机顶盒)、DVD 播放机、电子阅读机。

嵌入式应用领域近几年发展起来的一项概念和技术就是嵌入式 Internet 实际应用，也代表着它是指设备通过嵌入式模块而非 PC 系统直接接入 Internet，以 Internet 为介质实现信息交互的过程，通常又称为非 PC 的 Internet 接入。

嵌入式 Internet 将世界各地不同地方、不同种类及不同应用的设备联系起来，实现信息资源的实时共享。嵌入式 Internet 技术的广泛应用满足了二十一世纪人们要求随时随地获取信息、处理信息的需求。对 IT 产业发展规律进行总结，如果说前 20 年 PC 机的广泛应用是集成电路、IT 相关技术发展的驱动器，并且极大的促进 IT 相关技术发展的话，那么后 20 年除了 PC 技术要继续高速发展之外，主要驱动器应该是与 Internet 结合的可移动的（Mobile）、便携的（Portable）、实时嵌入式 Internet 信息处理设备。目前嵌入式 Internet 还是局限于智能家居（家电上网）、工业控制和智能设备的应用等。随着相关应用技术的发展，嵌入式技术必将会和许多领域的实际应用相结合，以难以想象的应用范围和速度发展，这必然会极大拓展嵌入式应用的广度和深度，体现嵌入式更加广泛的与实际应用密切结合的实用价值。

嵌入式 Internet 应用的核心是高性能、低功耗的各种基于网络信息处理的嵌入式系统芯片 SOC 及相关应用技术，它们是以 Internet 为介质，通过嵌入式网络处理器实现信息交互过程的非 PC Internet 设备接入。

随着 Internet 技术的成熟、带宽的运行速度的提高，ICP 和 ASP 在网上提供的信息内容日趋丰富、应用项目也多种多样。像手机、电话座机及电冰箱、微波炉等嵌入式电子设备的功能已不再单一，电气结构也更为复杂。为了满足应用功能的升级，系统设计技术人员一方面采用更强大的嵌入式处理器如 32 位、64 位 RISC 芯片或信号处理器 DSP 增强处理能力；同时还采用实时多任务编程技术和交叉开发工具技术来控制功能的复杂性，简化应用程序设计过程、保障软件质量和缩短产品开发周期。

目前，市面上已有几千种嵌入式芯片可供选择。由于面向应用的需要，许多产品设计人员还是根据自己产品特点设计自己的嵌入式芯片。通常设计人员首先获得嵌入式微处理器核的授权，然后增加他们应用产品所需的专门特点的接口模块。例如，针对数码像机处理器有可能加一个电荷耦合芯片；对网络应用产品处理则可能加一个以太网接口，而嵌入式微处理器核应用会越来越多，选用不同的核，会使电路的性能差别很大。

ARM 系列处理器核是英国先进 RISC 机器公司（Advanced RISC Machines, ARM）的产品。ARM 公司自成立以来，一直以 IP(Intelligence Property)提供者的身份向各大半导体制造商出售知识产权，而自己从不介入芯片的生产销售，它提供一些高性能、低功耗、低成本和高可靠性的 RISC 处理器核、外围部件和系统级芯片的应用解决设计方案。

ARM 处理器核具有低功耗、低成本等卓越性能和显著优点，越来越多的芯片厂商早已看好 ARM 的前景。ARM 处理器核得到了众多的半导体厂家和整机厂商的大力支持，在 32 位嵌入式应用领域获得了巨大的成功,如 Intel、Motorola、IBM、NS、Atmel、Philips、NEC、OKI、SONY 等世界上几乎所有的半导体公司获得 ARM 授权，开发具有自己特色的基于 ARM 的嵌入式系统芯片。

目前非常流行的 ARM 芯核有 ARM7TDMI, ARM720T, ARM9TDMI, ARM920T, ARM940T, ARM946T, ARM966T, XScale 等。ARM 公司 2003 在美国加利福尼亚州圣荷西市召开的嵌入式处理器论坛上公布了四个新的 ARM11 系列微处理器内核（ARM1156T2-S 内核、ARM1156T2F-S 内核、ARM1176JZ-S 内核和 ARM11JZF-S 内核）、应用 ARM1176JZ-S 和 ARM11JZF-S 内核系列的 PrimeXsys

平台、相关的 CoreSight 技术。ARM 公司日前发布最新的 Cortex 处理器，分为 R、M、A 三个系列，其中 A 系列与应用（Application）有关；如应用于高清电视、手机通用芯片等；R 系列与实时嵌入系统（Realtime）有关；M 系列与微控制器（MCU）有关。它将给消费和低功耗移动产品带来重大变革，使得最终用户可以享受到更高水准的娱乐和创新。此外，ARM 芯片还获得了许多实时操作系统(Real Time Operating System)供应商的支持，比较知名的有：Windows CE、uCLinux、pSOS、VxWorks、Nucleus、EPOC、uC/OS、BeOS、Palm OS、QNX 等。

ARM 公司具有完整的产业链，ARM 的全球合作伙伴主要为半导体和系统伙伴、操作系统伙伴、开发工具伙伴、应用伙伴、ARM 技术共享计划（ATAP），ARM 的紧密合作伙伴已发展为 122 家半导体和系统合作伙伴、50 家操作系统合作伙伴，35 家技术共享合作伙伴，并在 2002 年在上海成立中国全资子公司。早在 1999 年，ARM 就已突破 1.5 亿个，市场份额超过了 50%，而在最新的市场调查表明，在 2001 年度里，ARM 占据了整个 32、64 位嵌入式微处理器市场的 75%，在 2002 年度里，占据了整个 32、64 位嵌入式微处理器市场的 79.5%，全世界已使用了 20 多亿个 ARM 核。ARM 已经成为业界的龙头老大，“每个人口袋中装着 ARM”，是毫不夸张的。因为几乎所有的手机、移动设备、PDA 几乎都是用具有 ARM 核的系统芯片开发的。

1.2 基于 ARM 的嵌入式开发环境概述

ARM 技术是高性能、低功耗嵌入式芯片的代名词，在嵌入式尤其是在基于嵌入式 Internet 方面应用广泛。因此，学习嵌入式系统的开发应用技术，应该是基于某种 ARM 核系统芯片应用平台基础上进行，在讲述嵌入式系统开发应用之前，应该对基于 ARM 的嵌入式开发环境进行了解，本节主要对如何构造 ARM 嵌入式开发环境等基本情况介绍。

1.2.1 交叉开发环境

作为嵌入式系统应用的 ARM 处理器，其应用软件的开发属于跨平台开发，因此需要一个交叉开发环境。交叉开发是指在一台通用计算机上进行软件的编辑编译，然后下载到嵌入式设备中进行运行调试的开发方式。用来开发的通用计算机可以选用比较常见的 PC 机、工作站等，运行通用的 Windows 或 Unix 操作系统。开发计算机一般称为主机，嵌入式设备称为目标机，在主机上编译好的程序，下载到目标机上运行，交叉开发环境提供调试工具对目标机上运行的程序进行调试。

交叉开发环境一般由运行于主机上的交叉开发软件(最少必须包含编译调试模块)、主机到目标机的调试通道组成。

运行于主机上的交叉开发软件最少必须包含编译调试模块，其编译器为交叉编译器。作为主机的一般为基于 x86 体系的桌上型计算机，而编译出的代码必须在 ARM 体系结构的目标机上运行，这就是所谓的交叉编译了。在主机上编译好目标代码后，通过主机到目标机的调试通道将代码下载到目标机，然后由运行于主机的调试软件控制代码在目标机上运行调试。为了方便调试开发，交叉开发软件一般为一个整合编辑、编译汇编链接、调试、工程管理及函数库等功能模块的集成开发环境 IDE（Integrated Development Environment）。

组成 ARM 交叉开发环境的主机到目标机的调试通道一般有以下三种：

1) 基于 JTAG 的 ICD(In-Circuit Debugger)。

JTAG 的 ICD 也称为 JTAG 仿真器，是通过 ARM 芯片的 JTAG 边界扫描口进行调试的设备。JTAG 仿真器通过 ARM 处理器的 JTAG 调试接口与目标机通信，通过并口或串口、网口、USB 口与主机通

讯。JTAG 仿真器比较便宜，连接比较方便。通过现有的 JTAG 边界扫描口与 ARM CPU 核通信，属于完全非插入式(即不使用片上资源)调试，它无需目标存储器，不占用目标系统的任何应用端口。通过 JTAG 方式可以完成：

- 读出/写入 CPU 的寄存器，访问控制 ARM 处理器内核。
- 读出/写入内存，访问系统中的存储器。
- 访问 ASIC 系统。
- 访问 I/O 系统
- 控制程序单步执行和实时执行
- 实时地设置基于指令地址值或者基于数据值的断点。

基于 JTAG 仿真器的调试是目前 ARM 开发中采用最多的一种方式。

2)Angel 调试监控软件。

Angel 调试监控软件也称为驻留监控软件，是一组运行在目标板上的程序，可以接收宿主主机上调试器发送的命令，执行诸如设置断点、单步执行目标程序、读写存储器、查看或修改寄存器等操作。宿主主机上的调试软件一般通过串行端口、以太网口、并行端口等通讯端口与 Angel 调试监控软件进行通信。与基于 JTAG 的调试不同，Angel 调试监控程序需要占用一定的系统资源，如内存、通信端口等。驻留监控软件是一种比较低廉有效的调试方式，不需要任何其他的硬件调试和仿真设备。Angel 调试监控程序的不便之处在于它对硬件设备的要求比较高，一般在硬件稳定之后才能进行应用软件的开发，同时它占用目标板上的一部分资源，如内存、通信端口等，而且不能对程序的全速运行进行完全仿真，所以对一些要求严格的情况不是很适合。

3)在线仿真器 ICE(In-Circuit Emulator)。

在线仿真器 ICE 是一种模拟 CPU 的设备，在线仿真器使用仿真头完全取代目标板上的 CPU，可以完全仿真 ARM 芯片的行为，提供更加深入的调试功能。在和宿主主机连接的接口上，在线仿真器也是通过串行端口或并行端口、网口、USB 口通信。在线仿真器为了能够全速仿真时钟速度很高的 ARM 处理器，通常必须采用极其复杂的设计和工艺，因而其价格比较昂贵。在线仿真器通常用在 ARM 的硬件开发中，在软件的开发中较少使用，其价格昂贵，也是在线仿真器难以普及的因素。

1.2.2 模拟开发环境

在很多时候为保证项目进度，硬件和软件开发往往同时进行，这时作为目标机的硬件环境还没有建立起来，软件的开发就需要一个模拟环境来进行调试。模拟开发环境建立在交叉开发环境基础之上，是对交叉开发环境的补充。这时，除了宿主主机和目标机之外，还需要提供一个在宿主主机上模拟目标机的环境，使得开发好的程序直接在这个环境里运行调试。模拟硬件环境是非常复杂的，由于指令集模拟器与真实的硬件环境相差很大，即使用户使用指令集模拟器调试通过的程序也有可能无法在真实的硬件环境下运行，因此软件模拟不可能完全代替真正的硬件环境，这种模拟调试只能作为一种初步调试，主要是用作用户程序的模拟运行，用来检查语法、程序的结构等简单错误，用户最终还必须在真实的硬件环境中实际运行调试，完成整个应用的开发。

1.2.3 评估电路板

评估电路板，也称作开发板，一般用来作为开发者学习板、实验板，可以作为应用目标板出来之前的软件测试、硬件调试的电路板。尤其是对应用系统的功能没有完全确定、初步进行嵌入式开发且没有相关开发经验的非常重要。开发评估电路板并不是 ARM 应用开发必须的，对于有经验的工程师完

全可以自行独立设计自己的应用电路板和根据开发需要设计实验板。好的评估电路板一般文档齐全，对处理器的常用功能模块和主流应用都有硬件实现，并提供电路原理图和相关开发例程与源代码供用户设计自己的应用目标板和应用程序作参考。选购合适于自己实际应用的开发板可以加快开发进度，可以减少自行设计开发的工作量。

1.2.4 嵌入式操作系统

随着嵌入式应用的迅猛发展，以前不怎么知名的嵌入式操作系统概念开始流行起来，以至很多初学者认为嵌入式开发必须采用嵌入式操作系统。实际上，一个嵌入式应用是否采用嵌入式操作系统，采用哪种嵌入式操作系统完全由项目的复杂程度、实时性要求、应用软件规模、目标板硬件资源以及产品成本等因素决定。早期的嵌入式系统并没有操作系统，只不过有一个简单的控制循环而已，对很简单的嵌入式系统开发来说，这可能满足开发需求。随着嵌入式系统在复杂性上的增长，一个操作系统显得重要起来，有些复杂的嵌入式系统也许是因为设计者坚持不要操作系统才使系统开发过程非常复杂。

嵌入式操作系统一般可以提供内存管理、多任务管理、外围资源管理，给应用程序设计带来很多好处，但嵌入式操作系统同时也会占用一定的系统资源，并且要在用户自己的目标板上运行起来，并基于操作系统来设计自己的应用程序，也会相应地带来很多新的问题。所以对于不太复杂的应用完全可以不用操作系统，而对于应用软件规模较大的场合，采用操作系统则可以省掉很多麻烦。嵌入式操作系统是嵌入式开发中一个非常大的课题，目前已有专门的书籍做详细讲解，这里就不进行讨论了。

关于各种嵌入式操作系统及其对 ARM 处理器的支持情况，用户也可以访问网站 <http://www.embed.com.cn> 了解，该网站对目前流行的大多数嵌入式操作系统都有介绍。

用户选用 ARM 处理器开发嵌入式系统时，建立嵌入式开发环境是非常重要的，以上对嵌入式开发环境的基本情况作了简单介绍，一般来说一套具备最基本功能的交叉开发环境是 ARM 嵌入式开发必不可少的，至于嵌入式实时操作系统、评估板等其他开发工具则可以根据应用软件规模和开发计划选用。

1.3 各种 ARM 开发工具简介

用户选用 ARM 处理器开发嵌入式系统时，选择合适的开发工具可以加快开发进度，节省开发成本，用户在建立自己的基于 ARM 嵌入式开发环境时，可供选择的开发工具是非常多的，目前世界上有几十多家公司提供不同类别的 ARM 开发工具产品，根据功能的不同，分别有编译软件、汇编软件、链接软件、调试软件、嵌入式操作系统、函数库、评估板、JTAG 仿真器、在线仿真器等。有些工具是成套提供的，有些工具则需要组合使用。在本节中，我们将简要介绍几种比较流行的 ARM 开发工具，包括 ARM SDT、ARM ADS、Multi 2000、RealViewMDK 等集成开发环境以及 OPENice32-A900 仿真器、Multi-ICE 仿真器、ULink 2 仿真器等。

1.3.1 ARM 的 SDT

ARM SDT 的英文全称是 ARM Software Development Kit，是 ARM 公司 (www.arm.com) 为方便用户在 ARM 芯片上进行应用软件开发而推出的一整套集成开发工具。ARM SDT 经过 ARM 公司逐年的维护和更新，目前的最新版本是 2.5.2，但从版本 2.5.1 开始，ARM 公司宣布推出一套新的集成开发工具 ARM ADS 1.0，取 ARM SDT 而代之，今后将不会再看到 ARM SDT 的新版本。

ARM SDT 由于价格适中，同时经过长期的推广和普及，目前拥有最广泛的 ARM 软件开发用户群体，也被相当多的 ARM 公司的第三方开发工具合作伙伴集成在自己的产品中，比如美国 EPI 公司的 JEENI 仿真器。

ARM SDT（以下关于 ARM SDT 的描述均是以版本 2.50 为对象）可在 Windows95、98、NT 以及 Solaris 2.5/2.6、HP-UX 10 上运行，支持最高到 ARM9（含 ARM9）的所有 ARM 处理器芯片的开发，包括 Strong ARM。

ARM SDT 包括一套完整的应用软件开发工具：

- armcc ARM 的 C 编译器，具有优化功能，兼容于 ANSI C。
- tcc THUMB 的 C 编译器，同样具有优化功能，兼容于 ANSI C。
- armasm 支持 ARM 和 THUMB 的汇编器。
- armlink ARM 连接器，连接一个和多个目标文件，最终生成 ELF 格式的可执行映像文件。
- armsd ARM 和 THUMB 的符号调试器。

以上工具为命令行开发工具，均被集成在 SDT 的两个 Windows 开发工具 ADW 和 APM 中，用户无需直接使用命令行工具。

- APM Application Project Manager，ARM 工程管理器，完全图形界面，负责管理源文件，完成编辑、编译、链接并最终生成可执行映像文件等功能，见下图。

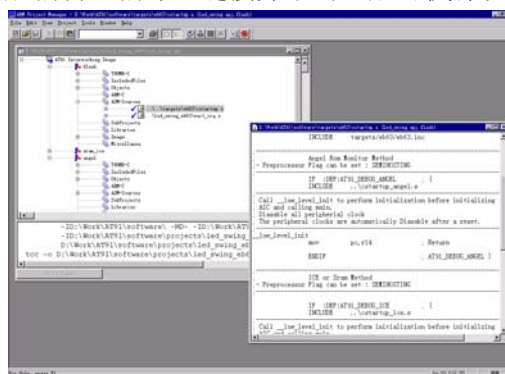


图 1-1 APM 项目管理器窗口

- ADW Application Debugger Windows，ARM 调试工具，ADW 提供一个调试 C、C++ 和汇编源文件的全窗口源代码级调试环境，在此也可以执行汇编指令级调试，同时可以查看寄存器、存储区、栈等调试信息。

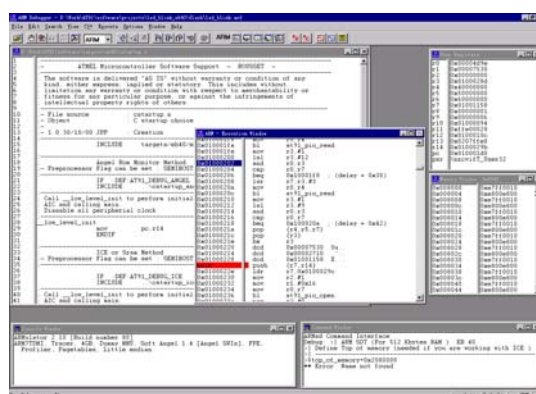


图 1-2 ADW 窗口

ARM SDT 还提供一些实用程序，如 fromELF、armprof、decaxf 等，可以将 ELF 文件转换为不同的格式，执行程序分析以及解析 ARM 可执行文件格式等。

ARM SDT 集成快速指令集模拟器，用户可以在硬件完成以前完成一部分调试工作；ARM SDT 提

供 ANSI C、C++、Embedded C 函数库，所有库均以 lib 形式提供，每个库都分为 ARM 指令集和 THUMB 指令集两种，同时在各指令集中也分为高字节结尾（big endian）和低字节结尾（little endian）两种。

用户使用 ARM SDT 开发应用程序可选择配合 Angel 驻留模块或者 JTAG 仿真器进行，目前大部分 JTAG 仿真器均支持 ARM SDT。

1.3.2 ARM 的 ADS

ARM ADS 的英文全称为 ARM Developer Suite，是 ARM 公司推出的新一代 ARM 集成开发工具，用来取代 ARM 公司以前推出的开发工具 ARM SDT。

ARM ADS 起源于 ARM SDT，对一些 SDT 的模块进行了增强并替换了一些 SDT 的组成部分，用户可以感受到的最强烈的变化是 ADS 使用 CodeWarrior IDE 集成开发环境替代了 SDT 的 APM，使用 AXD 替换了 ADW，现代集成开发环境的一些基本特性如源文件编辑器语法高亮，窗口驻留等功能在 ADS 中才得以体现。

ARM ADS 支持所有 ARM 系列处理器包括最新的 ARM9E 和 ARM10，除了 ARM SDT 支持的运行操作系统外还可以在 Windows2000/Me 以及 RedHat Linux 上运行。

ARM ADS 由六部分组成：

- 代码生成工具（Code Generation Tools）

代码生成工具由源程序编译、汇编、链接工具集组成。ARM 公司针对 ARM 系列每一种结构都进行了专门的优化处理，这一点除了作为 ARM 结构的设计者的 ARM 公司，其他公司都无法办到，ARM 公司宣称，其代码生成工具最终生成的可执行文件最多可以比其他公司工具套件生成的文件小 20%。

- 集成开发环境（CodeWarrior IDE from Metrowerks）

CodeWarrior IDE 是 Metrowerks 公司一套比较有名的集成开发环境，有不少厂商将它作为界面工具集成在自己的产品中。CodeWarrior IDE 包含工程管理器、代码生成接口、语法敏感编辑器、源文件和类浏览器、源代码版本控制系统接口、文本搜索引擎等，其功能与 Visual Studio 相似，但界面风格比较独特。ADS 仅在其 PC 机版本中集成了该 IDE。



图 1-3 源程序窗口

- 调试器（Debuggers）

调试器部分包括两个调试器：ARM 扩展调试器 AXD（ARM eXtended Debugger）、ARM 符号调试器 armsd（ARM symbolic debugger）。

AXD 基于 Windows9X/NT 风格，具有一般意义上调试器的所有功能，包括简单和复杂断点设置、

栈显示、寄存器和存储区显示、命令行接口等。

Armsd 作为一个命令行工具辅助调试或者用在其他操作系统平台上。

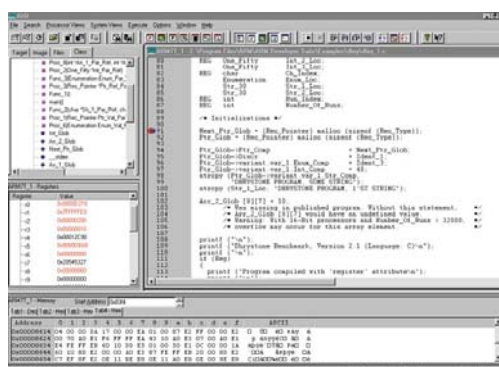


图 1-4 AXD 窗口

- 指令集模拟器（Instruction Set Simulators）

用户使用指令集模拟器无需任何硬件即可在 PC 机上完成一部分调试工作。

- ARM 开发包（ARM Firmware Suite）

ARM 开发包由一些底层的例程和库组成，帮助用户快速开发基于 ARM 的应用和操作系统。具体包括系统启动代码、串行口驱动程序、时钟例程、中断处理程序等，Angel 调试软件也包含在其中。

- ARM 应用库（ARM Applications Library）

ADS 的 ARM 应用库完善和增强了 SDT 中的函数库，同时还包括一些相当有用的提供了源代码的例程。

用户使用 ARM ADS 开发应用程序与使用 ARM SDT 完全相同，同样是选择配合 Angel 驻留模块或者 JTAG 仿真器进行，目前大部分 JTAG 仿真器均支持 ARM ADS。

ARM ADS 的零售价为 5500 美元，如果选用不固定的许可证方式则需要 6500 美元。

1.3.3 Multi 2000

Multi 2000 是美国 Green Hills 软件公司(www.ghs.com)开发的集成开发环境，支持 C/C++/Embedded C++/Ada 95/Fortran 编程语言的开发和调试，可运行于 Windows 平台和 Unix 平台，并支持各类设备的远程调试。

Multi 2000 支持 Green Hills 公司的各类编译器以及其它遵循 EABI 标准的编译器，同时 Multi 2000 支持众多流行的 16 位、32 位和 64 位处理器和 DSP，如 PowerPC、ARM、MIPS、x86、Sparc、TriCore、SH-DSP 等，并支持多处理器调试。

Multi 2000 包含完成一个软件工程所需要的所有工具，这些工具可以单独使用，也可集成第三方系统工具。Multi 2000 各模块相互关系以及和应用系统相互作用如下图所示：

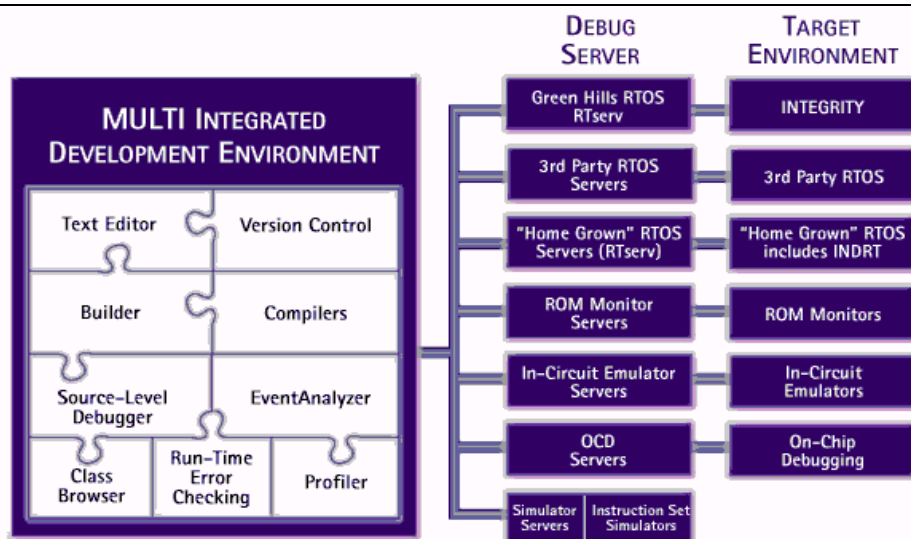


图 1-5 MULTI2000 模块与应用系统

- 工程生成工具（Project Builder）

工程生成工具实现对项目源文件、目标文件、库文件以及子项目的统一管理，显示程序结构，检测文件相互依赖关系，提供编译和链接的图形设置窗口，并可对编程语言的进行特定环境设定。

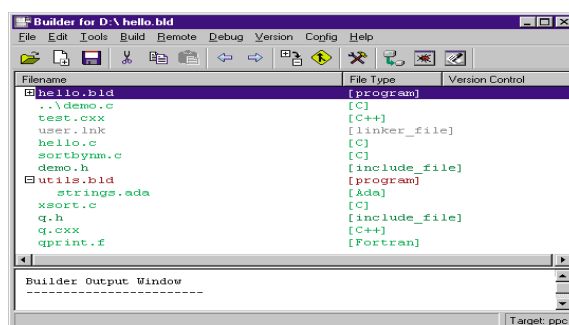


图 1-6 项目生成工具界面

- 源代码调试器（Source-Level Debugger）

源代码调试器提供程序装载、执行、运行控制和监视所需要的强大的窗口调试环境，支持各类语言的显示和调试，同时可以观察各类调试信息。

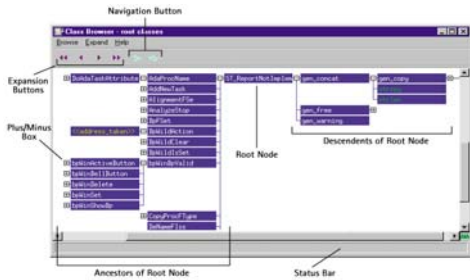


图 1-10 浏览器界面

● 文本编辑器（Text Editor）

Multi 2000 的文本编辑器是一个具有丰富特性的用户可配置的文本图形化编辑工具，提供关键字高亮显示、自动对齐等辅助功能。

● 版本控制工具（Version Control System）

Multi 2000 的版本控制工具和 Multi 2000 环境紧密结合，提供对应用工程的多用户共同开发功能。Multi 2000 的版本控制工具通过配置对支持很多流行的版本控制程序，如 Rational 公司的 ClearCase 等。

1.3.4 RealView MDK

MDK（Microcontroller Development Kit）是 Keil 公司（An ARM Company）开发的 ARM 开发工具，是用来开发基于 ARM 核的系列微控制器的嵌入式应用程序的开发工具。它适合不同层次的开发者使用，包括专业的应用程序开发工程师和嵌入式软件开发的入门者。MDK 包含了工业标准的 C 编译器、宏汇编器、调试器、实时内核等组件，支持所有基于 ARM 的设备，能帮助工程师按照计划完成项目。

Keil ARM 开发工具集集成了很多有用的工具（如图 1-11 所示），正确的使用它们，可以有助于快速完成项目开发。

| 组件 | Part Number | |
|---------------------------|------------------------|--------|
| | MDK-ARM ^{2,3} | DB-ARM |
| μVision IDE | ✓ | ✓ |
| RealView C/C++ Compiler | ✓ | |
| RealView Macro Assembler | ✓ | |
| RealView Utilities | ✓ | |
| RTL-ARM Real-Time Library | ✓ | |
| μVision Debugger | ✓ | ✓ |
| GNU GCC1 | ✓ | ✓ |

图 1-11 MDK 开发工具的组件

以下是 MDK 所包含组件的一些说明：

- μVision IDE 集成开发环境和 μVision Debugger 调试器可以创建和测试应用程序，可以用 RealView、CARM 或者 GNU 的编译器来编译这些应用程序；
- MDK-ARM 是 PK-ARM 的一个超集；
AARM 汇编器、CARM C 编译器、LARM 连接器和 OHARM 目标文件到十六进制的转换器仅包含在 MDK-ARM 开发工具集中。

MDK 的最新版本是 μVision 3，可以开发基于 ARM7、ARM9、Cortex-M3 的微控制器应用程序，它易学易用且功能强大。以下是它的一些主要特性：

- μVision 3 集成了一个能自动配置工具选项的设备数据库；
- 工业标准的 RealView C/C++编译器能产生代码容量最小、运行速度最快的高效应用程序，同

时它包含了一个支持 C++ STL 的 ISO 运行库；

- 集成在 μ Vision 3 中的在线帮助系统提供了大量有价值的信息，可加速应用程序开发速度；
- 包含大量的例程，帮助开发者快速配置 ARM 设备，以及开始应用程序的开发；
- μ Vision 3 集成开发环境能帮助工程人员开发稳健、功能强大的嵌入式应用程序；
- μ Vision 3 调试器能够精确地仿真整个微控制器，包括其片上外设，使得在没有目标硬件的情况下也能测试开发程序；
- 包含标准的微控制器和外部 Flash 设备的 Flash 编程算法；
- ULINK USB-JTAG 仿真器可以实现 Flash 下载和片上调试；
- RealView RL-ARM 具有网络和通信的库文件以及实时软件；
- 还可使用第三方工具扩展 μ Vision 3 的功能；
- μ Vision 3 还支持 GNU 的编译器。

本教程的所有例程均在 MDK 下开发，在第二章中将对 MDK 的使用作详细介绍。

1.3.5 OPENice32-A900 仿真器

OPENice32-A900 仿真器是韩国 AIJI 公司(www.aijisystem.com)生产的。OPENice32-A900 是 JTAG 仿真器，支持基于 ARM7/ARM9/ARM10 核的处理器以及 Intel Xscale 处理器系列。它与 PC 之间通过串口或 USB 口或网口连接，与 ARM 目标板之间通过 JTAG 口连接。OPENice32-A900 仿真器主要特性如下：

- 支持多核处理器和多处理器目标板。
- 支持汇编与 C 语言调试。
- 提供在板(on-board)flash 编程功能。
- 提供存储器控制器设置 GUI。
- 可通过升级软件的方式支持更新的 ARM 核。

OPENice32-A900 仿真器自带宿主调试软件 AIJI Spider，但需要使用第三方编译器。AIJI Spider 调试器支持 ELF/DWARF1/DWARF2 等调试符合信息文件格式，可以通过 OPENice32-A900 仿真器下载 BIN 文件到目标板，控制程序在目标板上的运行并进行调试。支持单步、断点设置、查看寄存器/变量/内存以及 Watch List 等调试功能。

OPENice32-A900 仿真器也支持一些第三方调试器，包括 Linux GDB 调试器和 EWARM、ADS/SDT 等调试工具。

1.3.6 Multi-ICE 仿真器

Multi-ICE 是 ARM 公司自己的 JTAG 在线仿真器，目前的最新版本是 2.1 版。

Multi-ICE 的 JTAG 链时钟可以设置为 5 kHz 到 10 MHz，实现 JTAG 操作的一些简单逻辑由 FPGA 实现，使得并行口的通信量最小，以提高系统的性能。Multi-ICE 硬件支持低至 1V 的电压。Multi-ICE 2.1 还可以外部供电，不需要消耗目标系统的电源，这对调试类似于手机等便携式、电池供电设备是很重要的。

Multi-ICE 2.x 支持该公司的实时调试工具 MultiTrace，MultiTrace 包含一个处理器，因此可以跟踪触发点前后的轨迹，并且可以在不终止后台任务的同时对前台任务进行调试，在微处理器运行时改变存储器的内容，所有这些特性使延时降到最低。

Multi-ICE 2.x 支持 ARM7、ARM9、ARM9E、ARM 10 和 Intel Xscale 微结构系列。它通过 TAP 控制器串联，提供多个 ARM 处理器以及混合结构芯片的片上调试。它还支持低频或变频设计以及超低压核的调试，并且支持实时调试。

Multi-ICE 提供支持 Windows NT4.0、Windows95/98/2000/Me、HPUX 10.20 和 Solaris V2.6/7.0 的驱动程序。

Multi-ICE 主要优点：

- 快速的下载和单步速度。
- 用户控制的输入/输出位。
- 可编程的 JTAG 位传送速率。
- 开放的接口，允许调试非 ARM 的核或 DSP。
- 网络连接到多个调试器。
- 目标板供电，或外接电源。

1.3.7 ULINK 2 仿真器

ULINK 是 Keil 公司提供的 USB-JTAG 接口仿真器，目前最新的版本是 2.0。它支持诸多芯片厂商的 8051、ARM7、ARM9、Cortex M3、Infineon C16x、Infineon XC16x、Infineon XC8xx、STMicroelectronics μ PSD 等多个系列的处理器。ULINK 2 内部实物如图 1-12 所示，电源由 PC 机的 USB 接口提供。ULINK2 不仅包含了 ULINK USB-JTAG 适配器具有的所有特点，还增加了串行线调试（SWD）支持，返回时钟支持和实时代理功能。ULINK2 适用与标准的 Windows USB 驱动等功能。

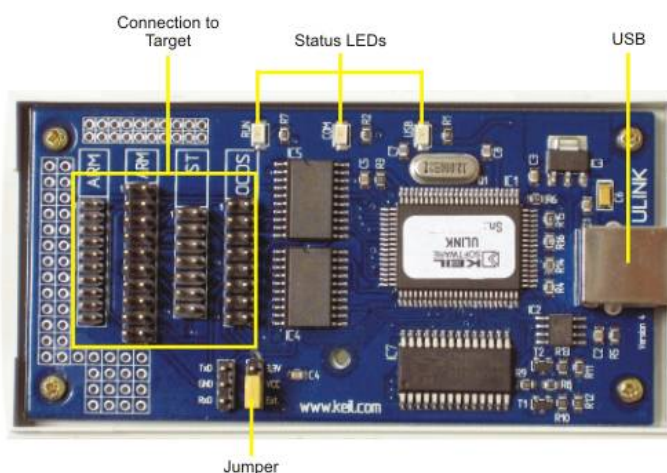


图 1-12 ULINK 2（无盖）

ULINK 2 的主要功能：

- 下载目标程序；
- 检查内存和寄存器；
- 片上调试，整个程序的单步执行；
- 插入多个断点；
- 运行实时程序；
- 对 FLASH 存储器进行编程。

ULINK2 新特点

- 标准 Windows USB 驱动支持，也就是 ULINK2 即插即用；
- 支持基于 ARM Cortex-M3 的串行线调试；
- 支持程序运行期间的存储器读写、终端仿真和串行调试输出；
- 支持 10/20 针连接器。

本教程中所有的例程使用的 ULINK USB-JTAG 仿真器套件即 ULINK 2 仿真器。

1.4 如何学习基于 ARM 嵌入式系统开发

ARM 微处理器因其卓越的低功耗、高性能在 32 位嵌入式应用中已位居世界第一，是高性能、低功耗嵌入式处理器的代名词，为了顺应当今世界技术革新的潮流，了解、学习和掌握嵌入式技术，就

必然要学习和掌握以 ARM 微处理器为核心的嵌入式开发环境和开发平台，这对于研究和开发高性能微处理器、DSP 以及开发基于微处理器的 SOC 芯片设计及应用系统开发是非常必要的。

那么究竟如何学习嵌入式的开发和应用？技术基础是关键。技术基础决定了学习相关知识、掌握相关技能的潜能。嵌入式技术融合具体应用系统技术、嵌入式微处理器/DSP 技术、系统芯片 SOC 设计制造技术、应用电子技术和嵌入式操作系统及应用软件技术，具有极高的系统集成性，可以满足不断增长的信息处理技术对嵌入式系统设计的要求。因此学习嵌入式系统首先是基础知识学习，主要是相关的基本硬件知识，如一般处理器及接口电路（Flash/ SRAM/SDRAM /Cache, UART, Timer, GPIO, Watchdog、USB、IIC 等...）等硬件知识，至少了解一种 CPU 的体系结构；至少了解一种操作系统（中断，优先级，任务间通信，同步...）。对于应用编程，要掌握 C、C++ 及汇编语言程序设计（至少会 C），对处理器的体系结构、组织结构、指令系统、编程模式、一般对应用编程要有一定的了解。在此基础上必须在实际工程实践中掌握一定的实际项目开发技能。

其次对于嵌入式系统开发的学习，必须要有一个较好的嵌入式开发教学平台。功能全面的开发平台一方面为学习提供了良好的开发环境，另一方面开发平台本身也是一般的典型实际应用系统。在教学平台上开发一些基础例程和典型实际应用例程，对于初学者和进行实际工程应用也是非常必要的。

嵌入式系统的学习必须对基本内容有深入的了解。在处理器指令系统、应用编程学习的基础上，重要的是加强外围功能接口应用的学习，主要是人机接口、通讯接口，如 USB 接口、A/D 转换、GPIO、以太网、IIC 串行数据通信、音频接口、触摸屏等知识的掌握。

嵌入式操作系统也是嵌入式系统学习重要的一部分，在此基础上才能进行各种设备驱动应用程序的开发。

第二章 Embest ARM 实验教学系统

2.1 教学系统介绍

Embest ARM 教学系统包括 μ Vision IDE 集成开发环境，ULINK USB-JTAG 仿真器，Embest EduKit-III 开发板、各种连接线、电源适配器以及实验指导书等。基本实验模型示意图如 2-1 所示：



图 2-1 实验模型示意图

2.1.1 μ Vision 3 集成开发环境

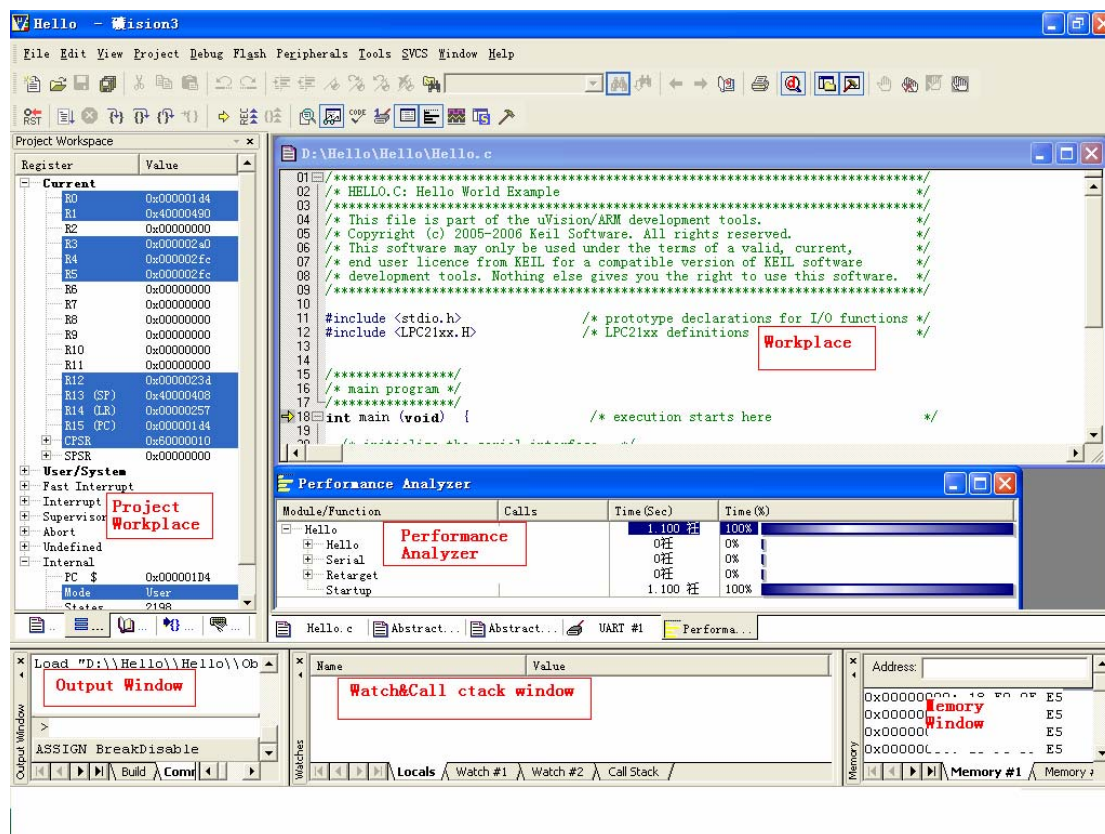
1) μ Vision 3 是一个基于窗口的软件开发平台，它集成了功能强大的编辑器、工程管理器以及 make 工具。 μ Vision3 IDE 集成的工具包括 C 编译器、宏汇编器、链接/定位器和十六进制文件生成器。 μ Vision 有编译和调试两种工作模式，两种模式下设计人员都可查看并修改源文件。图 2-2 是编译模式下典型的窗口配置， μ Vision IDE 由多个窗口、对话框、菜单栏、工具栏组成。其中菜单栏和工具栏用来实现快速的操作命令；工程工作区（Project Workspace）用于项目文件管理、寄存器调试、函数管理、手册管理等；输出窗口（Output Window）用于显示编译信息、搜索结果以及调试命令交互灯；内存窗口（Memory Window）可以不同格式显示内存中的内容；观测窗口（Watch & Call Stack Window）用于观察、修改程序中的变量以及当前的函数调用关系；工作区（Workspace）用于文件编辑、反汇编输出和一些调试信息显示；外设对话框（Peripheral Dialogs）帮助设计者观察片内外围接口的工作状态。

2) μ Vision IDE 主要功能特点及组件

μ Vision IDE 可在 Windows 98、2000、NT 及 XP 等操作系统上运行，主要支持基于 ARM7、ARM9、Cortex-M3 系列处理器，目前最新的版本为 μ Vision 3，其主要特点如下：

- μ Vision 3 集成了一个能自动配置工具选项的设备数据库；
- 工业标准的 RealView C/C++ 编译器能产生代码容量最小、运行速度最快的高效应用程序，同时它包含了一个支持 C++ STL 的 ISO 运行库；
- 集成在 μ Vision 3 中的在线帮助系统提供了大量有价值的信息，可加速应用程序开发速度；
- 包含大量的例程，帮助开发者快速配置 ARM 设备，以及开始应用程序的开发；
- μ Vision 3 集成开发环境能帮助工程人员开发稳健、功能强大的嵌入式应用程序；

- μ Vision 3 调试器能够精确地仿真整个微控制器，包括其片上外设，使得在没有目标硬件的情况下也能测试开发程序；
- 包含标准的微控制器和外部 Flash 设备的 Flash 编程算法；
- ULINK USB-JTAG 仿真器可以实现 Flash 下载和片上调试；
- RealView RL-ARM 具有网络和通信的库文件以及实时软件；
- 还可使用第三方工具扩展 μ Vision 3 的功能；
- μ Vision 3 还支持 GNU 的编译器。

图 2-2 μ Vision IDE 开发环境软件界面

μ Vision IDE 包含以下功能组件，能加速嵌入式应用程序开发过程：

- 功能强大的源代码编辑器；
- 可根据开发工具配置的设备数据库；
- 用于创建和维护工程的工程管理器；
- 集汇编、编译和链接过程于一体的编译工具；
- 用于设置开发工具配置的对话框；
- 真正集成高速 CPU 及片上外设模拟器的源码级调试器；
- 高级 GDI 接口，可用于目标硬件的软件调试和 Keil ULINK 仿真器的连接；
- 用于下载应用程序到 Flash ROM 中的 Flash 编程器；
- 完善的开发工具手册、设备数据手册和用户向导。

μ Vision IDE 使用简单、功能强大，是保证设计者完成设计任务的重要保证。 μ Vision IDE 还提供了大量的例程及相关信息，有助于开发人员快速开发嵌入式应用程序。

μVision IDE 有编译和调试两种工作模式。编译模式用于维护工程文件和生成应用程序；调试模式下，则可以用功能强大的 CUP 和外设仿真器来测试程序，也可以使用调试器经 Keil ULINK USB-JTAG 适配器（或其他 AGDI 驱动器）来连接目标系统测试应用程序。ULINK 仿真器能用于下载应用程序到目标系统的 Flash ROM 中。

在嵌入式软件开发时，完成设计和编码后，即开始调试程序，这是软件开发的第三步。一个几千行的程序，其编译可达到没有一个警告，然而在运行时却可能达不到正常的设计需求、甚至系统无法运行起来而崩溃，更为难以查找的是系统运行只是在偶然的情况下出现问题或崩溃。当程序不能顺利运行，而又不能简单、直观的分析、知道问题的症结所在时，就该使用调试器来监视此程序的运行了。μVision IDE 调试器提供程序装载、执行、运行控制和监视所需要的强大的窗口调试环境，支持源码显示和调试，同时可以观察各类调试信息。μVision IDE 具有功能强大的调试器，μVision 3 调试器用于调试和测试应用程序，它提供了两种操作模式：仿真模式和 GDI 驱动器模式。可以在 Options for Target – Debug 对话框内进行选择，如图 2-3 所示。

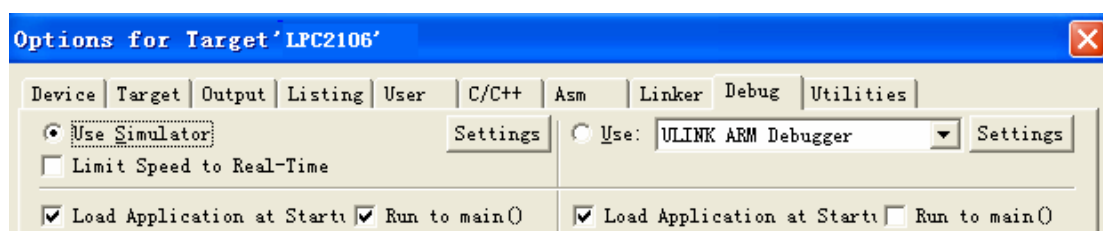


图 2-3 调试器操作模式的选择

◆ 仿真模式

仿真模式可在无目标系统硬件情况下，仿真微控器的许多特性。可在目标硬件准备好之前，把 μVision 3 调试器配置为软件仿真，可以测试和调试所开发的嵌入式应用，μVision 3 能仿真大量的外围设备包括串口、外部 I/O 及时钟等。在为目标程序选择 CPU 时，相应外围接口就被从设备库中选定。

◆ GDI 驱动器模式

GDI 驱动模式下，可使用高级 GDI 驱动器，例如 ULINK ARM Debugger 来连接目标硬件。对 μVision 3 来说，以下几种驱动器均可用于连接目标硬件：

- JTAG/OCDS 适配器：连接到片上的调试系统，如 AMR Embedded ICE；
- 监视器：可以集成在用户硬件上、也可以在许多评估板上；
- 仿真器：连接到目标硬件的 CPU 引脚上
- 测试硬件：例如 Infineon SmartCard ROM 监视器，或 Philips SmartMX DBox。

2.1.2 ULINK USB-JTAG 仿真器

JTAG 仿真器也称为 JTAG 调试器，是通过 ARM 芯片的 JTAG 边界扫描口进行调试的设备。JTAG 仿真器连接比较方便，通过现有的 JTAG 边界扫描口与 ARM CPU 核通信，属于完全非插入式(即不使用片上资源)调试，它无需目标存储器，不占用目标系统的任何端口，而这些是驻留监控软件所必需的。

另外，由于 JTAG 调试的目标程序是在目标板上执行，仿真更接近于目标硬件，因此，许多接口问题，如高频操作限制、AC 和 DC 参数不匹配，电线长度的限制等被最小化了。使用集成开发环境配

合 JTAG 仿真器进行开发是目前采用最多的一种调试方式。ULINK USB-JTAG 仿真器如图 2-4 (a) 所示。

ULINK USB-JTAG 仿真器支持众多 Philips、Samsung、Atmel、Analog Devices、Sharp、ST 等众多厂商 ARM7 及 ARM9 内核的 ARM 微控制器, 将其 PC 机的 USB 端口与用户的目标硬件相连(通过 JTAG 或 OCD), 使用户可在目标硬件上调试代码。通过使用 Keil uVision IDE/调试器和 ULINK USB-JTAG 仿真器, 用户可以方便地编辑、下载和在实际的目标硬件上测试嵌入式的程序, 并且具有三个 LED 灯分别显示 RUN, COM, 和 USB 状态。使用 ULINK USB-JTAG 仿真器可以实现以下功能:

- USB 通讯接口高速下载用户代码
- 存储区域/寄存器查看
- 快速单步程序运行
- 添加多个程序断点
- 运行实时程序
- 片内 Flash 编程
- 运行实时程序



(a) ULINK USB-JTAG 仿真器图

(b) ULINK2 USB-JTAG 仿真器图

图 2-4 ULINK 系列仿真器图

ULINK2 通过 JTAG, SWD, 或 OCDS 将目标硬件与您电脑的 USB 端口连接起来, 使用 ULINK2 您可以调试在目标硬件上运行的嵌入式程序。ULINK2 不仅包含了 ULINK USB-JTAG 适配器具有的所有功能, 而且具有如下新的特点:

- 标准 Windows USB 驱动支持 ULINK2 即插即用
- 支持基于 ARM Cortex-M3 的串行线调试
- 支持程序运行期间的存储器读写、终端仿真和串行调试输出
- 既支持 20 针引脚, 同时也支持 10 针引脚
- ULINK2 仿真器如图 2-4 (b) 所示。

2.1.3 Embest EduKit-III 嵌入式教学实验平台

Embest EduKit-III 嵌入式教学实验平台是英蓓特公司针对高校需求, 在 II 型的基础上研发的最新的第三代教学系统。Embest EduKit-II 自从推出市场以来, 受到了老师们的一致认可。在此基础上, 英蓓特公司综合了众多使用老师的意见, 在 II 型实验系统基础上整合了更多的资料, 整理了布局, 使整个实验系统更加合理, 更加精致, 推出了 Embest EduKit-III 教学实验系统。该硬件平台如图 2-5 所示。



图 2-5 实验系统硬件平台

Embest EduKit-III 开发板的基本资源如下：

- 采用多 CPU 子板
 - ARM7 S3C44B0 子板
 - ARM9 S3C2410 子板
 - DSP Blackfin 533 子板（选配）
 - Intel Xscale270 子板（选配）
- 64M NandFlash, 2—32M NorFlash
- 64M SDRAM
- 4Kbit IIC BUS 的串行 EEPROM
- 2 个 232 串口，一个 422 串口，一个 485 串口
- 两个中断按钮，一个复位按钮，4 个 LED
- 5.7 寸 320*240 STN 彩色 LCD 及 TSP 触摸屏
- 4×5 键盘
- 20 针 JTAG 接口
- PS/2 接口
- 2 个 USB 主口
- 1 个 USB 从口
- SD 接口模块
- PCI 扩展接口
- 以太网接口
- 8 段数码管
- 双 CAN 总线模块
- A/D 模块

- IDE 硬盘接口模块
- CF 卡接口模块
- CPLD 模块
- 直流电机模块
- 步进电机模块
- MICROPHONE 输入口
- IIS 音频信号输出口
- 高速 USB2.0 Ulink2 仿真器一个
- YS244-JTAG 简易仿真头一个
- 固态硬盘 32M × 8bit (选配)
- GPRS 模块 (选配)
- GPS 模块 (选配)
- 蓝牙 (选配)
- 摄像头模块 (选配)
- WIFI 模块 (选配)
- DAC 模块 (选配)

2.1.4 各种连接线与电源适配器

实验系统除了提供以上的组件以外,还提供了各种连接时候需要的电缆线。包括交叉网线,USB 线,交叉串口线,并口线和两根 JTAG 线(分别是 20 针和 14 针接口)。

2.2 教学系统安装

Embest ARM 教学系统包括 μ Vision 3 集成开发环境、ULINK USB-JTAG 仿真器、Embest EduKit-III、各种连接线等。其中 μ Vision 3 属软件平台部分,其余属于硬件平台部分。

本节主要介绍如何安装实验系统的软件平台、如何搭建和如何进行软件平台与硬件平台的连接。

在安装 μ Vision3 IDE 集成开发环境之前请首先阅读软件使用许可协议)

安装 μ Vision 3 评估软件必须满足的最小的系统要求为:

- 操作系统: Windows 98, Windows NT4, Windows 2000, Windows XP;
- 硬盘空间: 30M 以上;
- 内存: 128M 以上。

μ Vision IDE 集成开发环境的安装方法如下:

- (1) 购买 MDK 的安装程序, 或从 <http://www.realview.com.cn/xz-down.asp> 下载 MDK 的评估版;

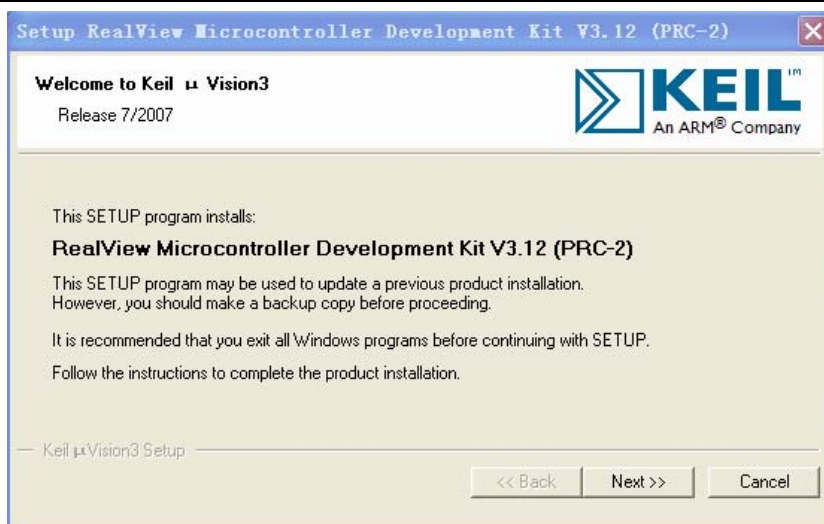


图 2-6 MDK 安装界面 1

(2) 双击安装文件，弹出如图 2-6 对话框。建议在安装之前关闭所有的应用程序，单击 Next，弹出如图 2-7 所示对话框；



图 2-7 MDK 安装界面 2

(3) 仔细阅读许可协议，选中 I agree to all the terms of the preceding License Agreement 选项，单击 Next，弹出如图 2-8 所示对话框；

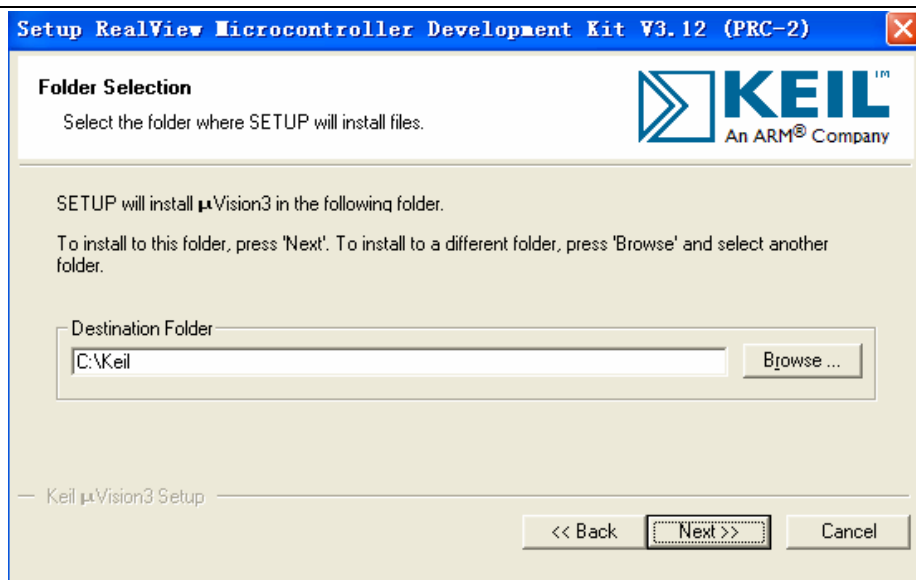


图 2-8 MDK 安装界面 3

(4) 单击 Browse 选择安装路径，然后单击 Next，弹出如图 2-9 所示对话框；

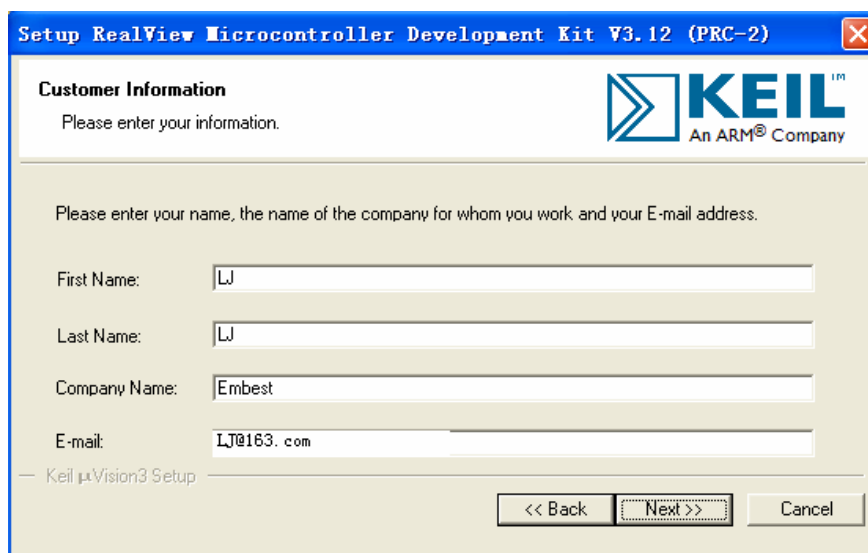


图 2-9 MDK 安装界面 4

(5) 输入 First Name、Last Name、Company Name 以及 E-mail 地址后，单击 Next；安装程序将在计算机上安装 MDK，依据机器性能的不同，安装程序大概耗时半分钟到两分钟不等，之后将会弹出如图 2-10 所示对话框，单击 Finish 结束安装。至此，开发人员就可在计算机上使用 MDK 软件来开发应用程序了。

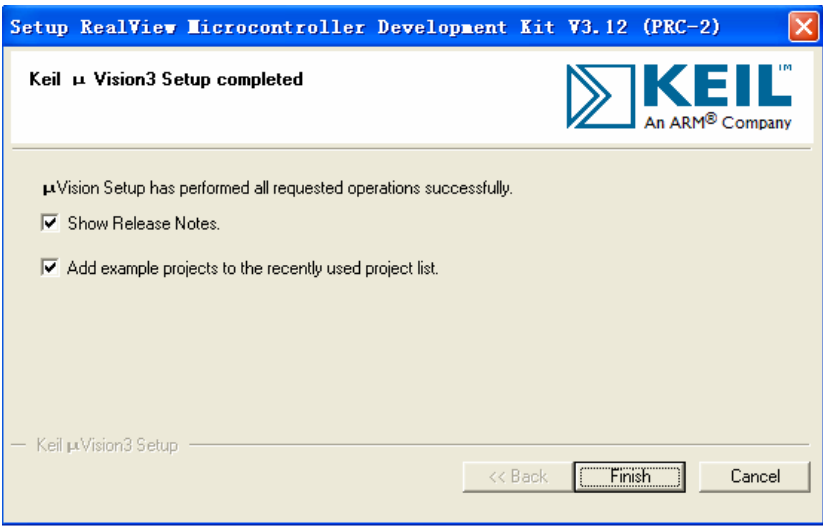



图 2-10 MDK 安装界面 5

μVision IDE 集成开发环境安装完毕后，点击 μVision IDE 的图标  即可运行 μVision IDE。第一次使用 μVision IDE 正式版时，用户必须注册。μVision 3 的有两种许可证：单用户许可证和浮动许可证。单用户许可证只允许单用户最多在二台计算机上使用 MDK，而浮动许可证则允许局域网众多台计算机分时使用 MDK。目前，所有的 Keil 软件均可使用单用户许可证注册，绝大多数 Keil 软件可使用浮动许可证注册。下面分别介绍两种许可证注册：

■ 单用户许可证注册过程

- (1) 安装好 μVision 3；
- (2) 在 μVision IDE 中，单击 File – License Management 菜单项进入许可证管理对话框；
- (3) 选择 Single-User License 页，在该页右边的 CID（Computer ID）文本框中会自动产生 CID；
- (4) 用CID和MDK提供的PSN（产品序列号）在 <https://www.keil.com/license/embest.htm>注册，确保输入邮箱的正确性；
- (5) 通过注册后，在所填写注册信息的邮箱中将会收到许可证 ID 码 LIC(License ID Code)；
- (6) 将得到的许可证 ID 输入 New License ID Code (LIC) 文本框，然后单击右边的 Add LIC 按钮，此时这册成功，如图 2-11 所示：

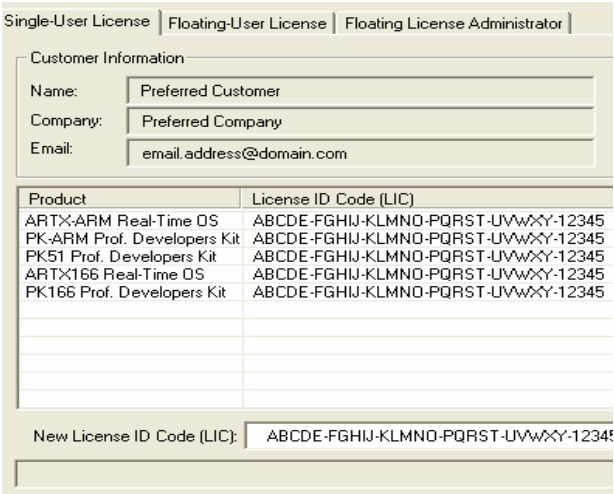


图 2-11 单用户许可证注册成功后界面

■ 浮动许可证注册过程

浮动许可证的注册过程比单用户许可证的注册过程复杂。由于它是局域网上多用户分时复用的，所以必须保证浮动许可证在公用的网络服务器上，从而保证开发团队中的每一个人都能使用该许可证。下面是浮动许可证的注册过程：

- (1) 安装好 μ Vision 3；
- (2) μ Vision IDE 中，单击 File – License Management 菜单项进入许可证管理对话框；
- (3) 选择 Floating License 页，在该页右边的 CID 文本框中会自动产生 Computer ID；
- (4) 单击“增加产品”按钮，选择浮动许可证文件（由浮动许可证管理员创建）的路径然后单击“确认”按钮，它会自动打开 <https://www.keil.com/license/lic30floating.asp> 浮动许可证注册页面；
- (5) 在上述页面中输入 CID 和浮动许可证码 PSN 以及其他相关信息，确保输入的电子邮箱地址的正确性；
- (6) 通过注册后，在所填写注册信息的邮箱中将会收到许可证 ID 码；
- (7) 将得到的许可证 ID 输入 New License ID Code (LIC) 文本框，然后单击右边的 Add LIC 按钮，此时注册成功，如图 2-12 所示。

| Product | License ID Code (LIC) | Support |
|-----------------------------------|-------------------------------------|------------|
| ARTX-ARM Real-Time OS | ABCDE-FGHIJ-KLMNO-PQRST-UVWXY-12345 | Expires: . |
| PK-ARM Prof. Developers Kit | ABCDE-FGHIJ-KLMNO-PQRST-UVWXY-12345 | Expires: . |
| PK51 Prof. Developers Kit | ABCDE-FGHIJ-KLMNO-PQRST-UVWXY-12345 | Expires: . |
| ARTX166 Real-Time OS | ABCDE-FGHIJ-KLMNO-PQRST-UVWXY-12345 | Expires: . |
| PK166 Prof. Developers Kit 3 user | ABCDE-FGHIJ-KLMNO-PQRST-UVWXY-12345 | Expires: . |

图 2-12 浮动许可证注册成功后界面

使用浮动许可证注册的某个网络用户，如果要使用 MDK，可单击图 2-12 对话框右边的 Check out 按钮，该用户将获得许可证，此时开发小组的其他用户将只能使用 MDK 评估版的功能。该用户使用完之后，需及时单击 Check in 按钮（默认的情况下一个小时后 MDK 将自动 Check in），否则其它用户将无法正常使用 MDK。

■ 浮动许可证的管理

- (1) 安装好 μ Vision 3；
- (2) μ Vision IDE 中，单击 File – License Management 菜单项进入许可证管理对话框；
- (3) 选择 Floating License Administrator 页，如图 2-13 所示在 Path 文本框中设置正确的服务器共享文件夹的路径；
- (4) 在 PSN 文本框中输入正确的产品序列号；
- (5) 单击 Create FLE 按钮，即在服务器共享文件夹中产生注册浮动许可证时需要的 FLE 文件。

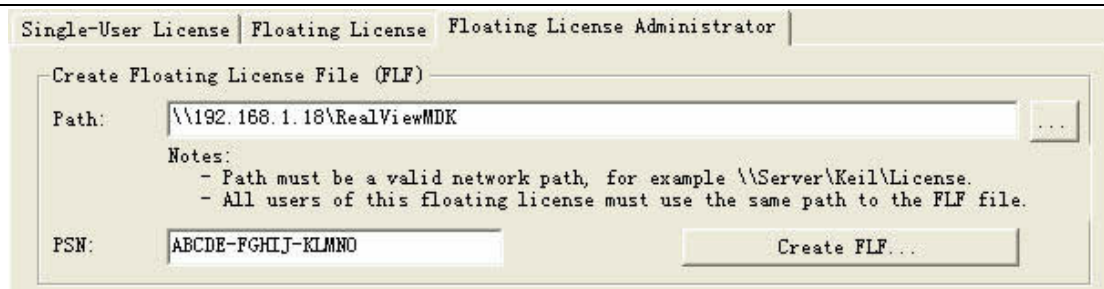


图 2-13 正确创建 FLE 文件后的界面

软件安装完毕后，请详细阅读相关软件说明及软件使用手册。

实验软件平台和硬件平台的连接如图 2-1 所示，PC 端与仿真器通过实验系统提供的并口线连接，仿真器和开发板通过一根 20 针的 JTAG 线连接。

其中需要注意：

1) 仿真器驱动程序在安装 μ Vision IDE 时自动安装，第一次使用 ULINK 仿真器时，PC 机会自动加载其驱动程序，该驱动程序已在 Windows 2000、Windows XP 和 Windows XP SP2 上测试通过，如果驱动不能自动加载，可以访问 <http://www.keil.com/support/>。

2) 硬件平台最好预先参照 Embest EduKit-III 用户手册（在 Embest ARM 教学系统光盘中）进行基本硬件检测。

2.3 集成开发环境使用说明

2.3.1 μ Vision IDE 主框架窗口

μ Vision IDE 由如图 2-2 所示的多个窗口、对话框、菜单栏、工具栏组成。其中菜单栏和工具栏用来实现快速的操作命令；工程工作区（Project Workspace）用于文件管理、寄存器调试、函数管理、手册管理等；输出窗口（Output Window）用于显示编译信息、搜索结果以及调试命令交互灯；内存窗口（Memory Window）可以不同格式显示内存中的内容；观测窗口（Watch & Call Stack Window）用于观察、修改程序中的变量以及当前的函数调用关系；工作区（Workspace）用于文件编辑、反汇编输出和一些调试信息显示；外设对话框（Peripheral Dialogs）帮助设计者观察片内外围接口的工作状态。

本节将主要介绍 μ Vision IDE 的菜单栏、工具栏、常用快捷方式，以及各种窗口的内容和使用方法，以便让读者能快速了解 μ Vision IDE，并能对 μ Vision IDE 进行简单和基本的操作。

μ Vision IDE 集成开发环境的菜单栏可提供如下菜单功能：编辑操作、工程维护、开发工具配置、程序调试、外部工具控制、窗口选择和操作，以及在线帮助等。工具栏按钮可以快速执行 μ Vision 3 的命令。状态栏显示了编辑和调试信息，在 View 菜单中可以控制工具栏和状态栏是否显示。键盘快捷键可以快速执行 μ Vision 3 的命令，它可以通过菜单命令 Edit – Configuration - Shortcut Key 来进行配置。

2.3.2 工程管理

1. 工程管理介绍

在 μ Vision IDE 集成开发环境中，工程是一个非常重要的概念，它是用户组织一个应用的所有源文件、设置编译链接选项、生成调试信息文件和最终的目标 Bin 文件的一个基本结构。一个工程管理一个应用程序的所有源文件、库文件、其它输入文件，并根据实际情况进行相应的编译链接设置，一个工程须生成一个相对应的目录，以进行文件管理。

μVision IDE 工程管理提供以下功能：

μVision IDE 的工作区由五部分组成，分别为 Files（文件）页、工作区如图 2-14 所示，它显示了工程结构。

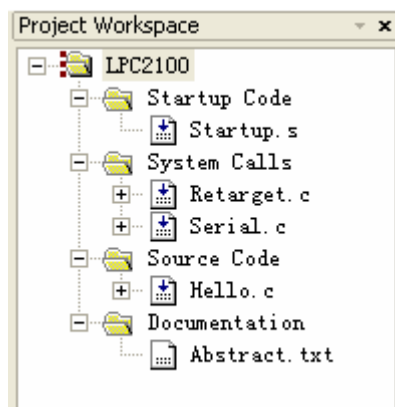


图 2-14 工程管理区结构图

- 以工程为单位定义设置应用程序的各选项，包括目标处理器和调试设备的选择与设置，调试相关信息的配置，以及编译、汇编、链接等选项的设置等。系统提供一个专门的对话框来设置这些选项。
- 提供 build 菜单和工具按钮，让用户轻松进行工程的编译、链接。编译、链接信息输出到输出窗口中的 Build 标签窗中，如图 2-15 所示，编译链接出现的错误，通过鼠标左键双击错误信息提示行来定位相应的源文件行。

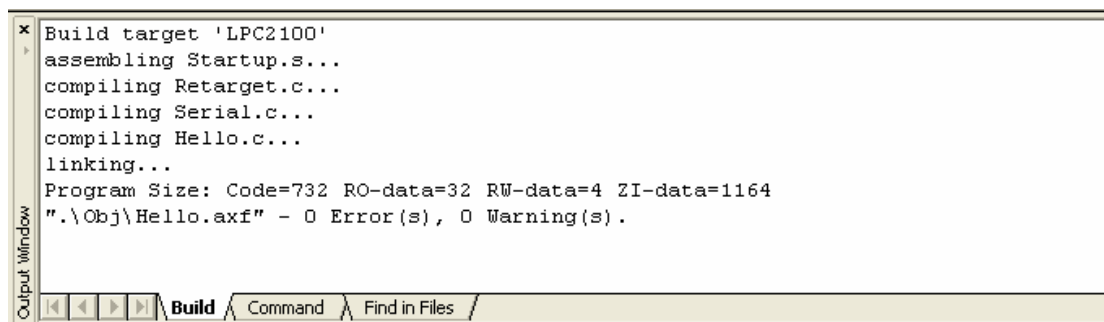


图 2-15 编译链接输出子窗口

- 一个应用工程编译链接后根据编译器的设置生成相应格式的调试信息文件，调试通过的程序转换成二进制格式的可执行文件后最终在目标板上运行。

2. 工程的创建

μVision 3 所提供的工程管理，使得基于 ARM 处理器的应用程序设计开发变得越来越方便。通常使用 μVision 3 创建一个新的工程需要以下几步：选择工具集、创建工程并选择处理器、创建源文件及文件组、配置硬件选项、配置对应启动代码、最后编译链接生成 HEX 文件。

1) 选择工具集

利用 μVision 3 创建一个基于处理器的应用程序，首先要选择开发工具集。单击 Project – Manage-Components, Environment, and Books 菜单项，在如图 2-16 所示对话框中，可选择所使用的工具集。在 μVision 3 中既可使用 ARM RealView 编译器、GNU GCC 编译器，也可以使用 Keil CARM 编译器。当使用 GNU GCC 编译器时，需要安装相应的工具集。在本例程中选择 ARM RealView 编译器，MDK 环境默认的编译器，可不用配置。

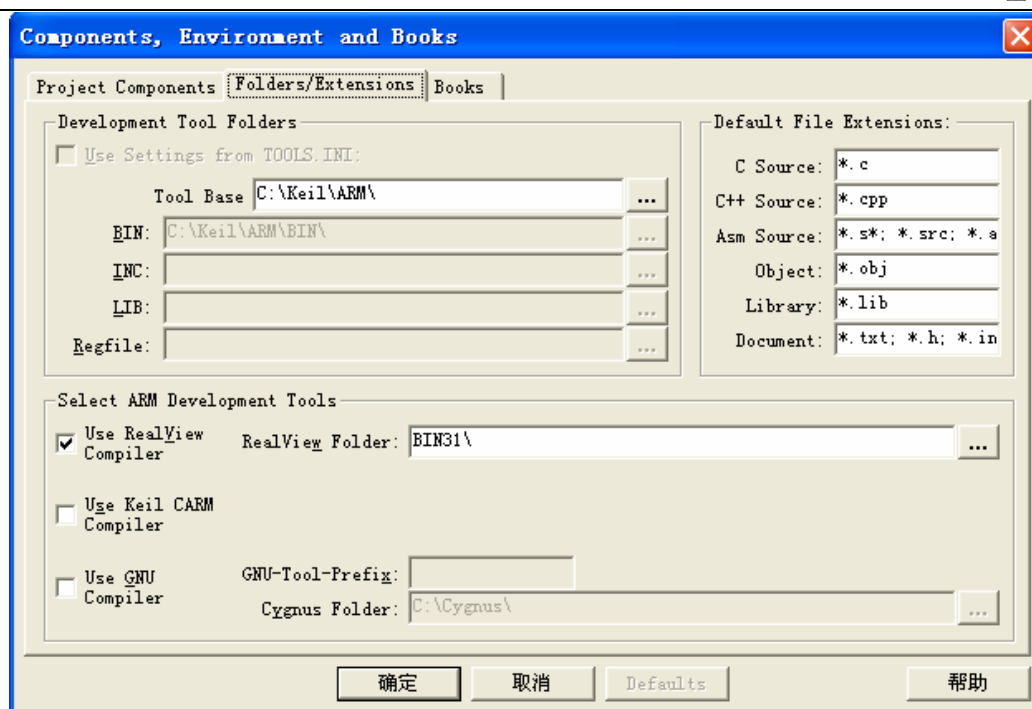


图 2-16 选择工具集

2) 创建工程并选择处理器

单击 Project – New μ Vision Project.... 菜单项， μ Vision 3 将打开一个标准对话框，输入希望新建工程的名字即可创建一个新的工程，建议对每个新建工程使用独立的文件夹。这里先建立一个新的文件夹 Hello，在前述对话框中输入 Hello， μ Vision 将会创建一个以 Hello.UV2 为名字的新工程文件，它包含了一个缺省的目标（target）和文件组名。这些内容在 Project Workspace 窗口中可以看到。

创建一个新工程时， μ Vision 3 要求设计者为工程选择一款对应处理器，如图 2-17 所示，该对话框中列出了 μ Vision 3 所支持的处理器设备数据库，也可单击 Project - Select Device... 菜单项进入此对话框。选择了某款处理之后， μ Vision 3 将会自动为工程设置相应的工具选项，这使得工具的配置过程简化。

对于大部分处理器设备， μ Vision 3 会提示是否在目标工程里加入 CPU 的相关启动代码。如图 2-18 所示。启动代码是用来初始化目标设备的配置，完成运行时系统的初始化工作，对于嵌入式系统开发而言是必不可少的，单击 Ok 便可将启动代码加入工程，这使得系统的启动代码编写工作量大大减少。

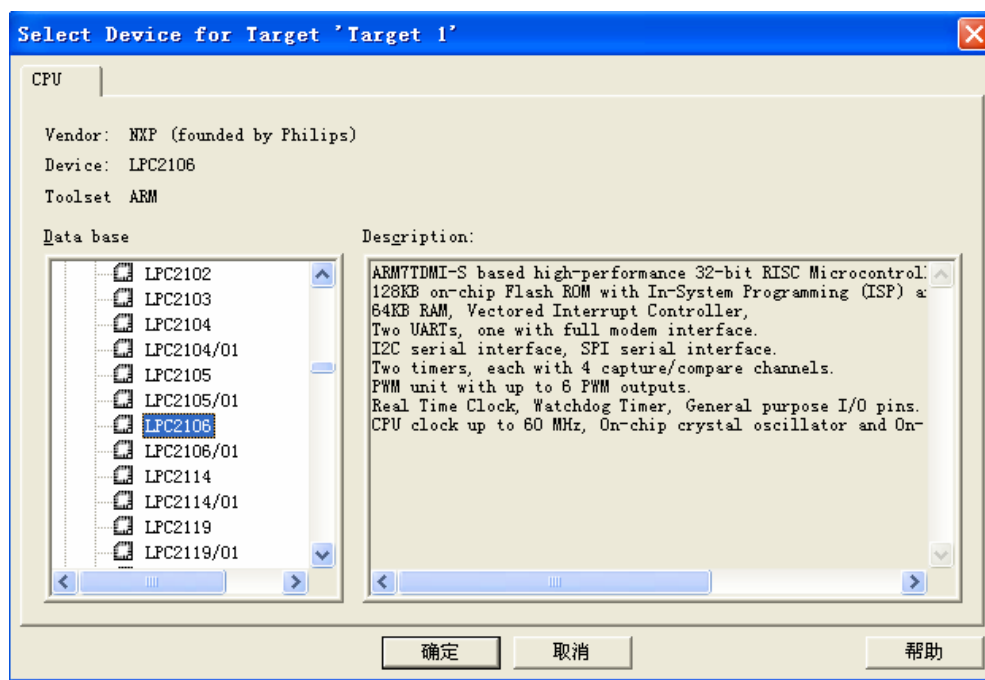


图 2-17 选择处理器

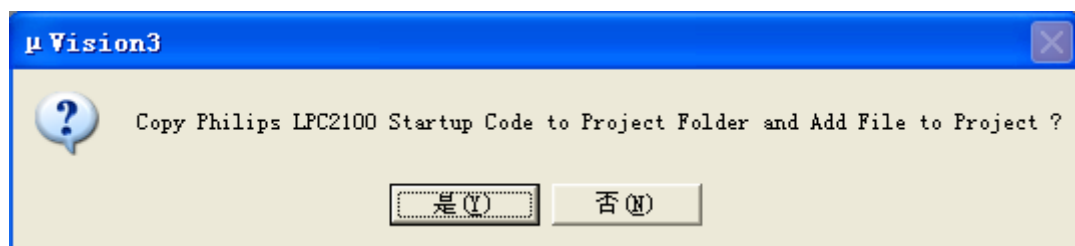


图 2-18 加入启动代码

在设备数据库中为工程选择 CPU 后，Project Workspace - Books 内就可以看到相应设备的用户手册，以供设计者参考，如图 2-19 所示。

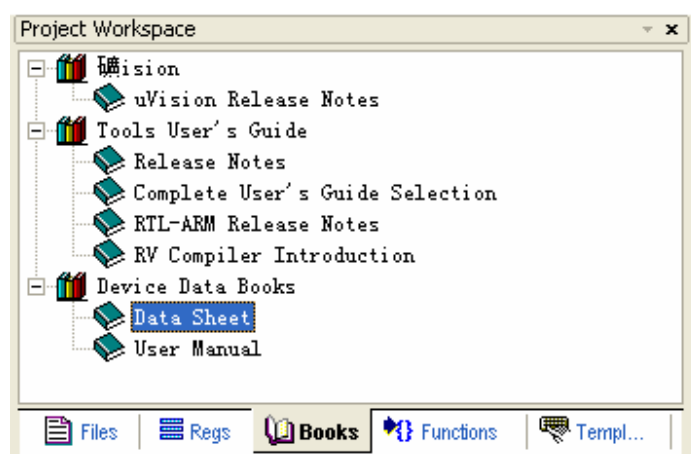


图 2-19 相应设备数据手册

3. 建立一个新的源文件

创建一个工程之后，就应开始编写源程序。选择菜单项 File - New 可创建新的源文件， μ Vision IDE 将会打开一个空的编辑窗口用以输入源程序。在输入完源程序后，选择 File - Save As... 菜单项保存源程序，当以 *.C 为扩展名保存源文件时， μ Vision IDE 将会根据语法以彩色高亮字体显示源程序。

4. 工程中文件的加入

创建完源文件后便可以在工程里加入此源文件， μ Vision 提供了多种方法加入源文件到工程中。例如，在 Project Workspace - Files 菜单项中选择文件组，右击将会弹出如图 2-20 所示快捷菜单，单击选项 Add Files to Group... 打开一个标准文件对话框，将已创建好的源文件加入到工程中。

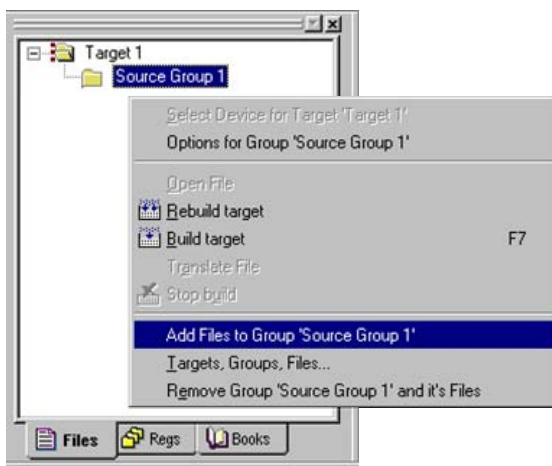


图 2-20 加入源文件到工程中

通常，设计人员应采用文件组来组织大的工程，将工程中同一模块或者同一类型的源文件放在同一文件组中。例如，可在 Project -Manage- Components, Environment, Books... 对话框中创建自己的文件组 Syssem Files 来管理 CPU 启动代码和其它系统配置文件等，如图 2-23 所示。可使用 New (Insert)按钮可创建新的文件组，或在 Groups 文件组中选定一个文件组，然后点击 **Add Files** 为其添加文件。

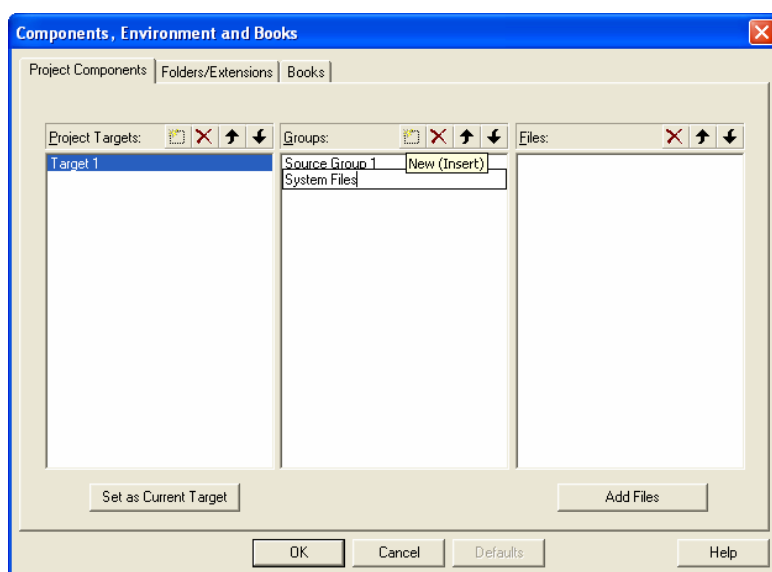



图 2-21 创建新的文件组

5. 设置活动工程

在 μ Vision IDE 中可以存在几个同时打开的工程，但只有一个工程处于活动状态并显示在工程区中，处于活动状态的工程才可以作为调试工程。可在图 2-21 中的工程目标框中选择需要激活的工程，然后单击 `Set as Current Target` 按钮即可。

2.3.3 工程基本配置

1. 硬件选项配置

μ Vision 3 可根据目标硬件的实际情况对工程进行配置。通过单击目标工具栏图标或者单击菜单项 `Project - Options for Target`，在弹出的 `Target` 页面可指定目标硬件和所选择设备片内组件的相关参数，如图 2-22 所示。

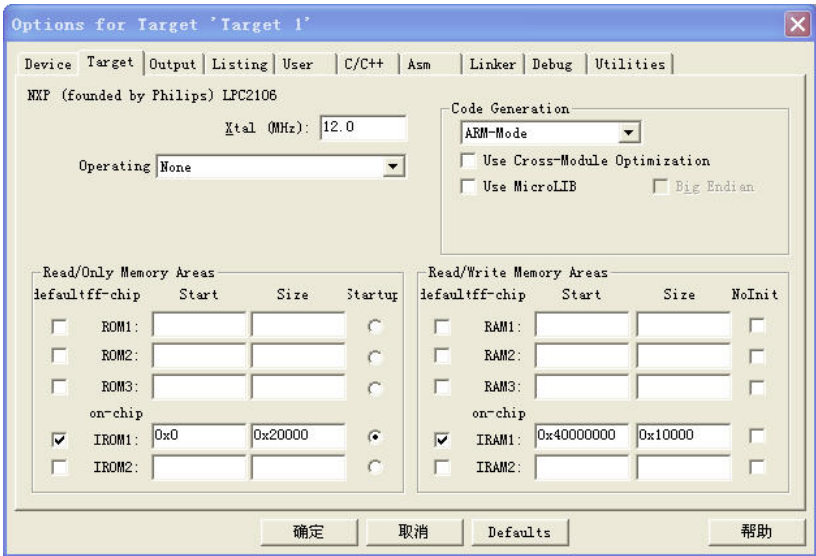


图 2-22 处理器配置对话框

表 2-1 对 `Target` 页面选项作一个简要说明：

表 2-1 目标硬件配置选项说明表

| 选项 | 描述 |
|--------------|---|
| 晶振 | 设备的晶振频率。大部分基于 ARM 的微控制器使用片内 PLL 作为 CPU 时钟源。多数情况下 CPU 时钟和晶振频率是不一致的，依据硬件设备不同设置其相应的值 |
| 使用片内 ROM/RAM | 定义片内的内存部件的地址空间以供链接器/定位器使用。注意对于一些设备来说需要在启动代码中反映出这些配置 |
| 操作系统 | 允许为目标工程选择一个实时操作系统 |

2. 处理器启动代码配置

通常情况下，ARM 程序都需要初始化代码用来配置所对应的目标硬件。如 2.3.2 节所述，当创建一个应用程序时 μ Vision 3 会提示使用者自动加入相应设备的启动代码。

μ Vision 3 提供了丰富的启动代码文件，可在相应文件夹中获得。例如，针对 Keil 开发工具的启动代码放在 `..\ARM\Startup` 文件夹下，针对 GNU 开发工具的在 `..\ARM\GNU\Startup` 文件夹下，针对 ADS 开发工具的在文件夹 `..\ARM\ADS\Startup` 下。以 LPC2106 处理器为例，其启动代码文件为 `...\Startup\Philips\Startup.s`，可把这个启动代码文件复制到工程文件夹下。在图 2-16 中双击 `Startup.s`

源文件，根据目标硬件作相应的修改即可使用。 μ Vision 3 里大部分启动代码文件都有一个配置向导（Configuration Wizard），如图 2-23 所示，它提供了一种菜单驱动方式来配置目标板的启动代码。

开发工具提供缺省的启动代码，对于大部分单芯片应用程序来说是一个很好的起点，但是开发者必须根据目标硬件来调整部分启动代码的配置，否则很可能是无法使用的。例如，CPU/PLL 时钟和总线系统往往会根据目标系统的不同而不同，不能够自动地配置。一些设备还提供了片上部件的使能/禁止可选项，这就需要开发者对目标硬件有足够的了解，能够确保启动代码的配置和目标硬件完全匹配。在图 2-25 中的 Text Editor 页面中，提供了标准文本编辑窗口可打开并修改相应的启动代码。

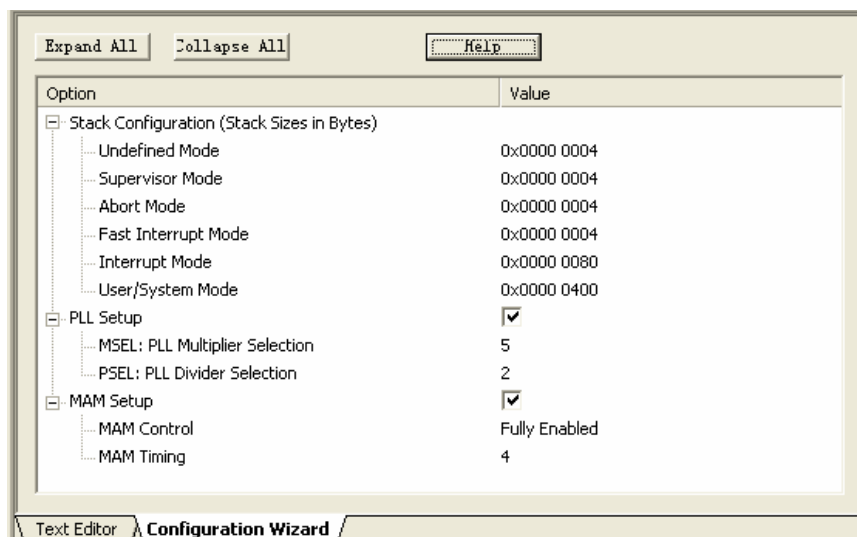


图 2-23 启动代码文件配置向导

3. 仿真器配置


选择菜单项 **Project->Project-Option for Target** 或者直接单击 ，打开 Option for Target 对话框的 Debug 页，弹出如下对话框，进行仿真器的连接配置。



图 2-24 Option for Target 对话框 Debug 页

使用 ULINK 仿真器时，为仿真器选择合适的驱动以及为应用程序和可执行文件下载进行配置，对图 2-24 对话框的设置如图 2-25、图 2-26 所示：

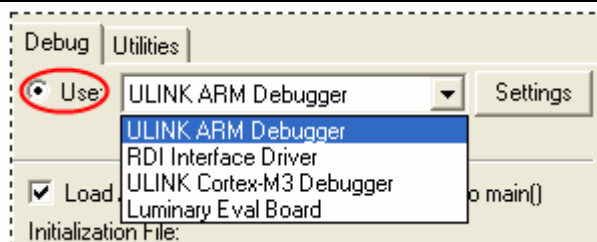


图 2-25 仿真器驱动配置图

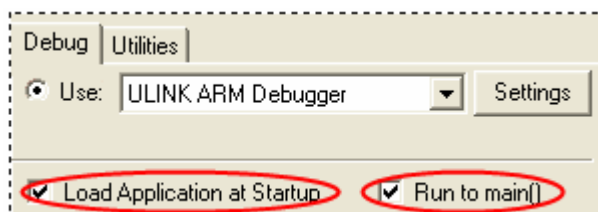


图 2-26 仿真器下载应用程序配置图

PC 机通过 ULINK USB-JTAG 仿真器与目标板连接成功之后，可以打开图 2-26 中的 Settings 选项查看 ULINK 信息，如图 2-28 所示：

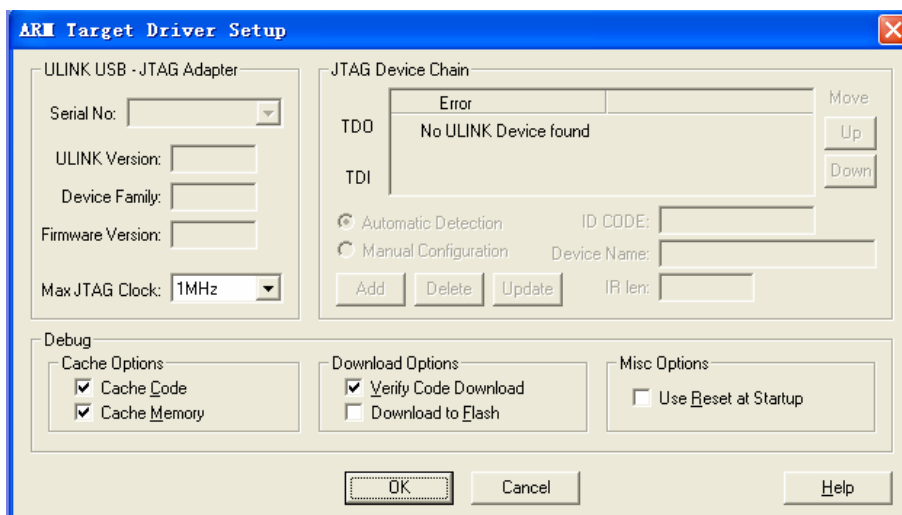


图 2-27 未连 ULINK 仿真器前

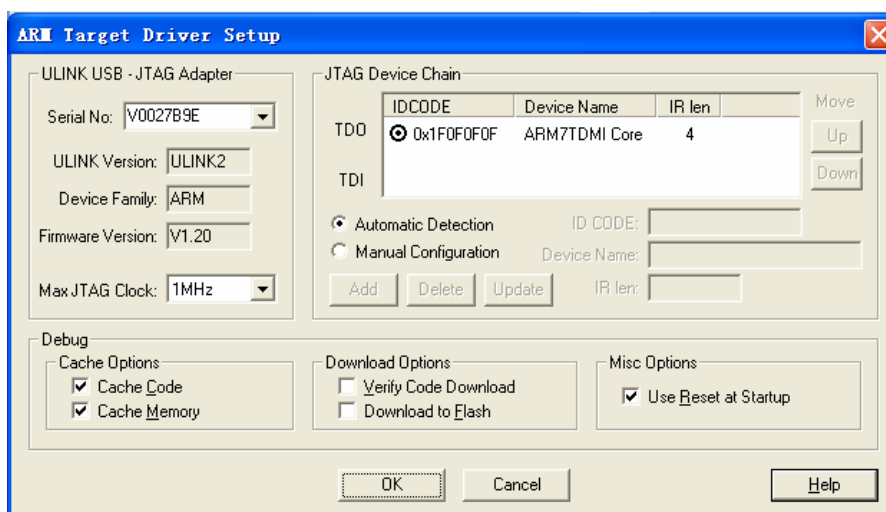


图 2-28 连接 ULINK 仿真器后

4. 工具配置

工具选项主要设置 Flash 下载选项。打开菜单栏的 **Project->Project-Option for Target** 对话框选择其“Utilities”页，或者打开菜单 **Flash->Configure Flash Tools...**，将弹出如图 2-29 所示的对话框。

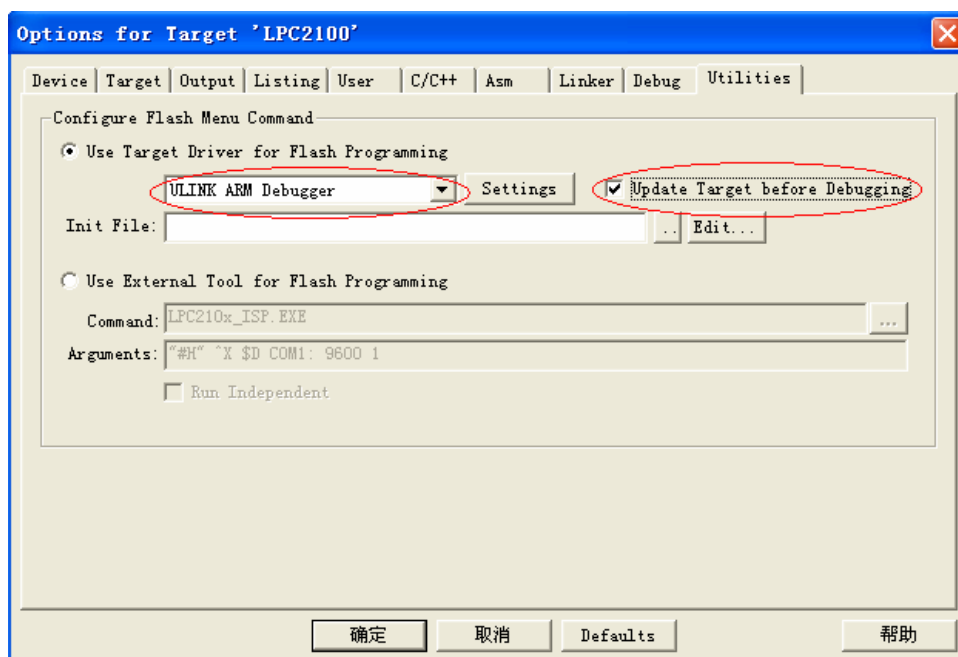


图 2-29 选择 ULINK 下载代码到 FLASH

在图 2-29 对话框中选“Use Target Driver for Flash Programming”，再选择“ULINK ARM Debugger”，同时钩上“Update Target before Debugging”选项。这时还没有完成设置，还需要选择编程算法，点击“Settings”将弹出如图 2-32 所示的对话框。

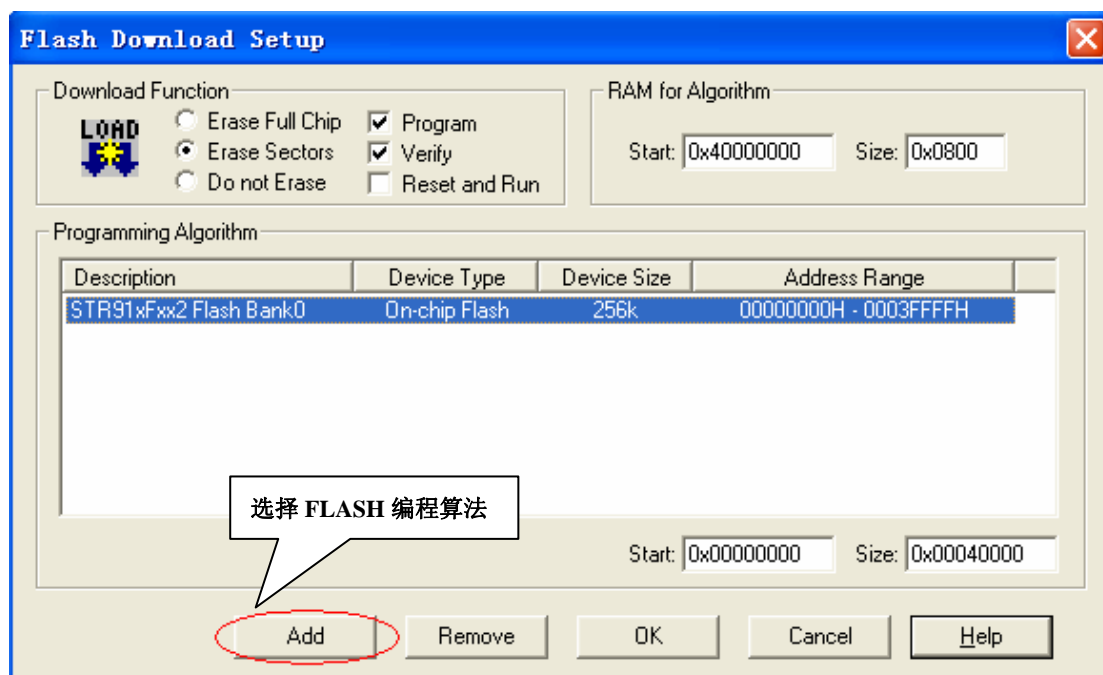


图 2-30 FLASH 下载设置

点击图 2-30 的对话框中的“Add”，将弹出如图 2-31 所示的对话框，在该对话框中选对需要的 FLASH 编程算法。例如对 STR912FW 芯片，由于其 FLASH 为 256kB 则需要选择如下图 2-31 所标注的 FLASH 编程算法。

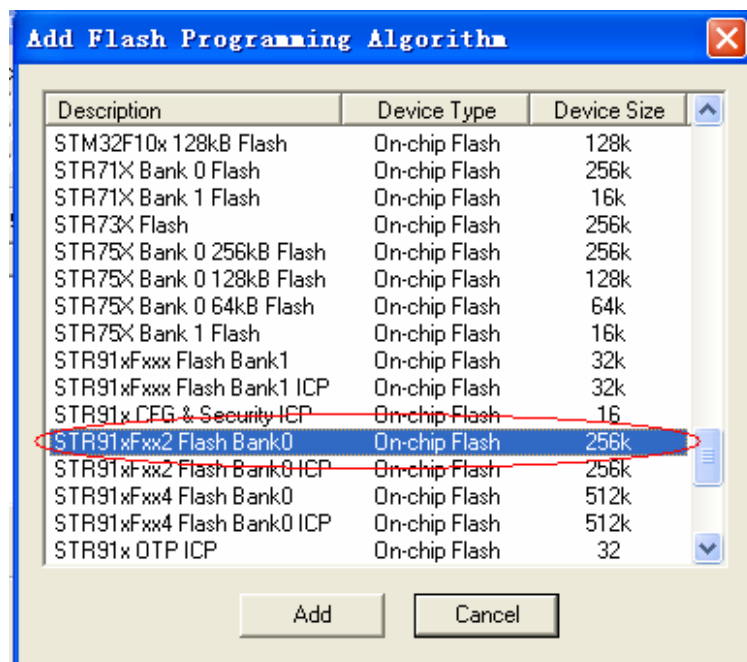


图 2-31 选择 FLASH 编程算法

5. 调试设置

μVision 3 调试器提供了两种调试模式，可以从 Project->Options for Target 对话框的 Debug 页内选择操作模式，如图 2-32 所示。

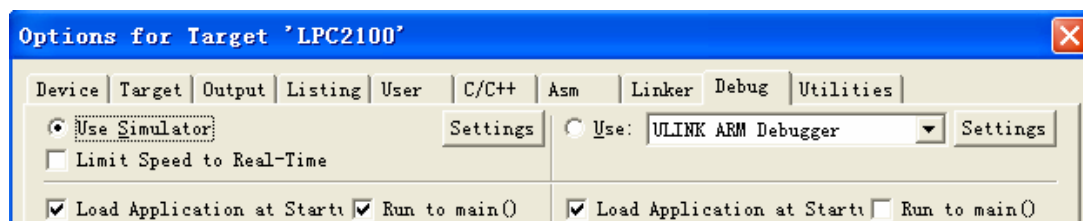


图 2-32 调试器的选择

软件仿真模式：在没有目标硬件情况下，可以使用仿真器（Simulator）将 μVision3 调试器配置为软件仿真器。它可以仿真微控制器的许多特性，还可以仿真许多外围设备包括串口、外部 I/O 口及时钟等。所能仿真的外围设备在为目标程序选择 CPU 时就被选定了。在目标硬件准备好之前，可用这种方式测试和调试嵌入式应用程序。

GDI 驱动模式：使用高级 GDI 驱动设备连接目标硬件来进行调试，例如使用 ULINK Debugger。对 μVision 3 来说，可用于连接的驱动设备有：

- JTAG/OCDS 适配器：它连接到片上调试系统，例如 AMR Embedded ICE。
- Monitor(监视器)：它可以集成在用户硬件上、也可以用在许多评估板上。
- Emulator(仿真器)：它连接到目标硬件的 CPU 引脚上。
- In-System Debugger(系统内调试器)：它是用户应用程序的一部分，可以提供基本的测试功能。
- Test Hardware(测试硬件)：如 Philips SmartMX DBox 、 Infineon SmartCard ROM Monitor RM66P 等。


使用仿真器调试时，选择菜单项 Project->Project-Option for Target 或者直接单击 ，打开 Option for Target 对话框的 Debug 页，弹出如下对话框，可进行调试配置。



图 2-33 选择 ULINK USB-JTAG 仿真器调试

如果目标板已上电，并且与 ULINK USB-JTAG 仿真器连接上，点击图 2-33 中的“Settings”，将弹出如图 2-34 所示的对话框，正常则可读取目标板芯片 ID 号。如果读不出 ID 号，则需要检查 ULINK USB-JTAG 仿真器与 PC 或目标板的连接是否正确。

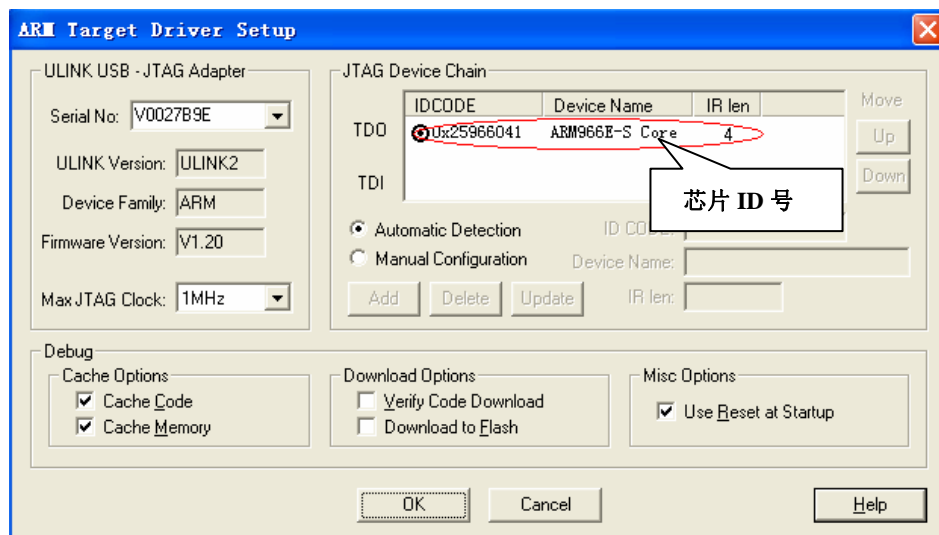


图 2-34 读取设备 ID

6. 编译配置

选择编译器：

μVision IDE 目前支持 RealView、Keil CARM 和 GNU 这三种编译器，从菜单栏的 Project->Manage->Component,Environment,Books...或者直接单击工具栏中的  图标，打开其 Folder/Extensions 页进入编译器选择界面。我们使用 RealView 编译器如图 2-35 所示。

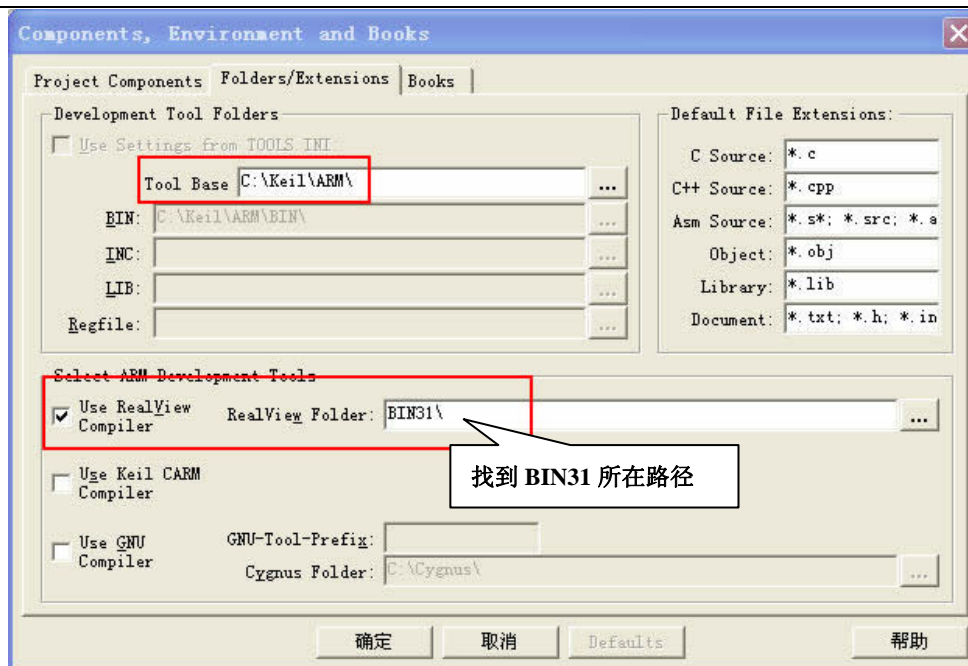



图 2-35 选择编译器

配置编译器：

选择好编译器后，单击图标，打开 Option for Target 对话框的 C/C++ 页，出现如图 2-36 的编译属性配置页面（这里主要说明 RealView 编译器的编译配置）：

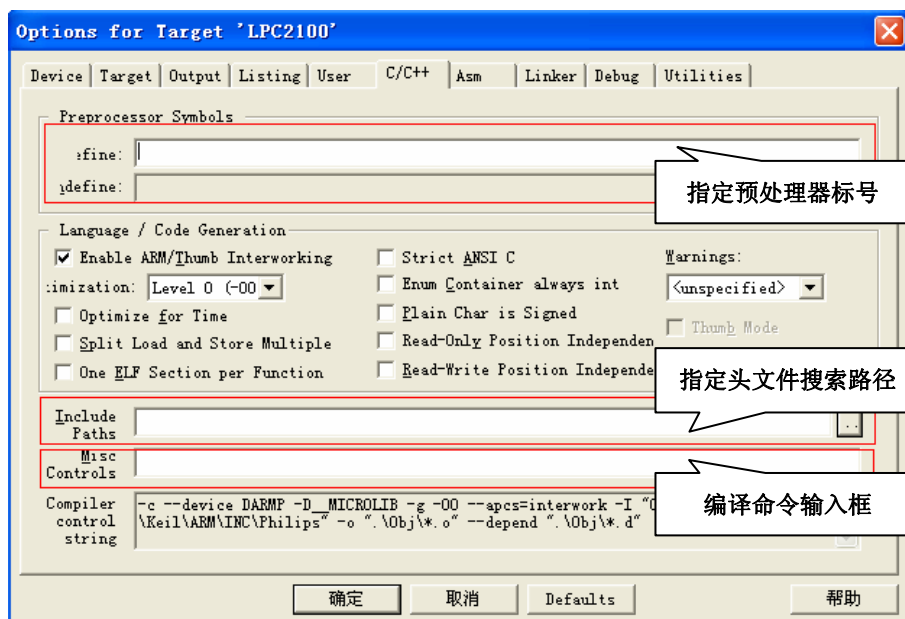


图 2-36 编译器配置页

各个编译选项说明如下：

Enable ARM/Thumb Interworking: 生成 ARM/Thumb 指令集的目标代码，支持两种指令之间的函数调用。

Optimization: 优化等级选项，分四个档次。

Optimize for Time: 时间优化。

Split Load and Store Multiple: 非对齐数据采用多次访问方式。

One ELF Section per Function: 每个函数设置一个 ELF 段。

Strict ANSI C: 编译标准 ANSI C 格式的源文件。

Enum Container always int: 枚举值用整型数表示。

Plain Char is Signed: Plain Char 类型用有符号字符表示。

Read-Only Position Independent: 段中代码和只读数据的地址在运行时候可以改变。

Read-Write Position Independent: 段中的可读/写的数据地址在运行期间可以改变。

Warning: 编译源文件时, 警告信息输出提示选项。

7. 汇编选项设置

单击图标, 打开 Option for Target 对话框的 Asm 页, 出现如图 2-37 的汇编属性配置页面:

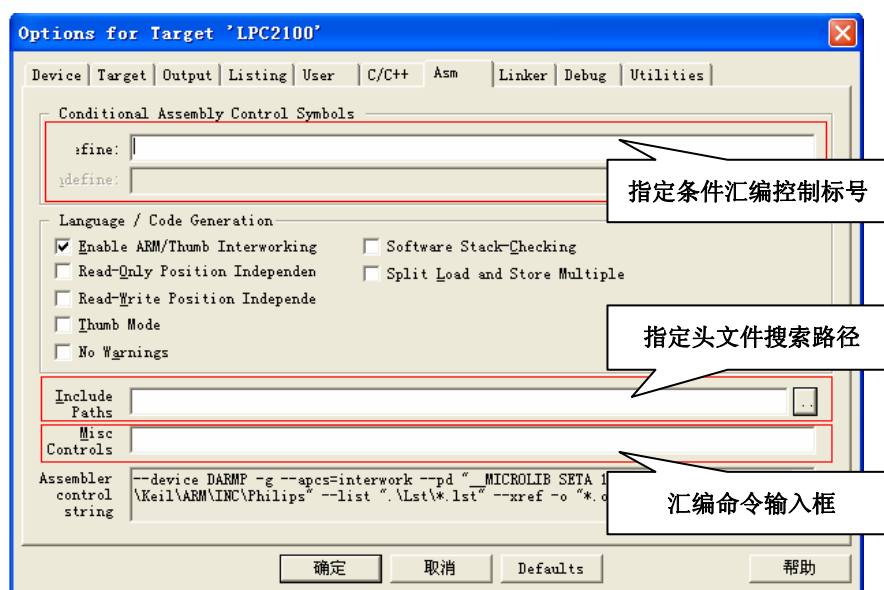


图 2-37 汇编配置页

各个汇编选项说明如下:

Enable ARM/Thumb Interworking: 生成 ARM/Thumb 指令集的目标代码, 支持两种指令之间的函数调用。

Read-Only Position Independent: 段中代码和只读数据的地址在运行时候可以改变。

Read-Write Position Independent: 段中的可读/写的数据地址在运行期间可以改变。


Thumb Mode: 只编译 THUMB 指令集的汇编源文件。

No Warnings: 不输出警告信息。

Software Stack-Checking: 软件堆栈检查。

Split Load and Store Multiple: 非对齐数据采用多次访问方式。

8. 链接选项设置

链接器/定位器用于将目标模块进行段合并, 并对其定位, 生成程序。既可通过命令行方式使用链接器, 也可在 μVision IDE 中使用链接器。单击图标, 打开 Option for Target 对话框的 Linker 页, 出现如图 2-38 的链接属性配置页面:

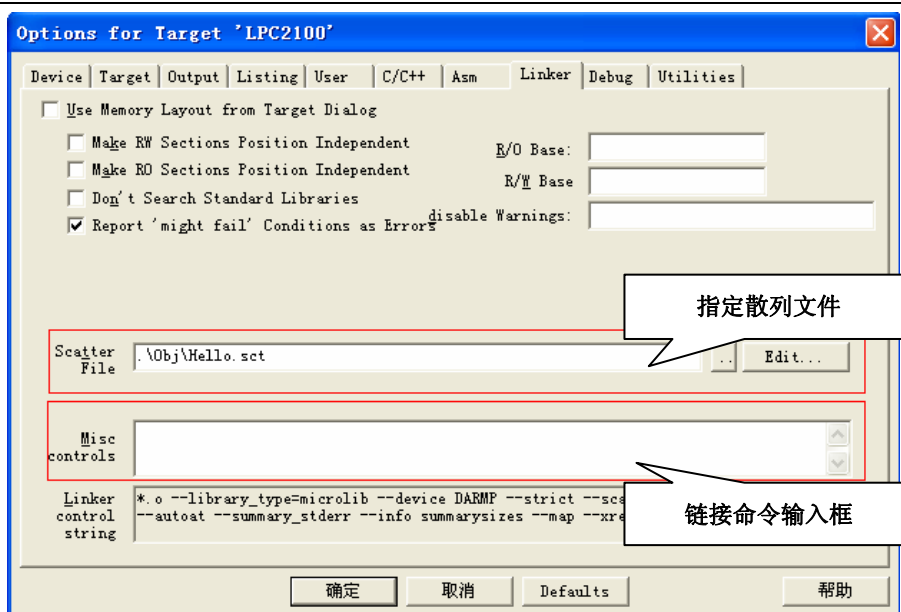


图 2-38 链接配置页

各个链接选项配置说明如下：

Make RW Sections Position Independent: RW 段运行时可改变。

Make RO Sections Position Independent: RO 段运行时可改变。

Don't search Standard Libraries: 链接时不搜索标准库。

Report 'might fail' Conditions as Err: 将'might fail'报告为错误提示输出。

R/O Base: R/O 段起始地址输入框。

R/W Base: R/W 段起始地址输入框。

9. 输出文件设置

在 Project->Option for Target 的 Output 页中配置输出文件，如图 2-39 所示。

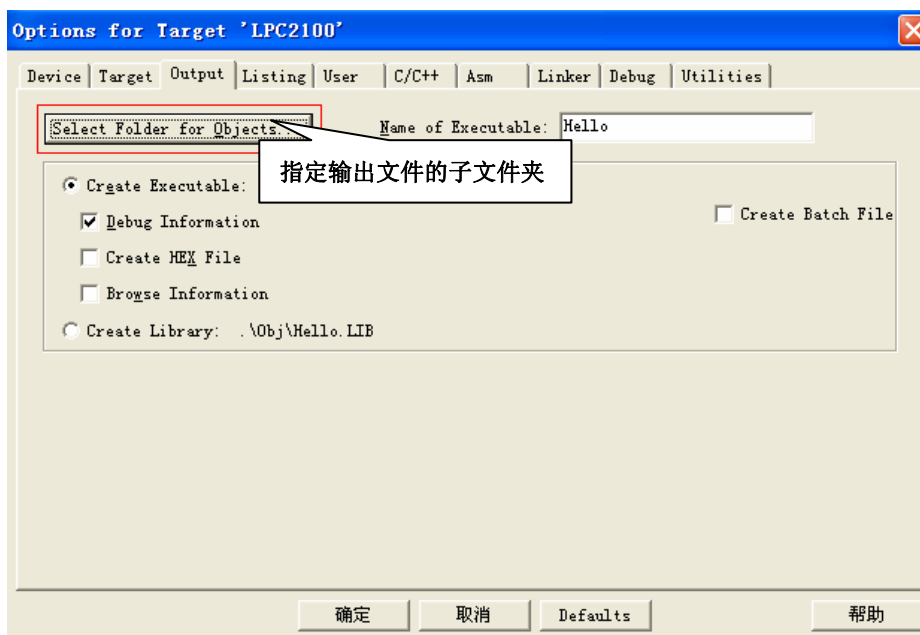


图 2-39 输出文件配置页

输出文件配置选项说明如下

Name of Executable: 指定输出文件名。


Debug Information: 允许时, 在可执行文件内存储符号的调试信息。

Create HEX File: 允许时, 使用外部程序生成一个 HEX 文件进行 Flash 编程。

Big Endian: 输出文件采用大端对齐方式。

Create Batch File: 创建批文件

2.3.4 工程的编译链接

完成工程的设置后, 就可以对工程进行编译链接了。用户可以通过选择主窗口 Project 菜单的 Build target 项或工具条  按钮, 编译相应的文件或工程, 同时将在输出窗的 Build 子窗口中输出有关信息。如果在编译链接过程中, 出现任何错误, 包括源文件语法错误和其它错误时, 编译链接操作立刻终止, 并在输出窗的 Build 子窗口中提示错误, 如果是语法错误, 用户可以通过鼠标左键双击错误提示行, 来定位引起错误的源文件行。

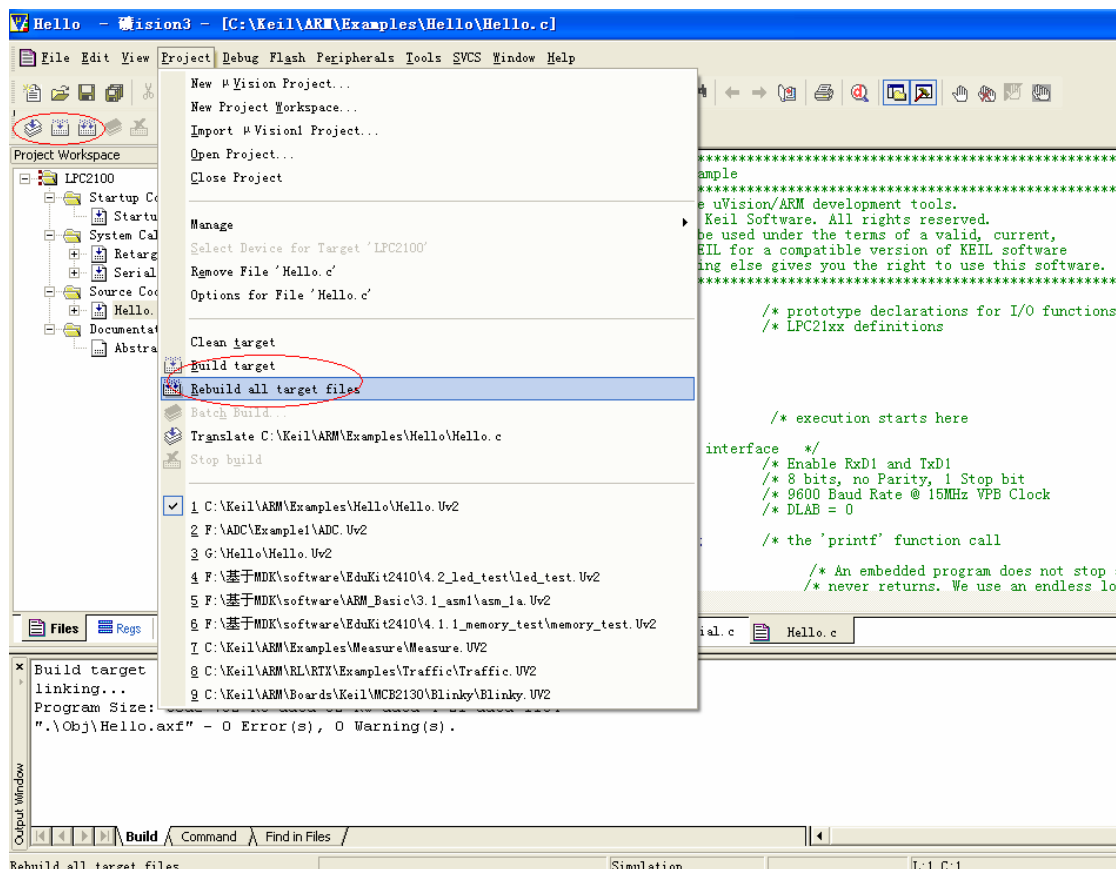


图 2-40 工程 Project 菜单和工具条

2.3.5 加载调试

μVision 3 调试器提供了软件仿真和 GDI 驱动两种调试模式, 采用 ULINK 仿真器调试时, 首先将集成环境与 ULINK 仿真器连接, 按照前面 2.4.3 小节中的工程配置方法对要调试的工程进行配置后, 点击 Flash->Download 菜单项可将目标文件下载到目标系统的指定存储区中, 文件下载后即可进行在线仿真调试。

1. 断点和单步

调试器可以控制目标程序的运行和停止，并反汇编正在调试的二进制代码，同时可通过设置断点来控制程序的运行，辅助用户更快的调试目标程序。μVision IDE 的调试器可以在源程序、反汇编程序、以及源程序汇编程序混合模式窗口中设置和删除断点。在 μVision3 中设置断点的方式非常灵活，甚至可以在程序代码被编译前

在源程序中设置断点。定义和修改断点的方式有如下几种：

使用文件工具栏，只要在编辑窗口或反汇编窗口中选中要插入断点的行，然后再单击工具栏上的按钮就可以定义或修改断点；

使用快捷菜单上的断点命令，在编辑窗口或反汇编窗口中单击右键即可打开快捷菜单；

在 Debug-Breakpoints...对话框中，可以查看、定义、修改断点。这个对话框可以定义及访问不同属性的断点；

在 Output Window-Command 页中，使用 BreakSet、BreakKill、BreakList、BreakEnable、BreakDisable 命令对断点进行管理。

在断点对话框中可以查看及修改断点，如图 2-41 所示。可以在 Current Breakpoints 列表中通过单击复选框来快捷地 Disable 或 Enable 一个断点。在 Current Breakpoints 列表中双击可以修改选定的断点。

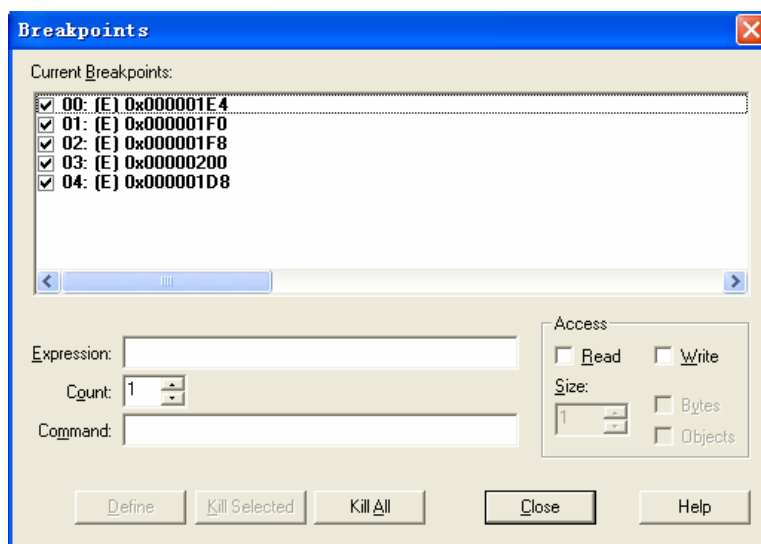


图 2-41 断点对话框

如图 2-41 所示，可以在断点对话框表达式的文本框中输入一个表达式来定义断点。根据表达式的类型可以定义如下类型的断点：

当表达式是代码地址时，一个类型为 Execution Break(E)的断点被定义，当执行到指定的代码地址时，此断点有效。输入的代码地址要参考每条 CPU 指令的第一个字节；

当对话框中一个内存访问类型（可读、可写或既可读又可写）被选中时，那么将会定义一个类型为 Access Break(A)的断点。当指定的内存访问发生时，此断点有效。可以字节方式指定内存访问的范围，也可以指定表达式的目标范围。Access Break 类型的表达式必须能转化为内存地址及内存类型。在 Access Break 类型的断点停止程序执行或执行命令之前操作符(&, &&, <, <=, >, >=, =, !=)可用于比较变量的值；

当表达式不能转换为内存地址时，一个 Conditional Break(C)类型的断点将被定义，当指定的条件表达式为真时，此断点有效。在每个 CPU 指令后，均需要重新计算表达式的值，因此，程序执行速度会明显降低。

在 command 文本框中可以为断点指定一条命令，程序执行到断点时将执行该命令，μVision 3 执行命令后会继续执行目标程序。在此指定的命令可以是 μVision 3 的调试命令或信号函数。μVision 3 中，可以使用系统变量 _break_ 来停止程序的执行。Count 的值用于指定断点触发前断点表达式为真的次数。

2. 反汇编窗

反汇编窗用于显示反汇编二进制代码后得到的汇编级代码，可以混合源代码显示，也可以混合二进制代码显示。反汇编窗可以设置和清除汇编级别断点，并可按照 ARM 或 THUMB 格式的反汇编二进制代码。

如图 2-42 所示，反汇编窗口可用于将源程序和反汇编程序一起显示也可以只显示反汇编程序。通过 Debug -> View Trace Records 可以查看前面指令的执行记录。为了实现这一功能，需要设置 Debug -> Enable/Disable Trace Recording。

若选择反汇编窗口作为当前窗口，那么程序的执行是以 CPU 指令为单位的而不是以源程序中的行为单位的。可以用工具条上的按钮或快捷菜单命令为选中的行设置断点或对断点进行修。还可以使用对话框 Debug -> Inline Assembly...来修改 CPU 指令，它允许设计这对调试的目标程序进行临时修改。

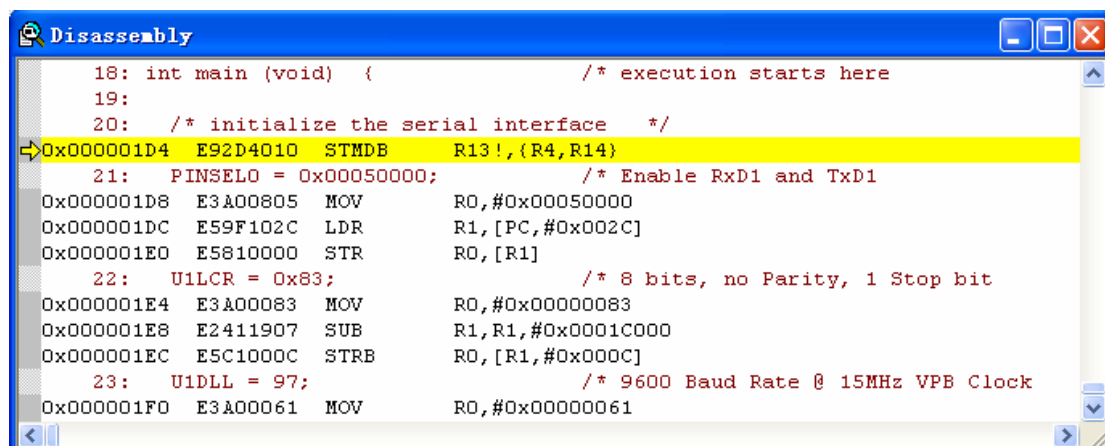


图 2-42 源文件与反汇编指令交叉显示窗口

3. 寄存器窗

在 Project Workspace - Regs 页中列出了 CPU 的所有寄存器，按模式排列共有八组，分别为 Current 模式寄存器组、User/System 模式寄存器组、Fast Interrupt 模式寄存器组、Interrupt 模式寄存器组、Supervisor 模式寄存器组、Abort 模式寄存器组、Undefined 模式寄存器组以及 Internal 模式寄存器组，如图 2-43 所示。在每个寄存器组中又分别有相应的寄存器。在调试过程中，值发生变化的寄存器将会以蓝色显示。选中指定寄存器单击或按 F2 键便可以出现一个编辑框，从而可以改变此寄存器的值。

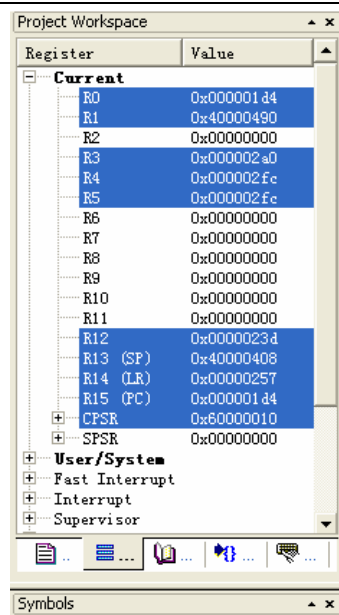


图 2-43 Regs 页

4. 存储区窗

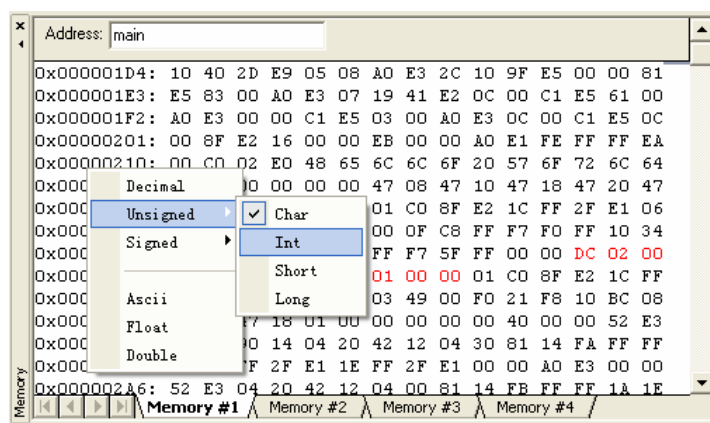


图 2-44 内存窗口

通过内存窗口可以查看与显示存储情况，View-Memory Window 可以打开存储器窗口，如图 2-44 所示。 μ Vision 3 可仿真了高达 4GB 的存储空间，这些空间可以通过 MAP 命令或 Debug -Memory Map 打开内存映射对话框来映射为可读的、可写的、可执行的，如图 2-45。 μ Vision 3 能够检查并报告非法的存储访问。

从图 2-44 中可看出内存窗口有四个 Memory 页，分别为 Memory#1、Memory#2、Memory#3、Memory#4，即可同时显示四个指定存储区域的内容。在 Address 域内，输入地址即可显示相应地址中的内容。需要说明的是，它支持表达式输入，只要这个表达式代表了某个区域的地址即可，例如图 2-44 中所示的 main。双击指定地址处会出现编辑框，可以改变相应地址处的值。在存储区内单击右键可以打开如图 2-44 所示的快捷菜单，在此可以选择输出格式。通过选中 View -> Periodic Window Update，可以在运行时实时更新此内存窗口中的值。在运行过程中，若某些地址处的值发生变化，将会以红色显示。

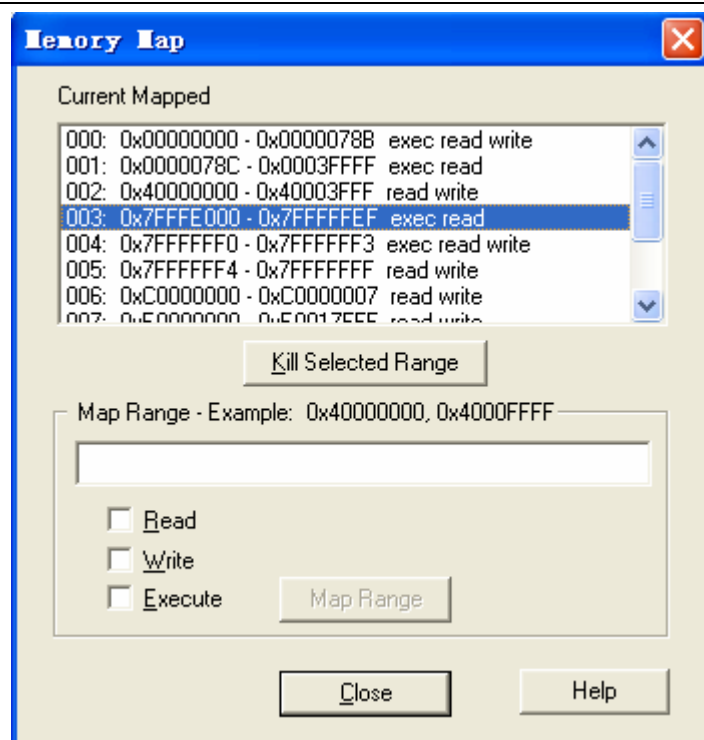


图 2-45 内存映射对话框

内存映射对话框可以用来设定哪些地址空间用于存储数据、哪些地址空间用于存储程序。也可以用 MAP 命令来完成上述工作。在载入目标应用时，μVision 3 自动地对应用进行地址映射，一般不需要映射额外的地址空间，但被访问的地址空间没有被明确声明时必须进行地址映射，如存储映射 I/O 空间。如上图所示，每一个存储空间均可指定为可读、可写、可执行，若在编辑框内输入“MAP 0X0C000000,0X0E000000 READ WRITE EXEC”，此命令就是将从 0X0C000000 到 0X0E000000 这部分区域映射为可读的、可写的、可执行的。在目标程序运行期间，μVision 3 使用存储映射来保证程序没有访问非法的存储区。

5. 观测窗口

观测窗口（Watch Windows）用于查看和修改程序中变量的值，并列出了当前的函数调用关系。在程序运行结束之后，观测窗口中的内容将自动更新。也可通过菜单 View - Periodic Window Update 设置来实现程序运行时实时更新变量的值。观测窗口共包含四个页：Locals 页、Watch #1 页、Watch #2 页、Call Stack 页，分别介绍如下。

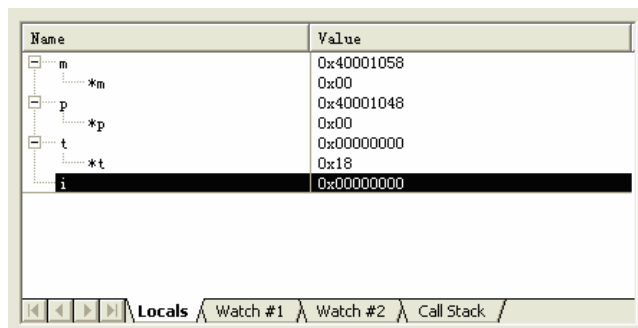


图 2-46 Watch 窗口之 Locals 页

Locals 页：如图 2-46 所示，此页列出了程序中当前函数中全部的局部变量。要修改某个变量的值，只需选中变量的值，然后单击或按 F2 即可弹出一个文本框来修改该变量的值。

Watch 页：如图 2-47 所示，观测窗口有 2 个 Watch 页，此页列出了用户指定的程序变量。有三种方式可以把程序变量加到 Watch 页中：

- 在 Watch 页中，选中<type F2 to edit>，然后按 F2，会出现一个文本框，在此输入要添加的变量名即可，用同样的方法，可以修改已存在的变量；
- 在工作空间中，选中要添加到 Watch 页中的变量，右击会出现快捷菜单，在快捷菜单中选择 Add to Watch Window，即可把选定的变量添加到 Watch 页中；
- 在 Output Window 窗口的 Command 页中，用 WS(WatchSet) 命令将所要添加的变量添入 Watch 页中。

若要修改某个变量的值，只需选中变量的值，再单击或按 F2 即可出现一个文本框修改该变量的值。若要删除变量，只需选中变量，按 Delete 键或在 Output Window 窗口的 Command 页中用 WK(WatchKill) 命令就可以删除变量。

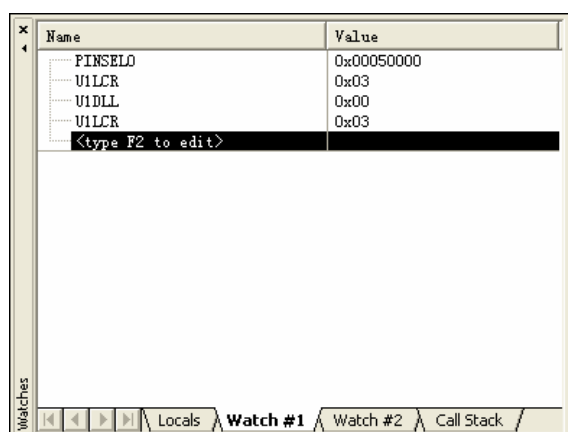


图 2-47 Watch 窗口之 Watch 页

Call Stack：如图 2-48 所示，此页显示了函数的调用关系。双击此页中的某行，将会在工作区中显示该行对应的调用函数以及相应的运行地址。

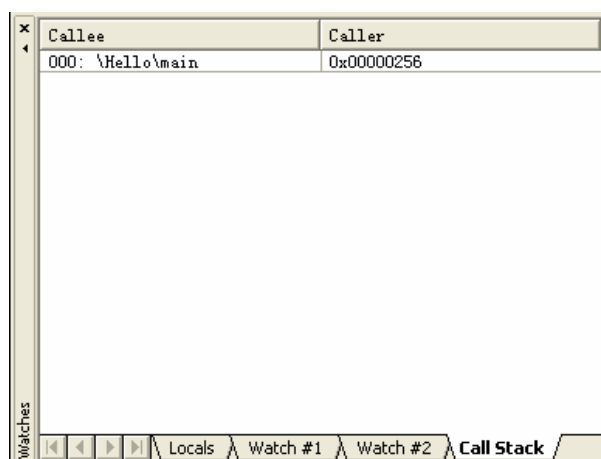


图 2-48 Watch 窗口之 Call Stack 页

6. 代码统计对话框

μVision 3 提供了一个统计代码 (Code Coverage) 执行情况的功能, 这个功能以代码统计对话框的形式表示出来, 如图 2-51 所示。在调试窗口中, 已执行的代码行在左侧以绿色标出。当测试嵌入式应用程序时, 可以用此功能来查看那些程序还没有被执行。

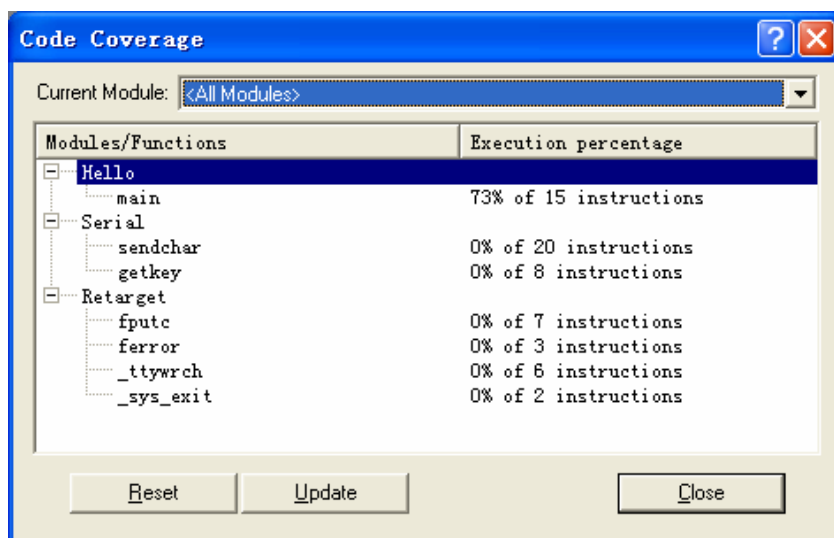


图 2-49 代码统计对话框

如图 2-51 所示代码统计对话框提供了程序中各个模块及函数的执行情况。在 Current Module 下拉列表框中列出了程序所有要模块, 而在下面的则显示了相应模块中指令的执行情况, 即每个模块或函数的指令执行百分比, 只要是执行了的部分均以绿色标出。在 Output Window – Command 页中可以用 COVERAGE 调试命令将此信息输出到输出窗口中。

7. 执行剖析器

μVision ARM 仿真器包含一个执行剖析器, 它可以记录执行全部程序代码所需的时间。可以通过选中 Debug - Execution Profiling 来使能此功能。它具有两种显示方式: Call (显示执行次数) 和 Time (显示执行时间)。将鼠标放在指定的入口处, 即可显示有关执行时间及次数的详细信息。如图 2-50 所示。

对 C 源文件, 可能使用编辑器的源文件的大纲视图特性, 用此特性可以将几行源文件代码收缩为一行, 以此可查看某源文件块的执行时间。

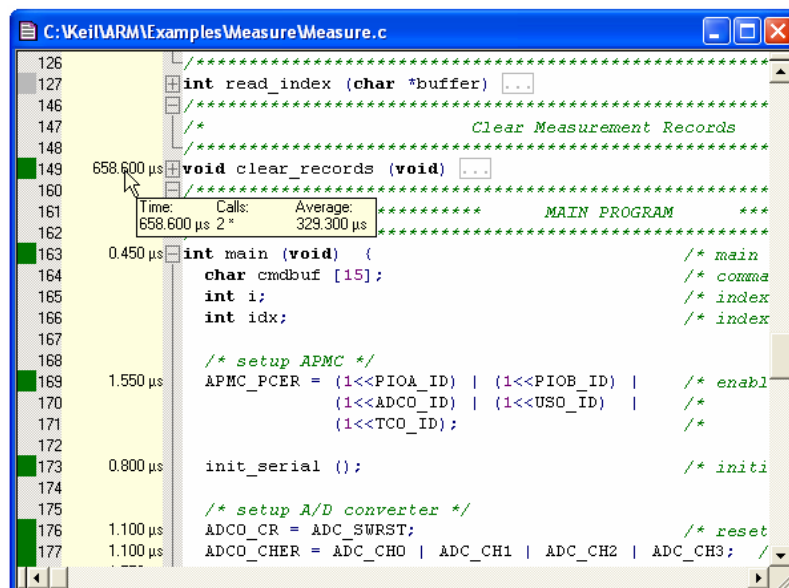


图 2-50 执行剖析器

在反汇编窗口中，可以显示每条汇编指令的执行信息，如图 2-51 所示。

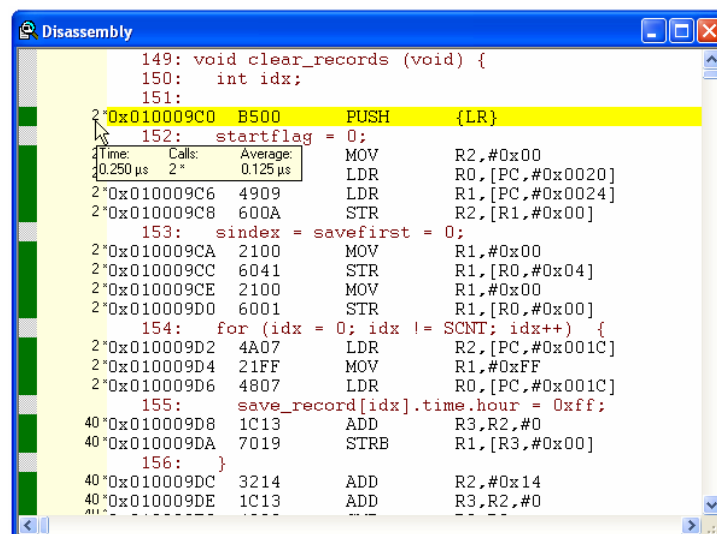


图 2-51 反汇编窗口

需要注意：执行剖析器得到的执行时间是基于当前的时钟设置，当代码以不同的时钟执行多次时，可能会得到一个错误的执行时间。另外，目前执行剖析器仅能用于 ARM 仿真器。

8. 性能分析仪

μVision 3 ARM 仿真器的执行剖析器能够显示已知地址区域的执行统计的信息。对没有调试信息的地址区域，显示列表中是不会显示这块区域的执行情况，例如 ARM ADS/RealView 工具集的浮点库。

μVision 3 性能分析仪则可用于显示整个模块的执行时间及各个模块被调用的次数。μVision 3 的仿真器可以记录整个程序代码的执行时间及函数调用情况，如图 2-52 所示。

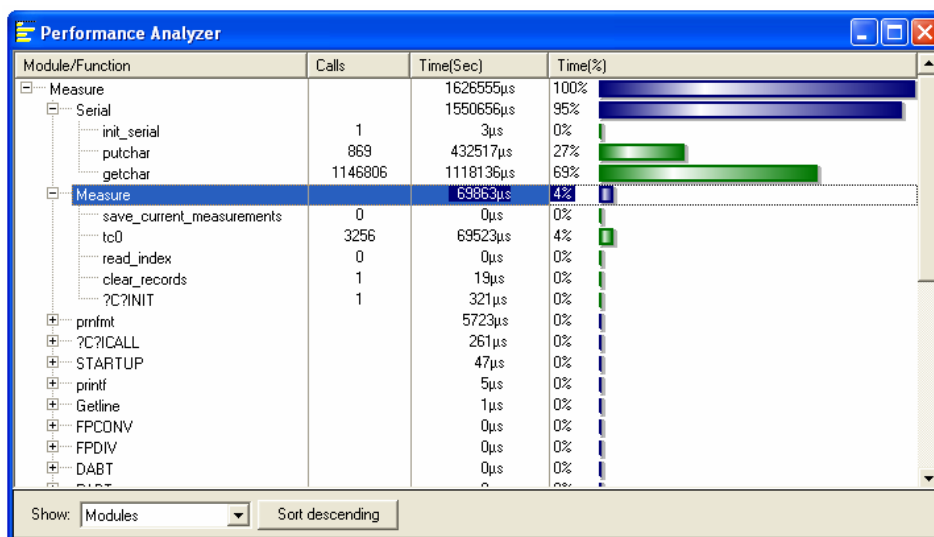


图 2-52 性能分析仪

图 2-52 中的 Show 下拉框用于选择以模块或函数的形式进行显示。Sort descending 按钮则用于以降序来排列各模块或函数的执行时间。表头各项含义分别为：Module/Funcation 是模块或函数名；Calls 是函数的调用次数；Time(Sec)是花费在函数或模块区域内的执行时间；Time (%) 是花费在函数或模块区域内的时间百分比。

9. 串行窗口

μVision 3 提供了两个串行窗口用于串行输入及输出，如图 2-53 所示。从被仿真的处理器所输出的数据会在此窗口中显示，在此窗口中输入的字符也会被输入到被仿真的 CPU 中。

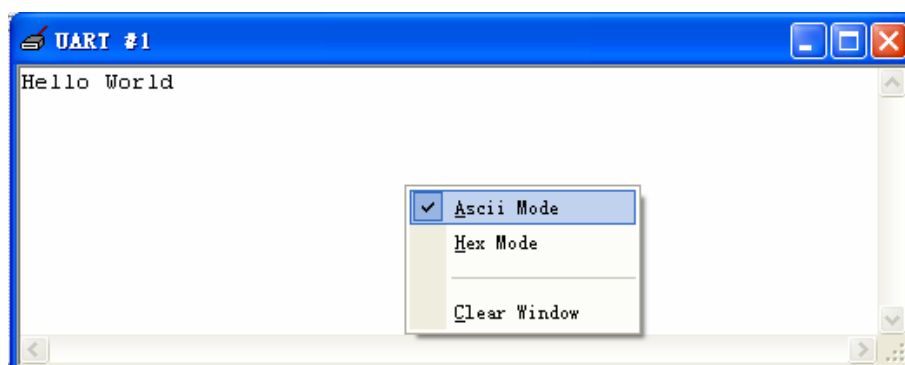


图 2-53 串行窗口

利用串行窗口，在不需要外部硬件的情况下也可以仿真 CPU 的 UART。在 Output Window – Command 页中使用 ASSIGN 命令也可以将串口输出指定为 PC 的 COM 口。

10. 工具箱

如图 2-54 所示，工具箱中包含用户可配置的按钮，单击工具箱上的按钮可以执行相关的调试命令或调试函数。工具箱按钮可以在任何时间执行，甚至是运行测试程序时。

在 Output Window-Command 页中用 DEFINE BUTTON 命令可定义工具箱按钮，语法格式：

```
>DEFINE BUTTON "button_label", "command"
```

其中，button_label 是显示在工具箱按钮上的名字；command 是按下此按钮被时要执行的命令。

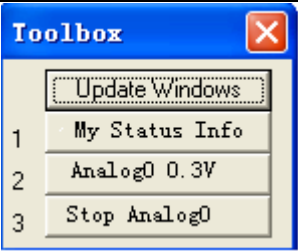


图 2-54 工具箱

下面的例子说明了如何使用命令来定义图 2-54 所示的工具箱按钮：

```
>DEFINE BUTTON "Decimal Output", "radix=0x0A">DEFINE BUTTON "Hex Output", "radix=0x10"

>DEFINE BUTTON "My Status Info", "MyStatus ()" /* call debug function */
>DEFINE BUTTON "Analog0..5V", "analog0 ()" /* call signal function */
>DEFINE BUTTON "Show R15", "printf (\\"R15=%04XH\\n\\")"
```

11. 输出窗口调试命令对话框

通过在 Output Window – Command 页中键入命令，可以交互的方面使用 μVision 3 调试器。此窗口还能提供一般的调试输出信息，并允许键入用于查看或修改变量和寄存器的表达式（expressions），也可用于调用调试函数（debug functions）。图 2-55 为输出窗口命令对话框。

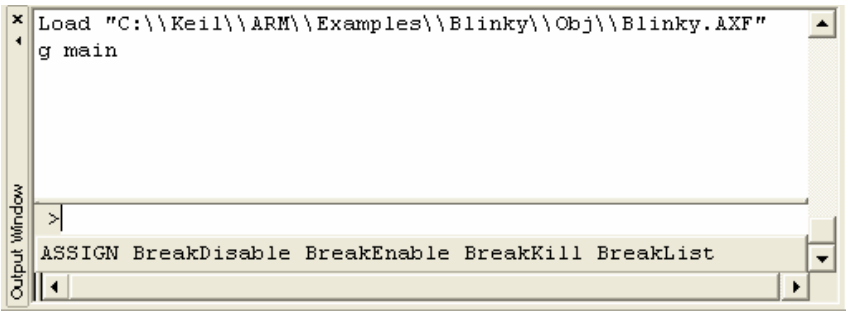


图 2-55 输出窗口命令对话框

调试命令 在调试窗口的“>”提示符后可以输入调试命令，仅用首字母来键入命令，例如 WatchSet 命令仅需要键入 WS。

还可以在命令窗口中显示和改变变量，寄存器和存储位置，例如，可以在命令提示符中输入如表 2-2 所示的文本命令。

表 2-2 利用命令修改变量及寄存器

| 命令 | 结果 |
|-------------|-----------------|
| R7 = 12 | 为寄存器 R7 分配值 12. |
| CPSR | 显示寄存器 CPSR 的值. |
| time.hour | 显示时间结构体的成员：小时. |
| time.hour++ | 时间结构体的成员小时递增. |

index = 0 为 index 分配值 0.

调试函数

还可以在命令提示符处输入调试函数来进行程序调试，例如：

ListInfo (2)

在命令键入处，有语法生成器可以帮助显示命令、选项以及参数。随着命令的键入，μVision 3 会自动减少所列出的命令以与所键入的字符相匹配。例如图 2-56 所示。

若键入 B,语法生成器会减少所列出的命令。

```
>B
BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

若命令确定的话可用的命令选项会被列出。

```
>BS|
BreakSet
[Navigation Icons] Build Command Find in Files /
```

语法生成器指导进入命令并避免错误。

```
>BS WRITE saverecords[0],1,"_break_1"
"<command>" <cr>
[Navigation Icons] Build Command Find in Files /
```

图 2-56 命令输入的语法提示

12. 符号窗

在符号窗口 View - Symbol Window 中显示了定义在当前被载入的应用程序中的公有符号、局部符号及行号信息。CPU 特殊功能寄存器 SFR 符号也显示在此窗口中，如图 2-57 所示。

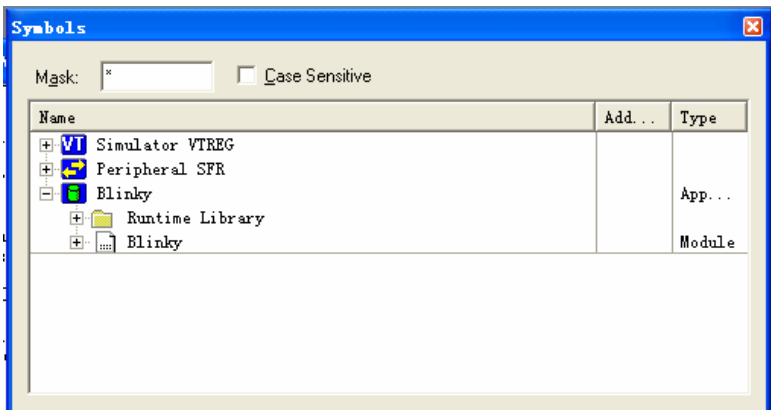


图 2-57 符号窗口

可以选择符号类型并用符号窗口中的选项过滤信息，如表 2-3 所示。

表 2-3 符号窗口各选项含义

| 选项 | 描述 |
|----------------|--|
| Mode | 选择 PUBLIC、LOCALS 或 LINE。公有 PUBLIC 符号的作用域是整个应用程序；局部 LOCALS 函数的作用被限制在一个模块或函数中；行 LINE 是与源文本中的行号信息相关的。 |
| Current Module | 选择其信息应该被显示的源模块。 |
| Mask | 指定一个通配字符串以用于匹配符号名。通配字符串由文字数字符及通配字符组成： <div><div>#</div><div>匹配一位数字 (0 – 9)</div></div> |

| | | |
|-------|--------------------|-------------|
| | \$ | 匹配任意字符 |
| | * | 匹配 0 个或多个字符 |
| Apply | 应用 mask，并显示更新的符号列表 | |

表 2-4 通配字符用法

| 通配字符 | 匹配符号名 ... |
|--------|--|
| * | 匹配任意符号。这是符号浏览器中的默认掩码。 |
| *** | 匹配在任意位置处包含一个数字位的符号 |
| _a\$#* | 以一个下划线开始，后面是个字母 a，再后面是任意个字符，再后面是一个数字位，以 0 个或多个字符结束，例如 _ab1 or _a10value. |
| _*ABC | 以一个下划线开始，后面是 0 个或多个字符，以 ABC 结束。 |

2.3.6 Flash 编程工具

μVision 3 集成了 Flash 编程工具，所有的相关配置将被保存在当前工程中。在菜单项 Project-Options-Utilities 下可配置当前工程所使用的 Flash 编程工具，开发人员既可使用外部的命令行驱动工具（通常由芯片销售商提供），也可使用 Keil ULINK USB-JTAG 适配器等工具。

用户可通过 Flash 菜单启动 Flash 编程器，若设置了 Project-Options-Utilities-Update Target before Debugging，那么在调试器启动之前 Falsh 编程器也将启动。

μVision 3 为 Flash 编程工具提供了一个命令接口，在 Project-Option for Target 对话框的 Utilities 页中可配置 Flash 编程器，通过菜单项 Flash-Configure Flash Tools 也可进入此对话框，如图 2-58 所示。一旦配置好了命令接口方式，就可以通过 Flash 菜单下载(Download)或擦除(Erase)目标板中 Flash 存储器的内容。

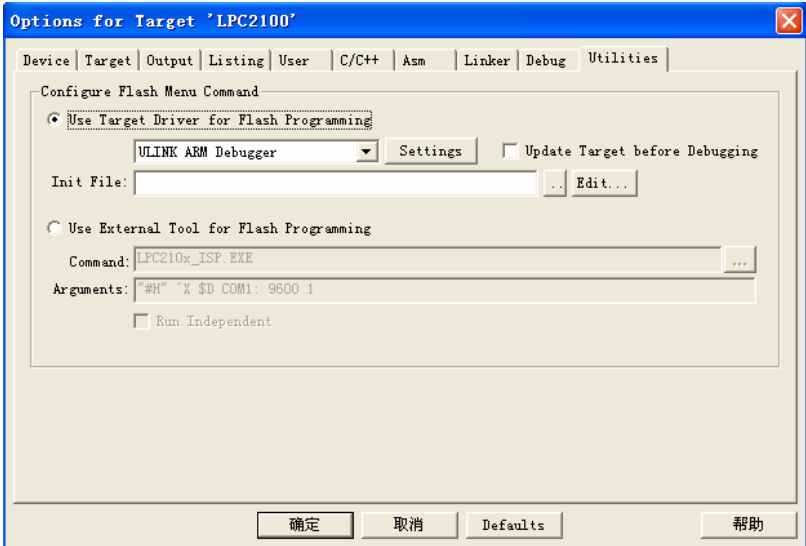


图 2-58 Flash 编程器的配置对话框

μVision 3 提供了两种 Flash 编程的方法：目标板驱动和外部工具。

- 目标板驱动:

μVision 3 提供了 3 种 Flash 编程驱动, ULINK ARM Debugger、ULINK Cortex-M3 Debugger 及 RDI Interface Driver。选择一个 Flash 编程驱动程序, 单击右边的 Settings 按钮, 弹出图 2-59 所示的对话框(根据选择的驱动程序不同, 弹出的对话框略有差异)。最右边复选框决定是否在调试前更新目标板中 Flash 的内容。Init File 文本框中的初始化文件, 包括总线的配置、附加程序的下载及发或调试函数。对大多数 Flash 芯片而言, μVision 3 的设备数据库已经提供了片上 Flash Rom 的正确配置, 例如图 2-59 所示的 LPC2104 Flash。

● 外部工具:

使用第三方的基于命令行的 Flash 编程工具。通过命令行及参数调用下载工具, 如图 2-60 中使用的是 LPC210x_ISP.exe 在线编程器。使用键码序列(Key Sequences)指定输出文件名、设备名及 Flash 编程器的时钟频率等。设置 Run Independent 复选框决定编程工具是否能独立运行。

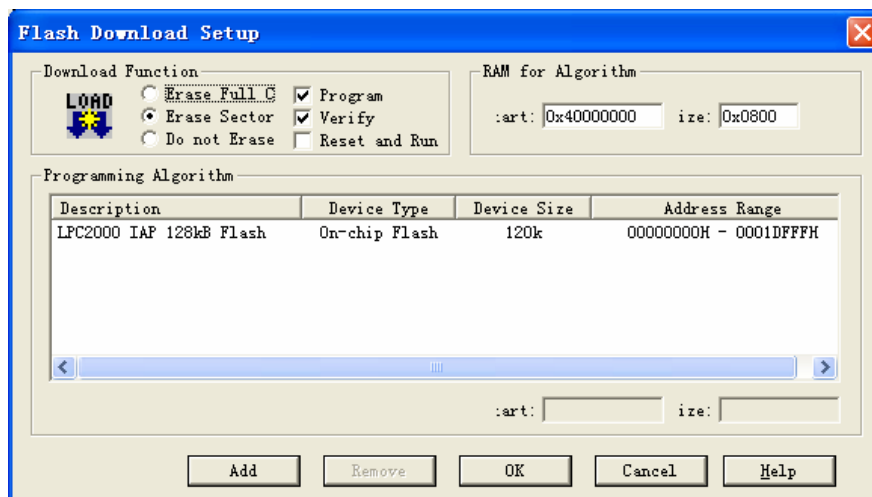


图 2-59 ULINK ARM Debugger 的 Flash 下载设置对话框

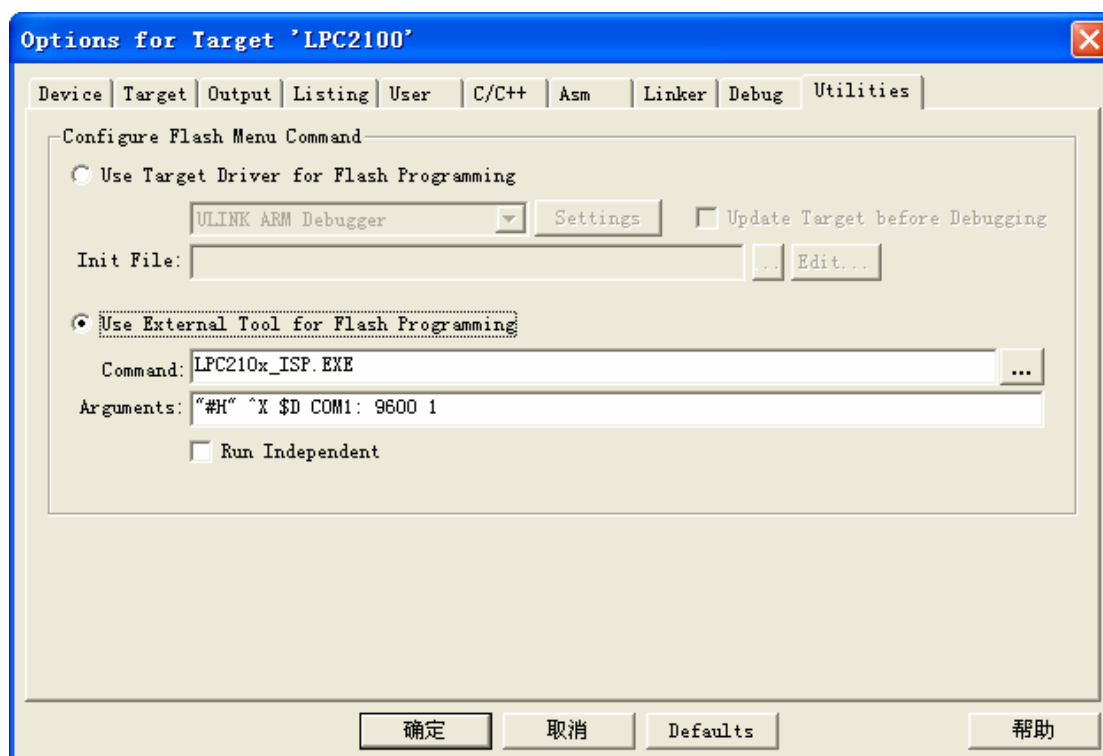


图 2-60 使用外部 Flash 编程工具

第三章 嵌入式软件开发基础实验

本章将介绍几个简单的基础开发实例，既能帮助读者掌握 MDK 嵌入式软件的基本开发过程，同时也能让读者了解 ARM 处理器的基本结构、指令集、存储系统以及基本接口编程。

本书所有的实例均在 Embest EduKit-III（选用 S3C44B0 处理器子板）教学实验平台上运行通过。Embest EduKit-III 是一款功能强大的 32 位的嵌入式开发板，可任意选用以下处理器子板：ARM7 的三星的 S3C44BOX、ARM9 的三星的 S3C2410A 芯片、ARM10 的 Intel 的 XScale 芯片和 ADI 的 Blackfin533 芯片。该实验板除了提供键盘、LED、LCD、触摸屏和串口等一些常用的功能模块外，还具有 IDE 硬件接口、PCI 接口、CF 存储卡接口、以太网接口、USB 接口、IrDA 接口、II S 接口、SD 卡接口以及步进电机等丰富的接口模块，并可根据用户要求扩展 GPRS 等模块。Embest EduKit-III 实验平台能给用户在 32 位 ARM 嵌入式领域进行实验和开发提供丰富的支持和极大的便利。

3.1 ARM 汇编指令实验一

3.1.1 实验目的

- 初步学会使用 μ Vision3 IDE for ARM 开发环境及 ARM 软件模拟器；
- 通过实验掌握简单 ARM 汇编指令的使用方法。

3.1.2 实验设备

- 硬件：PC 机。
- 软件： μ Vision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.1.3 实验内容

- 熟悉开发环境的使用并使用 ldr/str, mov 等指令访问寄存器或存储单元；
- 使用 add/sub/lsl/lsr/and/orr 等指令，完成基本算术/逻辑运算。

3.1.4 实验原理

ARM 处理器共有 37 个寄存器：

- 31 个通用寄存器，包括程序计数器(PC)。这些寄存器都是 32 位的；
- 6 个状态寄存器。这些寄存器也是 32 位的，但是只是使用了其中的 12 位。

这里简要介绍通用寄存器，关于状态寄存器的介绍，请参照下一节。

3.1.4.1 ARM 通用寄存器

通用寄存器（R0-R15）可分为三类：

- 不分组寄存器 R0~R7；
- 分组寄存器 R8~R14；
- 程序计数器 R15。

(1) 不分组寄存器 R0~R7

不分组寄存器 R0~R7 在所有处理器模式下，它们每一个都访问一样的 32 位寄存器。它们是真正的通用寄存器，没有体系结构所隐含的特殊用途。

(2) 分组寄存器 R8~R14

分组寄存器 R8~R14 对应的物理寄存器取决于当前的处理器模式。若要访问特定的物理寄存器而不依赖当前的处理器模式，则使用规定的名字。

寄存器 R8~R12 各有两组物理寄存器：一组为 FIQ 模式，另一组为除了 FIQ 以外的所有模式。寄存器 R8~R12 没有任何指定的特殊用途，只是在作快速中断处理时使用。寄存器 R13, R14 各对应 6 个分组的物理寄存器，1 个用于用户模式和系统模式，其它 5 个分别用于 5 种异常模式。寄存器 R13 通常用做堆栈指针，称为 SP；寄存器 R14 用作子程序链接寄存器，也称为 LR。

(3) 程序计数器 PC

寄存器 R15 用做程序计数器 (PC)。

在本实验中，ARM 核工作在用户模式，R0~R15 可用。

3.1.4.2 存储器格式

ARM 体系结构将存储器看作是从零地址开始的字节的线性组合。字节零到字节三放置第一个字 (WORD)，字节四到字节七存储第二个字，以此类推。

ARM 体系结构可以用两种方法存储字数据，分别称为大端格式和小端格式。

● 大端格式

在这种格式中，字数据的高位字节存储在低地址中，而字数据的低位字节则存放在高地址中，如图 3-1 所示。

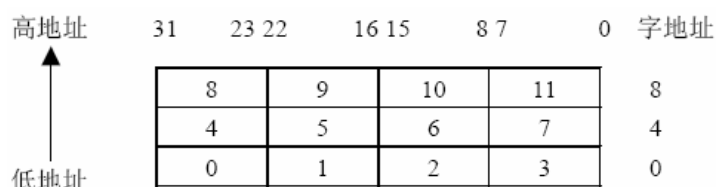


图 3-1 大端格式

● 小端格式

在这种格式中，字数据的高位字节存储在高地址中，而字数据的低位字节则存放在低地址中，如图 3-2 所示。

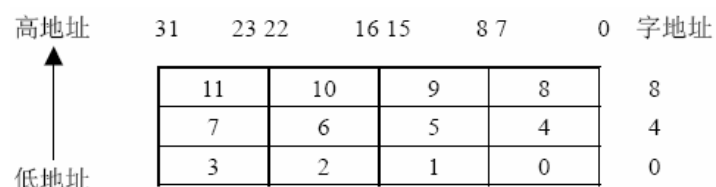


图 3-2 小端格式

3.1.4.4 REALVIEW 基础知识

μVision3 IDE 集成了 REALVIEW 汇编器 AARM、编译器 CARM、链接器 LARM，若采用 GNU 编译器则需要下载安装相应的工具包。本书所有例程代码均按照 REALVIEW 的语法和规则来书写。关于 AARM、CARM 和 LARM 的规范和具体使用，可参照 μVision3 IDE 所带的帮助文档，在此不再赘述。这里简单介绍几个相关基本知识：

● ENTRY

设置程序默认入口点，一个程序可有多多个 ENTRY，但一个源文件最多只有一个 ENTRY。

● EQU

EQU 伪操用于将数字常量、基于寄存器的值和程序中的标号定义为一个字符名称。语法格式：

symbol EQU expression

其中, expression 可以是一个寄存器的名字, 也可由程序标号、常量或者 32 位的地址常量组成的表达式。symbol 是 EQU 伪操作所定义的字符名称。示例: COUNT EQU 0X1FFF

- EXTERN/IMPORT

IMPORT (EXTERN 功能完全相同)用于声明在其他模块中定义但需要在本文中使用的符号。EXTRN 声明的变量必须是在其他模块中用 EXPORT 或 GLOBAL 声明过的。语法格式:

```
IMPORT class (symbol, symbol ...)
```

其中, class 为变量的类型, 可以为 ARM、CODE16、CODE32、DATA、CONST、THUMB; symbol 为所声明的变量名。

- EXPORT/GLOBAL

EXPORT (GLOBAL 功能完全相同) 用于声明在本文件中定义但能在其他模块中使用的变量, 相当于定义了一个全局变量。语法格式:

```
EXPORT symbol, symbol...
```

其中, symbol 为所声明的变量名。

- AREA

AREA 用于定义一个代码段或数据段, ARM 汇编程序设计采用分段式设计, 一个 ARM 源程序至少有一个代码段, 大的程序会有若干个代码段和数据段。语法格式:

```
AREA segment-name, class-name, attributes ,...
```

其中, segment-name 为所定义段的名称; class-name 为所定义段的类型名称, 可以为系统类型 (CODE, CONST, DATA, ERAM) 或用户定义类型; attributes 为段的属性。

- END

END 用于标记汇编文件的结束行, 即标号后的代码不作处理。

3.1.5 实验操作步骤

拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下 (如果已经拷贝, 可跳过此步骤)

3.1.5.1 新建工程

首先在\Keil\ARM\Examples\目录下建立文件夹命名为 Asm1_a, 运行 μ Vision3 IDE 集成开发环境, 选择菜单项 Project – New...– μ Vision Project, 系统弹出一个对话框, 按照图 3-3 所示输入相关内容。点击“保存”按钮, 将创建一个新工程 asm_1a.Uv2。

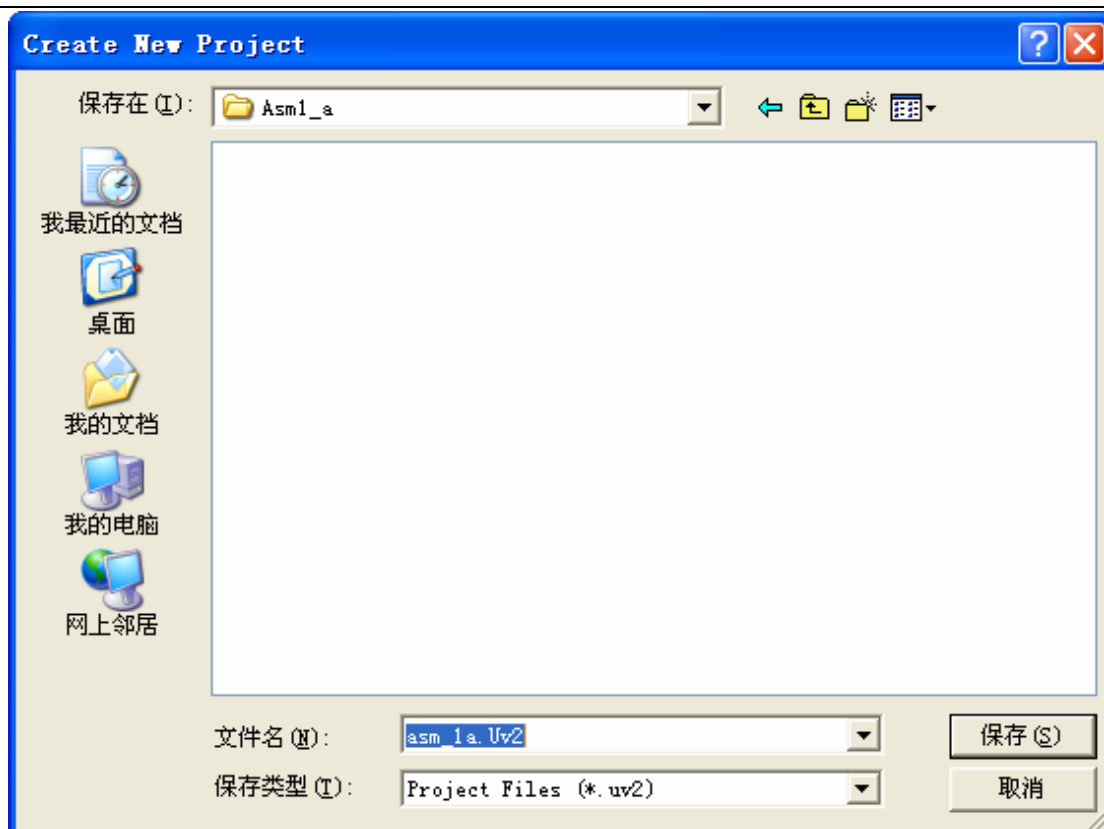


图 3-3 新建工程

3.1.5.2 为工程选择 CPU

新建工程后，要为工程选择 CPU，如图 3-4 所示，在此选择 SAMSUNG 的 S3C44BOX

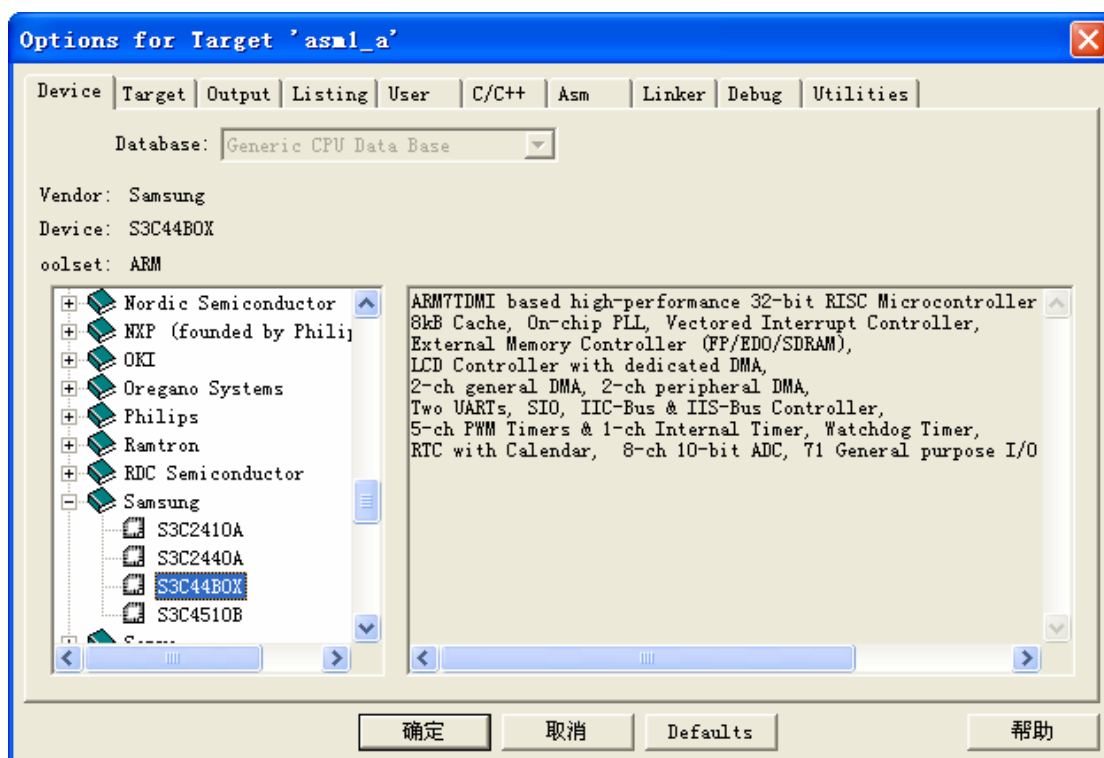


图 3-4 选择 CPU

3.1.5.3 添加启动代码

在图 3-4 中点“确定”后，会弹出一个对话框，问是否要添加启动代码。如图 3-5 所示。

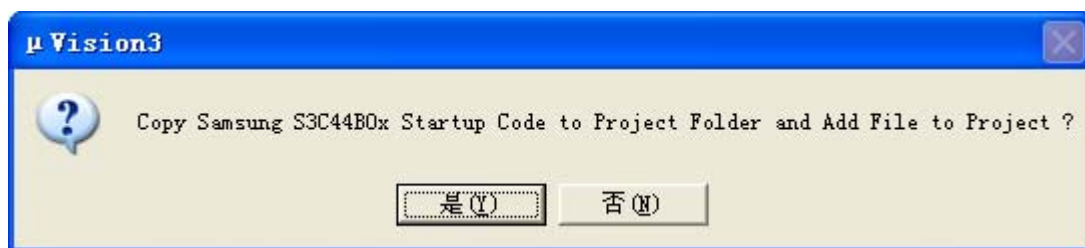


图 3-5 添加启动代码

由于本实验是简单的汇编实验，因此不需要启动代码，选择否。

3.1.5.4 选择开发工具

要为工程选择开发工具，在 Project - Manage - Components, Environment and Books - Folder/Extensions 对话框的 Folder/Extensions 页内选择开发工具，如图 3-6 所示。

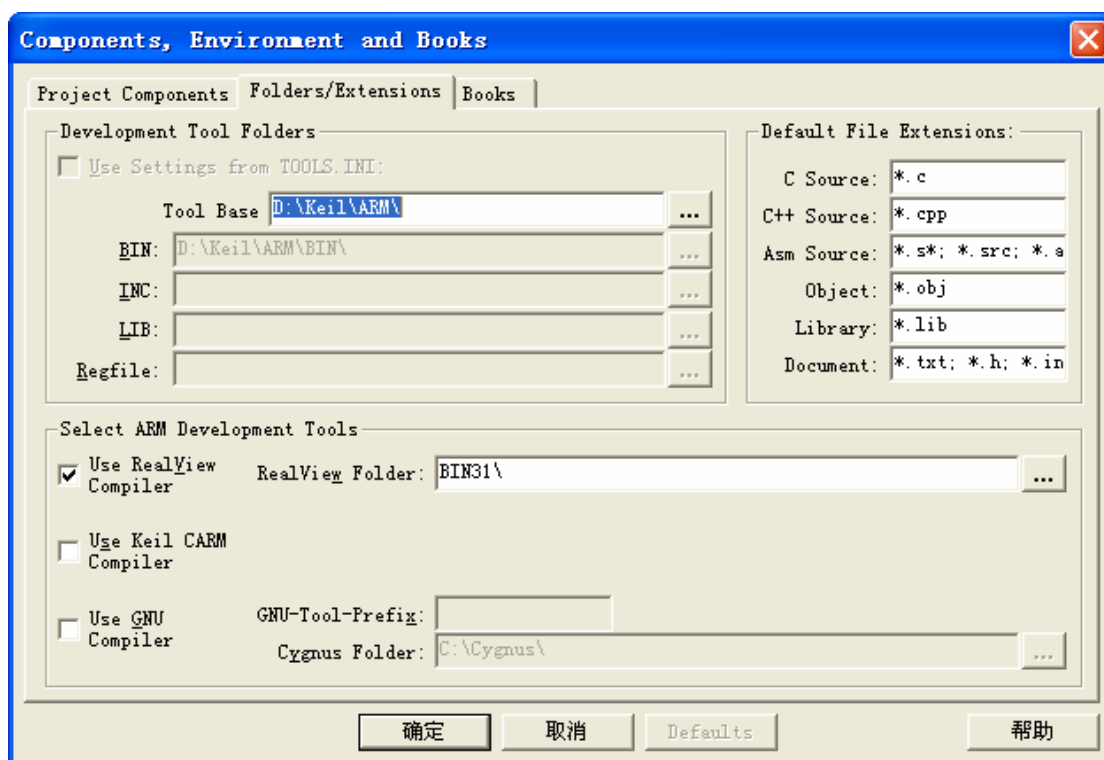


图 3-6 选择开发工具

从图中可以看到，有三个开发工具可选，在此选择 RealView Compiler。

3.1.5.5 建立源文件

点击菜单项 File - New，系统弹出一个新的、没有标题的文本编辑窗，输入光标位于窗口中第一行，按照实验参考程序编辑输入源文件代码。编辑完后，保存文件 asm1_a.s。(源代码可以参考光盘 \software\Asm1_a 中的 asm1_a.s 文件)

3.1.5.6 添加源文件

单击工程管理窗口中的相应右键菜单命令，选择 **Add Files to...**，会弹出文件选择对话框，在工程目录下选择刚才建立的源文件 asm1_a.s。如图 3-7 所示。

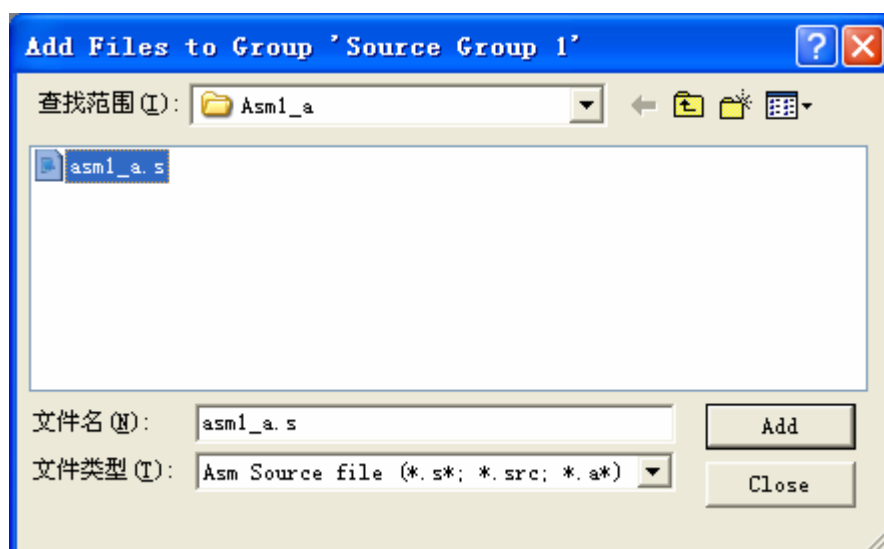


图 3-7 添加源文件

3.1.5.7 工程配置

把光盘 software\EduKit44B0_for_MDK\cmmon 目录中的 DebugINRam.ini 文件拷贝到\K eil\ARM\Examples\Asm1_a 目录下。选择菜单项 Project->Option for Target..., 将弹出工程设置对话框, 如图 3-8 所示。对话框会因所选开发工具的不同而不同, 在此仅对 Target 选项页、Link er 选项页及 Debug 选项页进行配置。Target 选项页的配置如图 3-8; Linker 选项页的配置如图 3-9; Debug 选项页的配置如图 3-10。需要注意, 在 Debug 选项页内需要一个初始化文件: DebugI NRam.ini。此.INI 文件用于设置生成的.AXF 文件下载到目标中的位置, 以及调试前的寄存器、内存的初始化等配置操作。它是由调试函数及调试命令组成调试命令脚本文件。

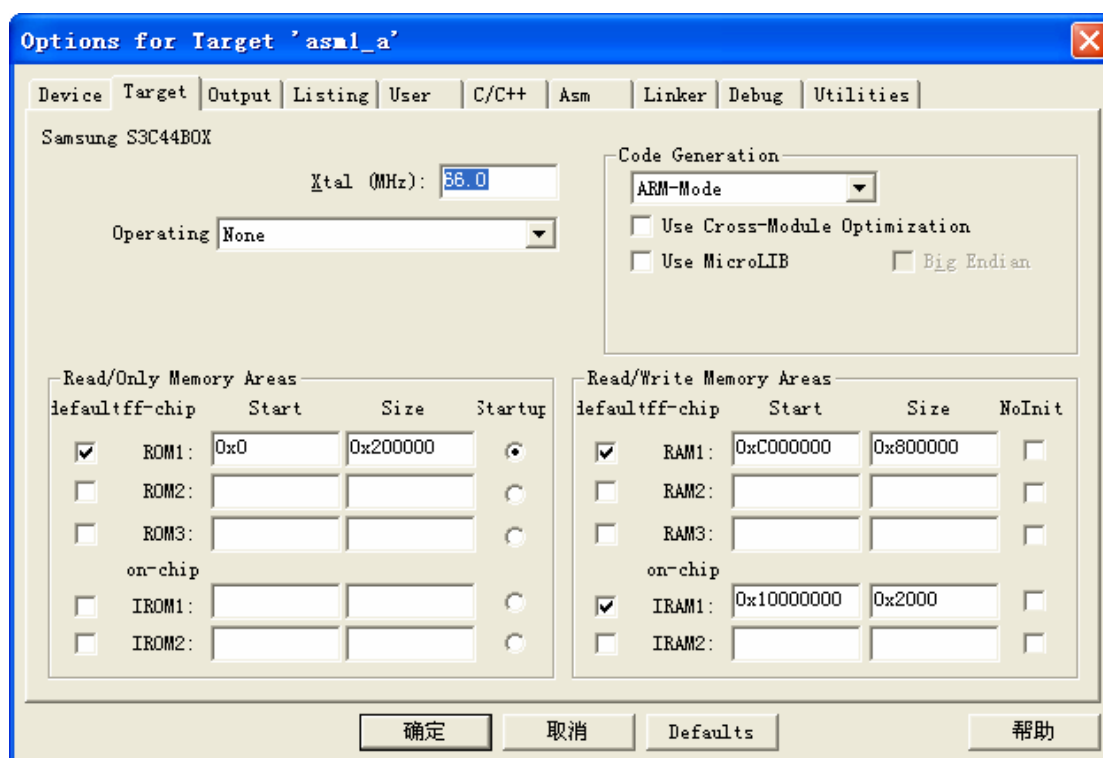


图 3-8 基本配置—Target

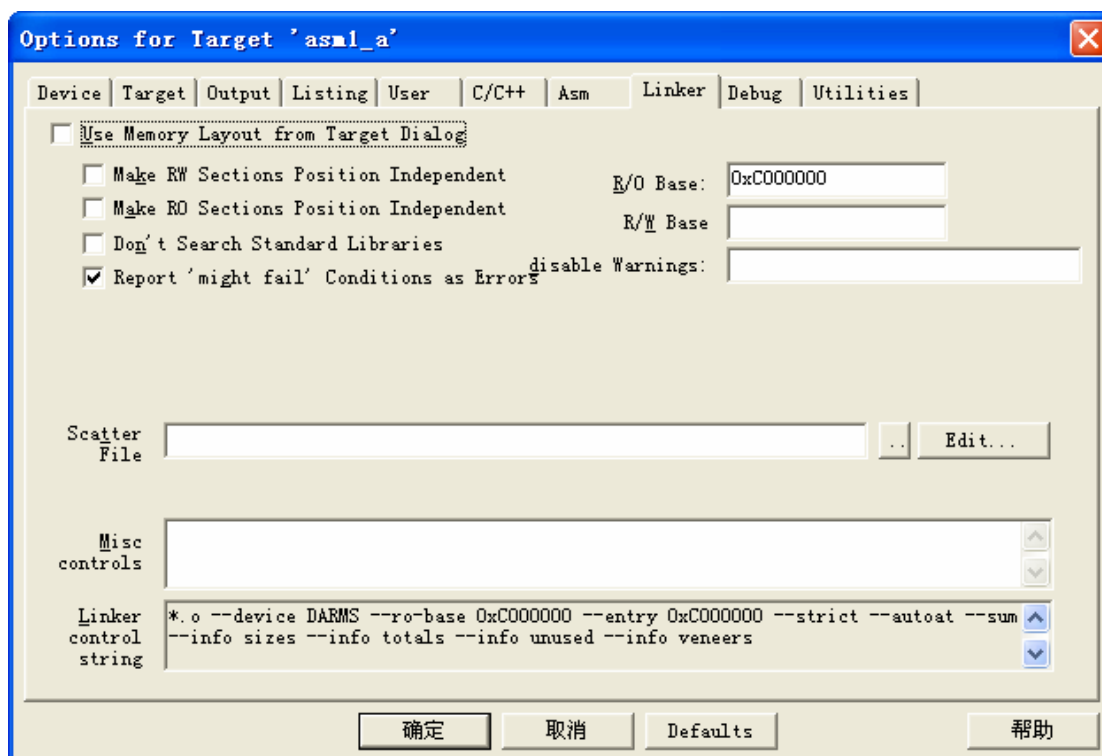


图 3-9 基本配置—Linker

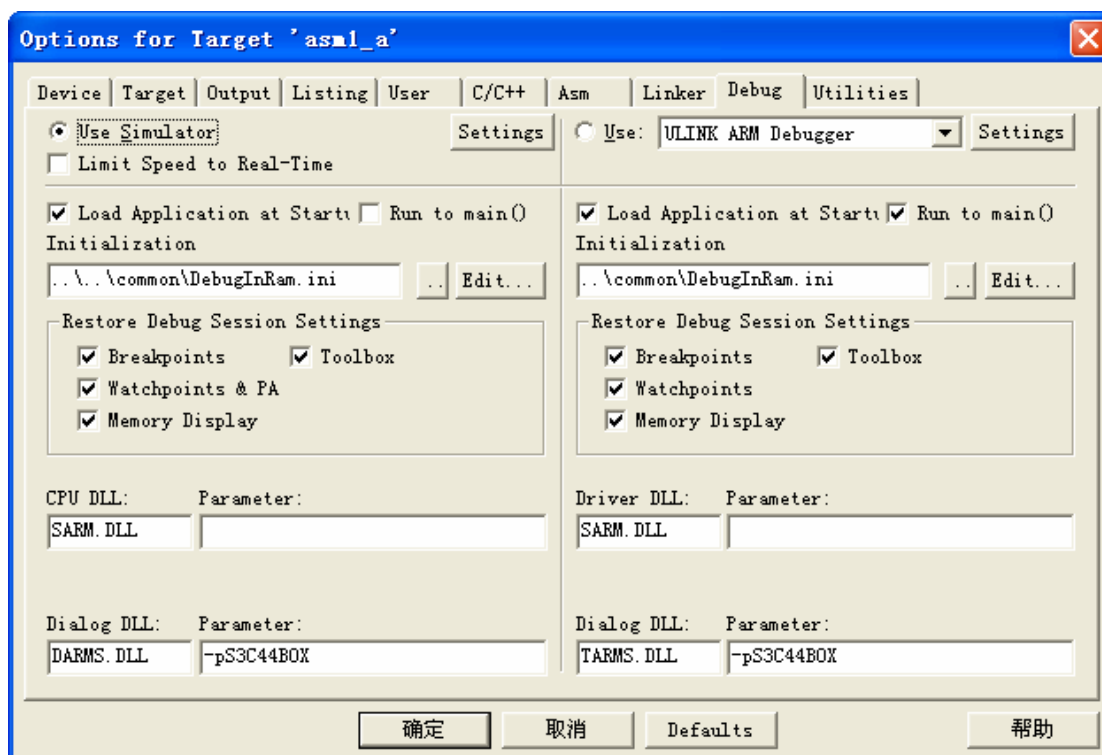


图 3-10 基本配置—Debug

3.1.5.8 生成目标代码

选择菜单项 Project - Build target 或快捷键 F7，生成目标代码。在此过程中，若有错误，则进行修改，直至无错误。若无错误，则可进行下一步的调试。

3.1.5.9 调试

选择菜单项 Debug - Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。若没有目标硬件，可以用 μ Vision 3 IDE 中的软件仿真器。如果使用 MDK 试用版，则在进入调试模式前，会有如下对话框弹出，如图 3-11 所示。

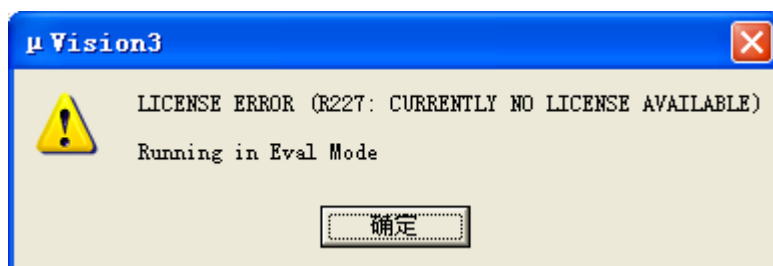


图 3-11 在软件仿真下调试程序

确定后即可调试了，做如下调试工作：

- ◆ 打开memory 窗口，单步执行，观察地址0x8000~0x801f 的内容与地址 0xff0~0xff的内容；
- ◆ 单步执行，观察寄存器的变化；
- ◆ 结合实验内容和相关资料，观察程序运行，通过实验加深理解ARM指令的使用；
- ◆ 理解和掌握实验后，完成实验练习题。

实验B与上述步骤完全相同，只要把对应的asm1_a.s文件改成asm1_b.s以及工程名即可。

3.1.6 实验参考程序

(1) 实验 A

汇编程序

```

;*****
; NAME:      asm1_a.s
*
; Author:    TYW/WUHAN R&D Center,Embest
*
; Desc:      ARM instruction examples
*
; History:   2007.5.1
*
;*****

; /*----- */
; /*          constant define
*/
; /*----- */
x      EQU 45          ; x=45
y      EQU 64          ; y=64/
stack_top EQU 0x30200000 ; define the top address for stacks

export Reset_Handler

; /*----- */
; /*          code
*/
; /*----- */

```

```

        AREA text, CODE, READONLY
        export
Reset_Handler                ; code start */
        ldr    sp, =stack_top
        mov    r0, #x                ; put x value into R0
        str    r0, [sp]              ; save the value of R0 into stacks
        mov    r0, #y                ; put y value into R0
        ldr    r1, [sp]              ; read the data from stack, and put it into
R1
        add    r0, r0, r1              ; R0=R0+R1
        str    r0, [sp]
        stop
        b      stop                  ; end the code f-cycling
        end

```

调试命令脚本文件

```

/** <<< Use Configuration !disalbe! Wizard in Context Menu >>> */
/*Name: DebugINRam.ini*/

FUNC void Setup (void)
{
    // <o> Program Entry Point, .AXF File download Address
    PC = 0x03000000;
}
map 0x00000000,0x00200000 read write exec //Map this memory to be read、write and exec
map 0x30000000,0x34000000 read write exec //Map this memeory to be read,write and exec
Setup();                      // Setup for Running
//g, main

```

(2) 实验 B

汇编程序

```

;#*****
;# NAME:      asm1_b.s
*
;# Author:    WUHAN R&D Center, Embest
*
;# Desc:      ARM instruction examples
*
;# History:    TianYunFang 2007.05.12
*
;#*****
;#----- */
;#
;#                      constant define
*/
;#----- */
x      EQU 45                ;/* x=45 */
y      EQU 64                ;/* y=64 */
z      EQU 87                ;/* z=87 */
stack_top EQU 0x30200000      ;/* define the top address for stacks*/
                        export Reset_Handler

;#----- */
;#                      code

```

```

*/
; /*----- */
        AREA text, CODE, READONLY

Reset_Handler                                ; /* code start */

        mov     r0, #x                                ; /* put x value into R0 */
        mov     r0, r0, lsl #8                    ; /* R0 = R0 << 8 */
        mov     r1, #y                                ; /* put y value into R1 */
        add     r2, r0, r1, lsr #1                ; /* R2 = (R1>>1) + R0 */
        ldr     sp, =stack_top
        str     r2, [sp]

        mov     r0, #z                                ; /* put z value into R0 */
        and     r0, r0, #0xFF                    ; /* get low 8 bit from R0 */
        mov     r1, #y                                ; /* put y value into R1 */
        add     r2, r0, r1, lsr #1                ; /* R2 = (R1>>1) + R0 */

        ldr     r0, [sp]                                ; /* put y value into R1 */
        mov     r1, #0x01
        orr     r0, r0, r1
        mov     r1, R2                                ; /* put y value into R1 */
        add     r2, r0, r1, lsr #1                ; /* R2 = (R1>>1) + R0 */

stop
        b       stop                                ; /* end the code f-cycling */
        END

```

调试命令脚本文件与实验 A 相同。

3.1.7 练习题

1. 编写程序循环对 R4~R11 进行累加 8 次赋值, R4~R11 起始值为 1~8, 每次加操作后把 R4~R11 的内容放入 SP 栈中, SP 初始设置为 0x800。最后把 R4~R11 用 LDMFD 指令清空赋值为 0。
2. 更改实验 A 中 X、Y 的值, 观察执行结果。

3.2 ARM 汇编指令实验二

3.2.1 实验目的

- 通过实验掌握使用 ldm/stm, b, bl 等指令完成较为复杂的存储区访问和程序分支;
- 学习使用条件码, 加强对 CPSR 的认识;

3.2.2 实验设备

- 硬件: PC 机。
- 软件: μVision IDE for ARM 集成开发环境, Windows 98/2000/NT/XP。

3.2.3 实验内容

- 熟悉开发环境的使用并完成一块存储区的拷贝。
- 完成分支程序设计, 要求判断参数, 根据不同参数, 调用不同的子程序。

3.2.4 实验原理

1. ARM 程序状态寄存器

在所有处理器模式下都可以访问当前的程序状态寄存器 CPSR。CPSR 包含条件码标志，中断禁止位，当前处理器模式以及其它状态和控制信息。每种异常模式都有一个程序状态保存寄存器 SPSR。当异常出现时，SPSR 用于保存 CPSR 的状态。

CPSR 和 SPSR 的格式如下：

| | | | | | | | | | | | | | |
|----|----|----|----|----|-------------|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | DNM (RAZ) | I | F | T | M | M | M | M | M |

1) 条件码标志：

N, Z, C, V 大多数指令可以检测这些条件码标志以决定程序指令如何执行。

2) 控制位：

最低 8 位 I, F, T 和 M 位用做控制位。当异常出现时改变控制位。当处理器在特权模式下也可以由软件改变。

- 中断禁止位：I 置1 则禁止IRQ 中断；F 置1 则禁止FIQ 中断。
- T 位：T=0 指示ARM 执行；T=1 指示Thumb 执行。在这些体系结构系统中，可自由地使用能在ARM 和Thumb 状态之间切换的指令。
- 模式位：M0, M1, M2, M3 和M4 (M[4:0]) 是模式位.这些位决定处理器的工作模式.如表 12-1 所示。

表 3-1 ARM 工作模式 M[4:0]

| M[4:0] | 模式 | 可访问的寄存器 |
|---------|-----|--|
| 0b10000 | 用户 | PC, R14~R0, CPSR |
| 0b10001 | FIQ | PC, R14_fiq~R8_fiq, R7~R0, CPSR, SPSR_fiq |
| 0b10010 | IRQ | PC, R14_irq~R8_fiq, R12~R0, CPSR, SPSR_irq |
| 0b10011 | 管理 | PC, R14_svc~R8_svc, R12~R0, CPSR, SPSR_svc |
| 0b10111 | 中止 | PC, R14_abt~R8_abt, R12~R0, CPSR, SPSR_abt |
| 0b11011 | 未定义 | PC, R14_und~R8_und, R12~R0, CPSR, SPSR_und |
| 0b11111 | 系统 | PC, R14~R0, CPSR |

3) 其他位

程序状态寄存器的其他位保留，用作以后的扩展。

2. 本实验涉及到的汇编指令语法及规则

ldr

ldr 伪指令将一个 32 位的常数或者一个地址值读取到寄存器中。当需要读取到寄存器中的数据超过了 mov 或者 mnv 指令可以操作的范围时，可以使用 ldr 伪指令将该数据读取到寄存器中。在汇编编译器处理源程序时，如果该常数没有超过 mov 或者 mnv 可以操作的范围，则 ldr 指令被这两

条指令中的一条所替代，否则，该常数将被放在最近的一个文字池内（literal pool），同时，本指令被一条基于 PC 的 ldr 指令替代。

语法格式：

ldr <register> , = <expression>

其中，expression 为需要读取的32 位常数；register 为目标寄存器。

示例：

```
ldr r1, =0xff
ldr r0, =0xffff0000
```

adr

adr 指令将基于 PC 的地址值或者基于寄存器的地址值读取到寄存器中。在汇编编译器处理源程序时，adr 伪指令被编译器替换成一条合适的指令。通常，编译器用一条 add 指令或者 sub 指令来实现该伪指令的功能。如果标号超出范围或者标号在同一文件（和同一段）内没有定义，则会产生一个错误。该指令不使用文字池（literal pool）。

语法格式：

adr <register> , <label>

其中，register 为目标寄存器；label为基于PC 或者寄存器的地址表达式。

示例：

```
label1
    mov r0, #25
    adr r2, label1
```

ltorg

ltorg 用于声明一个文字池。

语法格式：

ltorg

3.2.5 实验操作步骤

(1) 实验 A

1) 在\Keil\ARM\Examples\目录下建立文件夹命名为 armcode1，参考 3.1.5 小节实验的操作步骤建立一个新的工程，命名为 ArmCode1；

2) 参考 3.1.5 小节实验的步骤和实验参考程序编辑输入源代码，保存文件为 asmcode1.s(源代码可以参考光盘\software\EduKit44B0_for_MDK\3.2_asm2\Armcode1 中的 asmcode1.s 文件)；

3) 在 Project workspace 工作区中右击 target1->Source Group 1，在弹出菜单中选择“Add file to Group ‘Source’ ”，在随后弹出的文件选择对话框中，选择刚才建立的源文件 armcode1.s；

4) 把光盘\software\EduKit44B0_for_MDK\common 目录中的 DebugINRam.ini 文件拷贝到\Keil\ARM\Examples\Armcode1 目录下。选择菜单项 Project->Option for Target...，将弹出工程设置对话框，设置与 3.1.5 小节实验中的工程配置相同。

5) 选择菜单项 Project ->Build target 或快捷键 F7，生成目标代码；

6) 选择菜单项 Debug ->Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。这里使用的是 μVision3 IDE 中的软件仿真器；

7) 选择菜单项 Debug ->run 或 F5，即可运行代码；

8) 打开 memory 窗口，观察地址 0x8054~0x80A0 的内容，与地址 0x80A4~0x80f0 的内容

9) 单步执行程序并观察和记录寄存器与 memory 的值变化，注意观察步骤 8 里面的地址的内容变化，当执行 stmfd, ldmfd, ldmia 和 stmia 指令的时候，注意观察其后面参数所指的地址段或寄存器段的内容变化；

10) 结合实验内容和相关资料，观察程序运行，通过实验加深理解 ARM 指令的使用；

11) 理解和掌握实验后，完成实验练习题。

(2) 实验 B

1) 参考 3.1.5 小节实验的操作步骤建立一个新的工程，命名为 ArmCode2；

2) 参考 3.1.5 小节实验的步骤和实验参考程序编辑输入源代码，保存文件为 asmcode2.s（如果存在，可以调过此步骤）；

3) 在 Project workspace 工作区中右击 target1->Source Group 1，在弹出菜单中选择“Add file to Group ‘Source’”，在随后弹出的文件选择对话框中，选择刚才建立的源文件 ArmCode2.s；

4) 参照实验 A 的步骤完成目标代码的生成与调试。

5) 理解和掌握实验后，完成实验练习题。

3.2.6 实验参考程序

(1) 实验 A 参考源代码：

```

;#*****
;# NAME:   ARMcode.s
;#
;# Author:   EWUHAN R & D Center, st
;#
;# Desc:   ARMcode examples
;#
;#          copy words from src to dst
;#
;# History:   shw.He 2005.02.22
;#
;#*****
;/*----- */
;/*                               code
;/*
;/*----- */

GLOBAL Reset_Handler
area start,code,readwrite
entry
code32
num EQU 20 ;/* Set number of
words to be copied */

Reset_Handler
    ldr    r0, =src ;/* r0 = pointer to source block */
    ldr    r1, =dst ;/* r1 = pointer to destination block

```

```

*/
    mov     r2, #num                                /* r2 = number of words to copy */

    ldr     sp, =0x30200000                          /* set up stack pointer (r13) */
blockcopy
    movs    r3,r2, LSR #3                            /* number of eight word multiples */
    beq     copywords                                /* less than eight words to move ? */

    stmfd   sp!, {r4-r11}                            /* save some working registers */
octcopy
    ldmia   r0!, {r4-r11}                            /* load 8 words from the source */
    stmia   r1!, {r4-r11}                            /* and put them at the destination */
    subs    r3, r3, #1                                /* decrement the counter */
    bne     octcopy                                  /* ... copy more */

    ldmfd   sp!, {r4-r11}                            /* don't need these now - restore
originals */

copywords
    ands    r2, r2, #7                                /* number of odd words to copy */
    beq     stop                                      /* No words left to copy ? */
wordcopy
    ldr     r3, [r0], #4                              /* a word from the source */
    str     r3, [r1], #4                              /* store a word to the destination */
    subs    r2, r2, #1                                /* decrement the counter */
    bne     wordcopy                                  /* ... copy more */

stop
    b       stop

/*----- */
/*
    make a word pool
*/
/*----- */
    ltorg
src
    dcd     1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4
dst
    dcd     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
end

```

(2) 实验 B 参考源代码:

```

;#*****
;# NAME:   ARM_code2.s
;#
;# Author:   WUHAN R&D Center,Embest
;#
;# Desc:   ARM instruction examples
;#
;#           Example for Condition Code
;#

```



```

;# History:    shw.He 2005.02.22
*
;#*****

;/*----- */
                                code
;/*----- */

        area start,code,readwrite
        entry
        code32
num equ    2                                ;/* Number of entries in jump table */
        export Reset_Handler
Reset_Handler
        mov     r0, #0                      ;/* set up the three parameters */
        mov     r1, #3
        mov     r2, #2
        bl      arithfunc                   ;/* call the function */

stop
        b       stop

;# *****
;# * According R0 valude to execute the code
*
;# *****
arithfunc                                ;/* label the function */
        cmp     r0, #num                    ;/* Treat function code as unsigned
integer */
        bhs     DoAdd                       ;/* If code is >=2 then do operation 0.
*/

        adr     r3, JumpTable                ;/* Load address of jump table */
        ldr     pc, [r3,r0,LSL#2]            ;/* Jump to the appropriate routine */

JumpTable
        dcd     DoAdd
        dcd     DoSub

DoAdd
        add     r0, r1, r2                  ;/* Operation 0, >1 */
        bx      lr                          ;/* Return */

DoSub
        sub     r0, r1, r2                  ;/* Operation 1 */
        bx      lr                          ;/* Return */

end                                          ;/* mark the end of this file */

```

3.2.7 练习题

1. 打开开发板光盘中的启动文件，观察启动文件中复位异常的编写及 Itorg 的使用及其功能；
2. 新建工程，并自行编写汇编程序，分别使用 ldr、str、ldmia、stmia 操作，实现对某段连续存储单元写入数据，并观察操作结果。

3.3 Thumb 汇编指令实验

3.3.1 实验目的

- 通过实验掌握 ARM 处理器 16 位 Thumb 汇编指令的使用方法。

3.3.2 实验设备

- 硬件：PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.3.3 实验内容

- 使用 THUMB 汇编语言，完成基本 reg/mem 访问，以及简单的算术/逻辑运算。
- 使用 THUMB 汇编语言，完成较为复杂的程序分支，push/pop，领会立即数大小的限制，并体会 ARM 工作状态与 THUMB 工作状态的区别。

3.3.4 实验原理

(1) ARM 处理器工作状态

ARM 处理器共有两种工作状态：

ARM：32 位，这种状态下执行字对准的 ARM 指令；

Thumb：16 位，这种状态下执行半字对准的 Thumb 指令。

在 Thumb 状态下，程序计数器 PC 使用位 1 选择另一个半字。

注意：ARM 和 Thumb 之间状态的切换不影响处理器的模式或寄存器的内容。

ARM 处理器在两种工作状态之间可以切换。

进入 Thumb 状态。当操作数寄存器的状态位（位 0）为 1 时，执行 BX 指令进入 Thumb 状态。如果处理器在 Thumb 状态进入异常，则当异常处理（IRQ，FIQ，Undef，Abort 和 SWI）返回时，自动切换到 Thumb 状态。

进入 ARM 状态。当操作数寄存器的状态位（位 0）为 0 时，执行 BX 指令进入 ARM 状态。此外，在处理器进行异常处理（IRQ，FIQ，Undef，Abort 和 SWI）时，把 PC 放入异常模式链接寄存器中，从异常向量地址开始执行也可进入 ARM 状态。

(2) Thumb 状态的寄存器集

Thumb 状态下的寄存器集是 ARM 状态下寄存器集的子集。程序员可以直接访问 8 个通用的寄存器(R0~R7)，PC,SP,LR 和 CPSP。每一种特权模式都有一组 SP，LR 和 SPSR。

Thumb 状态的 R0~R7 与 ARM 状态的 R0~R7 是一致的；

Thumb 状态的 CPSR 和 SPSR 与 ARM 状态的 CPSR 和 SPSR 是一致的；

Thumb 状态的 SP 映射到 ARM 状态的 R13；

Thumb 状态的 LR 映射到 ARM 状态的 R14；

Thumb 状态的 PC 映射到 ARM 状态的 PC (R15)。

Thumb 寄存器与 ARM 寄存器的关系如图 3-7 所示。

(3) 本实验涉及到的伪操作

Code [16|32]

code 伪操作用于选择当前汇编指令的指令集。参数 16 选择 Thumb 指令集，参数 32 选择 ARM 指令集。

语法格式：

code[16|32]

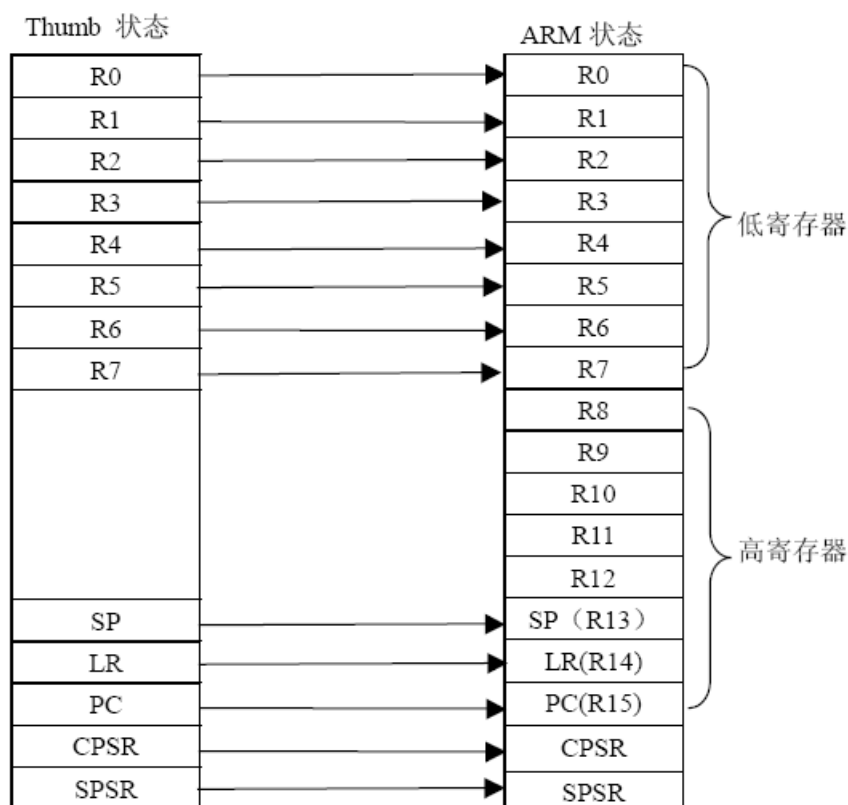


图 3-7 寄存器状态图

thumb

同 code 16。

arm

同 code 32。

align

align 伪指令通过添加补丁字节使当前位置满足一定的对齐方式。

语法格式：

align {expr{, offset}}

其中：expr 为数字表达式，用于指定对齐的方式。取值为 2 的 n 次幂，如 1、2、4、8 等，不能为 0。若没有 expr，则默认为字对齐方式。

Offset 为数字表达式。当前位置对齐到下面形式的地址处： $\text{offset} + n * \text{expr}$

示例

align 4, 3 ; 字对齐

3.3.5 实验操作步骤

(1) 实验 A

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 打开实验例程目录 3.3_thumbcode\thumbcode1 子目录下的 thumbcode1.Uv2 例程；
- 3) 选择菜单项 Project ->Build target 或快捷键 F7，生成目标代码；
- 4) 选择菜单项 Debug ->Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。这里使用的是 μ Vision3 IDE 中的软件仿真器；
- 5) 选择菜单项 Debug ->run 或 F5，即可运行代码；
- 6) 记录代码执行区中每条指令的地址，注意指令最后尾数的区别；
- 7) 观察 Project workspace 工作区中寄存器的变化，特别是 R0 和 R1 的值的变化；
- 8) 结合实验内容和相关资料，观察程序运行，通过实验加深理解 ARM 指令和 Thumb 指令的不同；
- 9) 理解和掌握实验后，完成实验练习题。

(2) 实验 B

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 打开实验例程目录 3.3_thumbcode\thumbcode2 子目录下的 thumbcode2.Uv2 例程；
- 3) 选择菜单项 Project ->Build target 或快捷键 F7，生成目标代码；
- 4) 选择菜单项 Debug ->Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。这里使用的是 μ Vision3 IDE 中的软件仿真器；
- 5) 选择菜单项 Debug ->run 或 F5，即可运行代码；
- 6) 注意并记录 ARM 指令下和 Thumb 指令状态下 stmfd, ldmfd, ldmia 和 stmia 指令执行的结果，指令的空间地址数值，数据存储的空间大小等等；
- 7) 理解和掌握实验后，完成实验练习题。

3.3.6 实验参考程序

(1) 实验 A 参考源代码：

```

;#*****
;# NAME:   ThumbCode.s
;# Author:   Wuhan R&D Center, Embest
;# Desc:   ThumbCode examples
;# History:   WuHan R&D Center 2007.01.12
;#*****
;/*-----*/
;/*          unable to locate source file.          code*/
;/*-----*/

        area start,code,readonly
        entry

```

```

        code32                /* Subsequent instructions are ARM */
        export Reset_Handler
Reset_Handler
        adr    r0, Tstart + 1    /* Processor starts in ARM state, */
        bx     r0                /* so small ARM code header used */
                                   /* to call Thumb main program. */
        nop
        code16
Tstart
        mov    r0, #10           /* Set up parameters */
        mov    r1, #3
        bl     doadd             /* Call subroutine */

stop
        b      stop

/*-----*/
/* Subroutine code:R0 = R0 + R1 and return */
/*-----*/
doadd
        add    r0, r0, r1        /* Subroutine code */
        mov    pc, lr           /* Return from subroutine. */
        end                /* Mark end of file */

```

本节可使用 3.1 节的调试脚本文件。

(2) 实验 B 参考源代码:

```

;*****
; NAME:      Thumbcode2.s
; Author:    WuHan R&D Center,Embest
; Desc:      Thunmbcode examples
;            copy words from src to dst
; History:   WuHan R&D Center 2007.01.12
;*****
;
;            code
;-----
        area start,code,readonly
        entry
        code32                /* Subsequent instructions are ARM */
num equ 20                /* Set number of words to be copied */
        export Reset_Handler
Reset_Handler
                                   /* Subsequent instructions are ARM header
*/
        ldr    sp, =0x30200000    /* set up user_mode stack pointer (r13)
*/
        adr    r0, Tstart + 1    /* Processor starts in ARM state,
*/
        bx     r0                /* so small ARM code header used */
                                   /* to call Thumb main program. */
        code16                /* Subsequent instructions are Thumb. */
Tstart

```

```

        ldr    r0, =src                /* r0 = pointer to source block */
        ldr    r1, =dst                /* r1 = pointer to destination block */
        mov    r2, #num                /* r2 = number of words to copy */
blockcopy
        lsr    r3, r2, #2              /* number of four word multiples */
        beq    copywords               /* less than four words to move ? */
        push   {r4-r7}                 /* save some working registers */
quadcopy
        ldmia  r0!, {r4-r7}            /* load 4 words from the source */
        stmia  r1!, {r4-r7}            /* and put them at the destination */
        sub    r3, #1                  /* decrement the counter */
        bne    quadcopy                /* ... copy more */
        pop    {r4-r7}                 /* don't need these now - restore originals */
copywords
        mov    r3, #3                  /* bottom two bits represent
number... */
        and    r2, r3                  /* ...of odd words left to copy */
        beq    stop                    /* No words left to copy ? */
wordcopy
        ldmia  r0!, {r3}                /* a word from the source */
        stmia  r1!, {r3}                /* store a word to the destination */
        sub    r2, #1                  /* decrement the counter */
        bne    wordcopy                /* ... copy more */
stop
        b      stop

/*----- */
/* make a word pool */
/*----- */
        align
src
        dcd    1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4
dst
        dcd    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
end

```

3.3.7 练习题

编写程序从 ARM 状态切换到 Thumb，在 ARM 状态下把 R2 赋值为 0x12345678，在 Thumb 状态下把 R2 赋值为 0x87654321。同时观察并记录 CPSR，SPSR 的值，分析各个标志位。

3.4 ARM 处理器工作模式实验

3.4.1 实验目的

- 通过实验掌握学会使用 msr/mrs 指令实现 ARM 处理器工作模式的切换；
- 观察不同模式下的寄存器，加深对 CPU 结构的理解；
- 通过实验进一步熟悉 ARM 汇编指令。

3.4.2 实验设备

- 硬件：PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.4.3 实验内容

通过 ARM 汇编指令，在各种处理器模式下切换并观察各种模式下寄存器的区别；掌握 ARM 不同模式的进入与退出。

3.4.4 实验原理

1. ARM 处理器模式

ARM 体系结构支持下表 3-2 所列的 7 种处理器模式。

表 3-2 处理器模式

| 处理器模式 | 说明 |
|---------|-----------------|
| 用户 usr | 正常程序执行模式 |
| FIQ fiq | 支持高速数据传送或通道处理 |
| IRQ irq | 用于通用中断处理 |
| 管理 svc | 操作系统保护模式 |
| 中止 abt | 实现虚拟存储器和/或存储器保护 |
| 未定义 und | 支持硬件协处理器的软件仿真 |
| 系统 sys | 运行特权操作系统任务 |

在软件控制下可以改变模式，外部中断或异常处理也可以引起模式发生改变。

大多数应用程序在用户模式下执行。当处理器工作在用户模式时，正在执行的程序不能访问某些被保护的系统资源，也不能改变模式，除非异常(exception)发生。这允许通过合适地编写操作系统来控制系统资源的使用。

除用户模式外的其他模式称为特权模式。它们可以自由的访问系统资源和改变模式。其中的 5 种称为异常模式，即：

- FIQ (Fast Interrupt request)；
- IRQ (Interrupt ReQuest)；
- 管理 (Supervisor)；
- 中止(Abort) ；
- 未定义(Undefined) 。

当特定的异常出现时，进入相应的模式。每种模式都有某些附加的寄存器，以避免异常出现时用户模式的状态不可靠。

剩下的模式是系统模式。仅 ARM 体系结构 V4 及其以上的版本有该模式。不能由任何异常而进入该模式。它与用户模式有完全相同的寄存器，然而它是特权模式，不受用户模式的限制。它供需要访

问系统资源的操作系统任务使用，但希望避免使用与异常模式有关的附加寄存器。避免使用附加寄存器保证了当任何异常出现时，都不会使任务的状态不可靠。

2. 程序状态寄存器

前一节提到的程序状态寄存器 CPSR 和 SPSR 包含了条件码标志，中断禁止位，当前处理器模式以及其他状态和控制信息。每种异常模式都有一个程序状态保存寄存器 SPSR。当异常出现时，SPSR 用于保留 CPSR 的状态。

CPSR 和 SPSR 的格式如下：

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----------|--|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | DNM(RAZ) | | I | F | T | M | M | M | M | M | M |

1) 条件码标志：

N, Z, C, V 大多数指令可以检测这些条件码标志以决定程序指令如何执行

2) 控制位：

最低 8 位 I, F, T 和 M 位用作控制位。当异常出现时改变控制位。当处理器在特权模式下也可以由软件改变。

- 中断禁止位:I 置 1 则禁止 IRQ 中断；F 置 1 则禁止 FIQ 中断。
- T 位:T=0 指示 ARM 执行；T=1 指示 Thumb 执行。在这些体系结构的系统中，可自由的使用能在 ARM 和 Thumb 状态之间切换的指令。
- 模式位:M0, M1, M2, M3 和 M4 (M[4:0]) 是模式位.这些位决定处理器的工作模式.如表 3-3 所示。

表 3-3 ARM 工作模式 M[4:0]

| M[4:0] | 模式 | 可访问的寄存器 |
|---------|-----|---|
| 0b10000 | 用户 | PC, R14~R0,CPSR |
| 0b10001 | FIQ | PC, R14_fiq~R8_fiq,R7~R0,CPSR,SPSR_fiq |
| 0b10010 | IRQ | PC, R14_irq~R8_fiq,R12~R0,CPSR,SPSR_irq |
| 0b10011 | 管理 | PC, R14_svc~R8_svc,R12~R0,CPSR,SPSR_svc |
| 0b10111 | 中止 | PC, R14_abt~R8_abt,R12~R0,CPSR,SPSR_abt |
| 0b11011 | 未定义 | PC, R14_und~R8_und,R12~R0,CPSR,SPSR_und |
| 0b11111 | 系统 | PC, R14~R0,CPSR |


3) 其他位

程序状态寄存器的其他位保留，用作以后的扩展。

3.4.5 实验操作步骤

1) 参考 3.1.5 小节实验 A 的步骤建立一个新的工程，命名为 ARMMode，处理器选择 S3C44B0X；

2) 参考 3.1.5 小节实验 A 的步骤和实验参考程序编辑输入源代码，编辑完毕后，保存文件为 armmode.s；

3) 单击工具栏的  图标，或单击工程管理窗口中的相应右键菜单 Manage Components 命令，弹出 Componets,Environment and Books 对话框，在该对话框中为相应的文件组添加刚才新建的源文件 ARMMode.s；

- 4) 参考 3.1.5 小节实验 A 的步骤进行相应设置、生成目标代码和下载目标代码调试；
- 5) 打开寄存器窗，单步执行，观察并记录寄存器 R0 和 CPSR 的值的变化和每次变化后执行寄存器赋值后的 36 个寄存器的值的变化情况，尤其注意各个模式下 R13 和 R14 的值；
- 6) 结合实验内容和相关资料，观察程序运行，通过实验加深理解 ARM 各种状态下寄存器的使用；
- 7) 理解和掌握实验后，完成实验练习题。

3.4.6 实验参考程序

```

;*****
;
; NAME:      ARMmode.s                                *
; Author:    Wuhan R&D Center, Embest                  *
; Desc:      ARM instruction examples                  *
;            Example for ARM mode                      *
; History:    JianYing, Wang 2007.05.15                *
;*****
;

;-----*/
;/*                                */
;/*                                constant define          */
;/*-----*/

EXPORT start

;-----*/
;/*                                */
;/*                                code                      */
;/*-----*/

AREA    |.text|, CODE, READONLY

start
;-----*/
;/* Setup interrupt / exception vectors                */
;/*-----*/

    b      Reset_Handler
Undefined_Handler
    b      Undefined_Handler
    b      SWI_Handler
Prefetch_Handler
    b      Prefetch_Handler
Abort_Handler
    b      Abort_Handler
    nop                                ;/* Reserved vector */
IRQ_Handler
    b      IRQ_Handler
FIQ_Handler
    b      FIQ_Handler
SWI_Handler
    bx lr

Reset_Handler

```

```

visitmen
; /*-----*/
; /*   into System mode                                     */
; /*-----*/
    mrs   r0,cpsr                ; /* read CPSR value          */
    bic   r0,r0,#0x1f            ; /* clear low 5 bit        */
    orr   r0,r0,#0x1f            ; /* set the mode as System mode */
    msr   cpsr_cxfs,r0

    mov r0, #1                    ; /* initialization the register in System mode */
    mov r1, #2
    mov r2, #3
    mov r3, #4
    mov r4, #5
    mov r5, #6
    mov r6, #7
    mov r7, #8
    mov r8, #9
    mov r9, #10
    mov r10, #11
    mov r11, #12
    mov r12, #13
    mov r13, #14
    mov r14, #15

; /*-----*/
; /*   into FIQ mode                                       */
; /*-----*/
    mrs   r0,cpsr                ; /* read CPSR value          */
    bic   r0,r0,#0x1f            ; /* clear low 5 bit        */
    orr   r0,r0,#0x11            ; /* set the mode as FIQ mode */
    msr   cpsr_cxfs,r0

    mov r8, #16                    ; /* initialization the register in FIQ mode */
    mov r9, #17
    mov r10, #18
    mov r11, #19
    mov r12, #20
    mov r13, #21
    mov r14, #22

; /*-----*/
; /*   into SVC mode                                       */
; /*-----*/
    mrs   r0,cpsr                ; /* read CPSR value          */
    bic   r0,r0,#0x1f            ; /* clear low 5 bit        */
    orr   r0,r0,#0x13            ; /* set the mode as SVC mode */

```

```

msr cpsr_cxfs,r0

mov r13, #23                /* initialization the register in SVC mode */
mov r14, #24

/*-----*/
/* into Abort mode */
/*-----*/
mrs r0,cpsr                /* read CPSR value */
bic r0,r0,#0x1f            /* clear low 5 bit */
orr r0,r0,#0x17            /* set the mode as Abort mode */
msr cpsr_cxfs,r0

mov r13, #25                /* initialization the register in Abort mode */
mov r14, #26

/*-----*/
/* into IRQ mode */
/*-----*/
mrs r0,cpsr                /* read CPSR value */
bic r0,r0,#0x1f            /* clear low 5 bit */
orr r0,r0,#0x12            /* set the mode as IRQ mode */
msr cpsr_cxfs,r0

mov r13, #27                /* initialization the register in IRQ mode */
mov r14, #28

/*-----*/
/* into UNDEF mode */
/*-----*/
mrs r0,cpsr                /* read CPSR value */
bic r0,r0,#0x1f            /* clear low 5 bit */
orr r0,r0,#0x1b            /* set the mode as UNDEF mode */
msr cpsr_cxfs,r0

mov r13, #29                /* initialization the register in UNDEF mode */
mov r14, #30

b Reset_Handler            /* jump back to Reset_Handler */

END

.##*****
.##
;# NAME:      abort.s      *
;# Author:    Embest      *
```

```

;# Desc:      ARM instruction examples
;#           Example for Abort mode
;# History:   shw.He 2005.02.22
;#*****

;-----*/
;/*                               constant define */
;-----*/

;-----*/
;/*                               code */
;-----*/

        area start,code,readonly
        entry
        export START
START

;-----*/
;/* Setup interrupt / exception vectors
*/
;-----*/

        b      Reset_Handler
Undefined_Handler
        b      Undefined_Handler
        b      SWI_Handler
Prefetch_Handler
        b      Prefetch_Handler
Abort_Handler
        b      Abort_Handler
        nop                               ;/* Reserved vector */
IRQ_Handler
        b      IRQ_Handler
FIQ_Handler
        b      FIQ_Handler

SWI_Handler
        bx     lr

Reset_Handler

visitmen
        ldr    r1,=0xf0000000           ;/* R1 = 0xf0000000 */
        ldr    r2,[r1]                 ;/* read the value in 0xf0000000 into R2 */
;-----*/
;/* the non-exist memory access will cause cpu into Abort mode (only a real hardware) */

```

```

;-----*/
mrs r0,spsr                                ;/* read CPSR value
*/
mov r13, #1                                ;/* initialization the register in Abort mode*/
mov r14, #2

;-----*/
;/* into User mode
*/
;-----*/

mrs r0,cpsr                                ;/* read CPSR value */
bic r0,r0,#0x1f                             ;/* clear low 5 bit */
orr r0,r0,#0x10                             ;/* set the mode as User mode */
msr cpsr_cxfs,r0                             ;/* write the data into CPSR */
mov r13, #3                                ;/* initialization the register in User mode */

mov r14, #4
stop
b stop

end

```

本节可使用 3.1 节的调试脚本文件。

3.4.7 练习题

参考第一个例子，把其中系统模式程序更改为用户模式程序，编译调试，观察运行结果，检查是否正确，如果有错误，分析其原因；（提示：不能从用户模式直接切换到其他模式，可以先使用 SWI 指令切换到管理模式）。

3.5 C 语言实例一

3.5.1 实验目的

学会使用 μ Vision IDE for ARM 开发环境编写简单的 C 语言程序；

学会编写和使用调试函数；

掌握通过 memory/register/watch/variable 窗口分析判断运行结果。

3.5.2 实验设备

- 硬件：PC 机。
- 软件： μ Vision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.5.3 实验内容

利用函数初始化栈指针，并使用 c 语言完成延时函数。

3.5.4 实验原理

1. 调试函数

μVision3 具有强大的调试功能，其中之一就是它的调试函数。μVision3 有一个内嵌的调试函数编辑器，可以通过 Debug -> Function Editor 打开。在此编辑器中，可以写编写调试函数并可以编译此函数。调试函数的功能有：

扩展的 μVision3 Debugger 的能力；

产生外部中断；

生成存储器内容文件；

定期更新模拟输入值；

输入串行数据到片上串口；

其它。

具体来说，用户在集成环境与目标板连接时、软件调试过程中以及复位目标板后，有时需要集成环境自动完成一些特定的功能，比如复位目标板、清除看门狗、屏蔽中断寄存器、存储区映射等，这些特定的功能可以通过执行一组命令序列完成。而这一组命令序列可以写在调试函数中。

2. 调试函数的执行方法

编写好调试函数后，可以使用 INCLUDE 命令读取并处理调试函数。若保存调试函数的脚本文件名为 MYFUNCS.INI，在命令窗口中输入如下命令 μVision3 就可读取并解释 MYFUNCS.INI 中的内容。

```
>INCLUDE MYFUNCS.INI
```

MYFUNCS.INI 可以包含调试命令和函数定义，当然也可以通过如下的方式来执行：把此文件放入 Options for Target -> Debug -> Initialization File 内，这样每当启动 μVision3 Debugger 器时，MYFUNCS.INI 中的内容就会被处理。

3. 常用命令介绍

GO

GO 用于指定程序从那里执行及在那里结束。

指令格式：Go startaddr, stopaddr

若 startaddr 被指定，则程序从 startaddr 处开始执行，否则从当前地址开始执行。若 stopaddr 被指定，则程序在 stopaddr 处结束，否则，运行到最近的断点处。

命令举例：

```
G,main //从 main 处开始执行
```

Display

Display 用于显示存储区域的内容。

指令格式：Display startaddr, endaddr

在命令窗口或存储器窗口（若打开）显示从 startaddr 到 endaddr 区域中的内容。可以以各种格式来显示存储器中的内容。

命令举例：

```
D main /* Output beginning at main */
```

其它命令请参考帮助文档 DEBUG COMMAND

3.5.5 实验操作步骤

- 1) 参考 3.1.5 小节实验的操作步骤建立一个新的工程，命名为 3.5_c1；
- 2) 参考 3.1.5 小节实验的步骤和实验参考程序编辑输入源代码，保存文件为 C1.c；
- 3) 按照实验参考程序建立调试脚本文件 DebugInRam.ini；
- 4) 在 Project workspace 工作区中右击 target1->Source Group 1，在弹出菜单中选择“Add file to Group ‘Source’”，在随后弹出的文件选择对话框中，选择刚才建立的源文件 C1.c；
- 5) 选择菜单项 Project ->Build target 或快捷键 F7，生成目标代码；
- 6) 选择菜单项 Debug ->Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。这里使用的是 μVision3 IDE 中的软件仿真器；在 Option For Target 对话框的 Debug 页中将 Initialization 文本框的内容清空；
- 7) 选择菜单项 Debug ->run 或 F5，即可运行代码；
- 8) 在 Output Windows 中的 Command 输入栏中输入“Include DebugInRam.ini”命令；
- 9) 单步执行，通过 memory、register、watch&call stack 等窗口分析判断结果，在 watch 框中输入要观察变量 I 和变量 J 的值，并记录下来。特别注意观察变量 I 的变化并记录下来；
- 10) 结合实验内容和相关资料，学习和尝试一些调试命令，观察程序运行，通过实验；
- 11) 理解和掌握实验后，完成实验练习题。

3.5.6 实验参考程序

1. C 程序

```

/*-----*/
/* NAME      :      C1.C                      */
/* AUTHOR     :      Wuhan R&D Center, Embest    */
/* DESC       :      C EXAMPLE                  */
/* HISTORY    :      1.8.2006                   */
/* MODIFY     :      TYW,Wuhan R&D Center        */
/*-----*/

/*-----/*
           function declare                      */
/*-----*/
void delay(int nTime);
/*-----*/
/* NAME      :      START                      */
/* FUNC       :      ENTRY POINT                */
/* PARA       :      NONE                      */
/* RET        :      NONE                      */
/* MODIFY     :                                  */
/* COMMENT    :                                  */
/*-----*/

main()
{
    int i = 5;
    for( ; ; )
    {
        delay(i);
    }
}

```

```

    }
}
/*-----*/
/* NAME      :      DELAY                      */
/* FUNC      :      DELAY SOME TIME            */
/* PARA      :      nTime -- INPUT             */
/* RET       :      NONE                       */
/* MODIFY    :                                     */
/* COMMENT   :                                     */
/*-----*/
void delay(int nTime)
{
    int i, j = 0;
    for(i = 0; i < nTime; i++)
    {
        for(j = 0; j < 10; j++)
        {
        }
    }
}

```

2. 调试脚本文件

```

//*** <<< Use Configuration !disalbe! Wizard in Context Menu >>> ***
FUNC void Setup (void)
{
    // <o> Program Entry Point
    PC =main;;
}
//LOAD debug_in_RAM\Project.axf INCREMENTAL      // Download
//map 0x000,0x200000 READ WRITE EXEC
map 0x30000000,0x30200000 READ WRITE exec
Setup();                      // Setup for Running
//g, main

```

3.5.7 练习题

参考汇编实验，编写程序，实现从汇编语言中使用 B 或 BL 命令跳转到 C 语言程序的 Main() 函数中执行，并从 Main() 函数中调用 delay() 函数。

3.6 C 语言实验程序二

3.6.1 实验目的

掌握建立基本完整的 ARM 工程，包含启动代码，连接属性的配置等；

了解 ARM7 的启动过程，学会使用 MDK 编写简单的 C 语言程序和汇编启动代码并进行调试；

掌握如何指定代码入口地址与入口点；

掌握通过 memory、register、watch、Local 等窗口分析判断结果。

3.6.2 实验设备

- 硬件：PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.6.3 实验内容

用 c 语言编写延时函数，使用嵌入汇编。

3.6.4 实验原理

1. ARM 异常向量表

当正常的程序执行流程暂时挂起时，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，必须保存当前的处理器状态，以便从异常程序返回时可以继续执行当前的程序。ARM 异常向量表如下：

表 3-4 ARM 异常向量表

| 地址 | 异常 | 入口模式 |
|------------|-----------------------|------|
| 0x00000000 | RESET | 管理 |
| 0x00000004 | Undefined Instruction | 未定义 |
| 0x00000008 | Software interrupt | 管理 |
| 0x0000000C | Prefetch abort | 中止 |
| 0x00000010 | Data abort | 中止 |
| 0x00000014 | Reserved | 保留 |
| 0x00000018 | IRQ | IRQ |
| 0x0000001C | FIQ | FIQ |

处理器允许多个异常同时发生，这时，处理器会按照固定的顺序进行处理，参照下面的异常优先级。

高优先级：

1 ---- Reset

2 ---- Data abort

3 ---- FIQ

4 ---- IRQ

5 ---- Prefetch abort

低优先级：

6 ---- Undefined Instruction, Software interrupt

由上可见，Reset 入口，即为整个程序的实际入口点。因此，我们在编写代码的时候，第一条语句是在 0x00000000 处开始执行的。一般地，我们使用下面的代码：

```

area RESET,code,readonly
entry
    b    Reset_Handler
Undefined_Handler
    b    Undefined_Handler
SWI_Handler
    b    SWI_Handler
Prefetch_Handler
    b    Prefetch_Handler
Abort_Handler
    b    Abort_Handler
    nop
IRQ_Handler
    b    IRQ_Handler
FIQ_Handler

```

```

b      FIQ_Handler
Reset_Handler
    ldr sp, =0x0C002000
    .
    .
    .

```

2. 分散加载文件

Scatter file（分散加载描述文件）用于 LARM 链接器的输入参数，它指定映像文件内部各区域的 download 与运行时位置。LARM 将会根据 scatter file 生成一些区域相关的符号，它们是全局的供用户建立运行时环境时使用。通过这个文件可以指定程序的入口地址。在利用 MDK 进行实际应用程序开发时，常常需要使用道分散加载文件，例如以下情况：

存在复杂的地址映射：例如代码和数据需要分开放在在多个区域。

存在多种存储器类型：例如包含 Flash、ROM、SDRAM、快速 SRAM。需要根据代码与数据的特性把他们放在不同的存储器中，比如中断处理部分放在快速 SRAM 内部来提高响应速度，而把不常用的代码放到速度比较慢的 Flash 内。

函数的地址固定定位：可以利用 Scatter file 实现把某个函数放在固定地址，而不管其应用程序是否已经改变或重新编译。

利用符号确定堆与堆栈：

内存映射的 IO：采用 scatter file 可以实现把某个数据段放在精确的地指处。

因此对于实际的嵌入式系统来说 scatter file 是必不可少的，因为嵌入式系统通常采用了 ROM，RAM，和内存映射的 IO。关于 Scatter file 的相关知识非常多，详细内容可以参考 MDK 所带的帮助，下面给出一个简单实例。

```

LOAD_ROM 0x0000 0x8000
{
    EXEC_ROM 0x0000 0x8000
    {
        *(+RO)
    }
    RAM 0x10000 0x6000
    {
        *(+RW, +ZI)
    }
}

```

这个分散加载描述文件对应的分散加载映像如图 3-12 所示，文件中各项内容的含义分别是：

```

LOAD_ROM(下载区域名称) 0x0000(下载区域起始地址) 0x8000(下载区域最大字节数)
{
    EXEC_ROM(第一执行区域名称) 0x0000(第一执行区域起始地址) 0x8000(第一执行区域最大字节数)
    {
        *(+RO(代码与只读数据))
    }
    RAM(第二执行区域名称) 0x10000(第二执行区域起始地址) 0x6000(第二执行区域最大字节数)
    {
        *(+RW(读写变量), +ZI(未初始化变量))
    }
}

```

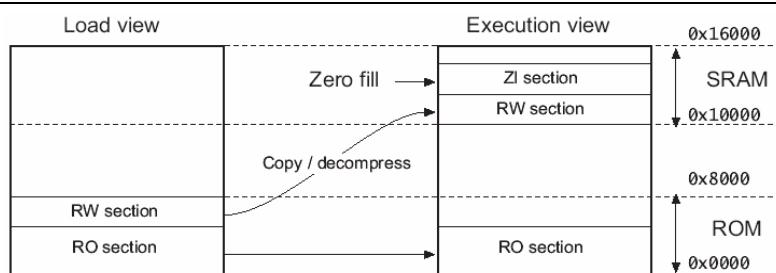


图 3-12 分散加载映像图

3. 内嵌汇编语言

编译 C 时，可以通过 `__asm` 汇编程序说明符调用内嵌汇编程序。说明符后面跟随有一列包含在大括号中的汇编程序指令。例如：

```
__asm
{
instruction [; instruction]
...
[instruction]
}
```

如果两条指令在同一行中，必须用分号将其分隔。如果一条指令占用多行，必须用反斜线符号 (\) 指定续行。可在内嵌汇编语言块内的任意位置处使用 C 注释。可在任何可以使用 C 语句的地方使用 `__asm` 语句。

3.6.5 实验操作步骤

1) 参考 3.1.5 小节实验的操作步骤建立一个新的工程，命名为 3.6_c2，注意在建立工程的过程中添加设备数据库中 S3C44B0 芯片自带的启动代码，也可手动添加启动代码 `startup.s`；

2) 参考 3.1.5 小节实验的步骤和实验参考程序编辑输入源代码，保存文件为 C2.c；

3) 在 Project workspace 工作区中右击 target1->Source Group 1，在弹出菜单中选择 “Add file to Group ‘Source’ ”，在随后弹出的文件选择对话框中，选择刚才建立的源文件 C2.c；

4) 在 Option for Target 对话框 Linker 页 Scatter File 对话框中添加位于 common 文件夹的分散加载描述文件 RuninRAM.sct，文件内容参考 3.3.6 小节。

5) 选择菜单项 Project ->Build target 或快捷键 F7，生成目标代码；

6) 选择菜单项 Debug ->Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。这里使用的是 μ Vision3 IDE 中的软件仿真器；

7) 选择菜单项 Debug ->run 或 F5，即可运行代码；

8) 打开 memory、register、watch、Local 窗口，单步执行，并通过 memory/register/watch/variable 窗口分析判断结果。注意观察程序如何从跳转进主程序 `__main`，在 call stack 窗口观察当前执行函数之间的调用。在 watch 框中输入要观察变量 I 的值，并记录下来。特别注意在 local 窗口观察变量 I 的变化并记录下来。

9) 结合实验内容和相关资料，观察程序运行；

10) 理解和掌握实验后，完成实验练习题。

3.6.6 实验参考程序

C2.c 的源代码

```

/*****
* File:      c2.c
* Author:    WUHAN R&D Center, embest
* Desc:      c language example 2
* History:   XU Liang Ping
*****/
/*****
* name:      _nop_
* func:      The following example embed assemble language
* para:      none
* ret:       none
* modify:
* comment:
*****/
void _nop_()
{
    int temp=0;
    __asm
    {
        mov temp,temp
    }
}

/*****
* name:      delay
* func:      delay time
* para:      none
* ret:       none
* modify:
* comment:
*****/
void delay(void)
{
    int i=0;

    for(; i <= 10; i++)
    {
        _nop_();
    }
}

/*****
* name:      delay10
* func:      delay time
* para:      none
* ret:       none
* modify:
* comment:
*****/

```

```

void delay10(void)
{
    int i;

    for(i = 0; i <= 10; i++)
    {
        delay();
    }
}

/*****
* name:      _main
* func:      c code entry
* para:      none
* ret:       none
* modify:
* comment:
*****/
__main()
{
    // int i = 5;

    for( ; ; )
    {
        delay10();
    }
}

```

Startup.s 的源代码

```

;#*****
;# File: startup.s
;# Author: Wuhan R&D Center, embest
;# Desc: C start up codes;#
;#*****
;#-----*/
;# global symbol define */
;#-----*/
; .global _start

;#-----*/
;# code */
;#-----*/ area

RESET,code,readonly
    entry
;# Set interrupt / exception vectors
    b Reset_Handler
Undefined_Handler
    b Undefined_Handler
SWI_Handler
    b SWI_Handler
Prefetch_Handler
    b Prefetch_Handler

```



```

Abort_Handler
    b      Abort_Handler
    nop                                /* Reserved vector */
IRQ_Handler
    b      IRQ_Handler
FIQ_Handler
    b      FIQ_Handler
Reset_Handler
;   ldr sp, =0x0C002000

;# *****
;# Branch on C code Main function (with interworking)      *

;# Branch must be performed by an interworking call as      *
;# either an ARM or Thumb.main C function must be          *
;# supported. This makes the code not position-independant.*
;# A Branch with link would generate errors                 *
;# *****

        IMPORT __main
        LDR     R0, =__main
        BX      R0
;   # jump to __main()

;# *****
;# * Loop for ever                                          *
;# * End of application. Normally, never occur.          *
;# * Could jump on Software Reset ( B 0x0 ).              *
;# *****

End
    b      End
end

```

CTest2.sct 的源代码

```

; *****
; *** Scatter-Loading Description File generated by uVision ***
; *****

LR_ROM1 0x30000000      {      ; load region
    ER_ROM1 0x30000000 0x00200000 { ; load address = execution address
        *.o (RESET, +First)
;    *(InRoot$$Sections)
        .ANY (+RO)
    }
    RW_RAM1 0x30300000 0x03D00000 { ; RW data
        .ANY (+RW +ZI)
    }
}

```

调试命令脚本文件可使用3.1节的。

3.6.7 练习题

1. 改进 C 语言使用实验一中的练习题，在 C 语言文件中定义全局及局部变量，并在编译链接时使用链接脚本文件，使用 Tools 菜单下的 Disassemble all 产生 objdump 文件，从该文件中观察代码及变量在目标输出代码中的存放情况。

2. 在以上实验例程 C 语言文件中，加入嵌入汇编语言，使用汇编指令实现读写某存储单元的值，初步掌握嵌入汇编语言的使用。

3.7 汇编与 C 语言相互调用实例

3.7.1 实验目的

阅读 S3C44B0 启动代码，观察处理器启动过程；

学会使用 MDK 集成开发环境辅助窗口来分析判断调试过程和结果；

学会在 MDK 集成开发环境中编写、编译与调试汇编和 C 语言相互调用的程序。

3.7.2 实验设备

- 硬件：PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.7.3 实验内容

使用汇编完成一个随机数产生函数，通过 C 语言调用该函数，产生一系列随机数，存放到数组里面。

3.3.4 实验原理

ARM过程调用ATPCS (ARM)

ATPCS是一系列用于规定应用程序之间相互调用的基本规则，这此规则包括：

支持数据栈限制检查；

支持只读段位制无关 (ROPI) ；

支持可读/写段位置无关 (RWPI) ；

支持 ARM 程序和 Thumb 程序的混合使用；

处理浮点运算。

使用以上规定的 ATPCS 规则时，应用程序必须遵守如下：

程序编写遵守 ATPCS；

变量传递以中间寄存器和数据栈完成；

汇编器使用 -apcs 开关选项。

关于其他ATPCS规则，用户可以参考ARM处理器相关书籍或登录ARM公司网站。

程序只要遵守ATPCS相应规则，就可以使用不同的源代码编写程序。程序间的相互调用最主要的是解决参数传递问题。应用程序之间使用中间寄存器及数据栈来传递参数，其中，第一个到第四个参数使用R0-R3，多于四个参数的使用数据栈进行传递。这样，接收参数的应用程序必须知道参数的个数。

但是，在应用程序被调用时，一般无从知道所传递参数的个数。不同语言编写的应用程序在调用时可以自定义参数传递的约定，使用具有一定意义的形式来传递，可以很好地解决参数个数的问题。常用的方法是把第一个或最后一个参数作为参数个数（包括个数本身）传递给应用程序。

ATPCS中寄存器的对应关系如表3-5所列：

表 3-5 ATPCS 规则中寄存器列表

| ARM 寄存器 | ATPCS 别名 | ATPCS 寄存器说明 |
|------------|------------|----------------------------------|
| R0-R3 4 | <==> a1-a4 | 参数/结果/scratch 寄存器 1-4 |
| R4 | <==> v1 | 局部变量寄存器 1 |
| R5 | <==> v2 | 局部变量寄存器 2 |
| R6 | <==> v3 | 局部变量寄存器 3 |
| R7 | <==> v4、wr | 局部变量寄存器 4 Thumb 状态工作寄存器 |
| R8 5 | <==> v5 | ARM 状态局部变量寄存器 |
| R9 | <==> v6、sb | ARM 状态局部变量寄存器 6 RWPI 的静态基址寄存器 |
| R10 | <==> v7、sl | ARM 状态局部变量寄存器 7 数据栈限制指针寄存器 |
| R11 | <==> v8 | ARM 状态局部变量寄存器 8 |
| R12 寄存器 | <==> ip | 子程序内部调用的临时 (scratch) 寄存器 |
| R13 | <==> sp | 数据栈指针寄存器 |
| R14 | <==> lr | 链接寄存器 |
| R15 | <==> PC | 程序计数器 |

3.7.5 实验操作步骤

1) 参考 3.1.5 小节实验的操作步骤建立一个新的工程，命名为 3.7_explasm，注意在建立工程的过程中添加设备数据库中 S3C44B0 芯片自带的启动代码，也可手动添加启动代码 startup.s；

2) 参考 3.1.5 小节实验的步骤和实验参考程序编辑输入源代码，保存文件为 randtest.c 和 random.s；

3) 在 Project workspace 工作区中右击 target1->Source Group 1，在弹出菜单中选择“Add file to Group ‘Source’”，在随后弹出的文件选择对话框中，选择刚才建立的源文件 randtest.c 和 random.s；

4) 在 Option for Target 对话框 Linker 页 Scatter File 对话框中添加分散加载描述文件 RuninRAM.sct，文件内容与 3.3.6 小节分散加载文件相同。

5) 选择菜单项 Project ->Build target 或快捷键 F7，生成目标代码；

6) 选择菜单项 Debug ->Start/Stop Debug Session 或快捷键 Ctrl+F5，即可进入调试模式。这里使用的是 μVision3 IDE 中的软件仿真器；

7) 选择菜单项 Debug ->run 或 F5，即可运行代码；

8) 打开 memory、register、watch、Local 窗口，单步执行，并通过 memory、register、watch、variable 窗口分析判断结果。注意观察程序如何从跳转进主程序__main，在 call stack 窗口观察当前执行函数之间的调用。

9) 结合实验内容和相关资料，观察程序运行；

10) 理解和掌握实验后，完成实验练习题。

3.7.6 实验参考程序

(1) randtest.c 参考源代码：

```

/*****
 * File:      randtest.c
 * Author:    Wuhan R&D Center, embest
 * Desc:      Random number generator demo program
 *            Calls assembler function 'randomnumber' defined in random.s
 * History:
 *****/

/*-----*/
/*
/*            extern function
*/
/*-----*/
extern unsigned int randomnumber( void );

/*****
 * name:      main
 * func:      c code entry
 * para:      none
 * ret:       none
 * modify:
 * comment:
 *****/

main()
{
    unsigned int i,nTemp;
    unsigned int unRandom[10];

    for( i = 0; i < 10; i++ )
    {
        nTemp = randomnumber();
        unRandom[i] = nTemp;
    }

    return(0);
}

```

(2) random.s 参考源代码：

```

;#*****
;# File:      random.s
;# Author:    embest
;# Desc:      Random number generator

```

```

*
;# This uses a 33-bit feedback shift register to generate a pseudo-randomly *
;# ordered sequence of numbers which repeats in a cycle of length 2^33 - 1 *
;# NOTE: randomseed should not be set to 0, otherwise a zero will be generated *
;# continuously (not particularly random!). *
;# This is a good application of direct ARM assembler, because the 33-bit *
;# shift register can be implemented using RRX (which uses reg + carry). *
;# An ANSI C version would be less efficient as the compiler would not use RRX.*
;# AREA |Random$$code|, CODE, READONLY *
;# History:

;#*****

;/*-----*/
;/*
;#                                global symbol define
*/
;/*-----*/
EXPORT randomnumber
EXPORT seed

;/*-----*/
;/*                                code */
;/*-----*/

AREA RAND,CODE,READONLY
randomnumber
;# on exit:
;# a1 = low 32-bits of pseudo-random number
;# a2 = high bit (if you want to know it)
ldr    ip, seedpointer
ldmia  ip, {a1, a2}
tst    a2, a2, lsr#1      ;/* to bit into carry */
movs   a3, a1, rrx        ;/* 33-bit rotate right */
adc    a2, a2, a2         ;/* carry into LSB of a2 */
eor     a3, a3, a1, lsl#12 ;/* (involved!) */
eor     a1, a3, a3, lsr#20 ;/* (similarly involved!)*
stmia  ip, {a1, a2}
mov     pc, lr

seedpointer
DCD     seed
seed     DCD     0x55555555
          DCD     0x55555555

END

```

3.7.7 练习题

参考3.3.6小节中“实验A参考源代码”，改进3.6节“C语言程序实验2”的练习题例程，使用嵌入汇编语言实现 $R1+R2=R0$ 的加法运算，运算结果保存在 $R0$ 。调试时打开Register窗口，观察嵌入汇编语句运行前后 $R0$ 、 $R1$ 、 $R2$ 、 SP 寄存器以及ATPCS寄存器对应的ARM寄存器内容的变化。

3.8 综合实验

3.8.1 实验目的

掌握处理器启动配置过程；

掌握使用 μ Vision IDE 辅助信息窗口来分析判断调试过程和结果，学会查找软件调试时的故障或错误；

掌握使用 μ Vision IDE 开发工具进行软件开发与调试的常用技巧。

3.8.2 实验设备

- 硬件：PC 机。
- 软件： μ Vision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

3.8.3 实验内容

完成一个完整的工程，要求包含启动代码，汇编函数和 C 文件，而且 C 文件包含 ARM 函数和 Thumb 函数，并可以相互调用。

3.8.4 实验原理

1. μ Vision IDE 开发调试辅助窗口

使用 μ Vision IDE 嵌入式开发环境，用户可以使用源代码编辑窗口编写源程序文件、使用反汇编窗口观察程序代码的执行、使用 Register 窗口观察程序操作及 CPU 状态、使用外围寄存器窗口观察当前处理器的配置、使用 Memory 窗口观察内存单元使用情况、使用 Watch 或 Variables 窗口观察程序变量、使用操作控制台执行特殊命令、使用性能分析仪窗口分析代码的性能、使用逻辑分析仪模拟处理器及其外设的时序、使用串行端口窗口模拟串口的工作，加上调试状态下丰富的右键菜单功能，用户可以使用 μ Vision IDE 实现或发现任何一部分应用软件、修改任何一个开发或运行时的错误。

结合 μ Vision IDE 随安装软件自带的丰富的样例程序，配合高性能的 Ulink2 仿真器，可以使用户把主要精力放在项目软件开发上。

2. 生成 HEX 文件

在 μ Vision 3 IDE 中选择菜单页 Project-Options for Target-Output，在弹出的对话框中，选择 Create HEX File 单选项即可；

单击“Select Folder for Objects...”按钮，为生成的十六进制文件选择存储路径；

在“Name of Executable”文本框中输入生成十六进制文件的名字；

选择 Create HEX File，并单击“确认”按钮；

重新编译该工程文件就可以得到十六进制文件。

3.8.5 实验操作步骤

1. 打开 Keil uVision3，在菜单中选择“Project->New->uVision project”创建新的工程，命名为 interwork，CPU 选择为 Samsung 的 S3C44B0X，接着会弹出如图 3-6 所示的对话框，选择“否”，则不会将 Samsung S3C44B0X 的启动代码拷贝到新工程。

2. 参考 3.8.6 的参考程序创建 C 语言程序 thumb.c、arm.c 和汇编语言程序 random.s，并将这些程序添加到新建的工程中。

3. 在菜单中选择“Project->Options for Target 'Target 1'”则会弹出如图 3-8 所示的工程配置界面，依据处理器及目标板的实际配置对工程进行配置。

4. 用鼠标右键点击“Project Workspace”中的文件“thumb.c”，如图 3-12 所示，选择“Options for File 'thumb.c'”，则出现文件“thumb.c”的配置对话框，在该对话框中选择“C/C++”标签，如图 3-13 所示，使“Thumb Mode”有效。“Thumb Mode”有效时在配置对话框下面的“Compiler control string”右边的框中会有“--thumb”选项。工程中的其它文件采用默认配置。

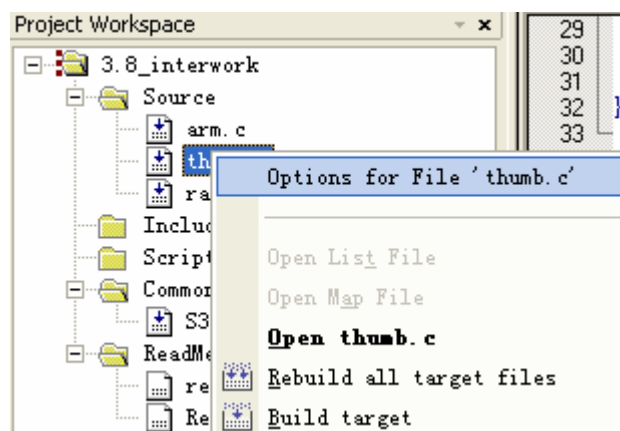


图 3-12 打开文件配置对话框

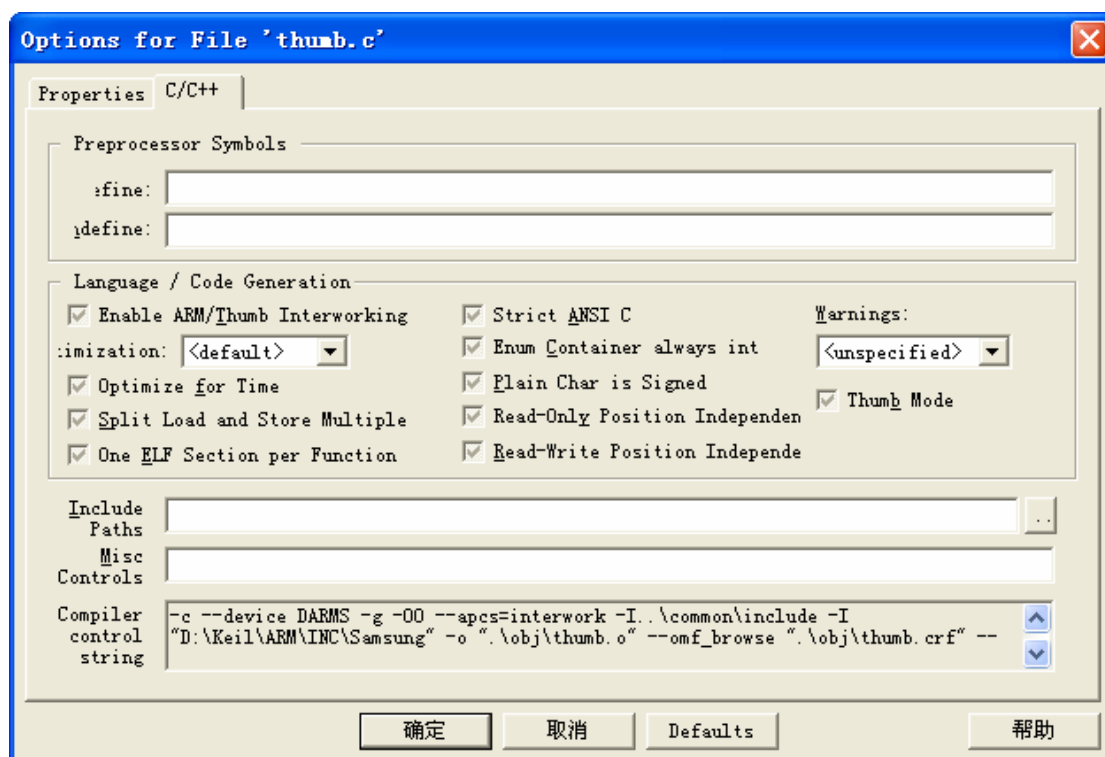


图 3-13 文件配置选项

5. 对工程进行编译、调试，使工程能够正确执行。
6. 在菜单中选择“Debug->Start/Stop Debug Session”或使用快捷键“Ctrl + F5”进入调试状态，单步执行程序，通过寄存器窗口、存储器窗口等分析判断运行结果。
7. 单步跟踪 ARM 函数与 Thumb 函数相互调用的反汇编代码，分析 arm 内核的状态切换过程；
8. 理解和掌握实验后，参考样例程序完成实验练习题。

3.8.6 参考程序

1. arm.c 参考程序:

```

/*****
* File:   arm.c
* Author:   WuHan R&D Center, Embest
* Desc:   arm instruction ,c program
* History:
*****/

/*-----*/
/*   extern variable           */
/*-----*/
extern char szArm[20];
extern int randomnumber(void);

/*****
* name:      delaya
* func:      delay function of arm instructon
* para:      nTime---input
* ret:       none
* modify:
* comment:
*****/
static void delaya(int nTime)
{
    int i, j, k;
    k = 0;

    for(i = 0; i < nTime; i++)
    {
        for(j = 0; j < 10; j++)
            k++;
    }
}

/*****
* name:      arm_function
* func:      example
* para:      none
* ret:       none
* modify:
* comment:
*****/
void arm_function(void)
{
    int i;

```



```

int nLoop;
unsigned int unRandom;
char *p = "Hello from ARM world";

for(i =0; i < 20; i++)
    szArm[i] = (*p++);

delaya(2);

for( nLoop = 0; nLoop < 10; nLoop++ )
{
    unRandom = randomnumber();
}
}

```

2. random.s 参考程序:

```

;#*****
;# Filef°    random.s                                *
;# Author:   embest                                  *
;# Descf°    Random                                number                generator
*
;#          This uses a 33-bit feedback shift register to generate a pseudo-randomly
*
;#          ordered sequence of numbers which repeats in a cycle of length 2^33 - 1
*
;#          NOTE: randomseed should not be set to 0, otherwise a zero will be generated
*
;#          continuously (not particularly random!).
*
;#          This is a good application of direct ARM assembler, because the 33-bit
*
;#          shift register can be implemented using RRX (which uses reg + carry).
*
;#          An ANSI C version would be less efficient as the compiler would not use RRX.
*
;#          AREA                |Random$$code|,    CODE,    READONLY
*
;# History:
*
;#*****

;/*----- */
;/*          global symbol define
*
;/*----- */
global randomnumber

```

```

global seed
code32
; /*----- */
; /*                                code
; /*                                */
; /*----- */
area tcode , code , readonly

EXPORT randomnumber
randomnumber
ldr    ip, seedpointer
ldmia  ip, {a1, a2}
tst    a2, a2, lsr #1      ; /* to bit into carry */
movs   a3, a1, rrx         ; /* 33-bit rotate right */
adc    a2, a2, a2          ; /* carry into LSB of a2 */
eor    a3, a3, a1, lsl #12 ; /* (involved!) */
eor    a1, a3, a3, lsr #20 ; /* (similarly involved!) */
stmia  ip, {a1, a2}
BX lr

seedpointer
DCD    seed

; area tdata, data, readwrite
seed
DCD    0x55555555
DCD    0x55555555

END

```

3. thumb.c 参考程序:

```

/*****
* File: thumb.c
* Author: Wuhan R&D Center, Embest
* Desc: c program of Thumb instruction
* History:
*****/

; /*----- */
; /* global variables */
; /*----- */
char szArm[22];
char szThumb[22];
unsigned long ulTemp = 0;

; /*----- */
; /* extern function */

```

```

/*-----*/
extern void arm_function(void);

/*****
* name:      delayt
* func:      delay functin with Thumb instruction
* para:      nTime --- input
* ret:       none
* modify:
* comment:
*****/
static void delayt(int nTime)
{
    int i, j, k;

    k = 0;
    for(i = 0; i < nTime; i++)
    {
        for(j = 0; j < 10; j++)
            k++;
    }
}

/*****
* name:      thumb_function
* func:      The following example use Thumb instruction
* para:      none
* ret:       none
* modify:
* comment:
*****/
int thumb_function(void)
{
    int i;
    char * p = "Hello from Thumb World";

    ulTemp++;
    arm_function();

    delayt(2);

    for(i = 0; i < 22; i++)
        szThumb[i] = (*p++);

    return 0;
}

```

3.8.7 练习题

1. 阅读 Keil uVision3 安装目录下的启动文件 S3C44BOX.s，尽量理解每一条语句的功能；
2. 编写一个汇编文件和一个 C 语言文件，实现从汇编语言中传递简单数学运算参数给由 C 语言程序编写的简单数学运算函数，并从 C 语言程序中返回运算结果；新建工程名为 smath 的工程，同时加入 S3C44B0A.s 文件，参照基础实验工程配置，对新建工程进行配置；编译、调试新工程；并在软件模拟器中下载到 0x0c000000 处执行，跟踪程序的执行。

第四章 基本接口实验

4.1 存储器实验

4.1.1 实验目的

- 通过实验熟悉 ARM 的内部存储空间分配。
- 熟悉使用寄存器配置存储空间的方法。
- 掌握对存储区进行访问的方法。

4.1.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.1.3 实验内容

掌握 S3C44BOX 处理器对存储空间的配置和读写访问的方法：

- 熟练使用命令脚本文件对 ARM 存储控制寄存器进行正确配置；
- 使用汇编编程，对 RAM 按字、半字和字节读写；
- C 语言编程，对 RAM 按字、半字和字节读写。

4.1.4 实验原理

1. 存储控制器

S3C44BOX 处理器的存储控制器可以为片外存储器访问提供必要的控制信号，它主要包括以下特点：

- 支持大、小端模式（通过外部引脚来选择）
- 地址空间：包含 8 个地址空间，每个地址空间的大小为 32M 字节，总共有 256M 字节的地址空间。
- 所有地址空间都可以通过编程设置为 8 位、16 位或 32 位对准访问。
- 8 个地址空间中，6 个地址空间可以用于 ROM、SRAM 等存储器，2 个用于 ROM、SRAM、FP/EDO/SDRAM 等存储器。
- 7 个地址空间的起始地址及空间大小是固定的。
- 1 个地址空间的起始地址和空间大小是可变的。
- 所有存储器空间的访问周期都可以通过编程配置。
- 提供外部扩展总线的等待周期。
- 支持 DRAM/SDARM 自动刷新。
- 支持地址对称或非地址对称的 DRAM。

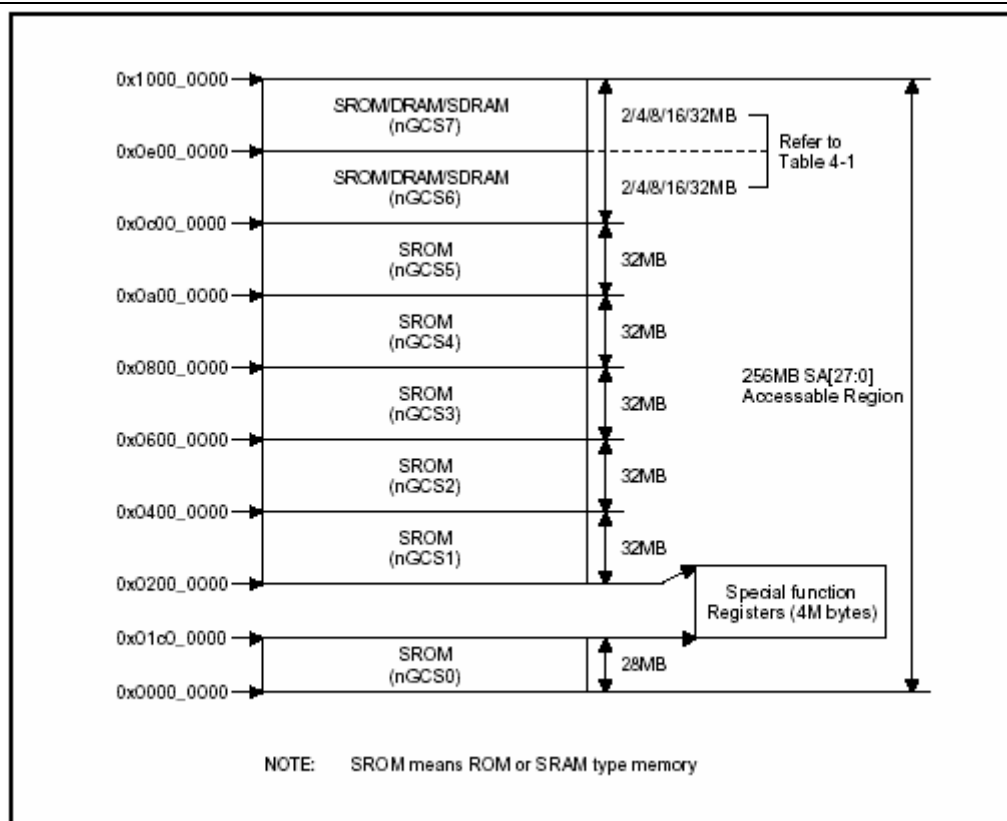


图 4-1 S3C44BOX 复位后的存储器地址分配

图 4-1 为 S3C44BOX 复位后的存储器地址分配图。从图中可以看出，特殊功能寄存器位于 0x01c00000 到 0x02000000 的 4M 空间内。Bank0-Bank5 的起始地址和空间大小都是固定的，Bank6 的起始地址是固定的，但是空间大小和 Bank7 一样是可变的，可以配置为 2/4/8/16/32M。Bank6 和 Bank7 的详细地址和空间大小的关系可以参考表 4-1

表 4-1 Bank6/Bank7 地址

| Address | 2MB | 4MB | 8MB | 16MB | 32MB |
|---------------|------------|------------|------------|------------|------------|
| Bank 6 | | | | | |
| Start address | 0xc00_0000 | 0xc00_0000 | 0xc00_0000 | 0xc00_0000 | 0xc00_0000 |
| End address | 0xc1f_fff | 0xc3f_fff | 0xc7f_fff | 0xcff_fff | 0xdff_fff |
| Bank 7 | | | | | |
| Start address | 0xc20_0000 | 0xc40_0000 | 0xc80_0000 | 0xd00_0000 | 0xe00_0000 |
| End address | 0xc3f_fff | 0xc7f_fff | 0xcff_fff | 0xdff_fff | 0xff_fff |

大/小 ENDIAN 模式选择

处理器复位时（nRESET 为低），通过 ENDIAN 引脚选择所使用的 ENDIAN 模式。ENDIAN 引脚通过下拉电阻与 Vss 连接，定义为 Little endian 模式；ENDIAN 引脚通过上拉电阻和 Vdd 连接，则定义为 Big endian 模式。如表 4-2 所示。

表 4-2 大/小 endian 模式

| ENDIAN Input @Reset | ENDIAN Mode |
|---------------------|---------------|
| 0 | Little endian |
| 1 | Big endian |

BANK0 总线宽度

BANK0 (nGCS0) 的数据总线宽度可以配置为 8 位、16 位或 32 位。因为 BANK0 为启动 ROM (映射地址为 0X00000000) 所在的空间, 所以必须在第一次访问 ROM 前设置 BANK0 数据宽度, 该数据宽度是由复位后 OM[1:0] 的逻辑电平决定的, 表 4-3 所示。

表 4-3 数据宽度选择

| OM1 (Operating Mode 1) | OM0 (Operating Mode 0) | Booting ROM Data width |
|------------------------|------------------------|------------------------|
| 0 | 0 | 8-bit |
| 0 | 1 | 16-bit |
| 1 | 0 | 32-bit |
| 1 | 1 | Test Mode |

- **存储器控制专用寄存器**

总线宽度/等待控制寄存器 (BWSCON)

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|--|-------------|
| BWSCON | 0x01C80000 | R/W | Bus Width & Wait Status Control Register | 0x000000 |

寄存器各位功能:

[ENDIAN]: 只读, 指示系统选定的大/小端模式, 0 表示小端模式, 1 表示大端模式。

[DWi]: $i=0\sim7$, 其中 DW0 为只读, 因为 bank0 数据总线宽度在复位后已经由 OM[1:0] 的电平决定。DW1~DW7 可写, 用于配置 bank1~bank7 的数据总线宽度, 00 表示 8 位数据总线宽度, 01 表示 16 位数据总线宽度, 10 表示 32 位数据总线宽度。

[SWi]: $i=1\sim7$, 写入 0 则对应的 banki 等待状态不使用, 写入 1 则对应的 banki 等待状态使能。

[STi]: $i=1\sim7$, 决定 SRAM 是否使用 UB/LB。0 表示不使用 UB/LB, 引脚[14:11]定义为 nWB E[3:0]; 1 表示使用 UB/LB, 引脚[14:11]定义为 nBE[3:0]。

Bank 控制寄存器(BANKCONn: nGCS0-nGCS5)

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|-------------------------|-------------|
| BANKCON0 | 0x01C80004 | R/W | Bank 0 control register | 0x0700 |
| BANKCON1 | 0x01C80008 | R/W | Bank 1 control register | 0x0700 |
| BANKCON2 | 0x01C8000C | R/W | Bank 2 control register | 0x0700 |
| BANKCON3 | 0x01C80010 | R/W | Bank 3 control register | 0x0700 |
| BANKCON4 | 0x01C80014 | R/W | Bank 4 control register | 0x0700 |
| BANKCON5 | 0x01C80018 | R/W | Bank 5 control register | 0x0700 |

Bank 控制寄存器(BANKCONn: nGCS6-nGCS7)

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|-------------------------|-------------|
| BANKCON6 | 0x01C8001C | R/W | Bank 6 control register | 0x18008 |
| BANKCON7 | 0x01C80020 | R/W | Bank 7 control register | 0x18008 |

刷新控制寄存器 (REFRESH)

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|-------------------------------------|-------------|
| REFRESH | 0x01C80024 | R/W | DRAM/SDRAM refresh control register | 0xac0000 |

BANK 大小寄存器 (BANKSIZE)

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|-----------------------------|-------------|
| BANKSIZE | 0x01C80028 | R/W | Flexible bank size register | 0x0 |

模式设置寄存器 (MRSR)

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|----------------------------------|-------------|
| MRSRB6 | 0x01C8002C | R/W | Mode register set register bank6 | xxx |
| MRSRB7 | 0x01C80030 | R/W | Mode register set register bank7 | xxx |

以上寄存器的详细定义可以查看 S3C44BOX 的数据手册。

下面列举了 13 个存储控制寄存器的配置示例：

```
#define rBWSCON                (*(volatile unsigned *)0x1c80000)
#define rBANKCON0              (*(volatile unsigned *)0x1c80004)
#define rBANKCON1              (*(volatile unsigned *)0x1c80008)
#define rBANKCON2              (*(volatile unsigned *)0x1c8000c)
#define rBANKCON3              (*(volatile unsigned *)0x1c80010)
#define rBANKCON4              (*(volatile unsigned *)0x1c80014)
#define rBANKCON5              (*(volatile unsigned *)0x1c80018)
#define rBANKCON6              (*(volatile unsigned *)0x1c8001c)
#define rBANKCON7              (*(volatile unsigned *)0x1c80020)
#define rREFRESH               (*(volatile unsigned *)0x1c80024)
#define rBANKSIZE              (*(volatile unsigned *)0x1c80028)
#define rMRSRB6               (*(volatile unsigned *)0x1c8002c)
#define rMRSRB7               (*(volatile unsigned *)0x1c80030)

BANKCON0_Val    EQU    0x00000600
BANKCON1_Val    EQU    0x00007FFC
BANKCON2_Val    EQU    0x00007FFC
BANKCON3_Val    EQU    0x00007FFC
BANKCON4_Val    EQU    0x00007FFC
BANKCON5_Val    EQU    0x00007FFC
BANKCON6_Val    EQU    0x00018000
BANKCON7_Val    EQU    0x00018000
BWSCON_Val      EQU    0x11119102
REFRESH_Val     EQU    0x00860459
BANKSIZE_Val    EQU    0x00000010
MRSRB6_Val      EQU    0x00000020
MRSRB7_Val      EQU    0x00000020
```

观察上面寄存器介绍中的寄存器地址可以发现，13 个寄存器分布在从 0x01c80000 开始的连续地址空间，所以上面的程序可以利用指令 “stmia r0, {r1-r13}” 实现将配置好的寄存器的值依次写入到相应的寄存器中。

存储器（SROM/DRAM/SDRAM）地址线连接如表 4-4 所示，数据宽度不同，连接方式也不同。

表 4-4 存储器地址线连接

| MEMORY ADDR. PIN | S3C44B0X ADDR. @ 8-bit DATA BUS | S3C44B0X ADDR. @ 16-bit DATA BUS | S3C44B0X ADDR. @ 32-bit DATA BUS |
|------------------|------------------------------------|-------------------------------------|-------------------------------------|
| A0 | A0 | A1 | A2 |
| A1 | A1 | A2 | A3 |
| A2 | A2 | A3 | A4 |
| A3 | A3 | A4 | A5 |
| ... | ... | ... | ... |

● 片选信号设置

Embest EduKit-III 实验板的片选信号设置如表 4-5 所示：

表 4-5 片选信号设置

| 片选信号 | | | | 选择的接口或器件 | 片选控制寄存器 |
|-------|-----|-----|-----|------------|----------|
| NGCS0 | | | | FLASH | BANKCON0 |
| NGCS6 | | | | SDRAM | BANKCON6 |
| nGCS1 | A22 | A21 | A20 | | BANKCON1 |
| | 0 | 0 | 0 | USB_CS | |
| | 0 | 0 | 1 | CAN_CS | |
| | 0 | 1 | 0 | CF_CS0 | |
| | 0 | 1 | 1 | CF_CS1 | |
| | 1 | 0 | 0 | | |
| | 1 | 0 | 1 | LCD_CD | |
| | 1 | 1 | 0 | | |
| nGCS2 | A22 | A21 | A20 | | BANKCON2 |
| | 0 | 0 | 0 | 扩展输出寄存器 1 | |
| | 0 | 0 | 1 | 扩展输入寄存器 1 | |
| | 0 | 1 | 0 | 扩展输入寄存器 2 | |
| | 0 | 1 | 1 | CF_MMRD/WR | |
| | 1 | 0 | 0 | CF_IORD/WR | |
| | | | | | |

| | | | | | |
|-------|---|---|---|----------|----------|
| | 1 | 0 | 1 | NO USE | |
| | 1 | 1 | 0 | NO USE | |
| | 1 | 1 | 1 | NO USE | |
| NGCS3 | | | | ETHERNET | BANKCON3 |

● 外围地址空间分配

板上外围地址空间分配如表 4-6 所列：

表 4-6 外围地址空间分配

| 外围器件 | 片选信号 | 控制寄存器 | S3C44B0X 地址空间 |
|-------------|--------|----------|-------------------------|
| FLASH | NGCS0 | BANKCON0 | 0X0000_0000~0X01BF_FFFF |
| SDRAM | NGCS6 | BANKCON6 | 0X0C00_0000~0X0DF_FFFF |
| USB 接口 | USB_CS | BANKCON1 | 0X0200_0000~0X020F_FFFF |
| CAN 接口 | CAN_CS | BANKCON1 | 0X0210_0000~0X021F_FFFF |
| CF_CS0 | CF_CS0 | BANKCON1 | 0X0220_0000~0X022F_FFFF |
| CF_CS1 | CF_CS1 | BANKCON1 | 0X0230_0000~0X023F_FFFF |
| IDE(IOR/W) | | BANKCON1 | 0X0240_0000~0X024F_FFFF |
| LCD 显示选通 | LCD_CD | BANKCON1 | 0X0250_0000~0X025F_FFFF |
| NO USE | | BANKCON1 | 0X0260_0000~0X03FF_FFFF |
| 扩展输出寄存器 1 | | BANKCON2 | 0X0400_0000~0X040F_FFFF |
| 扩展输入寄存器 1 | | BANKCON2 | 0X0410_0000~0X041F_FFFF |
| 扩展输入寄存器 2 | | BANKCON2 | 0X0420_0000~0X042F_FFFF |
| CF_MMRD/W R | | BANKCON2 | 0X0430_0000~0X043F_FFFF |
| CF_IORD/WR | | BANKCON2 | 0X0440_0000~0X044F_FFFF |
| NO USE | | BANKCON2 | 0X0450_0000~0X05FF_FFFF |
| Ethernet | NGCS3 | BANKCON3 | 0X0600_0000~0X07FF_FFFF |
| NO USE | NGCS4 | BANKCON4 | 0X0800_0000~0X09FF_FFFF |
| NO USE | NGCS5 | BANKCON5 | 0X0A00_0000~0X0BFF_FFFF |

| | | | |
|--------|-------|----------|-------------------------|
| NO USE | NGCS7 | BANKCON7 | 0X0E00_0000~0X1FFF_FFFF |
|--------|-------|----------|-------------------------|

2. 电路设计

Embest EduKit-III 实验板上的存储系统包括一片 $1\text{M} \times 16\text{bit}$ 的 Flash (SST39VF160) 和一片 $4\text{M} \times 16\text{bit}$ 的 SDRAM (HY57V65160B)。

如图 4-2 Flash 连接电路所示, 处理器是通过片选 nGCS0 与片外 Flash 芯片连接。由于是 16bit 的 Flash, 所以用 CPU 的地址线 A1-A20 来分别和 Flash 的地址线 A0-A19 连接。Flash 的地址空间是从 $0\text{x}00000000 \sim 0\text{x}00200000$ 。

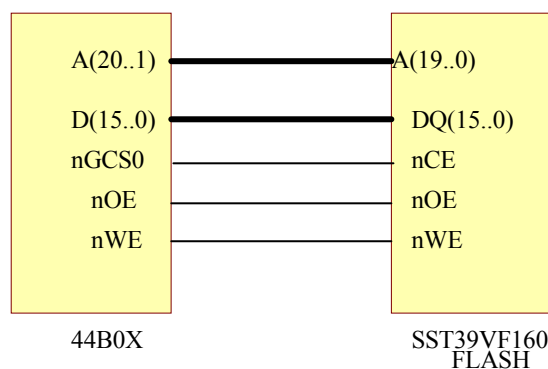


图 4-2 Flash 连接电路

如图 4-3 SDRAM 连接电路所示, SDRAM 分成 4 个 BANK, 每个 BANK 的容量为 $1\text{M} \times 16\text{bit}$ 。BANK 的地址由 BA1、BA0 决定, 00 对应 BANK0, 01 对应 BANK1, 10 对应 BANK2, 11 对应 BANK3。在每个 BANK 中, 分别用行地址脉冲选通 RAS 和列地址脉冲选通 CAS 进行寻址。本实验板还设置跳线, 可以为用户升级内存容量至 $4 \times 2\text{M} \times 16\text{bit}$ 。具体方法为使 SDRAM 的 BA0、BA1 分别接至 CPU 的 A21、A22, A23 脚。SDRAM 由 MCU 专用 SDRAM 片选信号 nSCS0 选通, 地址空间从 $0\text{x}0\text{C}000000 \sim 0\text{x}0\text{C}7\text{FFFFF}$ 。

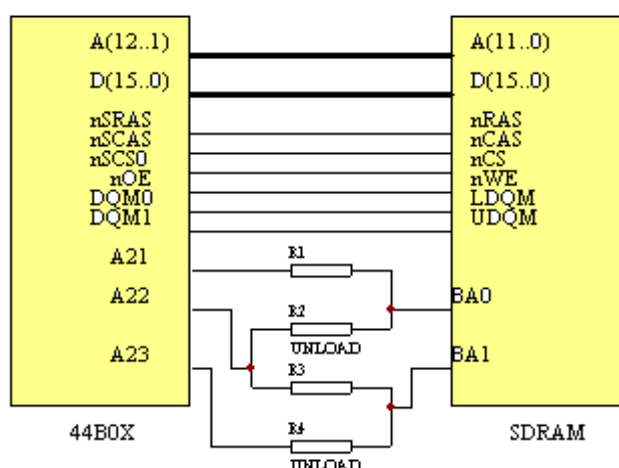


图 4-3 SDRAM 连接电路

4.1.5 实验操作步骤

1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.1_memory_test 子目录下的 memory_test. Uv2 例程，编译链接工程；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 打开 Memory 窗口，点击 Memory1 在地址输入栏中输入 0x0c010000，点击 Memory2 在地址输入栏中输入 0x0c020000；
- 6) 在工程管理窗口中双击 main.c 就会打开该文件，在 mem_test()、s_ram_test()、c_ram_test()处设置断点后，点击 Debug 菜单 Go 键运行程序；
- 7) 当程序停留到断点后点击 Debug 菜单下的 Step 或 F11 键，进入函数体程序，再点击 Step over 或 F10 键执行程序，并在 Memory 窗口观察地址单元 0x0c010000 和 0x0c020000 内容的变化；
- 8) 结合实验内容和实验原理部分，掌握汇编语言和高级语言程序访问 RAM 指令的使用方法。

4. 观察实验结果

在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...

Embest Arm EduKit III Evaluation Board

Memory Read/Write Access Test Example

Memory read  base Address : 0xc010000

Memory write base Address : 0xc020000

Memory Read/Write (ASM code,100Bytes) Test.
```

Memory Read/Write (C code,100Bytes) Test.

Access Memory (Word) Times : 25

Access Memory (half Word) Times : 50

Access Memory (Byte) Times : 100

Memory Test Success!

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

4.1.6 实验参考程序

```

;*****
;
; * name:      s_ram_test
; * func:      access memory by word, hard word, byte
; * para:      none
; * ret:       none
; * modify:
;
;*****/

        AREA text,CODE,READONLY
        export
s_ram_test
        stmdb    sp!, {r2-r4, lr}      ;/* push r0-r11, ip, lr */
        bl      init_ram
;@ read from RW_BASE and write them into RW_TARGET
        ldr      r2,=RW_BASE
        ldr      r3,[r2]                ;/* Read by Word */
        ldr      r2,=RW_TARGET
        str      r3,[r2]                ;/* Write by Word */

        bl      init_ram
;@ read from RW_BASE and write them into RW_TARGET
        ldr      r2,=RW_BASE
        ldrrh    r3,[r2],#2              ;/* Read by half Word */
        ldrrh    r4,[r2]                ;/* Read next half Word */
        ldr      r2,=RW_TARGET
        strrh    r3,[r2],#2              ;/* Write by half Word */
        strrh    r4,[r2]                ;/* Write next half Word */

;@ read from RW_BASE
        ldr      r2,=RW_BASE
        ldrrb    r3,[r2]                ;/* Read by Byte */

```

```

;@ write 0xDDBB2211 into RW_TARGET
ldr    r2,=RW_TARGET
ldr    r3,=0xDD
strb   r3,[r2],#1           ;/* Write by Byte */
ldr    r3,=0xBB
strb   r3,[r2],#1
ldr    r3,=0x22
strb   r3,[r2],#1
ldr    r3,=0x11
strb   r3,[r2]

ldmia   sp!, {r2-r4, lr}    ;/* pop r0-r11, ip, lr */
bx      lr                  ;/* return */
/*****

* name:      c_ram_test
* func:      access memory by word, hard word, byte
* para:      none
* ret:       none
* modify:
*****/

void c_ram_test(void)
{
    int i,nStep;

    // Access by Word.
    nStep = sizeof(int);
    for(i = 0; i < RW_NUM/nStep; i++)
    {
        (*(int*)(RW_BASE + i*nStep))    = 0x45563430;           // write memory
        (*(int*)(RW_TARGET + i*nStep)) = (*(int*)(RW_BASE + i*nStep)); // read memory
    }
    uart_printf("  Access Memory (Word) Times : %d\n",i);

    // Access by half Word.
    nStep = sizeof(short);
    for(i = 0; i < RW_NUM/nStep; i++)
    {
        (*(short*)(RW_BASE + i*nStep))  = 0x4B4F;              // write memory
        (*(short*)(RW_TARGET + i*nStep)) = (*(short*)(RW_BASE + i*nStep)); // read memory
    }
    uart_printf("  Access Memory (half Word) Times : %d\n",i);

    // Access by Byte.
    nStep = sizeof(char);

```

```

    for(i = 0; i < RW_NUM/nStep; i++)
    {
        (*(char*)(RW_BASE + i*nStep)) = 0x59;                // write memory
        (*(char*)(RW_TARGET + i*nStep)) = (*(char*)(RW_BASE + i*nStep)); // read memory
    }
    uart_printf("  Access Memory (Byte) Times : %d\n",i);
}
}

```

4.1.7 练习题

分别使用汇编语言及高级语言编写程序，实现对 RAM 连续地址空间的读和写。

4.2 I/O 接口实验

4.2.1 实验目的

- 掌握 S3C44B0X 芯片的 I/O 口控制寄存器的配置。
- 通过实验掌握 ARM 芯片使用 I/O 口控制 LED 显示。
- 了解 ARM 芯片中复用 I/O 口的使用方法。

4.2.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.2.3 实验内容

编写程序，控制实验平台的发光二极管 LED1206 和 LED1207，使它们有规律地点亮和熄灭：

LED1206 亮 -> LED1206 关闭 -> LED1207 亮 -> LED1206 和 LED1207 全亮 -> LED1207 关闭 -> LED1206 关闭

4.2.4 实验原理

S3C44B0X 芯片上共有 71 个多功能的输入输出管脚，它们分为 7 组 I/O 端口。

- 两个 9 位的输入/输出端口（端口 E 和 F）
- 两个 8 位的输入/输出端口（端口 D 和 G）
- 一个 16 位的输入/输出端口（端口 C）
- 一个 10 位的输出端口（端口 A）
- 一个 11 位的输出端口（端口 B）

每组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管脚的功能进行设置，如果某些管脚的复用功能没有使用，可以先将该管脚设置为 I/O 口。

1. S3C44BOX I/O 控制寄存器

端口控制寄存器 (PCONA-G)

在 S3C44BOX 芯片中, 大部分管脚是多路复用的, 所以在使用前要确定每个管脚的功能。对复用 I/O 管脚功能的配置, 可以通过配置寄存器 PCONn (端口控制寄存器) 来定义。

如果 PG0-PG7 作为掉电模式下的唤醒信号, 则这些端口必须配置成中断模式。

端口数据寄存器 (PDATA-G)

如果端口定义为输出口, 则输出数据可以写入 PDATn 中相应的位; 如果端口定义为输入口, 则输入的数据可以从 PDATn 相应的位中读入。

端口上拉寄存器 (PUPC-G)

通过配置端口上拉寄存器可以使该组端口和上拉电阻连接或断开。当寄存器中相应的位配置 0 时, 该管脚接上拉电阻; 当寄存器中相应的位配置 1 时, 该管脚不接上拉电阻。

外部中断控制寄存器 (EXTINT)

通过不同的信号方式可以使 8 个外部中断被请求, EXTINT 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。

以下表格为 Embest EduKit-III 实验板上各个端口的管脚定义。在端口 B 的表中可以看到, 管脚 PB9 和 PB10 被设置为输出口, 并且分别和 LED1206、LED1207 连接。

表 4-7 端口 A

| 端口 A | 管脚功能 | 端口 A | 管脚功能 | 端口 A | 管脚功能 |
|------|--------|------|--------|------|--------|
| PA0 | ADDR0 | PA4 | ADDR19 | PA8 | ADDR23 |
| PA1 | ADDR16 | PA5 | ADDR20 | PA9 | ADDR24 |
| PA2 | ADDR17 | PA6 | ADDR21 | | |
| PA3 | ADDR18 | PA7 | ADDR22 | | |

PCONA 寄存器地址: 0X01D20000

PDATA 寄存器地址: 0X01D20004

PCONA 复位默认值: 0X1FF

表 4-8 端口 B

| 端口 B | 管脚功能 | 端口 B | 管脚功能 | 端口 B | 管脚功能 |
|------|-------|------|------------------|------|------------------|
| PB0 | SCKE | PB4 | OUTPUT (LED1) | PB8 | NGCS3 |
| PB1 | SCLE | PB5 | OUTPUT (LED2) | PB9 | OUTPUT (NFCE) |
| PB2 | nSCAS | PB6 | nGCS1 | PB10 | OUTPUT (LCD) |

| | | | | | |
|------------|-------|------------|-------|--|--|
| PB3 | nSRAS | PB7 | NGCS2 | | |
|------------|-------|------------|-------|--|--|

PCONB 寄存器地址: 0X01D20008

PDATB 寄存器地址: 0X01D2000C

PCONB 复位默认值: 0X7FF

表 4-9 端口 C

| 端口 C | 管脚功能 | 端口 C | 管脚功能 | 端口 C | 管脚功能 |
|------------|---------------|-------------|-----------|-------------|------|
| PC0 | OUT (XMON) | PC6 | VD5 | PC12 | TXD1 |
| PC1 | OUT (YPON) | PC7 | VD4 | PC13 | RXD1 |
| PC2 | OUT (XMON) | PC8 | OUT (ALE) | PC14 | RTS0 |
| PC3 | OUT (YPON) | PC9 | OUT (CLE) | PC15 | CTS0 |
| PC4 | VD7 | PC10 | RTS1 | | |
| PC5 | VD6 | PC11 | CTS1 | | |

PCONC 寄存器地址: 0X01D20010

PDATC 寄存器地址: 0X01D20014

PUPC 寄存器地址: 0X01D20018

PCONC 复位默认值: 0X0FF0FFFF

表 4-10 端口 D

| 端口 D | 管脚功能 | 端口 D | 管脚功能 | 端口 D | 管脚功能 |
|------------|------|------------|-------|------------|--------|
| PD0 | VD0 | PD3 | VD3 | PD6 | VM |
| PD1 | VD1 | PD4 | VCLK | PD7 | VFRAME |
| PD2 | VD2 | PD5 | VLINE | | |

PCOND 寄存器地址: 0X01D2001C

PDATD 寄存器地址: 0X01D20020

PUPD 寄存器地址: 0X01D20024

PCOND 复位默认值: 0XA

表 4-11 端口 E

| 端口 E | 管脚功能 | 端口 E | 管脚功能 | 端口 E | 管脚功能 |
|------------|-----------------|------------|-------------|------------|-----------------|
| PE0 | OUT (PS2_IO) | PE3 | OUT (PWM1) | PE6 | OUT (L3DATA) |
| PE1 | TXD0 | PE4 | OUT (PWM2) | PE7 | OUT (L3MODE) |
| PE2 | RXD0 | PE5 | OUT (L3CLK) | PE8 | CODECLK |

PCONE 寄存器地址: 0X01D20028

PDATE 寄存器地址: 0X01D2002C

PUPE 寄存器地址: 0X01D20030

PCONE 复位默认值: 0X25529

表 4-12 Port F

| 端口 F | 管脚功能 | 端口 F | 管脚功能 | 端口 F | 管脚功能 |
|------------|---------|------------|------------|------------|--------|
| PF0 | IIC_SCL | PF3 | OUT (LED4) | PF6 | IISDO |
| PF1 | IIC_SDA | PF4 | OUT (LED3) | PF7 | IISDI |
| PF2 | nWAIT | PF5 | IISLRCLK | PF8 | IISCLK |

PCONF 寄存器地址: 0X01D20034

PDATF 寄存器地址: 0X01D20038

PUPF 寄存器地址: 0X01D2003C

PCONF 复位默认值: 0X00252A

表 4-13 Port G

| 端口 G | 管脚功能 | 端口 G | 管脚功能 | 端口 G | 管脚功能 |
|------------|--------|------------|--------|------------|--------|
| PG0 | EXINT0 | PG3 | EXINT3 | PG6 | EXINT6 |
| PG1 | EXINT1 | PG4 | EXINT4 | PG7 | EXINT7 |
| PG2 | EXINT2 | PG5 | EXINT5 | | |

PCONG 寄存器地址: 0X01D20040

PDATG 寄存器地址: 0X01D20044

PUPG 寄存器地址: 0X01D20048

PCONG 复位默认值: 0XFFFF

2. 电路设计

如图 4-4 所示，发光二极管 LED1 和 LED2 的正极与芯片的 47 脚 VDD33 连接，VDD33 可以输出 3.3V 的电压，负极通过限流电阻 R95、R96 和芯片的 13 脚（GPB4）、14（GPB5）脚连接。这两个管脚属于端口 B，已经配置为输出口。通过向 PDATB 寄存器中相应的位写入 0 或 1 可以使管脚 13、14 输出低电平或高电平。当 13、14 管脚输出低电平时，LED 点亮；当 13、14 管脚输出高电平时，LED 熄灭。

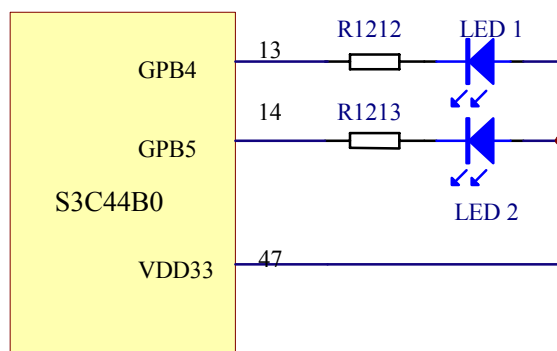


图 4-4 发光二极管控制电路

4.2.5 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.2_led_test 子目录下的 led_test. Uv2 例程，编译链接工程；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏“”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏“”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；

- 5) 在工程管理窗口中双击 led.c 就会打开该文件，在 led_test()处设置断点后，点击 Debug 菜单 Go 运行程序；
- 6) 观察当前 led 1206、led 1207 的状态，点击 Debug 菜单下的 Step over 或 F10 键执行程序，观察 led 1206、led 1207 的变化；
- 7) 结合实验内容和实验原理部分，掌握 ARM 芯片中复用 I/O 口的使用。

4. 观察实验结果

在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...

Embest Arm EduKit III Evaluation Board

Led Test Example
```

程序正确运行后，可以看到实验系统上 LED 1206 和 LED 1207 进行以下循环：

LED1206 亮 -> LED1206 关闭 -> LED1207 亮 -> LED1206 和 LED1207 全亮 -> LED1207 关闭 -> LED1206 关闭

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

4.2.6 实验参考程序

```
/*-----*/
/*                                global variables                                */
/*-----*/
int f_nLedState;                  // LED status

/*-----*/
/*                                function declare                                */
/*-----*/

void led_test();                 // LED test
void leds_on();                  // all leds on
void leds_off();                 // all leds off
void led1_on();                  // led 1 on
void led1_off();                 // led 1 off
void led2_on();                  // led 2 on
void led2_off();                 // led 2 off
void led3_on();                  // led 3 on
void led3_off();                 // led 3 off
void led4_on();                  // led 4 on
void led4_off();                 // led 4 off
void led_display(int nLedStatus); // led control

/*****
* name:      led_test
* func:      leds test function
* para:      none
*****/
```

```

* ret:      none
* modify:
* comment:
*****/

void led_test()
{

    leds_off();

    delay(3000);

    // 1 on -> 2 on -> all on -> 2 off -> 1 off
    led1_on();
    delay(1000);
    led1_off();
    delay(13000);
    led3_on();
    delay(1000);
    led1_on();
    delay(1000);
    beep(0);
    led3_off();
    delay(1000);
    led1_off();

}

/*****

* name:      leds_on,led_off
* func:      all leds light or close
* para:      none
* ret:      none
* modify:
* comment:
*****/

void leds_on()
{
    led_display(0xF);
}

void leds_off()
{
    led_display(0x0);
}

/****

```

```

* name:      led1_on,led1_off
* func:      led 1 light or close
* para:      none
* ret:       none
* modify:
* comment:
*****/

void led2_on()
{
    f_nLedState = f_nLedState | 0x1;
    led_display(f_nLedState);
}

void led2_off()
{
    f_nLedState = f_nLedState&0xfe;
    led_display(f_nLedState);
}

/*****
* name:      led2_on,led2_off
* func:      led 2 light or close
* para:      none
* ret:       none
* modify:
* comment:
*****/

void led4_on()
{
    f_nLedState = f_nLedState | 0x2;
    led_display(f_nLedState);
}

void led4_off()
{
    f_nLedState = f_nLedState&0xfd;
    led_display(f_nLedState);
}

/*****
* name:      led3_on,led3_off
* func:      led 3 light or close
* para:      none
* ret:       none
* modify:
* comment:

```

```

*****/

void led1_on()
{
    f_nLedState = f_nLedState | 0x4;
    led_display(f_nLedState);
}

void led1_off()
{
    f_nLedState = f_nLedState & 0xFB;
    led_display(f_nLedState);
}

/*****
* name:      led4_on,led4_off
* func:      led 4 light or close
* para:      none
* ret:       none
* modify:
* comment:
*****/

void led3_on()
{
    f_nLedState = f_nLedState | 0x8;
    led_display(f_nLedState);
}

void led3_off()
{
    f_nLedState = f_nLedState & 0xF7;
    led_display(f_nLedState);
}

/*****
* name:      led_display
* func:      light or close the Led 1,2
* para:      nLedStatus    -- input, light LED 1,2 according to the nLedStatus' bit[2:1]
*
*                                     nLedStatus' bit[2:1] = 00 : LED 2,1 off
*                                     nLedStatus' bit[2:1] = 11 : LED 2,1 on
*                                     nLedStatus    = 1 : LED 1 on
*                                     nLedStatus    = 2 : LED 2 on
* ret:       none
* modify:
* comment:
*****/

void led_display(int nLedStatus)

```

```

{
    f_nLedState = nLedStatus;

    // change the led's current status
    if((nLedStatus&0x01) == 0x01)
        rPDATC &= 0xFEFF;                // GPC8:LED1 (D1204) on
    else
        rPDATC |= (1<<8);                // off

    if((nLedStatus&0x02) == 0x02)
        rPDATC &= 0xFDF;                // GPC9:LED2 (D1205) on
    else
        rPDATC |= (1<<9);                // off

    if((nLedStatus&0x04) == 0x04)
        rPDATF &= 0xEF;                // GPF4:LED3 (D1206) on
    else
        rPDATF |= (1<<4);                // off

    if((nLedStatus&0x08) == 0x08)
        rPDATF &= 0xF7;                // GPF3:LED4 (D1207) on
    else
        rPDATF |= (1<<3);                // off
}

```

4.2.7 练习题

编写程序，实现使用 LED1206 和 LED1207 状态组合循环显示 00~11。

4.3 中断实验

4.3.1 实验目的

- 通过实验掌握 ARM 处理器的中断方式和中断处理。
- 熟悉 S3C44BOX 的中断控制寄存器的使用；
- 了解不同中断触发方式对中断产生的影响；
- 理解 S3C44BOX 处理器的中断响应过程；
- 熟练掌握如何进行 ARM 处理器中断处理的软件编程方法。

4.3.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.3.3 实验内容

编写中断处理程序，实现：

- 由 UART0 选择输入使用不同的中断触发方式，使能外部中断 Eint4,5,6,7；

- 在不同的中断触发方式下，使用按钮 SB1202 触发 EINT6，同时点亮 LED1204 一段时间后熄灭；
- 在不同的中断触发方式下，使用按钮 SB1203 触发 EINT7，同时点亮 LED1205 一段时间后熄灭。

文档中按钮标号、LED 标号均采用实验硬件平台上标号的简写形式：

按钮：SB1202 → SB2 SB1203 → SB3

LED：D1204 → LED1 D1205 → LED2

4.3.4 实验原理

1. ARM 处理器中断

S3C44BOX 的中断控制器可以接受来自 30 个中断源的中断请求。这些中断源来自 DMA、UART、SIO 等这样的芯片内部外围或芯片外部引脚。在这些中断源中，有 4 个外部中断（EINT4/5/6/7）是逻辑或的关系，它们共用一条中断请求线。UART0 和 UART1 的错误中断也是逻辑或的关系。

中断控制器的任务是在片内外围和外部中断源组成的多重中断发生时，选择其中一个中断通过 FIQ 或 IRQ 向 ARM7TDMI 内核发出中断请求。

实际上最初 ARM7TDMI 内核只有 FIQ（快速中断请求）和 IRQ（通用中断请求）两种中断，其它中断都是各个芯片厂家在设计芯片时定义的，这些中断根据中断的优先级高低来进行处理。例如，如果你定义所有的中断源为 IRQ 中断（通过中断模式寄存器设置），并且同时有 10 个中断发出请求，这时可以通过读中断优先级寄存器来确定哪一个中断将被优先执行。

一般的中断模式在进入所需的服务程序前需要很长的中断反应时间，为了解决这个问题，S3C44BOX 提供了一种新的中断模式叫做向量中断模式，它具有 CISC 结构微控制器的特征，能够降低中断反应时间。换句话说 S3C44BOX 的中断控制器硬件本身直接提供了对向量中断服务的支持。

当多重中断源请求中断时，硬件优先级逻辑会判断哪一个中断将被执行，同时，硬件逻辑自动执行由 0X18（或 0X1C）地址到各个中断源向量地址的跳转指令，然后再由中断源向量进入到相应的中断处理程序。和原来的软件实现的方式相比，这种方法可以显著地减少中断反应时间。

2. 中断控制

- 程序状态寄存器的 F 位和 I 位

如果 CPSR 程序状态寄存器的 F 位被设置为 1，那么 CPU 将不接受来自中断控制器的 FIQ（快速中断请求），如果 CPSR 程序状态寄存器的 I 位被设置为 1，那么 CPU 将不接受来自中断控制器的 IRQ（中断请求）。因此，为了使能 FIQ 和 IRQ，必须先将 CPSR 程序状态寄存器的 F 位和 I 位清零，并且中断屏蔽寄存器 INTMSK 中相应的位也要清零。

- 中断模式（INTMOD）

ARM7TDMI 提供了 2 种中断模式，FIQ 模式和 IRQ 模式。所有的中断源在中断请求时都要确定使用哪一种中断模式。

- 中断挂起寄存器（INTPND）

用于指示对应的中断是否被激活。如果挂起位被设置为 1，那么无论标志 I 或标志 F 是否被清零，都会执行相应的中断服务程序。中断挂起寄存器为只读寄存器，所以在中断服务程序中必须加入对 I_ISPC 和 F_ISPC 写 1 的操作来清除挂起条件。

- 中断屏蔽寄存器（INTMSK）

当 INTMSK 寄存器的屏蔽位为 1 时，对应的中断被禁止；当 INTMSK 寄存器的屏蔽位为 0 时，则对应的中断正常执行。如果一个中断的屏蔽位为 1，在该中断发出请求时挂起位还是会被设置为 1。

如果中断屏蔽寄存器的 global 位设置为 1，那么中断挂起位在中断请求时还会被设置，但所有的中断请求都不被受理。

3. S3C44BOX 中断源

在 30 个中断源中，有 26 个中断源提供给中断控制器，其中 4 个外部中断(EINT4/5/6/7)通过“或”的形式提供一个中断源送至中断控制器，2 个 URAT 错误中断（UERROR0/1）也是如此。

表 4-14 S3C44BOX 的中断源

| Sources | Descriptions | Master Group | Slave ID |
|-------------|----------------------------|--------------|----------|
| EINT0 | External interrupt 0 | mGA | sGA |
| EINT1 | External interrupt 1 | mGA | sGB |
| EINT2 | External interrupt 2 | mGA | sGC |
| EINT3 | External interrupt 3 | mGA | sGD |
| EINT4/5/6/7 | External interrupt 4/5/6/7 | mGA | sGKA |
| TICK | RTC Time tick interrupt | mGA | sGKB |
| INT_ZDMA0 | General DMA0 interrupt | mGB | sGA |
| INT_ZDMA1 | General DMA1 interrupt | mGB | sGB |
| INT_BDMA0 | Bridge DMA0 interrupt | mGB | sGC |
| INT_BDMA1 | Bridge DMA1 interrupt | mGB | sGD |
| INT_WDT | Watch-Dog timer interrupt | mGB | sGKA |
| INT_UERR0/1 | UART0/1 error Interrupt | mGB | sGKB |
| INT_TIMER0 | Timer0 interrupt | mGC | sGA |
| INT_TIMER1 | Timer1 interrupt | mGC | sGB |
| INT_TIMER2 | Timer2 interrupt | mGC | sGC |
| INT_TIMER3 | Timer3 interrupt | mGC | sGD |
| INT_TIMER4 | Timer4 interrupt | mGC | sGKA |
| INT_TIMER5 | Timer5 interrupt | mGC | sGKB |
| INT_URXD0 | UART0 receive interrupt | mGD | sGA |
| INT_URXD1 | UART1 receive interrupt | mGD | sGB |
| INT_IIC | IIC interrupt | mGD | sGC |
| INT_SIO | SIO interrupt | mGD | sGD |
| INT_UTXD0 | UART0 transmit interrupt | mGD | sGKA |
| INT_UTXD1 | UART1 transmit interrupt | mGD | sGKB |
| INT_RTC | RTC alarm interrupt | mGKA | - |
| INT_ADC | ADC EOC interrupt | mGKB | - |

4. 向量中断模式（仅针对 IRQ）

S3C44BOX 含有向量中断模式，可以减少中断的反应时间。通常情况下 ARM7TDMI 内核收到中断控制器的 IRQ 中断请求，ARM7TDMI 会在 0X00000018 地址处执行一条指令。但是在向量中断模式下，当 ARM7TDMI 从 0X00000018 地址处取指令的时候，中断控制器会在数据总线上加载分支指令，这些分支指令使程序计数器能够对应到每一个中断源的向量地址。这些跳转到每一个中断源向量地址的分支指令可以由中断控制器产生。例如，假设 EINT0 是 IRQ 中断，如表 4-15 所示，EINT0 的向量地址为 0X20，所以中断控制器必须产生从 0X18 到 0X20 的分支指令。因此，中断控制器产生的机器码为 0xea000000。在各个中断源对应的中断向量地址中，存放着跳转到相应中断服务程序的程序代码，在相应向量地址处分支指令的机器代码是这样计算的：

向量中断模式的指令机器代码 = 0xea000000 + ((<目标地址> - <向量地址> - 0x8) >> 2)

例如，如果 Timer 0 中断采用向量中断模式，则跳转到对应中断服务程序的分支指令应该存放在向量地址 0x00000060 处。中断服务程序的起始地址在 0x10000，下面就是计算出来放在 0x60 处的机器代码：

machine code@0x00000060 : 0xea000000+((0x10000-0x60-0x8)>>2) =
0xea000000+0x3fe6 = 0xea003fe6

通常机器代码都是反汇编后自动产生的，因此不必真正象上面这样去计算。

表 4-15 中断源的向量地址

| Interrupt Sources | Vector Address |
|-------------------|----------------|
| EINT0 | 0x00000020 |
| EINT1 | 0x00000024 |
| EINT2 | 0x00000028 |
| EINT3 | 0x0000002c |
| EINT4/5/6/7 | 0x00000030 |
| INT_TICK | 0x00000034 |
| INT_ZDMA0 | 0x00000040 |
| INT_ZDMA1 | 0x00000044 |
| INT_BDMA0 | 0x00000048 |
| INT_BDMA1 | 0x0000004c |
| INT_WDT | 0x00000050 |
| INT_UERR0/1 | 0x00000054 |
| INT_TIMER0 | 0x00000060 |
| INT_TIMER1 | 0x00000064 |
| INT_TIMER2 | 0x00000068 |
| INT_TIMER3 | 0x0000006c |
| INT_TIMER4 | 0x00000070 |
| INT_TIMER5 | 0x00000074 |
| INT_URXD0 | 0x00000080 |
| INT_URXD1 | 0x00000084 |
| INT_IIC | 0x00000088 |
| INT_SIO | 0x0000008c |
| INT_UTXD0 | 0x00000090 |
| INT_UTXD1 | 0x00000094 |
| INT_RTC | 0x000000a0 |
| INT_ADC | 0x000000c0 |

向量中断模式的程序举例

在向量中断模式下，当中断请求产生时，程序会自动进入相应的中断源向量地址，因此，在中断源向量地址处必须有一条分支指令使程序进入到相应的中断服务程序，如下：

| Vectors | | | |
|---------|---------------|-----------------------------|----|
| b | ResetHandler | ;/* for debug | */ |
| b | HandlerUndef | ;/* handlerUndef | */ |
| b | HandlerSWI | ;/* SWI interrupt handler*/ | |
| b | HandlerPabort | ;/* handlerPAbort | */ |
| b | HandlerDabort | ;/* handlerDAAbort | */ |
| b | . | ;/* handlerReserved | */ |

```

        ldr pc, =HandlerIRQ
        b      HandlerFIQ
        IF     VIM_SETUP <> 0

        VECTOR_BRANCH    ldr      pc, =HandlerEINT0                ;/*mGA      H/W interrupt vector
table */

        ldr pc, =HandlerEINT1                ;/*          */
        ldr pc, =HandlerEINT2                ;/*          */
        ldr pc, =HandlerEINT3                ;/*          */
        ldr pc, =HandlerEINT4567            ;/*          */
        ldr pc, =HandlerTICK                 ;/*mGA          */
        b .
        b .
        ldr pc, =HandlerZDMA0                ;/*mGB          */
        ldr pc, =HandlerZDMA1                ;/*          */
        ldr pc, =HandlerBDMA0                ;/*          */
        ldr pc, =HandlerBDMA1                ;/*          */
        ldr pc, =HandlerWDT                  ;/*          */
        ldr pc, =HandlerUERR01               ;/*mGB          */
        b .
        b .
        ldr pc, =HandlerTIMER0               ;/*mGC          */
        ldr pc, =HandlerTIMER1               ;/*          */
        ldr pc, =HandlerTIMER2               ;/*          */
        ldr pc, =HandlerTIMER3               ;/*          */
        ldr pc, =HandlerTIMER4               ;/*          */
        ldr pc, =HandlerTIMER5               ;/*mGC          */
        b .
        b .
        ldr pc, =HandlerURXD0                ;/*mGD          */
        ldr pc, =HandlerURXD1                ;/*          */
        ldr pc, =HandlerIIC                  ;/*          */
        ldr pc, =HandlerSIO                   ;/*          */
        ldr pc, =HandlerUTXD0                ;/*          */
        ldr pc, =HandlerUTXD1                ;/*mGD          */
        b .
        b .
        ldr pc, =HandlerRTC                  ;/*mGKA         */
        b .                                ;/*          */
        b .                                ;/*          */
        b .                                ;/*          */
        b .                                ;/*          */
        b .                                ;/*          */
        b .                                ;/*mGKA         */
        b .
        b .
        ldr pc, =HandlerADC                  ;/*mGKB          */

```

ENDIF

5. 中断控制专用寄存器

● 中断控制寄存器（INTCON）

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|----------------------------|-------------|
| INTCON | 0x01E00000 | R/W | Interrupt control Register | 0x7 |

| INTCON | Bit | Description | initial state |
|----------|-----|--|---------------|
| Reserved | [3] | 0 | 0 |
| V | [2] | This bit disables/enables vector mode for IRQ 0 = Vectored interrupt mode 1 = Non-vectored interrupt mode | 1 |
| I | [1] | This bit enables IRQ interrupt request line to CPU 0 = IRQ interrupt enable 1 = Reserved Note : Before using the IRQ interrupt this bit must be cleared. | 1 |
| F | [0] | This bit enables FIQ interrupt request line to CPU 0 = FIQ interrupt enable (Not allowed vectored interrupt mode) 1 = Reserved Note : Before using the FIQ interrupt this bit must be cleared. | 1 |

注意：FIQ 模式不支持向量中断模式。

从表中可以看出，INTCON 寄存器中

位[0] ---- 为 FIQ 中断使能位，写入 0 就使能 FIQ 中断；

位[1] ---- 为 IRQ 中断使能位，写入 0 就使能 IRQ 中断；

位[2] ---- 是选择 IRQ 中断为向量中断模式（V=0）还是普通模式（V=1）。

● 中断挂起寄存器（INTPND）

中断挂起寄存器 INTPND 共有 26 位，每一位对应着一个中断源，当中断请求产生时，相应的位会被设置为 1。该寄存器为只读寄存器，所以在中断服务程序中必须加入对 I_ISPC 和 F_ISPC 写 1 的操作来清除挂起条件。

如果有几个中断源同时发出中断请求，那么不管它们有没有被屏蔽，它们相应的挂起位都会置 1。只是优先级寄存器会根据它们的优先级高低来响应当前优先级最高的中断。

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|--|-------------|
| INTPND | 0x01E00004 | R | Indicates the interrupt request status. 0 = The interrupt has not been requested 1 = The interrupt source has asserted the interrupt request | 0x0000000 |

● 中断模式寄存器（INTMOD）

中断模式寄存器 INTMOD 共有 26 位，每一位对应着一个中断源，当中断源的模式位设置为 1 时，对应的中断会由 ARM7TDMI 内核以 FIQ 模式来处理。相反的，当模式位设置为 0 时，中断会以 IRQ 模式来处理。

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|---|-------------|
| INTMOD | 0x01E00008 | R/W | Interrupt mode Register 0 = IRQ mode 1 = FIQ mode | 0x0000000 |

● 中断屏蔽寄存器（INTMSK）

在中断屏蔽寄存器 INTMSK 中，除了全屏蔽位“global mask”外，其余的 26 位都分别对应一个中断源。当屏蔽位为 1 时，对应的中断被屏蔽；当屏蔽位为 0 时，该中断可以正常使用。如果全屏蔽位“global mask”被设置为 1，则所有的中断都不执行。

如果使用了向量中断模式，在中断服务程序中改变了中断屏蔽寄存器 INTMSK 的值，这时并不能屏蔽相应的中断过程，因为该中断在中断屏蔽寄存器之前已经被中断挂起寄存器 INTPND 锁定了。要解决这个问题，就必须在改变中断屏蔽寄存器后再清除相应的挂起位（INTPND）。

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|---|-------------|
| INTMSK | 0x01E0000C | R/W | Determines which interrupt source is masked. The masked interrupt source will not be serviced. 0 = Interrupt service is available 1 = Interrupt service is masked | 0x07ffff |

● IRQ 向量模式相关寄存器

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|--|-------------|
| I_PSLV | 0x01E00010 | R/W | IRQ priority of slave register | 0x1b1b1b1b |
| I_PMST | 0x01E00014 | R/W | IRQ priority of master register | 0x00001f1b |
| I_CSLV | 0x01E00018 | R | Current IRQ priority of slave register | 0x1b1b1b1b |
| I_CMST | 0x01E0001C | R | Current IRQ priority of master register | 0x0000xx1b |
| I_ISPR | 0x01E00020 | R | IRQ interrupt service pending register (Only one service bit can be set) | 0x00000000 |
| I_ISPC | 0x01E00024 | W | IRQ interrupt service clear register (Whatever to be set, INTPND will be cleared automatically) | Undef. |

S3C44BOX 中的优先级产生模块包含 5 个单元，1 个主单元和 4 个从单元。每个从优先级产生单元管理 6 个中断源。主优先级产生单元管理 4 个从单元和 2 个中断源。

每一个从单元有 4 个可编程优先级中断源（sGn）和 2 个固定优先级中断源（kn）。这 4 个中断源的优先级是由 I_PSLV 寄存器决定的。另外 2 个固定优先级中断源在 6 个中断源中的优先级最低。

主单元可以通过 I_PMST 寄存器来决定 4 个从单元和 2 个中断源的优先级。这 2 个中断源 INT_RTC 和 INT_ADC 在 26 个中断源中的优先级最低。

如果几个中断源同时发出中断请求，这时 I_ISPR 寄存器可以显示当前具有最高优先级的中断源。

● IRQ/FIQ 中断挂起清零寄存器(I_ISPC/F_ISPC)

通过对 I_ISPC/F_ISPC 相应的位写 1 来清除中断挂起位（INTPND）。

| Register | Address | R/W | Description | Reset Value |
|----------|------------|-----|--|-------------|
| I_ISPC | 0x01E00024 | W | IRQ interrupt service pending clear register | Undef. |
| F_ISPC | 0x01E0003C | W | FIQ interrupt service pending clear register | Undef. |

6. 电路原理

如图 4-5 中断实验电路所示，本实验选择的是外部中断 EXINT6 和 EXINT7。中断的产生分别来自按钮 SB2 和 SB3，当按钮按下时，EXINT6 或 EXINT7 和地连接，输入低电平，从而向 CPU 发出中断请求。当 CPU 受理中断后，进入相应的中断服务程序，实现 LED1 或 LED2 的显示功能。从前面介绍的中断源部分我们了解到，EXINT6 和 EXINT7 是共用一个中断控制器，所以在同一时间 CPU 只能受理其中一个中断，也就是说，当按钮 SB2 按下进入中断后，再按 SB3 是没用的，CPU 在处理完 EXINT6 中断前是不会受理来自 EXINT7 的中断，大家可以在实验中留意一下这个情况。

另外 8 段数码管显示部分电路在这里没有给出，需要的话可以参考 4.6 节。

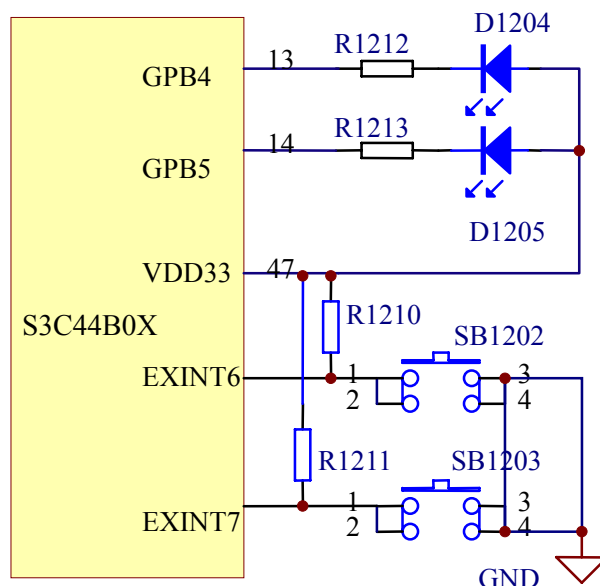


图 4-5 中断实验电路

4.3.5 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.3_int_test 子目录下的 int_test.Uv2 例程，编译链接工程；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 打开 View 菜单>Memory Windows 内存观察窗口，输入 0x01E00004 和 0x01E00024，重点观察 INTPND 和 I_ISPR 寄存器值的变化。

- 6) 在工程管理窗口中双击 `int_test.c` 就会打开该文件，分别在 “`uart_printf(" Press the buttons \n");`” 以及 “`if(f_uclntNesting)`” 设置断点后，点击 Debug 菜单 Go 或 F5 键运行程序，程序正确运行后，会在超级终端上输出如下信息

```
boot success...
External Interrupt Test
Please Select the trigger:
  1 - Falling trigger
  2 - Rising trigger
  3 - Both Edge trigger
  4 - Low level trigger
  5 - High level trigger
any key to exit...
```

- 7) 使用 PC 机键盘，输入所需设置的中断触发方式后，程序停留在第一个断点处，此时注意观察中断控制寄存器的值，即中断配置情况；
- 8) 再次点击 Debug 菜单 Go 或 F5 键运行程序，并等待按下按钮产生中断；当按下 SB2 或 SB3 后，程序停留到中断服务程序入口的断点，再次观察中断控制寄存器的值，注意观察[21]位值在程序运行前后的变化（提示：中断申请标志位应该被置位）；
- 9) 点击 Debug 菜单下的 Step over 或 F10 键执行程序，注意观察在执行完该函数返回前后，程序状态寄存器的变化（提示：CPSR 在返回时恢复中断产生前的值）；继续单步执行程序，从中断返回后，程序会判断被按下的按键点亮相应的 LED：按下 SB2 点亮 LED1 或 按下 SB3 点亮 LED2；
- 10) 结合实验内容和实验原理部分，掌握 ARM 处理器中断操作过程，如中断使能、设置中断触发方式和中断源识别等，重点理解 ARM 处理器的中断响应及中断处理的过程。

4. 观察实验结果

等待选择输入所需中断方式设置：

```
boot success...
External Interrupt Test
Please Select the trigger:
  1 - Falling trigger
  2 - Rising trigger
  3 - Both Edge trigger
  4 - Low level trigger
  5 - High level trigger
any key to exit...
```


在 PC 机键盘上输入 1 选择下降沿触发，并按下按钮 SB2

```
Press the buttons
push buttons may have glitch noise problem
EINT..
EINT6 had been occurred... LED1 (D1204) on
```

如果重复按下按钮 SB2 或再按下另一个按钮（在中断响应后到点亮 LED1 之间），将会报告当前正在处理第一个按钮产生的中断，并输出信息表示中断嵌套。

```
Press the buttons
push buttons may have glitch noise problem
EINT..
EINT6 had been occurred... LED1 (D1204) on
EINT..
f_ucIntNesting = 2
```

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

4.3.6 实验参考程序

1. 中断初始化程序

```
/*-----*/
/*          function declare          */
/*-----*/

void init_int(void);
void int_test(void);
void int4567_isr(void);// __attribute__((interrupt ("IRQ")));

/*-----*/
/*          global variables          */
/*-----*/

unsigned char f_ucIntNesting = 0;          // Interrupt nesting count
unsigned char f_ucWhichInt   = 0;          // interrupt source symbol

/*****
* name:      init_int
* func:      initialize the extern interrupt control
* para:      none
* ret:       none
* modify:
* comment:
*****/
```

```

void init_int(void)
{
    // interrupt settings
    rI_ISPC    = 0x3fffff;           // clear interrupt pending register
    rEXTINTPND = 0xf;               // clear EXTINTPND register
    rINTMOD     = 0x0;               // all for IRQ mode
    rINTCON     = 0x5;               // nonVectored mode, IRQ disable, FIQ disable
    rINTMSK     = ~(BIT_GLOBAL|BIT_EINT4567);

    // set EINT interrupt handler
    pISR_EINT4567 = (int)int4567_isr;

    // PORT G configuration
    rPCONG      = 0xffff;           // EINT7~0
    rPUPG       = 0x0;              // pull up enable
    rEXTINT     = rEXTINT | 0x22220020; // EINT4567 falling edge mode
    rI_ISPC |= BIT_EINT4567;
    rEXTINTPND = 0xf;               // clear EXTINTPND reg
}

```

2. 中断服务程序

```

/*****
* name:      int4567_isr
* func:
* para:      none
* ret:       none
* modify:
* comment:
*****/

void int4567_isr(void)
{
    f_ucWhichInt = rEXTINTPND;
    uart_printf(" EINT.. \n");
    if(f_ucIntNesting)
    {
        f_ucIntNesting++;           // an extern intrrupt had been occur before dealing with
one.
        uart_printf(" f_ucIntNesting = %d\n",f_ucIntNesting);
    }

    delay(1000);
    rEXTINTPND = 0xf;               // clear EXTINTPND reg.
    rI_ISPC |= BIT_EINT4567;       // clear pending_bit
}

```

4.3.7 练习题

1. 熟悉 S3C44B0X 芯片的时钟控制器及其相关的寄存器，掌握中断响应的完整过程。
2. 利用处理器内部时钟器中断，编写程序实现：
每隔 0.5 秒 LED1 点亮延时 1 秒后熄灭；每隔 1 秒 LED2 点亮延时 1 秒后熄灭。

4.4 串口通信实验

4.4.1 实验目的

- 了解 S3C44B0X 处理 UART 相关控制寄存器的使用。
- 熟悉 ARM 处理器系统硬件电路中 UART 接口的设计方法。
- 掌握 ARM 处理器串行通信的软件编程方法。

4.4.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.4.3 实验内容

编写 S3C44B0X 处理器的串口通信程序：

- 监视串行口 UART0 动作；
- 将从 UART0 接收到的字符串回送显示。

4.4.4 实验原理

1. S3C44B0X 串行通讯 (UART) 单元

S3C44B0X UART 单元提供两个独立的异步串行通信口，皆可工作于中断和 DMA 模式。最高波特率达 115.2Kbps。每一个 UART 单元包含一个 16 字节的 FIFO，用于数据的接收和发送。

S3C44B0X UART 包括可编程波特率，红外发送/接收，一个或两个停止位，5bit/6bit/ 7bit/ 8bit 数据宽度和奇偶校验。

2. 波特率的产生

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

$$\text{UBRDIVn} = (\text{round_off})(\text{MCLK}/(\text{bps} \times 16)) - 1$$

其中：MCLK 是系统时钟。UBRDIVn 的值必须在 1 到 (216-1) 之间。

例如：在系统时钟为 40MHz，当波特率为 115200 时，

$$\begin{aligned} \text{UBRDIVn} &= (\text{int})(40000000/(115200 \times 16) + 0.5) - 1 \\ &= (\text{int})(21.7 + 0.5) - 1 \\ &= 22 - 1 = 21 \end{aligned}$$

3. UART 通信操作

下面简略介绍 UART 操作，关于数据发送，数据接收，中断产生，波特率产生，回环模式，红外模式和自动流控制的详细介绍，请参照相关教材和数据手册。

发送数据帧是可编程的。一个数据帧包含一个起始位，5 到 8 个数据位，一个可选的奇偶校验位和 1 到 2 位停止位，停止位通过行控制寄存器 ULCONn 配置。

与发送类似，接收帧也是可编程的。接收帧由一个起始位，5 到 8 个数据位，一个可选的奇偶校验和 1 到 2 位行控制寄存器 ULCONn 里的停止位组成。接收器还可以检测过速错，奇偶校验错，帧错误和传输中断，每一个错误均可以设置一个错误标志。

过速错是指已接收到的数据在读取之前被新接收的数据覆盖。

奇偶校验错是指接收器检测到的校验和与设置的不符。

帧错误指没有接收的有效的停止位。

传输中断表示接收数据 RxDn 保持逻辑 0 超过一帧的传输时间。

在 FIFO 模式下，如果 RxFIFO 非空，而在 3 个字的传输时间内没有接收到数据，则产生超时。

4. UART 控制寄存器

1) UART 行控制寄存器 ULCONn，该寄存器的第 6 位决定是否使用红外模式，位 5~3 决定校验方式，位 2 决定停止位长度，位 1 和 0 决定每帧的数据位数。

2) UART 控制寄存器 UCONn，该寄存器决定 UART 的各种模式。UART FIFO 控制寄存器 UFCONn，UART MODEM 控制寄存器，分别决定 UART FIFO 和 MODEM 的模式。其中 UFCONn 的第 0 位决定是否启用 FIFO，UMCONn 的第 0 位是请求发送位。另外读写状态寄存器 UTRSTAT 以及错误状态寄存 UERSTAT，可以反映芯片目前的读写状态以及错误类型。FIFO 状态寄存器 UFSTAT 和 MODEM 状态寄存器 UMSTAT，通过前者可以读出目前 FIFO 是否满以及其中的字节数；通过后者可以读出目前 MODEM 的 CTS 状态。

3) 发送寄存器 UTXH 和接收寄存器 URXH，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，接收的数据必须被读出来，否则会引发下次溢出错误。

4) 波特率分频寄存器 UBRDIV。

在例程目录下的 common\include\44b.h 文件中有关于 UART 单元各寄存器的定义。

```
/* UART */
#define rULCON0      (*(volatile unsigned *)0x1d00000)
#define rULCON1      (*(volatile unsigned *)0x1d04000)
#define rUCON0       (*(volatile unsigned *)0x1d00004)
#define rUCON1       (*(volatile unsigned *)0x1d04004)
#define rUFCON0      (*(volatile unsigned *)0x1d00008)
#define rUFCON1      (*(volatile unsigned *)0x1d04008)
#define rUMCON0      (*(volatile unsigned *)0x1d0000c)
#define rUMCON1      (*(volatile unsigned *)0x1d0400c)
#define rUTRSTAT0    (*(volatile unsigned *)0x1d00010)
#define rUTRSTAT1    (*(volatile unsigned *)0x1d04010)
#define rUERSTAT0    (*(volatile unsigned *)0x1d00014)
#define rUERSTAT1    (*(volatile unsigned *)0x1d04014)
#define rUFSTAT0     (*(volatile unsigned *)0x1d00018)
#define rUFSTAT1     (*(volatile unsigned *)0x1d04018)
#define rUMSTAT0     (*(volatile unsigned *)0x1d0001c)
#define rUMSTAT1     (*(volatile unsigned *)0x1d0401c)
#define rUBRDIV0     (*(volatile unsigned *)0x1d00028)
#define rUBRDIV1     (*(volatile unsigned *)0x1d04028)
```

```

#ifdef __BIG_ENDIAN
#define rUTXH0      (*(volatile unsigned char *)0x1d00023)
#define rUTXH1      (*(volatile unsigned char *)0x1d04023)
#define rURXH0      (*(volatile unsigned char *)0x1d00027)
#define rURXH1      (*(volatile unsigned char *)0x1d04027)
#define WrUTXH0(ch) (*(volatile unsigned char *)0x1d00023)=(unsigned char)(ch)
#define WrUTXH1(ch) (*(volatile unsigned char *)0x1d04023)=(unsigned char)(ch)
#define RdURXH0()   (*(volatile unsigned char *)0x1d00027)
#define RdURXH1()   (*(volatile unsigned char *)0x1d04027)
#define UTXH0       (0x1d00020+3)          // byte_access address by BDMA
#define UTXH1       (0x1d04020+3)
#define URXH0       (0x1d00024+3)
#define URXH1       (0x1d04024+3)

#else //Little Endian
#define rUTXH0      (*(volatile unsigned char *)0x1d00020)
#define rUTXH1      (*(volatile unsigned char *)0x1d04020)
#define rURXH0      (*(volatile unsigned char *)0x1d00024)
#define rURXH1      (*(volatile unsigned char *)0x1d04024)
#define WrUTXH0(ch) (*(volatile unsigned char *)0x1d00020)=(unsigned char)(ch)
#define WrUTXH1(ch) (*(volatile unsigned char *)0x1d04020)=(unsigned char)(ch)
#define RdURXH0()   (*(volatile unsigned char *)0x1d00024)
#define RdURXH1()   (*(volatile unsigned char *)0x1d04024)
#define UTXH0       (0x1d00020)          // byte_access address by BDMA
#define UTXH1       (0x1d04020)
#define URXH0       (0x1d00024)
#define URXH1       (0x1d04024)
#endif

```

5. UART 初始化代码

下面列出的三个函数，是我们本实验用到的三个主要函数，包括 UART 初始化，字符的收、发，希望大家仔细阅读，理解每一行的含义。这几个函数可以在安装目录下\common\include\44bilib.c 文件内找到。

```

/*****
* name:      uart_init
* func:      initialize uart channel
* para:      nMainClk --   input,
*              nBaud,
* ret: none
* modify:
* comment:
*****/

void uart_init(int nMainClk, int nBaud)
{
    int i;

```

```

if(nMainClk==0)
    nMainClk=MCLK;
rUFCON0=0x0;                // FIFO disable
rUFCON1=0x0;
rUMCON0=0x0;
rUMCON1=0x0;
// UART0
rULCON0  = 0x3;              // Normal, No parity, 1 stop, 8 bit
rUCON0   = 0x245;           // rx=edge, tx=level, disable timeout int.,
                             enable rx error int., normal,interrupt or polling
rUBRDIV0 = ((int)(nMainClk/16./nBaud + 0.5) -1);
// UART1
rULCON1  = 0x3;
rUCON1   = 0x245;
rUBRDIV1 = ((int)(nMainClk/16./nBaud + 0.5) -1);
for(i=0; i<100; i++);
}

```

下面是接收字符的实现函数：

```

/*****
* name:      uart_getch
* func:      Get a character from the uart
* para:      none
* ret:  get a char from uart channel
* modify:
* comment:
*****/
char uart_getch(void)
{
    if(f_nWhichUart==0)
    {
        while(!(rUTRSTAT0 & 0x1));           // Receive data read
        return RdURXH0();
    }
    else
    {
        while(!(rUTRSTAT1 & 0x1));           // Receive data ready
        return  rURXH1;
    }
}

```

发送字符的实现函数：

```

/*****
* name:      uart_sendbyte
* func:      Send one byte to uart channel

```

```

* para:      nData    --    input, byte
* ret: none
* modify:
* comment:
*****/

void uart_sendbyte(int nData)
{
    if(f_nWhichUart == 0)
    {
        if(nData == '\n')
        {
            while(!(rUTRSTAT0 & 0x2));
            delay(10);                // because the slow response of
hyper_terminal
            WrUTXH0('\r');
        }
        while(!(rUTRSTAT0 & 0x2));    // Wait until THR is empty.
        delay(10);
        WrUTXH0(nData);
    }
    else
    {
        if(nData=='\n')
        {
            while(!(rUTRSTAT1 & 0x2));
            delay(10);                // because the slow response of
hyper_terminal
            rUTXH1 = '\r';
        }
        while(!(rUTRSTAT1 & 0x2));    // Wait until THR is empty.
        delay(10);
        rUTXH1 = nData;
    }
}
}

```

6. RS232 接口电路

EduKit-III 教学电路中, 串口电路如图 4-9 所示, 开发板上提供两个串口 DB9。其中 UART1 为主串口, 可与 PC 或 MODOM 进行串行通讯。由于 44B0X 未提供 DCD(载波检测)、DTR(数据终端准备好)、DSR(数据准备好)、RIC(振铃指示)等专用 I/O 口, 故用 MCU 的通用 I/O 口替代。UART0 只采用二根接线 RXD 和 TXD, 因此只能进行简单的数据传输及接收功能。全接口的 UART1 采用 MAX3243E 作为电平转换器, 简单接口的 UART0 则采用 MAX3221E 作为电平转换器。

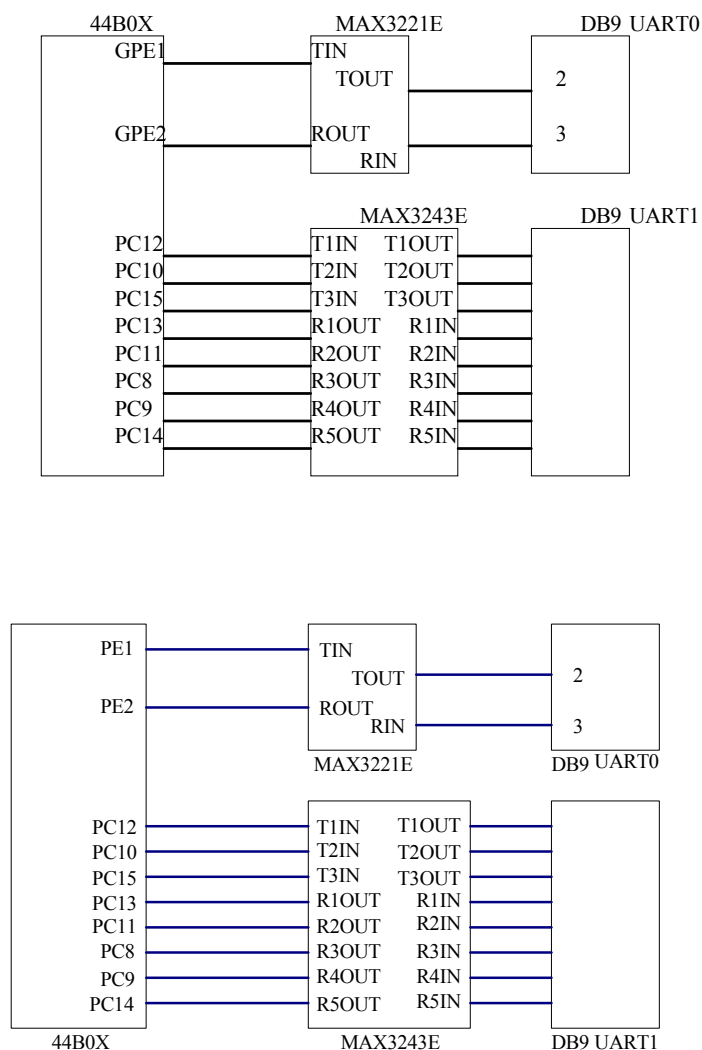


图 4-9 串口电路

4.4.4 实验操作步骤

1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。



2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序（如：串口精灵等），超级终端配置如图 4-10 所示。



图 4-10 Embest ARM 教学系统超级终端配置

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.4_uart_test 子目录下的 uart_test.Uv2 例程，编译链接工程；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 IDE 的 Debug 菜单 Go 或 F5 键运行程序；

4. 观察实验结果

执行程序，可以看到超级终端上输出等待输入字符，

```
boot success...
>
```

如果输入字符就会马上显示在超级终端上，输入回车符后打印一整串字符：

```
boot success...
>Hello,World!
Hello,World!
>
```

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

4.4.5 实验参考程序

本实验内容的主函数代码如下，大家可以修改该函数，以实现不同的通信目的。

```

/*****
* name:      Main
* func:      c code entry
* para:      none
* ret: none
* modify:
* comment:
*****/

void Main(void)
{
    char  cInput;                // user input char
    char  szLogo[17] = ">";
    char  szStr[256];
    char  *pStr = szStr;
    int    i;

    sys_init()                  // Initial 44BOX's Interrupt,Port and UART
    // printf interface
    uart_printf("\n");
    uart_printf(szLogo);

    while(1)
    {
        // waiting input from uart channel
        *pStr = uart_getch();
        uart_sendbyte(*pStr);

        if (*pStr == 0x0D)
        {
            uart_printf("\n");
            if (pStr != szStr)
            {
                // Send received string

```

```

        pStr = szStr;
        while (*pStr != 0x0D)
        {
            uart_sendbyte(*pStr);
            pStr++;
        }
        pStr = szStr;
    }

    // printf interface
    uart_printf("\n");
    uart_printf(szLogo);
}
else
    pStr++;
}
}

```

串口通信函数库中的其它函数：

```

/*****
* name:      uart_getString
* func:      Get string from uart channel and store the result to input address (*pString)
* para:      pString  --  input, string
* ret: none
* modify:
* comment:
*****/

void uart_getstring(char *pString)
{
    char *pString2 = pString;
    char c;
    while((c = uart_getch())!= '\r')
    {
        if(c == '\b')
        {
            if( (int)pString2 < (int)pString )
            {
                uart_printf("\b\b");
                pString--;
            }
        }
        else // store and echo on uart channel
        {
            *pString += c;
            uart_sendbyte(c);
        }
    }
}

```

```

    }
}
*pString = '\0';
uart_sendbyte('\n');
}

/*****
* name:      uart_getintnum
* func:      Get a numerical (Dec - default or Hex fromat) from the uart, with or without a signed
* para:      none
* ret:  nResult: the valid number which user input from uart
*           -- Dec format number (default)
*           -- Hex format number ('H/h' suffix or '0x' ahead)
* modify:
* comment:
*****/
int uart_getintnum(void)
{
    // 略
}

/*****
* name:      uart_sendstring
* func:      Send string to uart channel
* para:      pString  --  input, string
* ret:  none
* modify:
* comment:
*****/
void uart_sendstring(char *pString)
{
    while(*pString)
    {
        uart_sendbyte(*pString++);
    }
}

/*****
* name:      uart_printf
* func:      print format string
* para:      fmt  --  input,
* ret:  none
* modify:
* comment:
*****/

```

```
void uart_printf(char *fmt,...)
{
    // 略
}
```

4.4.6 练习题

1. 编写程序实现在 LCD 上显示从串口接收到的字符。
2. 思考题：怎样在本例程的基础上，增加错误检测功能？

4.5 实时时钟实验

4.5.1 实验目的

- 了解实时时钟的硬件控制原理及设计方法。
- 掌握 S3C44B0X 处理器的 RTC 模块程序设计方法。

4.5.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.5.3 实验内容

学习和掌握 Embest EduKit-III 实验平台中 RTC 模块的使用，进行以下操作：

- 编写应用程序，修改时钟日期及时间的设置。
- 使用 EMBEST ARM 教学系统的串口，在超级终端显示当前系统时间。

4.5.4 实验原理

1. 实时时钟（RTC）

实时时钟（RTC）器件是一种能提供日历/时钟、数据存储等功能的专用集成电路，常用作各种计算机系统的时钟信号源和参数设置存储电路。RTC 具有计时准确、耗电低和体积小等特点，特别是在各种嵌入式系统中用于记录事件发生的时间和相关信息，如通信工程、电力自动化、工业控制等自动化程度高的领域的无人值守环境。随着集成电路技术的不断发展，RTC 器件的新品也不断推出，这些新品不仅具有准确的 RTC，还有大容量的存储器、温度传感器和 A/D 数据采集通道等，已成为集 RTC、数据采集和存储于一体的综合功能器件，特别适用于以微控制器为核心的嵌入式系统。

RTC 器件与微控制器之间的接口大都采用连线简单的串行接口，诸如 I2C、SPI、MICROWIRE 和 CAN 等串行总线接口。这些串口由 2~3 根线连接，分为同步和异步。

2. S3C44B0X 实时时钟（RTC）单元

S3C44B0X 实时时钟（RTC）单元是处理器集成的片内外设。由开发板上的后备电池供电，可以在系统电源关闭的情况下运行。RTC 发送 8 位 BCD 码数据到 CPU。传送的数据包括秒、分、小时、星期、日期、月份和年份。RTC 单元时钟源由外部 32.768KHz 晶振提供，可以实现闹钟（报警）功能。

S3C44B0X 实时时钟（RTC）单元特性：

- BCD 数据：秒、分、小时、星期、日期、月份和年份
- 闹钟（报警）功能：产生定时中断或激活系统

- 自动计算闰年
- 无 2000 年问题
- 独立的电源输入
- 支持毫秒级时间片中断，为 RTOS 提供时间基准

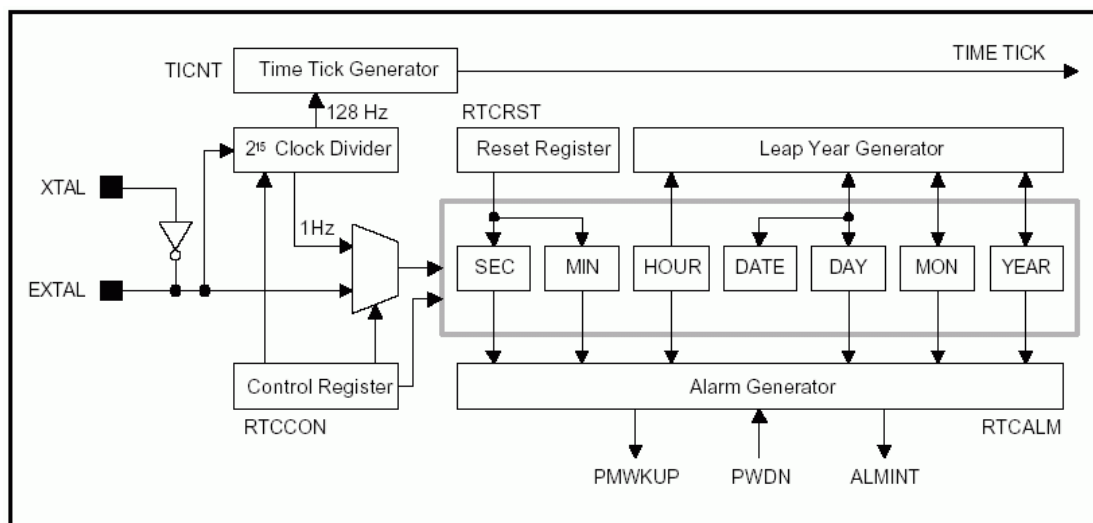


图 4-11 S3C44BOX 处理器 RTC 功能框图

● 读/写寄存器

访问 RTC 模块的寄存器，首先要设 RTCCON 的 bit0 为 1。CPU 通过读取 RTC 模块中寄存器 BCDSEC、BCDMIN、BCD HOUR、BCDDAY、BCDDATE、BCDMON 和 BCDYEAR 的值，得到当前的相应时间值。然而，由于多个寄存器依次读出，所以有可能产生错误。比如：用户依次读取年（1989）、月（12）、日（31）、时（23）、分（59）、秒（59）。当秒数为 1 到 59 时，没有任何问题，但是，当秒数为 0 时，当前时间和日期就变成了 1990 年 1 月 1 日 0 时 0 分。这种情况下（秒数为 0），用户应该重新读取年份到分钟的值（参考程序设计）。

● 后备电池：

RTC 单元可以使用后备电池通过管脚 RTCVDD 供电。当系统关闭电源以后，CPU 和 RTC 的接口电路被阻断，后备电池只需要驱动晶振和 BCD 计数器，从而达到最小的功耗。

● 闹钟功能

RTC 在指定的时间产生报警信号，包括 CPU 工作在正常模式和休眠（power down）模式下。在正常工作模式，报警中断信号（ALMINT）被激活。在休眠模式，报警中断信号和唤醒信号（PMWKUP）同时被激活。RTC 报警寄存器（RTCALM）决定报警功能的使能/屏蔽和完成报警时间检测。

● 时间片中断

RTC 时间片中断用于中断请求。寄存器 TICNT 有一个中断使能位和中断计数。该中断计数自动递减，当达到 0 时，则产生中断。中断周期按照下列公式计算：

$$\text{Period} = (n+1) / 128 \text{ second}$$

其中，n 为 RTC 时钟中断计数，可取值为（1-127）

● 置零计数功能

RTC 的置零计数功能可以实现 30、40 和 50 秒步长重新计数，供某些专用系统使用。当使用 50 秒置零设置时，如果当前时间是 11:59:49，则下一秒后时间将变为 12:00:00。

注意：所有的 RTC 寄存器都是字节型的，必须使用字节访问指令（STRB、LDRB）或字符型指针访问。

4.5.5 实验设计

1. 硬件电路设计

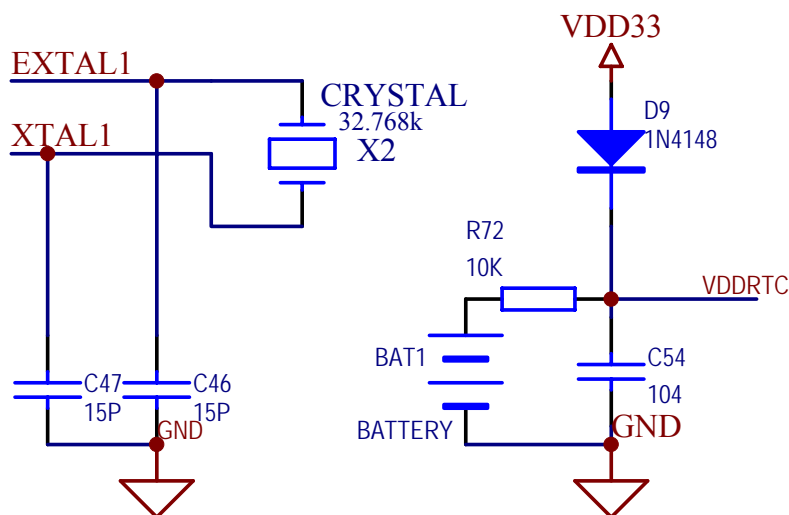


图 4-12 实时时钟外围电路

2. 软件程序设计

(1) 时钟设置

时钟设置程序必须实现时钟工作情况以及数据设置有效性检测功能。具体实现可以参考示例程序设计。

(2) 时钟显示

时钟参数通过实验系统串口 0 输出到超级终端，显示内容包括年月日时分秒。参数以 BCD 码形式传送，用户使用串口通信函数（参见串口通信实验）将参数取出显示。

```

/*****
* name:      rtc_read
* func:      read data from rtc
* para:      none
* ret:       none
* modify:
* comment:
*****/

void rtc_read(void)
{
    while(1)
    {
        // read the data from RTC registers
        if(rBCDYEAR == 0x99)
            g_nYear = 0x1999;
    }
}

```

```

else
    g_nYear = 0x2000 + rBCDYEAR;

    g_nMonth  = rBCDMON;
    g_nDay    = rBCDDAY;
    g_nWeekday = rBCDDATE;
    g_nHour   = rBCDHOUR;
    g_nMin    = rBCDMIN;
    g_nSec    = rBCDSEC;

    if(g_nSec != 0)
        break;
}
}

/*****
* name:      rtc_display
* func:      display data from rtc
* para:      none
* ret:       none
* modify:
* comment:
*****/

void rtc_display(void)
{
    rtc_read();
    uart_printf("\n\rCurrentTimeis%02x-%02x-%02x%s",
                g_nYear,g_nMonth,g_nDay,f_szdate[g_nWeekday]);
    uart_printf(" %02x:%02x:%02x\r\n",g_nHour,g_nMin,g_nSec);
}

```

4.5.6 实验操作步骤

1. 准备实验环境



使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.5_rtc_test 子目录下的 rtc_test.Uv2 例程，编译链接工程；

- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Go 或 F5 键运行程序。

4. 观察实验结果

- 1). 在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...
RTC Test Example
RTC Check(Y/N)?
```

- 2). 用户可以选择是否对 RTC 进行检查，检查正确的话，继续执行程序，检查不正确时也会提示是否重检查：

```
RTC Check(Y/N)? y
Set Default Time at 2004-12-31 FRI 23:59:59
Set Alarm Time at 2005-01-01 00:00:01
... RTC Alarm Interrupt O.K. ...
Current Time is 2005-01-01 SAT 00:00:01
RTC Working now. To set date(Y/N)?
```

- 3). 用户可以选择是否重新进行时钟设置，当输入不正确时也会提示是否重新设置：

```
RTC Working now. To set date(Y/N)? y Current date is (2005,01,01, SAT).
input new date (yy-mm-dd w): 5-2-23 3
Current date is: 2005-02-23 WED
RTC Working now. To set time(Y/N)? y Current time is (00:02:57).
To set time(hh:mm:ss): 19:32:5
```

- 4). 最终超级终端输出信息如下：

Current Time is 2005-02-23 WED 19:32:05

19:32:07

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

4.5.7 实验参考程序

1. 环境及函数声明

```
/*-----*/
/*                global variables                */
/*-----*/
int  g_nYear;
int  g_nMonth,g_nDay,g_nWeekday,g_nHour,g_nMin,g_nSec;
/*-----*/
/*                function declare                */
/*-----*/

int test_rtc_alarm(void);
void rtc_init(void);
void read_rtc(void);
void display_rtc(void);
void test_rtc_tick(void);

void rtc_int(void);
void rtc_tick(void);
```

2. 时钟设置控制程序

```
/*-----*/
* name:      rtc_set_date
* func:      get and check the DATE string from uart channel to set rtc
* para:      none
* ret:       cN09 = 0 : invalid string
*           cN09 = 1 : set date by input string and ok
* modify:
* comment:
/*-----*/

int rtc_set_date(char *pString)
{
    char cYn,cN09=1;
    char szStr[12];                // xxxx-xx-xx x
    int    i,nTmp;

    memcpy((void *)szStr, pString, 12);
    // check the format of the data
    nTmp = 0;
```

```

cN09 = 1;
for(i = 0; ((i < 12) & (szStr[i] != '\0')); i++)
{
    if((szStr[i] == '-') || (szStr[i] == ' '))
        nTmp += 1;
}

if(nTmp < 3)    // at least 2 '-' and 1 ' '
{
    cN09 = 0;
    uart_printf(" Invalid format!!\n\r");
}
else // check if number 0 - 9
{
    nTmp = i - 1;                                // adjust the account number

    // 1:MON 2:TUE 3:WED 4:THU 5:FRI 6:SAT 7:SUN
    if((szStr[nTmp] < '1' | szStr[nTmp] > '7'))    // check weekday
        cN09 = 0;

    for( i = nTmp; i >= 0; i--)
    {
        if(!((szStr[i] == '-') || (szStr[i] == ' ')))
            if((szStr[i] < '0' | szStr[i] > '9'))
                cN09 = 0;
    }
}

// write the data into rtc register
if(cN09)
{
    rRTCCON = 0x01;                                // R/W enable, 1/32768,
Normal(merge), No reset
    i = nTmp;
    nTmp = szStr[i]&0x0f;
    if(nTmp == 7)
        rBCDDATE = 1;                                // s3c44b0x: SUN:1 MON:2
TUE:3 WED:4 THU:5 FRI:6 SAT:7
    else
        rBCDDATE = nTmp+1;                            // -> weekday;

    nTmp = szStr[i-2]&0x0f;
    if(szStr[i-1] != '-')
        nTmp |= (szStr[i-2]<4)&0xff;
    if(nTmp > 0x31)
        cN09 = 0;
}

```

```

    rBCDDAY = nTmp;                                // -> day;

    nTmp = szStr[--i]&0x0f;
    if(szStr[--i] != '-')
        nTmp |= (szStr[i--]<<4)&0xff;
    if(nTmp > 0x12)
        cN09 = 0;
    rBCDMON = nTmp;                                // -> month;

    nTmp = szStr[--i]&0x0f;
    if(i)
        nTmp |= (szStr[--i]<<4)&0xff;
    if(nTmp > 0x99)
        cN09 = 0;
    rBCDYEAR = nTmp;                                // -> year;
    rRTCCON = 0x00;                                // R/W disable

    uart_printf(" Current date is: 20%02x-%02x-%02x %s\n"
                ,rBCDYEAR,rBCDMON,rBCDDAY,f_szdate[rBCDDATE]);

    if(!cN09)
        uart_printf(" Wrong value!\n");
    }else uart_printf(" Wrong value!\n");

    return (int)cN09;
}

/*****
* name:      rtc_set_time
* func:      get and check the TIME string from uart channel to set rtc
* para:      none
* ret:       cN09 = 0 : invalid string
*           cN09 = 1 : set time by input string and ok
* modify:
* comment:
*****/
int rtc_set_time(char *pString)
{
    char cYn,cN09=1;
    char szStr[8];                                // xx:xx:xx
    int i,nTmp;

    memcpy((void *)szStr, pString, 8);

    // check the format of the data
    nTmp = 0;

```

```

cN09 = 1;
for(i = 0;((i < 8)&(szStr[i] != '\0')); i++)
{
    if(szStr[i] == ':')
        nTmp += 1;
}

if(nTmp != 2)    // at least 3 ':'
{
    cN09 = 0;
    uart_printf(" InValid format!!\n\r");
}
else
{
    nTmp = i - 1;
    for( i = nTmp; i >= 0; i--)
    {
        if(szStr[i] != ':')
            if((szStr[i] < '0' | szStr[i] > '9'))
                cN09 = 0;
    }
}

// write the data into rtc register
if(cN09)
{
    rRTCCON  = 0x01;                                // R/W enable, 1/32768,
Normal(merge), No reset

    i = nTmp;
    nTmp = szStr[i]&0x0f;
    if(szStr[--i] != ':')
        nTmp |= (szStr[i--]<<4)&0xff;
    if(nTmp > 0x59)
        cN09 = 0;
    rBCDSEC  = nTmp;                                // -> second;

    nTmp = szStr[--i]&0x0f;
    if(szStr[--i] != ':')
        nTmp |= (szStr[i--]<<4)&0xff;
    if(nTmp > 0x59)
        cN09 = 0;
    rBCDMIN  = nTmp;                                // -> min;

    nTmp = szStr[--i]&0x0f;
    if(i)

```

```

        nTmp |= (szStr[--i]<<4)&0xff;
    if(nTmp > 0x24)
        cN09 = 0;
    rBCD HOUR   = nTmp;                // -> hour;

    rRTCCON   = 0x00;                // R/W disable

    if(!cN09)
        uart_printf(" Wrong value!\n");
    }else uart_printf(" Wrong value!\n");

    return (int)cN09;
}

```

4.5.8 练习题

编写程序检测 RTC 的时间片（RTC_Tick）功能。

4.6 数码管显示实验

4.6.1 实验目的

- 通过实验掌握 LED 的显示控制方法。
- 巩固实验 4.1 中所掌握的对存储区进行访问的方法。

4.6.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.6.3 实验内容

编写程序使实验板上八段数码管循环显示 0 到 9 字符。

4.6.4 实验原理

1. 八段数码管

嵌入式系统中，经常使用八段数码管来显示数字或符号，由于它具有显示清晰、亮度高、使用电压低、寿命长的特点，因此使用非常广泛。

● 结构

八段数码管由八个发光二极管组成，其中七个长条形的发光管排列成“日”字形，右下角一个点形的发光管作为显示小数点用，八段数码管能显示所有数字及部份英文字母。见图 4-13。

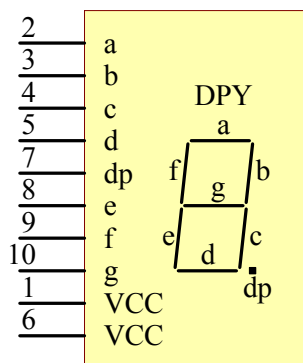


图 4-13 八段数码管的结构

● 类型

八段数码管有两种不同的形式：一种是八个发光二极管的阳极都连在一起的，称之为共阳极八段数码管；另一种是八个发光二极管的阴极都连在一起的，称之为共阴极八段数码管。

● 工作原理

以共阳极八段数码管为例，当控制某段发光二极管的信号为低电平时，对应的发光二极管点亮，当需要显示某字符时，就将该字符对应的所有二极管点亮；共阴极二极管则相反，控制信号为高电平时点亮。

电平信号按照 dp, g, e...a 的顺序组合形成的数据字称为该字符对应的段码，常用字符的段码表如下：

表 4-16 常用字符的段码表

| 字 符 | dp | g | f | e | d | c | b | a | 共阴极 | 共阳极 |
|--------|----|---|---|---|---|---|---|---|-----|-----|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3FH | C0H |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 06H | F9H |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5BH | A4H |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 4FH | B0H |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 66H | 99H |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 6DH | 92H |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 7DH | 82H |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 07H | F8H |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7FH | 80H |
| 9 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 6FH | 90H |
| A | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 77H | 88H |
| B | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 7CH | 83H |
| C | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 39H | C6H |
| D | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 5EH | A1H |
| E | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 79H | 86H |
| F | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 71H | 8EH |

| | | | | | | | | | | |
|----|---|---|---|---|---|---|---|---|-----|-----|
| - | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 40H | BFH |
| . | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80H | 7FH |
| 熄灭 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00H | FFH |

● 显示方式

八段数码管的显示方式有两种，分别是静态显示和动态显示。

静态显示是指当八段数码管显示一个字符时，该字符对应段的发光二极管控制信号一直保持有效。

动态显示是指当八段数码管显示一个字符时，该字符对应段的发光二极管是轮流点亮的，即控制信号按一定周期有效，在轮流点亮的过程中，点亮时间是极为短暂的（约 1ms），由于人的视觉暂留现象及发光二极管的余辉效应，数码管的显示依然是非常稳定的。

2. 电路原理

EMBEST EduKit-III 教学电路中，使用的是共阳极八段数码管，数码管的控制通过芯片 ZLG7290 控制，各段的控制信号是芯片 ZLG7290 的 SEGA~SEGG 引脚控制，需要显示的段码通过 IIC 总线传送到该芯片，见图 4-14、图 4-15。

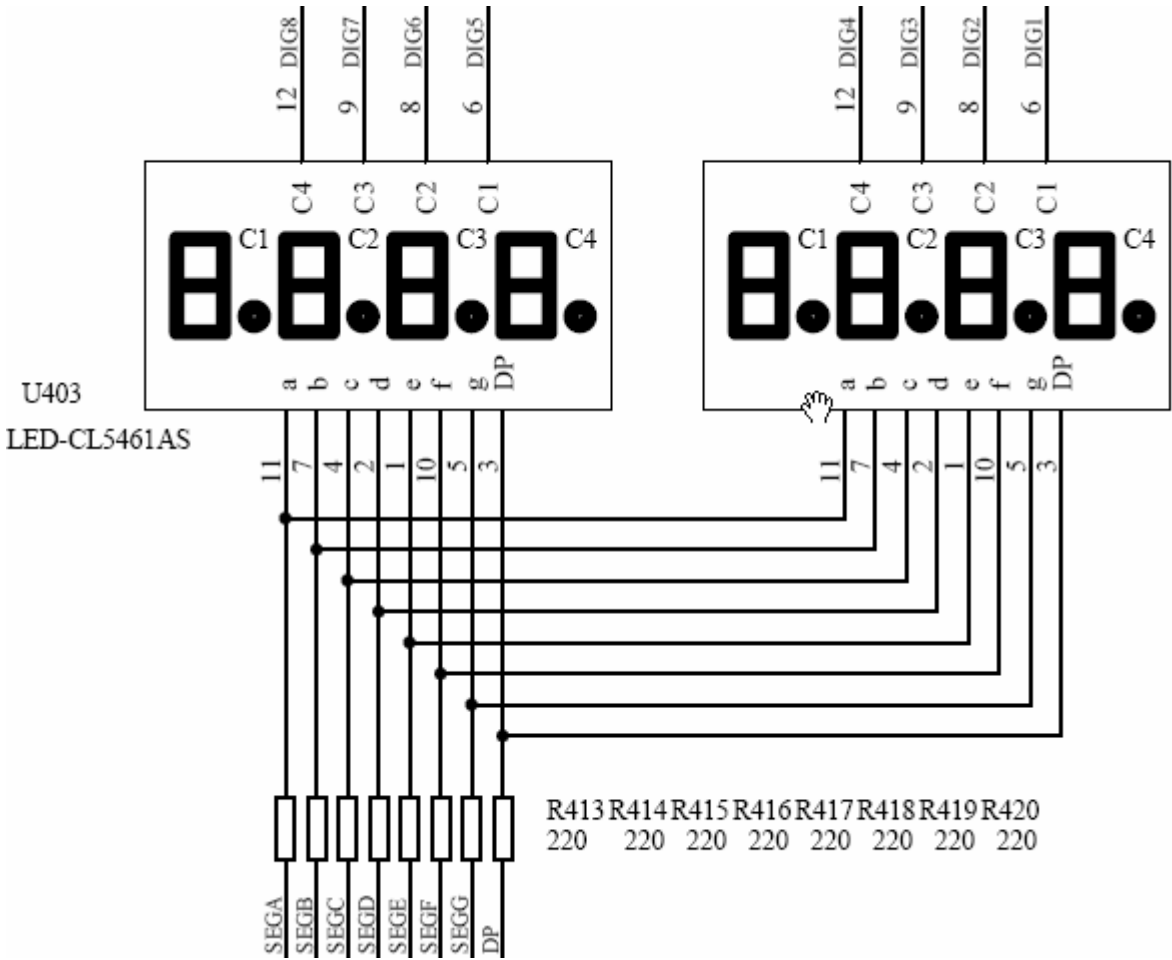


图 4-14 八段数码管连接电路

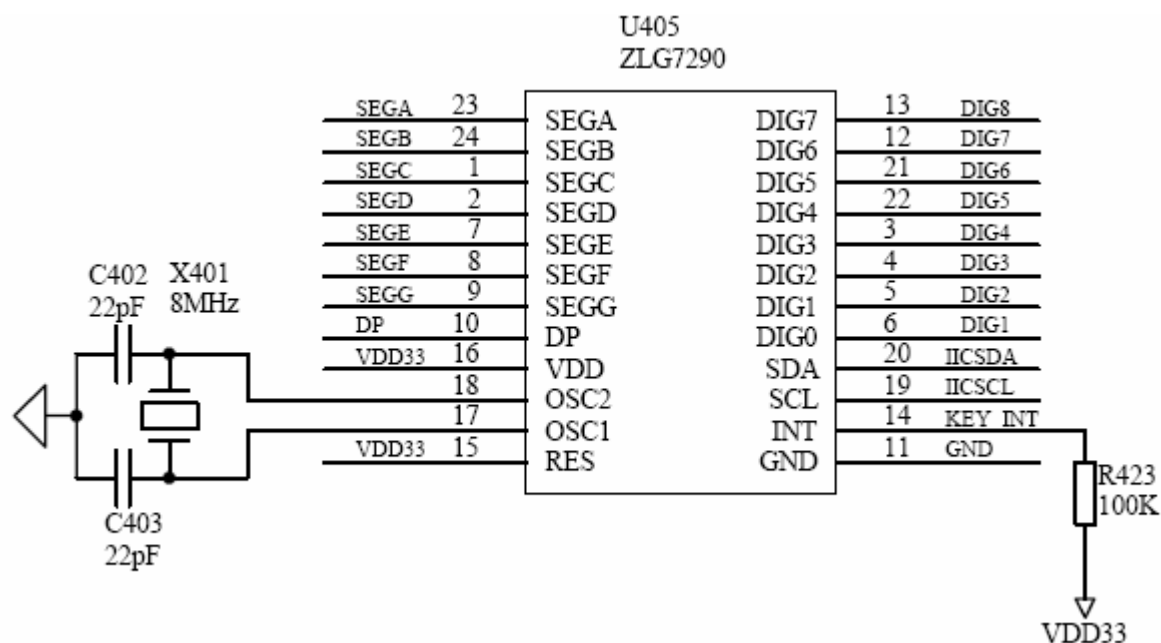


图 4-15 八段数码管控制电路

4.6.5 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.6_8led_test 子目录下的 8led_test.Uv2 例程，编译链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Go 或 F5 键运行程序。

4. 观察实验结果

1). 在 PC 机上观察超级终端程序主窗口, 可以看到如下界面:

```
boot success...

8-segment Digit LED Test Example (Please look at LED)
```

2). 实验系统八段数码管循环显示 0 ~ 9 字符。

5. 完成实验练习题

理解和掌握实验后, 完成实验练习题。

4.6.6 实验参考程序

```
/*-----*/
/*          function declare          */
/*-----*/

void led8_test(void);
void led8_disp_mem(int nMemory, int nLen, int nDirection);
void led8_disp(char cWhichS, char cWhichE, char uChar);

/*****
* name:      led8_test
* func:      test 8led
* para:      none
* ret:       none
* modify:
* comment:
*****/

void led8_test(void)
{
    int i, j, k;

    iic_init();
    for(;;)
    {
        for(j=0; j<10; j++)
        {
            for(i=0; i<8; i++)
            {
                k = 9-(i+j)%10;
                iic_write(0x70, 0x10+i, f_szDigital[k]);
            }
            delay(1000);
        }
    }
}
```

4.6.7 练习题

编写程序循环显示八段数码管的各段。

4.7 看门狗控制实验

4.7.1 实验目的

- 了解看门狗的作用。
- 掌握 S3C44BOX 处理器的看门狗控制器的使用。

4.7.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

4.7.3 实验内容

通过使用 S3C44BOX 处理器集成的看门狗模块，对其进行以下操作：

- 掌握看门狗的操作方式和用途。
- 对看门狗模块进行软件编程，实现看门狗的定时功能和复位功能。

4.7.4 实验原理

1. 看门狗概述

看门狗的作用是在微控制器受到干扰进入错误状态后，使系统在一定时间间隔内进行复位。因此看门狗是保证系统长期、可靠和稳定运行的有效措施。目前大部分嵌入式芯片片内都带有看门狗定时器，以此来提高系统运行的可靠性。

2. S3C44BOX 处理器的看门狗

S3C44BOX 里面的看门狗定时器是当系统被故障例如噪声和系统错误的干扰时，用来微处理器的复位操作的。同时看门狗定时器也可以当作一个通用的 16 位中断定时器来请求中断服务。看门狗定时器会在每 128 MCLK 发出一个复位信号。其主要特性如下：

- 16 位的看门狗定时器
- 当定时器溢出时发出中断请求或者系统复位。

看门狗功能框图

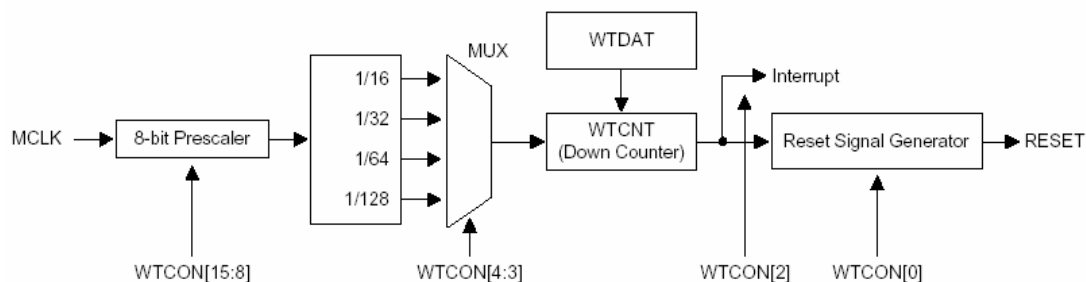


图 4.16 S3C44BOX 处理器的看门狗原理框图

看门狗模块包括预装比例因子放大器，一个四分频的分频器和一个 16 位的计数器。看门狗的时钟信号源自系统时钟（MCLK），为了得到比较宽范围的看门狗时钟信号，MCLK 先经过预装比例因子放大，然后再经过分配器进行分频。其中比例因子和之后的分频值，都可以由看门狗定时器的控制寄存器（WTCN）来决定。比例因子的有效范围值是 0~255。频率预分频可以有四个选择分别是 16 分频、32 分频、64 分频和 128 分频。

● 看门狗定时器时钟周期的计算

计算公式：

$$nWDTCountTime = 1/(MCLK/(Prescaler\ value + 1)/Division_factor)$$

-- $nWDTCountTime$: 看门狗定时器时钟周期

式中 MCLK 代表系统时钟，Prescaler value 为比例因子取值，Division_factor 为分频值。如果我将 MCLK=64MHz，Prescaler value=MCLK/1000000-1，Division_factor=128，则：

$$nWDTCountTime = 1/(64M/(63+1)/128) = 1.28\mu s$$

● 调试环境下的看门狗

当 S3C44BOX 用嵌入式 ICE 进行调试的时候，看门狗定时器复位功能将不起作用。看门狗定时器能够从 CPU 内核信号中确定当前模式是否是调试模式，如果看门狗定时器确定当前模式为调试模式，尽管看门狗定时器溢出，但看门狗定时器将不再发出复位信号。

● S3C44BOX 处理器看门狗的寄存器

（1）寄存器组

S3C44BOX 处理器集成的看门狗单元只使用到三个寄存器，即看门狗控制寄存（WTCN）、看门狗数据寄存器（WTDAT）、看门狗计数就寄存器（WTCNT）。

（2）看门狗控制寄存器（WTCN）

使用看门狗定时器控制寄存器（WTCN），可以开启和禁止看门狗定时器，可以从分频器中选择时钟信号源，中断使能和禁止，看门狗定时器复位使能和禁止。看门狗定时器将会在系统上电后，在系统出现故障时给予复位，如果系统不要求复位，看门狗定时器将不工作。

如果用户想把看门狗定时器用作一般的定时器来产生中断，那么必须使中断使能和看门狗定时器复位禁止。

| 名字 | 地址 | 访问 | 描述 | 复位值 |
|------|------------|-----|----------|--------|
| WTCN | 0x01D30000 | 读/写 | 看门狗控制寄存器 | 0x8021 |

表 4.17 WTCN 位描述

| WTCN | 位 | 描述 | 复位值 |
|--------|------|---------------------------|------|
| 预装比例因子 | 15:8 | 预装比例值，有效范围值 0~255 | 0x80 |
| 保留 | 7:6 | 保留 | 00 |
| 看门狗使能 | 5 | 使能和禁止看门狗定时器 0=禁止看门狗定时器 | 0 |

| | | | |
|------|-----|--|----|
| | | 1=使能看门狗定时器 | |
| 时钟选择 | 4:3 | 这两位决定时钟分频因素 00:1/16 01:1/32 10:1/64 11:1/128 | 00 |
| 中断使能 | 2 | 中断的禁止和使能 0=禁止中断产生 1=使能中断产生 | 0 |
| 保留 | 1 | 保留 | 0 |
| 复位使能 | 0 | 禁止和使能看门狗复位信号的输出 1=看门狗复位信号使能 0=看门狗复位信号禁止 | 1 |

(3) 看门狗数据寄存器 (WTDAT)

看门狗数据定时器 (WTDAT) 用于指定看门狗输出的时间。在对看门狗进行初始化操作的时候, 看门狗数据寄存器里面的内容不能自动地装载到看门狗计数器寄存器里面, 尽管如此, 第一个看门狗的输出可以由初始值 (0x8000) 来决定, 之后看门狗数据寄存器的值将自动装载到看门狗计数器寄存器里面。

| 名字 | 地址 | 访问 | 描述 | 复位值 |
|-------|------------|-----|----------|--------|
| WTDAT | 0x01D30004 | 读/写 | 看门狗数据寄存器 | 0x8000 |

表 4.18 WTDAT 位描述

| WTDAT | 位 | 描述 | 复位值 |
|--------|------|-----------|--------|
| 计数器转载值 | 15:0 | 看门狗计数器装载值 | 0x8000 |

(4) 看门狗计数寄存器 (WTCNT)

看门狗计数寄存器包含看门狗定时器在操作的时候计数器当前的计数值。注意, 当看门狗寄存器初始化的时候, 看门狗数据寄存器里面的值不能自动地装载到计数寄存器里面, 所以, 在看门狗定时器使能之前, 必须给看门狗计数器写上一个初始值。

| 名字 | 地址 | 访问 | 描述 | 复位值 |
|-------|------------|-----|-----------|--------|
| WTCNT | 0x01D30008 | 读/写 | 看门狗计数器寄存器 | 0x8000 |

表 4.19 WTCNT 位描述

| WTCNT | 位 | 描述 | 复位值 |
|-------|------|-------------|--------|
| 计数值 | 15:0 | 看门狗计数器当前计数值 | 0x8000 |

4.7.5 实验设计

1. 软件程序设计

由于看门狗只是对系统的复位或者中断的操作，所以不需要外围的硬件电路。要实现对看门狗的操作只需要对看门狗寄存器组进行操作，即对看门狗控制寄存器（WTCON）、看门狗数据寄存器（WTDAT）、看门狗计数寄存器（WTCNT）的操作。

一般操作流程如下：

- 1) 设置看门狗中断操作，包括全局中断和看门狗中断的使能和看门狗中断向量的定义。如果只是进行复位操作，不进行中断操作，这一步不用设置。
- 2) 对看门狗控制寄存器（WTCON）进行设置，包括设置比例因子、分频值、中断使能和复位使能等
- 3) 对看门狗数据寄存器（WTCNT）和计数寄存器（WTCON）进行设置。
- 4) 启动看门狗计数器。

2. 看门狗在 delay 里面的应用

在 44bllib.c 文件里面的 delay 用到了看门狗定时器来调整延时的时间。程序如下：

```

/*****
* name:      delay
* func:      delay time
* para:      nTime -- input, nTime=0: nAdjust the delay function by WatchDog timer.
*              nTime>0: the number of loop time, 100us resolution.
* ret:      none
* modify:
* comment:
*****/
void delay(int nTime)
{
    int nAdjust;
    int i;

    nAdjust = 0;

    if(nTime == 0)
    {
        nTime = 200;
        nAdjust = 1;
        f_nDelayLoopCount = 400;
        rWTCON = ((MCLK/1000000-1)<<8)|(<<3);           // 1M/64, Watch-dog, nRESET, interrupt
        disable

```

```

    rWTDAT = 0xffff;
    rWTCNT = 0xffff;
    rWTCON = ((MCLK/1000000-1)<<8)|(2<<3)|(1<<5);    // 1M/64, Watch-dog enable, nRESET,
interrupt disable
}

for(; nTime>0; nTime--)
{
    for(i=0; i<f_nDelayLoopCount; i++)
        ;
}

if(nAdjust==1)
{
    rWTCON = ((MCLK/1000000-1)<<8)|(2<<3);
    i = 0xffff-rWTCNT;                                // 1count/16us????????
    f_nDelayLoopCount = 8000000/(i*64);                // 400*100/(i*64/200)
}
}

```

在调整的过程中主要把看门狗看作一个定时器，在 $\text{time}=200$ ， $\text{Count_100us}=400$ 的时间内用看门狗定时器进行计数，然后，根据计数值来确定延时 100us 时的 Count_100us 。

根据程序和看门狗时钟周期频率输出的公式可以知道，在 $\text{time}=200$ ， $\text{Count_100us}=400$ 内看门狗计数器记了 i 个脉冲，而且看门狗时钟频率为 $\frac{64}{1M}s$ ，所以可以计算出一个 Count_100us 需要的时间，即 $\text{for}(i=0; i<\text{Count_100us}; i++)$ 执行一次的时间：

$$t = \frac{\frac{64}{1 \times 10^6} \times i}{200 \times 400}$$

当需要延时 100us 时所需要的 Count_100us 值为：

$$\text{delayloopcount} = 100 \times 10^{-6} / t = \frac{100 \times 10^{-6}}{\frac{64i}{\frac{1 \times 10^6}{200 \times 400}}} = \frac{8000000}{64i}$$

4.7.6 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 4.7_watchdog_test 子目录下的 watchdog_test.Uv2 例程，编译链接工程；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Go 或 F5 键运行程序。

4. 观察实验结果

在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...

WatchDog Timer Test Example

1 2 3 4 5 6 7 8 9 10

I will restart after 5 sec!!!
```

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

4.7.7 实验参考程序

1. 环境及函数声明

```
/*-----*/
/*                global variables                */
/*-----*/

volatile int f_nWdtIntnum;

/*-----*/
/*                function declare                */
/*-----*/

void wdtimer_test(void);
void wdt_int(void);// __attribute__ ((interrupt ("IRQ")));
```

2. 初始化程序

```
/*-----*/
```



```

* name:      Main
* func:      c code entry
* para:      none
* ret:       none
* modify:
* comment:
*****/

void Main(void)
{
    sys_init();                // Initial 44B0X's Interrupt,Port and UART

    wdtimer_test();            // test start
}

```

3. 看门狗控制程序

```

/*****
* name:      wdtimer _test
* func:      Test the interrupt function and reset function in watdch-dog
* para:      none
* ret:       none
* modify:
* comment:
*****/

void wdtimer_test(void)
{

    uart_printf("\n\r WatchDog Timer Test Example\n");
    // Enable interrupt
    rINTMSK = ~(BIT_GLOBAL | BIT_WDT);                // enable interrupt
    pISR_WDT = (unsigned)wdt_int;                      // set WDT interrupt handler
    f_nWdtIntnum = 0;

    //test interrupt function
    rWTCON = ((MCLK/1000000-1)<<8) | (3<<3) | (1<<2);    // nWDTCountTime = 1/64/128, interrupt
enable
    rWTDAT = 7812;                                        // 1s = 7812 * nWDTCountTime
    rWTCNT = 7812;
    rWTCON = rWTCON | (1<<5);                            // Enable Watch-dog timer

    while( f_nWdtIntnum != 10);

    //test reset function
    rWTCON = ((MCLK/1000000-1)<<8) | (3<<3) | (1);        // 1/66/128, reset enable
    uart_printf("\n\r will restart after 5 sec!!!\n");
    rWTCNT = 7812*5;                                        // waiting 5 seconds
}

```

```

rWTCON = rWTCON | (1<<5);           // Enable Watch-dog timer
while(1);

rINTMSK = BIT_GLOBAL;
}

/*****
* name:      wdt_int
* func:      wdt interrupt handler
* para:      none
* ret:       none
* modify:
* comment:
*****/
void wdt_int(void)
{
    rI_ISPC = BIT_WDT;
    uart_printf("%d ",++f_nWdtIntnum);
}

```

4.7.8 练习题

参考试验程序，重新调整看门狗定时器的分频数和计数值，让看门狗定时器 1s 发生一次中断，3s 后产生复位。

第五章 人机接口实验

5.1 液晶显示实验

5.1.1 实验目的

- 初步掌握液晶屏的使用及其电路设计方法。
- 掌握 S3C44B0X 处理器的 LCD 控制器的使用。
- 通过实验掌握液晶显示文本及图形的方法与程序设计。

5.1.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

5.1.3 实验内容

通过使用 Embest EduKit-III 实验板的 256 色彩色液晶屏（320x240）进行电路设计，掌握液晶屏作为人机接口界面的设计方法，并编写程序实现：

- 画出多个矩形框
- 显示 ASCII 字符
- 显示汉字字符
- 显示彩色位图

5.1.4 实验原理

1. 液晶显示屏（LCD）

液晶屏（LCD: Liquid Crystal Display）主要用于显示文本及图形信息。液晶显示屏具有轻薄、体积小、低耗电量、无辐射危险、平面直角显示以及影像稳定不闪烁等特点，因此在许多电子应用系统中，常使用液晶屏作为人机界面。

● 主要类型及性能参数

液晶显示屏按显示原理分为 STN 和 TFT 两种：

STN（Super Twisted Nematic，超扭曲向列）液晶屏

STN 液晶显示器与液晶材料、光线的干涉现象有关，因此显示的色调以淡绿色与橘色为主。STN 液晶显示器中，使用 X、Y 轴交叉的单纯电极驱动方式，即 X、Y 轴由垂直与水平方向的驱动电极构成，水平方向驱动电压控制显示部分为亮或暗，垂直方向的电极则负责驱动液晶分子的显示。STN 液晶显示屏加上彩色滤光片，并将单色显示矩阵中的每一像素分成三个子像素，分别通过彩色滤光片显示红、绿、蓝三原色，也可以显示出色彩。单色液晶屏及灰度液晶屏都是 STN 液晶屏。

TFT（Thin Film Transistor，薄膜晶体管）彩色液晶屏

随着液晶显示技术的不断发展和进步，TFT 液晶显示屏被广泛用于制作成电脑中的液晶显示设备。TFT 液晶显示屏既可在笔记本电脑上应用（现在大多数笔记本电脑都使用 TFT 显示屏），也常用于主流台式显示器。

使用液晶显示屏时，主要考虑的参数有外形尺寸、分辨率、点宽、色彩模式等。以下是 Embest EduKit-III 实验板所选用的液晶屏（LRH9J515XA STN/BW）主要参数：

表 5-1 LRH9J515XA STN/BW 液晶屏主要技术参数

| | | | | | |
|----|-------------|------|-----------------|----|--------|
| 型号 | LRH9J515XA | 外形尺寸 | 93.8×75.1×5mm | 重量 | 45g |
| 像素 | 320 × 240 | 画面尺寸 | 9.6cm (3.8inch) | 色彩 | 16 级灰度 |
| 电压 | 21.5V (25℃) | 点宽 | 0.24 mm/dot | 附加 | 带驱动逻辑 |

可视屏幕的尺寸及参数示意如图 5-1 所示：

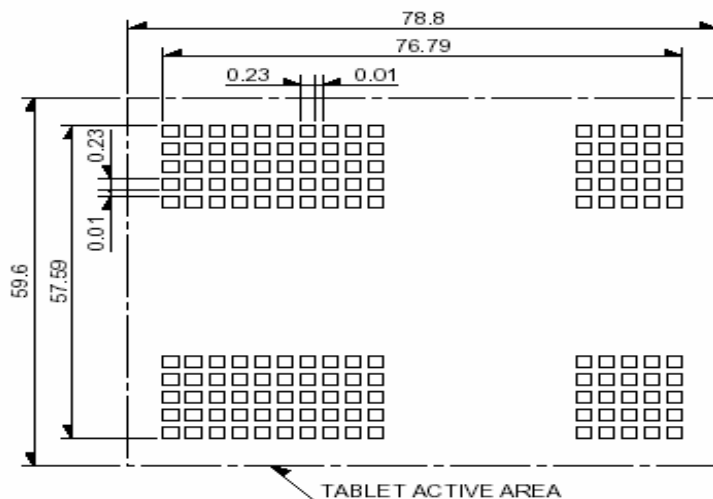


图 5-1 液晶屏参数示意图

液晶屏外形如图 5-2 所示：



图 5-2 LRH9J515XA STN/BW 液晶屏外形

● 驱动与显示

液晶屏的显示要求设计专门的驱动与显示控制电路。驱动电路包括提供液晶屏的驱动电源和液晶分子偏置电压，以及液晶显示屏的驱动逻辑；显示控制部分可由专门的硬件电路组成，也可以采用集成电路（IC）模块，比如 EPSON 的视频驱动器等；还可以使用处理器外围 LCD 控制模块。实验板的驱动与显示系统包括 S3C44BOX 片内外设 LCD 控制器、液晶显示屏的驱动逻辑以及外围驱动电路。

2. S3C44BOX LCD 控制器

● 介绍

S3C44BOX 处理器集成了 LCD 控制器，支持 4 位单扫描、4 位双扫描和 8 位单扫描工作方式。处理器使用内部 RAM 区作为显示缓存，并支持屏幕水平和垂直滚动显示。数据的传送采用 DMA（直接内存访问）方式，以达到最小的延迟。根据实际硬件水平和垂直像素点数、传送数据位数、时间线和帧速率方式等进行编程以支持多种类型的液晶屏。可以支持的液晶类型有：

- 单色液晶
- 4 级或 16 级灰度屏（基于时间抖动算法或帧速率控制--FRC）
- 256 色彩色液晶（STN 液晶）

● 显示控制

LCD 控制器主要提供液晶屏显示数据的传送、时钟和各种信号的产生与控制功能。S3C44BOX 处理器的 LCD 控制器主要部分框图如图 5-3 所示：

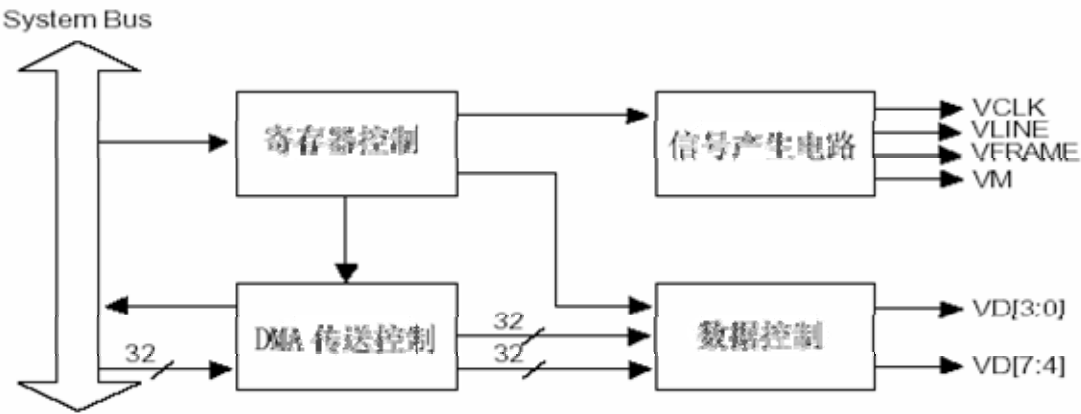


图 5-3 LCD 控制器框图

(1) LCD 控制器接口说明

表 5-2 S3C44BOX LCD 控制器接口说明

| | | |
|--------|--------|-------------------------|
| VCLK | 刷新时钟 | 为数据传送提供时钟信号（低于 16.5MHz） |
| VLINE | 水平同步脉冲 | 提供行信号，即行频率 |
| VFRAME | 帧同步信号 | 帧显示控制信号。显示完整帧后有效 |
| VM | 交流控制电压 | 极性的改变控制液晶分子的显示 |
| VD3:0 | 数据线 | 数据输入。双扫描时的高 4 位数据输入 |
| VD7:4 | 数据线 | 数据输入。双扫描时的低 4 位数据输入 |

(2) LCD 控制器信号时序

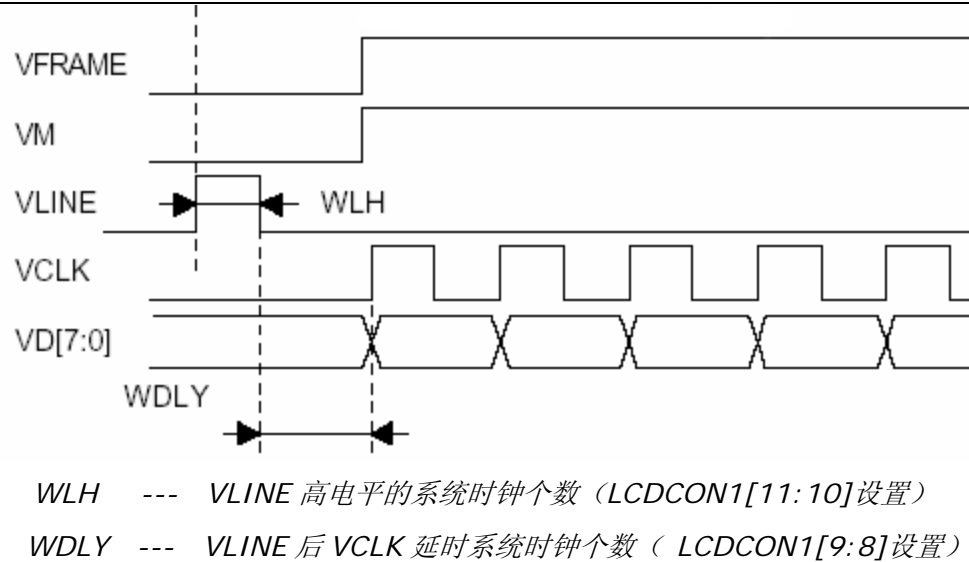


图 5-4 控制器信号时序（一个 LINE）

(3) 扫描模式支持

S3C44BOX 处理器 LCD 控制器扫描工作方式通过 DISMOD (LCDCON1[6:5]) 设置。

表 5-3 扫描模式选择

| DISMOD | 00 | 01 | 10 | 11 |
|--------|-----------|-----------|-----------|----|
| 模式 | 4-bit 双扫描 | 4-bit 单扫描 | 8-bit 单扫描 | 无 |

4 位单扫描 --- 显示控制器扫描线从左上角位置进行数据显示。显示数据从 VD[3:0]获得；彩色液晶屏数据位代表 RGB 色

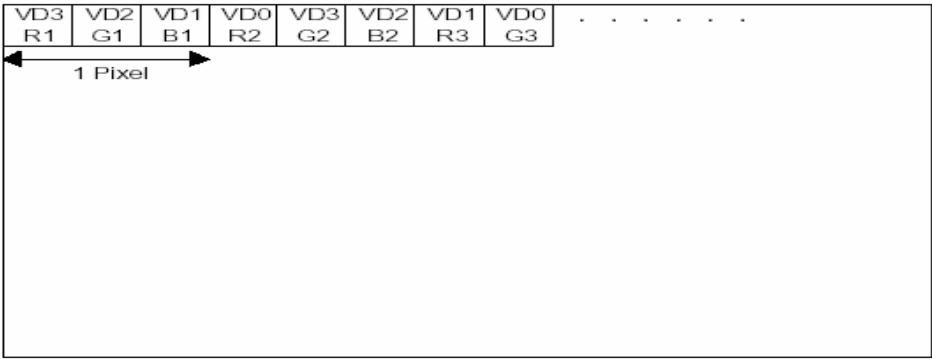


图 5-5 4 位单扫描

4 位双扫描 --- 显示控制器分别使用两个扫描线进行数据显示。显示数据从 VD[3:0]获得高扫描数据；VD[7:4]获得低扫描数据；彩色液晶屏数据位代表 RGB 色

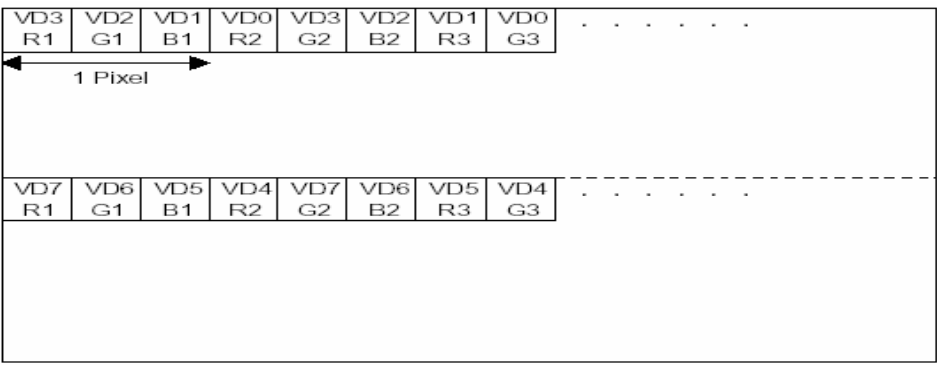


图 5-6 4 位双扫描

8 位单扫描 --- 显示控制器扫描线从左上角位置进行数据显示。显示数据从 VD[7:0] 获得；彩色液晶屏数据位代表 RGB 色

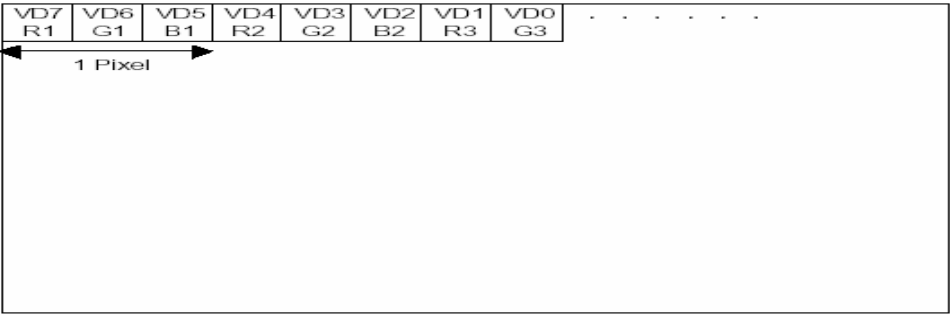


图 5-7 8 位单扫描

(4) 数据的存放与显示

液晶控制器传送的数据表示了一个像素的属性：4 级灰度屏用两个数据位；16 级灰度屏时使用 4 个数据位；RGB 彩色液晶屏使用 8 个数据位（R[7:5]、G[4:2]、B[1:0]）。

显示缓存中存放的数据必须符合硬件及软件设置，即要注意字节对齐方式。

在 4 位或 8 位单扫描方式时，数据的存放与显示如图 5-8 所示：

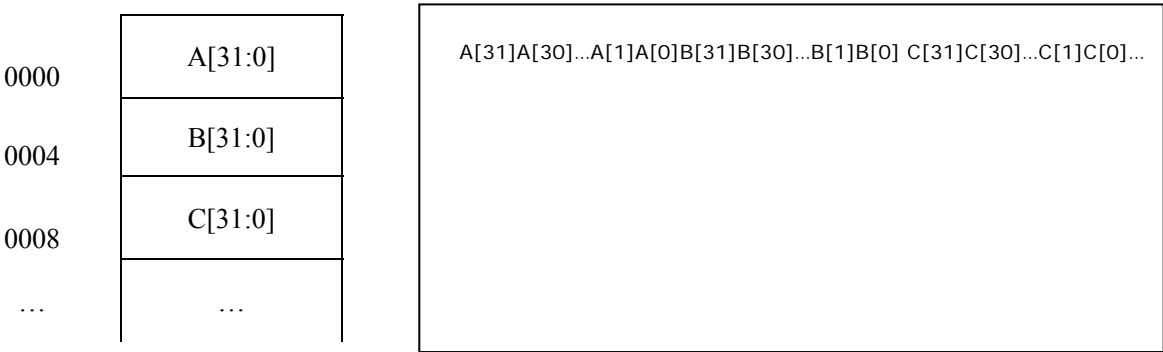


图 5-8 4 位或 8 位单扫描数据显示

在 4 位双扫描方式时，数据的存放与显示如图 5-9 所示：

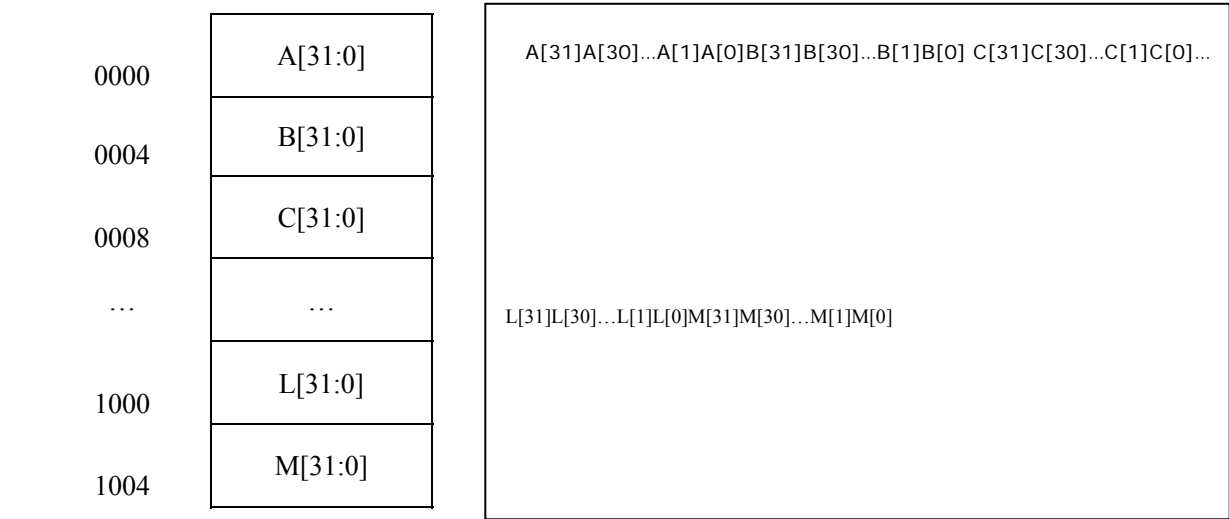


图 5-9 4 位双扫描数据显示

(5) LCD 控制器寄存器

S3C44BOX LCD 处理器所包含的可编程控制寄存器共有 18 个。

表 5-4 LCD 控制器寄存器列表

| 寄存器名 | 内存地址 | 读写 | 说 明 | |
|-----------|------------|-----|--------------|---------------|
| LCDCON1 | 0x01f00000 | R/W | LCD 控制寄存器 1 | 工作信号控制寄存器 |
| LCDCON2 | 0x01f00004 | R/W | LCD 控制寄存器 2 | 液晶屏水平垂直尺寸定义 |
| LCDCON3 | 0x01f00040 | R/W | LCD 控制寄存器 3 | 自测试设定，只用到最低位 |
| LCDSADDR1 | 0x01f00008 | R/W | 高位帧缓存地址寄存器 1 | 液晶类型和扫描模式定义 |
| LCDSADDR2 | 0x01f0000c | R/W | 低位帧缓存地址寄存器 2 | 设定显示缓存区信息 |
| LCDSADDR3 | 0x01f00010 | R/W | 虚屏地址寄存器 | 设定虚屏偏址和页面宽度 |
| REDLUT | 0x01f00014 | R/W | 红色定义寄存器 | 定义 8 组红色数据查找表 |
| GREENLUT | 0x01f00018 | R/W | 绿色定义寄存器 | 定义 8 组红色数据查找表 |
| BLUELUT | 0x01f0001c | R/W | 蓝色定义寄存器 | 定义 4 组红色数据查找表 |

| | | | | |
|-----------------|----------------|-----|----------|--------------------|
| DP1_2 | 0x01f000 20 | R/W | 1/2 抖动设定 | 推荐使用 0xa5a5 |
| DP4_7 | 0x01f000 24 | R/W | 4/7 抖动设定 | 推荐使用 0xba5da65 |
| DP3_5 | 0x01f000 28 | R/W | 3/5 抖动设定 | 推荐使用 0xa5a5f |
| DP2_3 | 0x01f000 2c | R/W | 2/3 抖动设定 | 推荐使用 0xd6b |
| DP5_7 | 0x01f000 30 | R/W | 5/7 抖动设定 | 推荐使用 0xeb7b5ed |
| DP3_4 | 0x01f000 34 | R/W | 3/4 抖动设定 | 推荐使用 0x7dbe |
| DP4_5 | 0x01f000 38 | R/W | 4/5 抖动设定 | 推荐使用 0x7ebdf |
| DP6_7 | 0x01f000 3c | R/W | 6/7 抖动设定 | 推荐使用 0x7fdfbfe |
| DITHMODE | 0x01f000 44 | R/W | 抖动模式寄存器 | 推荐使用 0x12210 或 0x0 |

注：以下实验说明中只是简单地介绍控制寄存器的含义，详细使用请参考 S3C44BOX 处理器数据手册。

(6) LCD 控制器主要参数设定

正确使用 S3C44BOX LCD 控制器，必须设置控制器所有 18 个寄存器。

控制器信号 VFRME、VCLK、VLIN 和 VM 要求配置控制寄存器 LCDCON1/2；液晶屏的显示与控制，以及数据的存取控制要求配置其他相关寄存器，详见以下说明。

- 设置 VM、VFRAME、VLIN

VM 信号通过改变液晶的行列电压的极性来控制像素的显示，VM 速率可以配置 LCDCON1 寄存器的 MMODE 位及 LCDCON2 寄存器的 MVAL[7:0]。

$$\text{VM 速率} = \text{VLIN 速率} / (2 * \text{MVAL})$$

VFRAME 和 VLIN 信号可以根据液晶屏的尺寸及显示模式，配置 LCDCON2 寄存器的 HOZVAL 和 LINEVAL 值，即：

$$\text{HOZVAL} = (\text{水平尺寸} / \text{VD 数据位}) - 1$$

彩色液晶屏时：水平尺寸 = 3 * 水平像素点数；

VD 数据位：=4 --- 4 位单/双扫描模式；

=8 --- 8 位单扫描模式

$$\text{LINEVAL} = \text{垂直尺寸} - 1 \quad \text{单扫描模式}$$

$$\text{LINEVAL} = (\text{垂直尺寸} / 2) - 1 \quad \text{双扫描模式}$$

- 设定 VCLK

VCLK 是 LCD 控制器的时钟信号, S3C44BOX 处理器在 66MHz 时钟频率时最高频率为 16.5MHz, 这可以支持现在所有液晶屏类型。VCLK 的计算需要先计算数据传送速率, 并由此设定的一个大于数据传送速率的值为 VCLKVAL (LCDCON1[21:12])。

数据传送速率 = 水平尺寸 × 垂直尺寸 × 帧速率 × 模式值(MV)

表 5-5 模式值

| 液晶类型和扫描模式 | 4 位单扫描 | 8 位单扫描或 4 位双扫描 |
|-----------|--------|----------------|
| 单色液晶 | 1/4 | 1/8 |
| 4 级灰度屏 | 1/4 | 1/8 |
| 16 级灰度屏 | 1/4 | 1/8 |
| 彩色液晶 | 3/4 | 3/8 |

帧速率可由以下公式得到

$$VCLK(Hz) = MCLK / (CLKVAL \times 2) \quad LKVAL \text{ 大于数据传送速率且不小于 } 2$$

帧速率(Hz) =

$$\left((1/VCLK) \times (HOZVAL+1) + (1/MCLK) \times (WLH+WDLY+LINEBLANK) \right) \times (LINEVAL+1)^{-1}$$

LINEBANK ---- 水平扫描信号 LINE 持续时间设置 (MCLK 个数)

LINEVAL ---- 显示屏的垂直尺寸

VCLK 的计算还可以使用以下公式:

$$VCLK(Hz) = (HOZVAL+1) / \left[(1 / (\text{帧速率} \times (LINEVAL+1))) - ((WLH+WDLY+LINEBLANK) / MCLK) \right]$$

- 设定数据帧显示控制 (LCDBASEU、LCDBASEL、PAGEWIDTH、OFFSIZE、LCDBANK)
 - (1) LCDBASEU 设置显示扫描方式中的开始地址 (单扫描方式) 或高位缓存地址 (双扫描方式);
 - (2) LCDBASEL 是设置双扫描方式的低位缓存开始地址。可用以下计算公式:

$$LCDBASEL = LCDBASEU + (PAGEWIDTH + OFFSIZE) \times (LINEVAL + 1)$$
 - (3) PAGEWDTH 是显示存储区的可见帧宽度 (半字数)
 - (4) OFFSIZE 半字数。是显示存储区的前行最后半字和后行第一个半字之间的半字数
 - (5) LCDBANK 是访问显示存储区的地址 A[27:22]值。ENVID=1 时该值不能改变。

(7) 液晶屏的支持与设定

对于 4 级灰度屏 (2 位数据), LCD 控制器通过设置 BULELUT[15:0]指定使用的灰度级, 并且从 0 - 4 级使用 BULELUT 的 4 个数据位。16 级灰度屏使用 BULELUT 的每一位来表示灰度级别。

使用 16 级灰度屏时, LCD 控制器参数设定可参考:

参考 1 LCD 液晶屏: 320*240; 16 级灰度; 单扫描模式

数据帧首地址 = 0xc300000; 偏移点数 = 2048 点 (512 个半字);

LINEVAL = 240 - 1 = 0xEF;

PAGEWIDTH = 320*4/16 = 0x50;

OFFSIZE = 512 = 0x200;

LCDBANK = 0xc300000 >> 22 = 0x30;

LCDBASEU = 0x100000 >> 1 = 0x80000;
LCDBASEL = 0x80000 + (0x50 + 0x200) * (0xef + 1) = 0xa2b00;

参考 2 LCD 液晶屏：320*240；16 级灰度；双扫描模式
数据帧首地址 = 0xc300000； 偏移点数 = 2048 点（ 512 个半字 ）；

LINEVAL = 120-1 = 0x77;
PAGEWIDTH = 320*4/16 = 0x50;
OFFSIZE = 512 = 0x200;
LCDBANK = 0xc300000 >> 22 = 0x30;
LCDBASEU = 0x100000 >> 1 = 0x80000;
LCDBASEL = 0x80000 + (0x50 + 0x200) * (0x77 + 1) = 0x91580;

彩色屏的 LCD 控制器参数设定参考实验程序。

5.1.5 实验设计

1. 电路设计

进行液晶屏控制电路设计时必须提供电源驱动、偏压驱动以及 LCD 显示控制器。由于 S3C44B0 X 处理器本身自带 LCD 控制器，而且可以驱动实验板所选用的液晶屏，所以控制电路的设计可以省去显示控制电路，只需进行电源驱动和偏压驱动的电

● 液晶电路结构框图

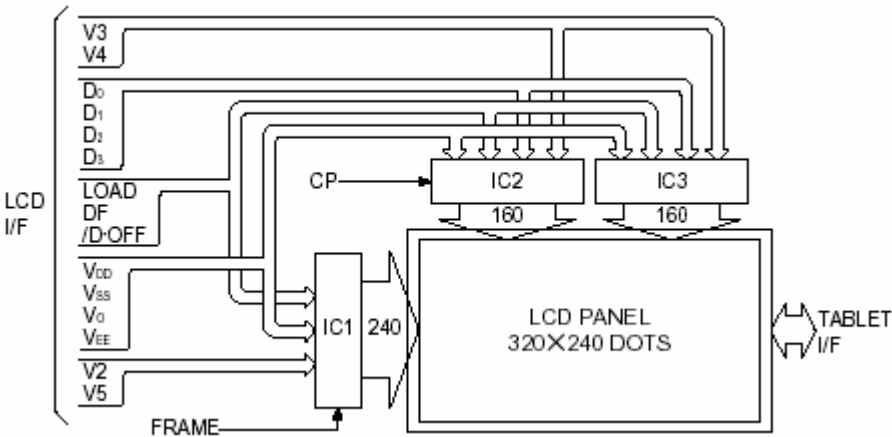


图 5-10 LCD 结构框图

● 引脚说明

表 5-6 液晶屏幕管脚

| | | | | | |
|---|------------|----|---------------|--------|----------|
| 1 | V5 偏压 5 | 6 | VO 电源地 | 11 | CP 时钟宽度 |
| 2 | V2 偏压 2 | 7 | LOAD 逻辑控制（内部） | 12 | V4 偏压 4 |
| 3 | VEE 驱动电压 | 8 | VSS 信号地 | 13 | V3 偏压 3 |
| 4 | VDD 逻辑电压 | 9 | DF 驱动交流信号 | 14 -17 | D3-D0 数据 |
| 5 | FRAME | 10 | /D-OFF 像素开关 | 18 | NC 未定义 |

● 控制电路设计

由前述可知实验板所选用的液晶屏的驱动电源是 21.5V，因此直接使用实验系统的 3V 或 5V 电源时需要电压升压控制，实验系统采用的是 MAX629 电源管理模块，以提供液晶屏的驱动电源。偏压电源可由系统升压后的电源分压得到。以下是 EduKit-III 实验板的电源驱动和偏压驱动参考电路。

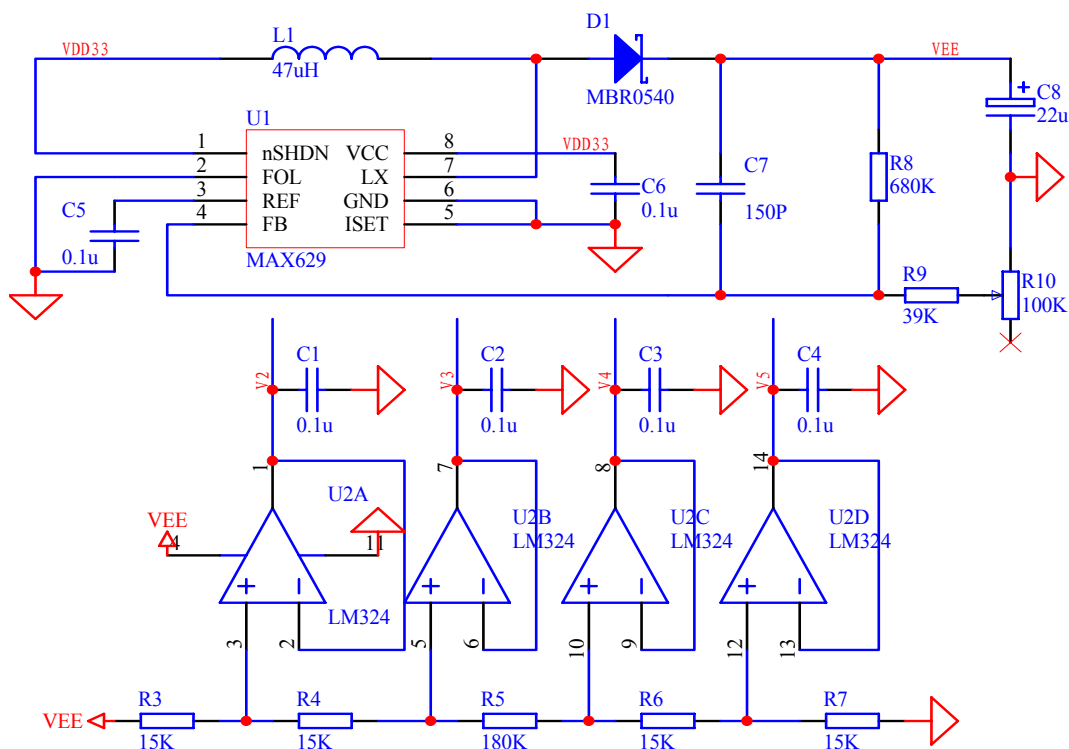


图 5-11 电源驱动与偏压驱动电路

2. 软件程序设计

由于实验要求在液晶显示屏上显示包括矩形、字符和位图文件，所以实验程序设计主要包括三部分。

● 设计思路

使用液晶屏显示最基本的是像素控制数据的使用，像素控制数据的存放与传送形式，决定了显示的效果。这也是所有显示控制的基本程序设计思想。图形显示可以直接使用像素控制函数实现；把像素控制数据按一定形式存放即可实现字符显示，比如 ASCII 字符、语言文字字符等。

Embest ARM 教学系统的像素控制函数按如下设计：

```

/*****
*   EduKit-III LCD 像素显示宏定义
*       LCD_PUT_PIXEL ----- 把像素发送到显示数据暂存区
*       LCD_ACTIVE_PUT_PIXEL ----- 发送到显示缓冲区（直接驱动 LCD）
*****/

#define LCD_PUT_PIXEL(x, y, c) \
    (*(UINT32T*)(LCD_VIRTUAL_BUFFER + (y) * SCR_XSIZE / 2 + ((x) / 8 * 4))) = \
    (*(UINT32T*)(LCD_VIRTUAL_BUFFER + (y) * SCR_XSIZE / 2 + ((x) / 8 * 4))) & \
    (~((0xf0000000 >> (((x) % 8) * 4))) | ((c) << (7 - ((x) % 8) * 4)))

#define LCD_ACTIVE_PUT_PIXEL(x, y, c) \

```

```
(*(UINT32T*)(LCD_ACTIVE_BUFFER + (y) * SCR_XSIZE / 2 + (319 - (x)) / 8 * 4)) = \
(*(UINT32T*)(LCD_ACTIVE_BUFFER + (y) * SCR_XSIZE / 2 + (319 - (x)) / 8 * 4)) & \
(~(0xf0000000 >> (((319 - (x))%8)*4))) |((c) << (7 - (319 - (x))%8) * 4)
```

● 矩形显示

矩形显示可以通过两条水平线和两条垂直线组成，因此在液晶显示屏上显示矩形实际就是画线函数的实现。画线函数则通过反复调用像素控制函数得到水平线或垂直线。

● 字符显示

字符的显示包括 ASCII 字符和汉字的显示。字符的显示可以采用多种形式字体，其中常用的字体大小有（W x H 或 H x W）：8x8、8x16、12x12、16x16、16x24、24x24 等，用户可以使用不同的字库以显示不同的字体。如实验系统中使用 8x16 字体显示 ASCII 字符，使用 16x16 字体显示汉字。

不管显示 ASCII 字符还是点阵汉字，都是通过查找预先定义好的字符表来实现，这个存储字符的表我们叫做库，相应的有 ASCII 库和汉字库。

```
const INT8U g_ucAscii8x16[]={ //ASCII 字符查找表 }
```

ASCII 字符的存储是把字符显示数据存放在以字符的 ASCII 值为下标的库文件（数组）中，显示时再按照字体的长与宽和库的关系取出作为像素控制数据显示。ASCII 库文件只存放 ANSI ASCII 的共 255 个字符。请参考样例程序。

```
const INT8U g_ucHZK16[]={ //点阵汉字查找表 }
```

点阵汉字库是按照方阵形式进行数据存放，所以汉字库的字体只能是方形的。汉字库的大小与汉字显示的个数及点阵数成正比。请参考样例程序。

● 位图文件显示

通过把位图文件转换成一定容量的显示数组，并按照一定的数据结构存放。与字符的显示一样，传送的数据需要设计软件控制程序。

Embest ARM 教学系统位图显示的存放数据结构及控制程序：

```
const INT8U g_ucBitmap [] = { // 位图文件数据};
```

位图显示（请参考样例程序）

```
void bitmap_view320x240x256(UINT8T *pBuffer);
```

5.1.6 实验操作步骤

1. 准备实验环境



使用 ULINK USB-JTAG 仿真器连接到目标板上，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；

- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上，打开实验例程目录 5.1_color_lcd_test 子目录下的 color_lcd _test.Uv2 例程，编译链接工程，直到链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境(工程默认已经配置正确)， 点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用！！！这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Run 或 F5 键全速运行程序，或者单步调试程序。

4. 观察实验结果

4. 观察实验结果

在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...

LCD display Test Example(please look at LCD screen)
```

观察 LCD 液晶屏，用户可以看到包含多个矩形框、ASCII 字符、汉字字符和彩色位图显示；

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

5.1.7 实验参考程序

1. 液晶屏初始化

```
/******
* name:      lcd_init()
* func:      Initialize LCD Controller
* para:      none
* ret:       none
* modify:
* comment:
*****/

void lcd_init(void)
{
    rDITHMODE = 0x12210;
```

```

rDP1_2 = 0xa5a5;
rDP4_7 = 0xba5da65;
rDP3_5 = 0xa5a5f;
rDP2_3 = 0xd6b;
rDP5_7 = 0xeb7b5ed;
rDP3_4 = 0x7dbe;
rDP4_5 = 0x7ebdf;
rDP6_7 = 0x7fdbf;

// disable, 8B_SINGL_SCAN, WDLY=16clk, WLH=16clk,
rLCDCON1 = (0x0)|(2<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_COLOR<<12);
// LINEBLANK=10 (without any calculation)
rLCDCON2 = (LINEVAL)|(HOZVAL_COLOR<<10)|(10<<21);
rLCDCON3 = 0;

// 256-color, LCDBANK, LCDBASEU
rLCSADDR1 = (0x3<<27) | ( ((unsigned int)g_unLcdActiveBuffer>>22)<<21 ) | M5D(((unsigned int)g_unLcdActiveBuffer>>1));
rLCSADDR2 = M5D((((unsigned int)g_unLcdActiveBuffer+(SCR_XSIZE*LCD_YSIZE))>>1)) | (MVAL<<21);
rLCSADDR3 = (LCD_XSIZE/2) | ( ((SCR_XSIZE-LCD_XSIZE)/2)<<9 );

//The following value has to be changed for better display.
rREDLUT =0xfdb96420; // 1111 1101 1011 1001 0110 0100 0010 0000
rGREENLUT=0xfdb96420; // 1111 1101 1011 1001 0110 0100 0010 0000
rBLUELUT =0xfb40; // 1111 1011 0100 0000

rLCDCON1=(0x1)|(2<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_COLOR<<12);
rPDATE=rPDATE&0x0e;
lcd_clr();
}

```

2. 显示像素和位图

```

#define LCD_PutPixel(x, y, c)\
    g_unLcdActiveBuffer[(y)][(x)/4]=(( g_unLcdActiveBuffer[(y)][(x)/4] & ~(0xff000000>>((x)%4)*8)) | ( (c)\
<<((4-1-((x)%4))*8) ));
#define LCD_ActivePutPixel(x, y, c)\
    g_unLcdActiveBuffer[(y)][(x)/4]=(( g_unLcdActiveBuffer[(y)][(x)/4] & ~(0xff000000>>((x)%4)*8)) | \

```

```

/*****
* name:      bitmap_view320x240x256
* func:      display bitmap
* para:      pBuffer -- input, bitmap data
* ret:       none
* modify:
* comment:
*****/

void bitmap_view320x240x256(UINT8T *pBuffer)
{
    UINT32T i, j;
    UINT32T *pView = (UINT32T*)g_unLcdActiveBuffer;

    for (i = 0; i < SCR_XSIZE * SCR_YSIZE / 4; i++)
    {
        *pView = ((*pBuffer) << 24) + ((*pBuffer+1)) << 16) + ((*pBuffer+2)) << 8) + (*pBuffer+3));
        pView++;
        pBuffer += 4;
    }
}

```

5.1.8 练习题

修改程序使用 DMA 方式在彩色液晶屏上显示彩色位图。

5.2 5x4 键盘控制实验

5.2.1 实验目的

- 通过实验掌握键盘控制与设计方法。
- 熟练编写 ARM 核处理器 S3C44BOX 中断处理程序。

5.2.2 实验设备

- 硬件: ULINK USB-JTAG 仿真器套件, PC 机。
- 软件: µVision IDE for ARM 集成开发环境, Windows 98/2000/NT/XP。

5.2.3 实验内容

- 使用实验板上 5x4 用户键盘, 编写程序接收键盘中断。
- 通过 IIC 总线读入键值, 并将读到的键值发送到串口。

5.2.4 实验原理

用户设计行列键盘接口, 一般常采用三种方法读取键值。一种是中断式, 另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。

扫描法

对键盘上的某一行送低电平，其他为高电平，然后读取列值，若列值中有一位是低，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，表明有键按下；接着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查表得到键码。

5.2.5 实验设计

1. 键盘硬件电路设计

(1) 键盘控制电路

键盘控制电路使用芯片 ZLG7290 控制，如图 5-12。

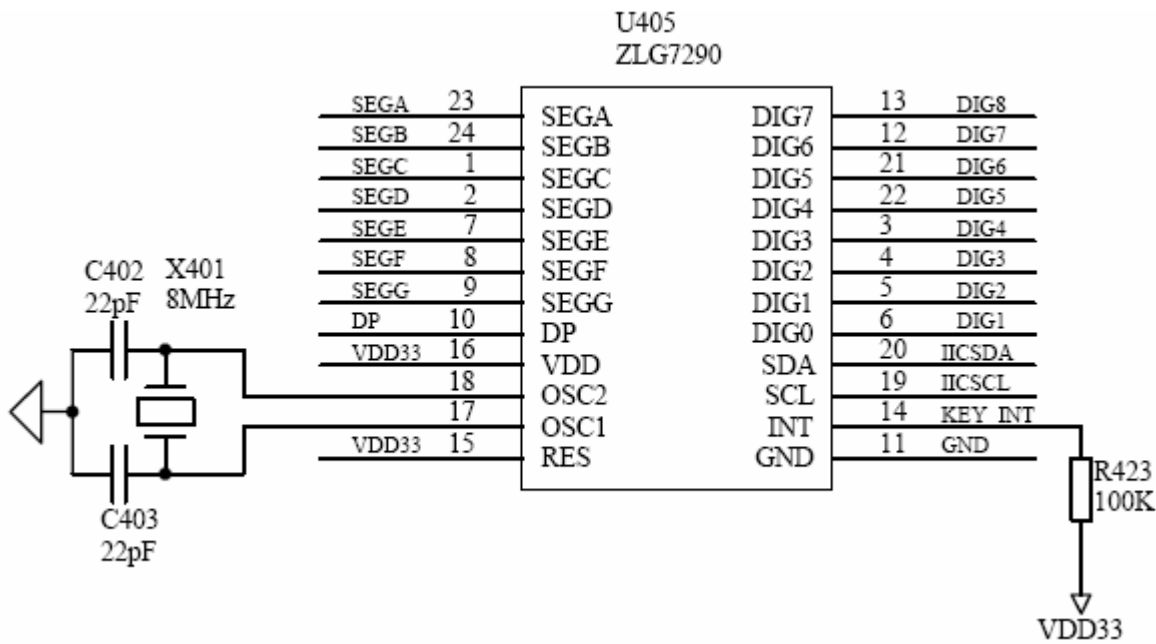


图 5-12 5x4 键盘控制电路

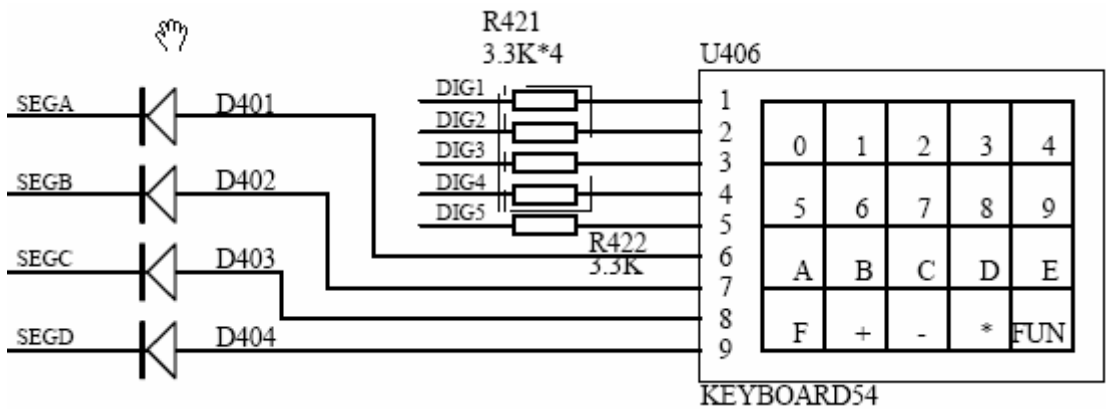


图 5-13 5x4 键盘连接电路

(2) 工作过程

键盘动作由芯片 ZLG7290 检测，当键盘按下时，芯片检测到后在 INT 引脚产生中断触发电平通知处理器，处理器通过 IIC 总线读取芯片中保存的键值。

5.2.6 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接到目标板上，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上，打开实验例程目录 5.2_keyboard_test 子目录下的 keyboard_test.Uv2 例程，编译链接工程，直到链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Run 或 F5 键全速运行程序，或者单步调试程序。

4. 观察实验结果

- 1). 在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...
Keyboard Test Example
```

- 2). 用户可以按下实验系统的 5x5 键盘，在超级终端上观察结果

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

5.2.7 实验参考程序

1. 键盘控制初始化

```

/*****
* name:      keyboard_test
* func:      test 8led
* para:      none
*****/

```

```

* ret:      none
* modify:
* comment:
*****/

void keyboard_test(void)
{
    int i, nTmp;
    UINT8T ucChar;

    iic_init();

    // initaze the 8led-function first of ZLG7290 (more to see ZLG7290.pdf)
    // precent the invalid sign generated while keyboard-function
    for(i=0; i<8; i++)
        iic_write(0x70, 0x10+i, 0xFF);

    iic_write(0x70, 0x10+3, f_szDigital[6]); // display 6 only

    // set EINT2 interrupt handler
    pISR_EINT2 = (int)keyboard_int;

    i = 8;
    uart_printf("\n");
    for(;;)
    {
        f_nKeyPress = 0;
        rINTMSK = rINTMSK & ~(BIT_GLOBAL|BIT_EINT2); // enable EINT2 int
        while(f_nKeyPress == 0);
        iic_read(0x70, 0x1, &ucChar);
        ucChar = key_set(ucChar); // key map for EduKitII

        nTmp = f_szDigital[ucChar];
        switch(ucChar)
        {
            case '*': nTmp = 0x90; break;
            case '+': nTmp = 0x92; break;
            case '-': nTmp = 0x02; break;
            default:
                break;
        }
        delay(5);
        led8_disp(i,i,nTmp);
        iic_init();
        if(i-- == 5) i = 8;

        if(ucChar < 10) ucChar += 0x30;
    }
}

```

```

else if(ucChar < 16) ucChar += 0x37;

if(ucChar < 255)
    uart_printf(" press key %c\n\r",ucChar);

if(ucChar == 0xFF)
{
    uart_printf(" press key FUN (exit now)\n\r");
    return;
}
}

}

```

2. 中断服务程序

```

/*****
* name:      keyboard_int
* func:      keyboard interrupt handler
* para:      none
* ret:       none
* modify:
* comment:
*****/

void keyboard_int(void)
{
    UINT8T ucChar;

    rINTMSK = rINTMSK | BIT_EINT2;           // disable EINT2 int
    rI_ISPC = BIT_EINT2;
    f_nKeyPress = 1;
}

```

5.2.8 练习题

编写程序实现双键同时按下时键盘的检测及处理程序。

5.3 触摸屏控制实验

5.3.1 实验目的

- 通过实验掌握触摸屏（TSP）的设计与控制方法。
- 掌握 S3C44B0X 处理器的 A/D 转换功能。

5.3.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。

- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

5.3.3 实验内容

- 点击触摸屏左下角和右上角，通过对角线定位方法确定触摸屏坐标范围；
- 点击触摸屏任意位置，将触摸屏坐标转换为液晶对应坐标后显示坐标位置。

5.3.4 实验原理

1. 触摸屏（TSP）

触摸屏（TSP: Touch Screen Panel）按其技术原理可分为五类：矢量压力传感式、电阻式、电容式、红外线式和表面声波式，其中电阻式触摸屏在嵌入式系统中用的较多。

表面声波触摸屏

表面声波触摸屏的边角有 X、Y 轴声波发射器和接收器，表面有 X、Y 轴横竖交叉的超声波传输。当触摸屏幕时，从触摸点开始的部分被吸收，控制器根据到达 X、Y 轴的声波变化情况和声波传输速度计算出声波变化的起点，即触摸点。

电容感应触摸屏

人相当于地，给屏幕表面通上一个很低的电压，当用户触摸屏幕时，手指头吸收走一个很小的电流，这个电流分别从触摸屏四个角或四条边上的电极中流出，并且理论上流经这四个电极的电流与手指到四角的距离成比例，控制器通过对这四个电流比例的计算，得出触摸点的位置。

红外线触摸屏

红外线触摸屏，是在显示器屏幕的前面安装一个外框，外框里有电路板，在 X、Y 方向排布红外发射管和红外接收管，一一对应形成横竖交叉的红外线矩阵。当有触摸时，手指或其它物体就会挡住经过该处的横竖红外线，由控制器判断出触摸点在屏幕的位置。

电阻触摸屏

电阻触摸屏是一个多层的复合膜，由一层玻璃或有机玻璃作为基层，表面涂有一层透明的导电层，上面再盖有一层塑料层，它的内表面也涂有一层透明的导电层，在两层导电层之间有许多细小的透明隔离点把它们隔开绝缘。工业中常用 ITO（Indium Tin Oxide 氧化锡）导电层。当手指触摸屏幕时，平常绝缘的两层导电层在触摸点位置就有了一个接触，控制器检测到这个接通后，其中一面导电层接通 Y 轴方向的 5V 均匀电压场，另一导电层将接触点的电压引至控制电路进行 A/D 转换，得到电压值后与 5V 相比即可得触摸点的 Y 轴坐标，同理得出 X 轴的坐标。这是所有电阻技术触摸屏共同的基本原理。电阻式触摸屏根据信号线数又分为四线、五线、六线...电阻触摸屏等类型。信号线数越多，技术越复杂，坐标定位也越精确。

四线电阻触摸屏，采用国际上评价很高的电阻专利技术：包括压模成型的玻璃屏和一层透明的防刮塑料，或经过硬化、清晰或抗眩光处理的尼龙，内层是透明的导体层，表层与底层之间夹着拥有专利技术的分离点（Separator Dots）。这类触摸屏适合于需要相对固定人员触摸的高精度触摸屏的应用场合，精度超过 4096x4096，有良好的清晰度和极微小的视差。主要优点还表现在：不漂移，精度高，响应快，可以用手指或其它物体触摸，防尘防油污等，主要用于专业工程师或工业现场。

Embest EduKit-III 采用四线式电阻式触摸屏，点数为 320x240，实验系统由触摸屏、触摸屏控制电路和数据采集处理三部分组成。

被按下的触摸屏状态如图 5-16 所示

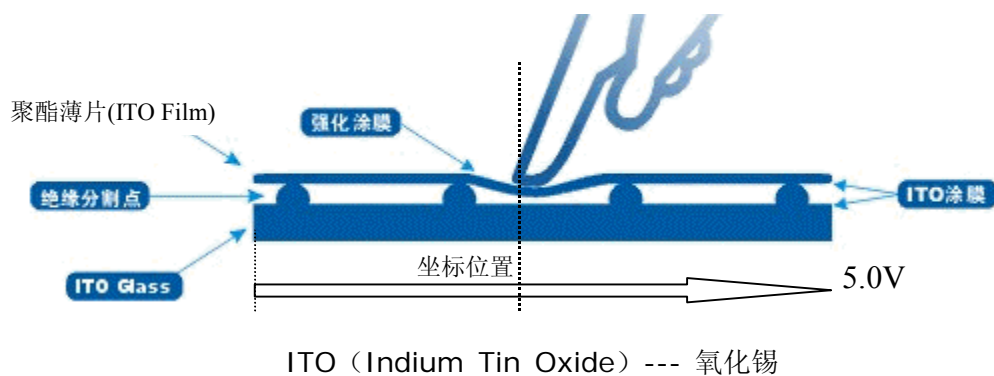


图 5-16 触摸屏按下



图 5-17 LRH9J515XA STN/BW 触摸屏

● 等效电路结构

电阻触摸屏采用一块带统一电阻外表面的玻璃板。聚酯表层紧贴在玻璃面上，通过小的透明的绝缘颗粒与玻璃面分开。聚酯层外表面坚硬耐用，内表面有一个传导层。当屏幕被触摸时，传导层与玻璃面表层进行电子接触。产生的电压就是所触摸位置的模拟表示。

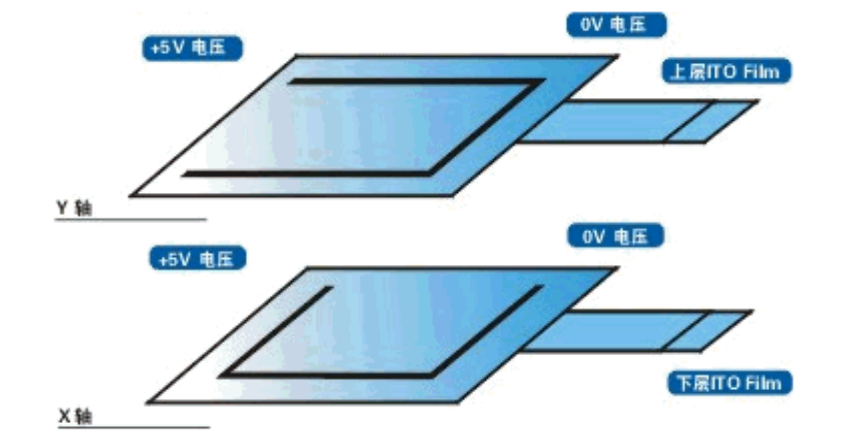


图 5-18 等效电路示意图

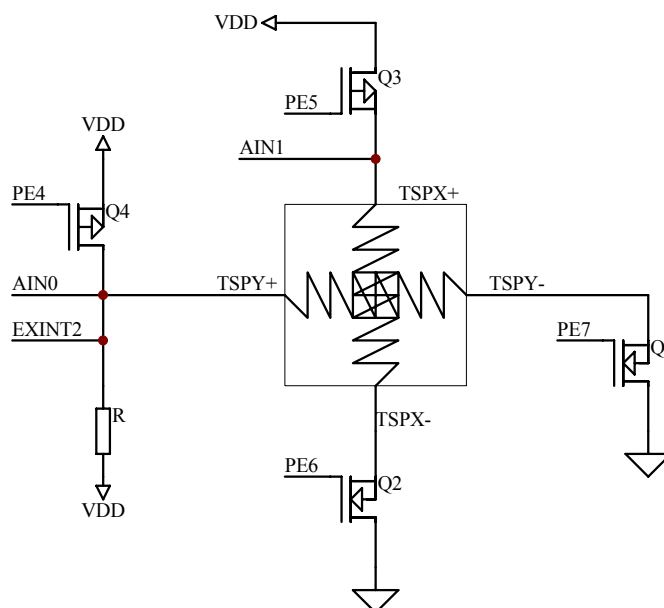


图 5-19 触摸屏的等效电路

● 触摸屏原点

电阻式触摸屏是通过电压的变化范围来判定按下触摸屏的位置，所以其原点就是触摸屏 X 电阻面和 Y 电阻面接通产生最小电压处。随着电阻的增大，A/D 转换所产生数值不断增加，形成坐标范围。

触摸原点的确定有很多种方法，比如常用的对角定位法、四点定位法、实验室法等。

对角定位法

系统先对触摸屏的对角坐标进行采样，根据数值确定坐标范围，可采样一条对角线或两条对角线的顶点坐标。这种方法简单易用，但是需要多次采样操作并进行比较，以取得定位的准确性。本实验板采用这种定位方法。

四点定位法

同对角定位法一样，需要进行数据采样，只是需要采样四个顶点坐标以确定有效坐标范围，程序根据四个采样值的大小关系进行坐标定位。这种方法的定位比对角定位法可靠，所以被现在许多带触摸屏的设备终端使用。

实验室法

触摸屏的坐标原点、坐标范围由生产厂家在出厂前根据硬件定义好。定位方法是按照触摸屏和硬件电路的系统参数，对批量硬件进行最优处理定义取得。这种方法适用于触摸屏构成的电路系统有较好的电气特性，且不同产品有较大相似性的场合。

● 触摸屏的坐标

触摸屏坐标值可以采用多种不同的计算方式。常用的有多次采样取平均值法、二次平方处理法等。Embest EduKit-III 教学系统的触摸屏坐标值计算采用取平均值法，首先从触摸屏的四个顶角得到两个最大值和两个最小值，分别标识为 Xmax、Ymax 和 Xmin、Ymin。

参照图 5-25 组成的坐标识别控制电路，X、Y 方向的确定见表 5-8。

表 5-8 确定 X、Y 方向

| | A/D 通道 | N-MOS | P-MOS |
|---|--------|----------------------|----------------------|
| X | AN0 | Q1(-) = 1; Q2(+) = 0 | Q3(-) = 0; Q4(+) = 1 |
| Y | AN1 | Q1(+) = 0; Q2(-) = 1 | Q3(+) = 1; Q4(-) = 0 |

当触摸屏被按下时，首先导通 MOS 管组 Q2 和 Q4，X+ 与 X- 回路加上 +5V 电源，同时将 MOS 管组 Q1 和 Q3 关闭，断开 Y+ 和 Y-；再启动处理器的 A/D 转换通道 0，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，计算出 X 轴的坐标。

接着先导通 MOS 管组 Q1 和 Q3，Y+ 与 Y- 回路加上 +5V 电源，同时将 MOS 管组 Q2 和 Q4 关闭，断开 X+ 和 X-；再启动处理器的 A/D 转换通道 1，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，计算出 Y 轴的坐标。

确定 X、Y 方向后坐标值的计算可通过以下方式求得（请参照程序设计）：

$$X = (X_{\max} - X_a) \times 240 / (X_{\max} - X_{\min}) \quad X_a = [X1 + X2 + \dots + X_n] / n$$

$$Y = (Y_{\max} - Y_a) \times 320 / (Y_{\max} - Y_{\min}) \quad Y_a = [Y1 + Y2 + \dots + Y_n] / n$$

通过计算，Embest ARM 教学系统的触摸屏的坐标情况（n=5）如图 5-20 所示。

| | | | |
|-----------|---------|-----------|-----------|
| (320,240) | | | |
| 0 | 1 | 2 | (240,180) |
| 4 | 5 | (160,120) | 7 |
| 8 | (80,60) | A | B |
| (0,0) | D | E | F |

* 触摸屏理论可以识别 1 个单位的变化

* 建议使用 10 个单位作为识别单位

图 5-20 触摸屏的坐标范围（坐标定位后）

2. A/D 转换器（ADC）

● A / D 转换器的类型

A / D 转换器种类繁多，分类方法也很多。其中常见的包括以下分类：

按照工作原理可分为：计数式 A / D 转换器、逐次逼近型、双积分型和并行 A / D 转换几类。

按转换方法可分为：直接 A / D 转换器和间接 A / D 转换器。所谓直接转换是指将模拟量转换成数字量；而间接转换则是指将模拟量转换成中间量，再将中间量转换成数字量。

按分辨率可分为：二进制的 4 位、6 位、8 位、10 位、12 位、14 位、16 位和 BCD 码的 3 位半、4 位半、5 位半等。

按转换速度可分为：低速（转换时间 $\geq 1\text{s}$ ）、中速（转换时间 $\leq 1\text{ms}$ ）、高速（转换时间 $\geq 1\mu\text{s}$ ）和超高速（转换时间 $\leq 1\text{ns}$ ）。

按输出方式可分为：并行、串行、串并行等。

● A / D 转换器的工作原理

A / D 转换的方法很多，下面介绍常用的 A / D 转换原理。

(1) 计数式

这种 A / D 转换原理最简单直观，它由 D / A 转换器、计数器和比较器组成，如图 5-21 所示。计数器由零开始计数，将其计数值送往 D / A 转换器进行转换，将生成的模拟信号与输入模拟信号在比较器内进行比较，若前者小于后者，则计数值加 1，重复 D / A 转换及比较过程。因为计数值是递增的，所以 D / A 输出的模拟信号是一个逐步增加的量，当这个信号值与输出模拟量比较相等时（在允许的误差范围内），比较器产生停止计数信号，计数器立即停止计数。此时 D / A 转换器输出的模拟量即为模拟输入值，计数器的值就是转换成的相应的数字量值。这种 A / D 转换器结构简单、原理清楚，但是转换速度与精度之间存在严重矛盾即若要提高转换速度，则转换器输出与输入的误差就越大，反之亦然。所以在实际中很少使用。

(2) 逐次逼近式

逐次逼近 A / D 转换器是由一个比较器、D / A 转换器、寄存器及控制逻辑电路组成，如图 5-22 所示。和计数式相同，逐次逼近式也要进行比较，以得到转换数字值。但在逐次逼近式中，是用一个寄存器控制 D / A 转换器。逐次逼近式是从高位到低位依次开始逐位试探比较。S3C44B0X 处理器集成了这种 A/D 转换器。

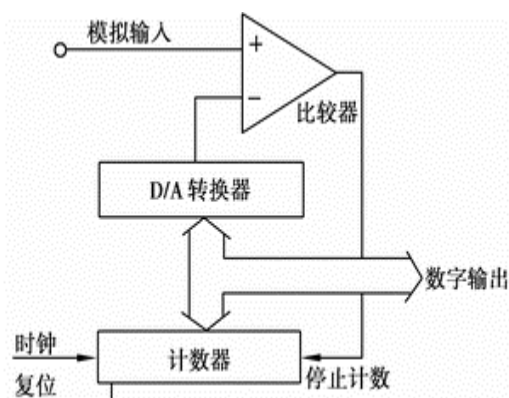


图 5-21 计数式 A / D 转换图

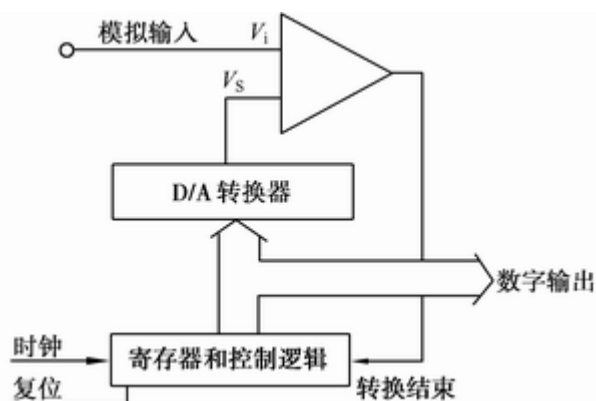


图 5-22 逐次逼近 A / D 转换图

逐次逼近式转换过程如下：初始时寄存器各位清为 0，转换时，先将最高位置 1，送入 D / A 转换器，经 D / A 转换后生成的模拟量送入比较器中与输入模拟量进行比较，若 $V_s < V_i$ ，该位的 1 被保留，否则被清除。然后次高位置为 1，将寄存器中新的数字量送入 D / A 转换器，输出的 V_s 再与 V_i 比较，若 $V_s < V_i$ ，保留该位的 1，否则清除。重复上述过程，直至最低位。最后寄存器中的内容即为输入模拟值转换成的数字量。

对于 n 位逐次逼近式 A / D 转换器，要比较 n 次才能完成一次转换。因此，逐次逼近式 A / D 转换器的转换时间取决于位数和时钟周期。转换精度取决于 D / A 转换器和比较器的精度，一般可达 0.01%，转换结果也可串行输出。逐次逼近式 A / D 转换器可应用于许多场合，是应用最为广泛的一种 A / D 转换器。

● A / D 转换器主要性能指标

(1) 分辨率

分辨率是指 A / D 转换器能分辨的最小模拟输入量。通常用能转换成的数字量的位数来表示，如 8 位、10 位、12 位、16 位等。位数越高，分辨率越高。如分辨率为 10 位，表示 A / D 转换器能分辨满量程的 $1 / 1024$ 的模拟增量，此增量亦可称为 1LSB 或最低有效位的电压当量。

(2) 转换时间

转换时间是 A / D 转换完成一次转换所需的时间。即从启动信号开始到转换结束并得到稳定数字输出量为止的时间。一般来说，转换时间越短则转换速度就越快。不同的 A / D 转换器转换时间差别较大，通常为微秒数量级。

(3) 量程

量程是指所能转换的输入电压范围。

(4) 绝对精度

A / D 转换器的绝对精度是指在输出端产生给定的数字代码的情况下，实际需要的模拟输入值与理论上要求的模拟输入值之差。

(5) 相对精度

相对精度是指 A / D 转换器的满刻度值校准以后，任意数字输出所对应的实际模拟输入值（中间值）与理论值（中间值）之差。线性 A / D 转换器的相对精度就是它的线性度。精度代表电气或工艺精度，其绝对值应小于分辨率，因此常用 1 LSB 的分数形式来表示。

3. S3C44BOX 处理器的 A/D 转换

处理器内部集成了采用近似比较算法（计数式）的 8 路 10 位 ADC，集成零比较器和内部产生的比较时钟信号；支持软件使能休眠模式，以减少电源损耗。其主要特性：

- 精度（Resolution）：10-bit
- 微分线性误差（Differential Linearity Error）： ± 1 LSB
- 积分线性误差（Integral Linearity Error）： ± 2 LSB (Max. ± 3 LSB)
- 最大转换速率（Maximum Conversion Rate）：100 KSPS
- 输入电压（Input voltage range）：0-2.5V
- 输入带宽（Input bandwidth）：0-100 Hz（无采样保持电路 S/H(sample&hold)）
- 低功耗（Low Power Consumption）

● A/D 功能框图

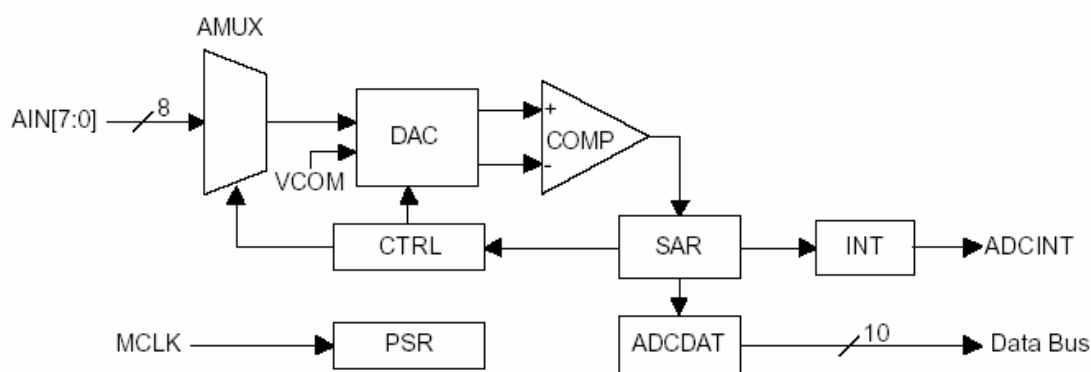


图 5-23 S3C44BOX 处理器的 ADC 框图

VCOM 是外部比较电压输入，实验系统中只需要接滤波电容到地，即比较范围为 5-0V。处理器的 AFREB、AREFT 引脚亦需要接滤波电容到地处理。

● S3C44BOX 处理器 A/D 转换器的使用

(1) 寄存器组

处理器集成的 ADC 只使用到三个寄存器，即 ADC 控制寄存器（ADCCON）、ADC 数据寄存器（ADCDAT）、ADC 预装比例因子寄存器（ADCPSR）。

ADC 控制寄存器（ADCCON）

| | [6] | [5] | [4:2] | [1] | [0] |
|--------|------|-------|-----------|-------------|--------------|
| ADCCON | FLAG | SLEEP | IN_SELECT | READ_ START | ENABLE_START |

FLAG ---- 0: A/D 转换正在进行; 1: A/D 转换结束

SLEEP ---- 0: 正常状态; 1: Sleep 模式

IN_SELECT ---- 选择转换通道[4:2]

000 = AIN0 001 = AIN1 010 = AIN2 011 = AIN3

100 = AIN4 101 = AIN5 110 = AIN6 111 = AIN7

READ_ START ---- 0: 禁止 Start-by-read 1: 允许 Start-by-read

ENABLE_START ---- 0: A/D 转换器不工作 1: A/D 转换器开始工作

ADC 预装比例因子寄存器（ADCPSR）

| | [7:0] |
|--------|-----------|
| ADCPSR | PRESCALER |

PRESCALER ---- 比例因子。该数据决定转换时间的长短，数据越大转换时间就越长。

ADC 数据寄存器（ADCDAT）

| | [9:0] |
|--------|--------|
| ADCDAT | ADCDAT |

ADCDAT ---- A/D 转换数据值

(2) A/D 转换的转换时间计算

例如系统时钟为 66MHz，PRESCALER=20；所有 10 位转换时间为：

$$66 \text{ MHz} / 2(20+1) / 16 = 98.2 \text{ KHz} = 10.2 \text{ us} \quad 16 \text{ 是指 } 10 \text{ 位转换所需最少周期数}$$

(3) 使用 A/D 转换器的注意事项：

- 1) ADC 的模拟信号输入通道没有采样保持电路，使用时可以设置较大的 ADCPSR 值，以减少输入通道因信号输出电阻过大而产生的信号电压。
- 2) ADC 的转换频率在 0~100Hz。
- 3) 通道切换时，应保证至少 15us 的间隔。
- 4) ADC 从 SLEEP 模式退出时，通道信号应保持 10ms 以使 ADC 参考电压稳定。
- 5) Start-by-read 可使用 DMA 传送转换数据。

5.3.5 实验设计

1. 电路设计

Embest EduKit-III 实验板所选用的触摸屏（LRH9J515XA STN/BW）的主要参数：

表 5-9 LRH9J515XA STN/BW 触摸屏主要技术参数

| | | | | | |
|----|------------|------|----------------|------|-------------|
| 型号 | LRH9J515XA | 外形尺寸 | 93.8×75.1×5mm | 重量 | 45g |
| 像素 | 320 × 240 | 画面尺寸 | 9.6cm（3.8inch） | 色彩 | 16 级灰度 |
| 电压 | 21.5V（25℃） | 点宽 | 0.24 mm/dot | 电阻 Ω | X:590 Y:440 |

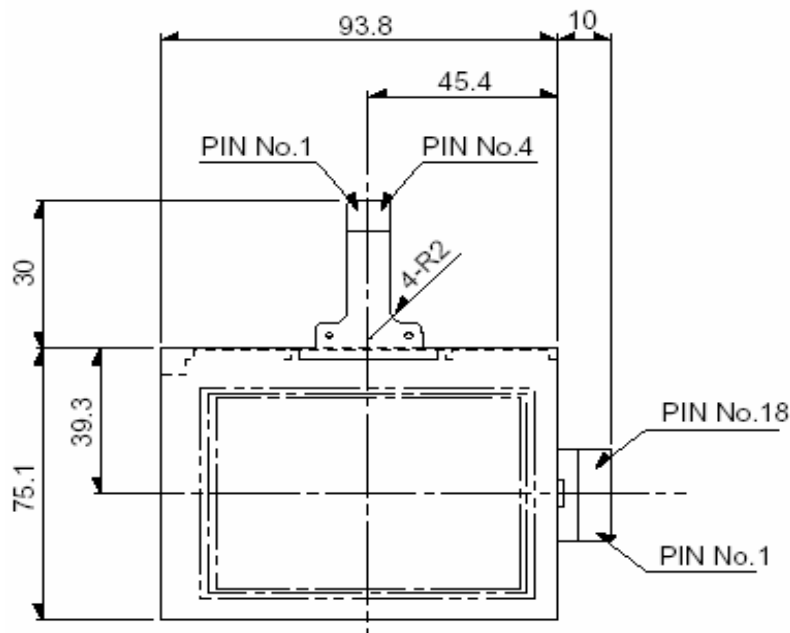


图 5-24 触摸屏的尺寸示意图

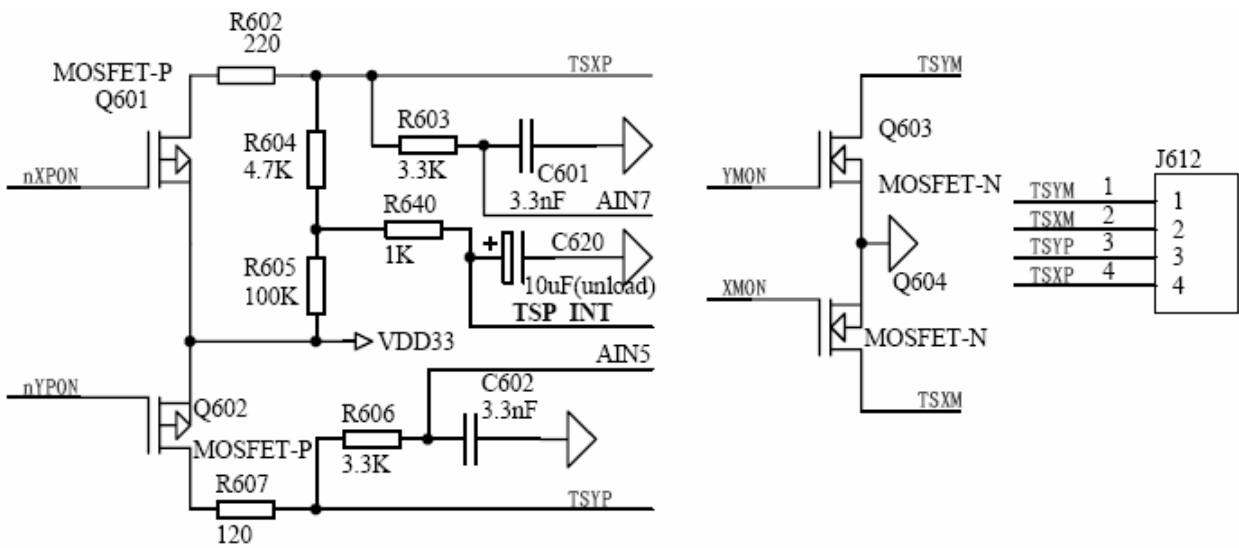


图 5-25 触摸屏坐标转换控制电路

当手指触摸屏幕时，平常绝缘的两层导电层在触摸点位置就有了一个接触，控制器检测到这个接通后，产生中断通知 CPU 进行 A/D 转换；中断处理程序通过导通不同 MOS 管组（见表 5-7），使接触部分与控制器电路构成电阻电路，并产生一个电压降作为坐标值输出。

2. 软件程序设计

触摸屏的控制程序软件包括串口数据传送、触摸屏定位、中断处理程序等。

根据实验原理的触摸屏定位方法，实验系统采用对角线定位方法。

中断处理程序中包括 A/D 转换、坐标存储。

5.3.6 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接到目标板上，使用 Embest EduKit-III 实验板自带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上，打开实验例程目录 5.3_touchscreen_test 子目录下的 touchscreen_test.Uv2 例程，编译链接工程，直到链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用！！！这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Run 或 F5 键全速运行程序，或者单步调试程序。

4. 观察实验结果

在 PC 上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...

Touch screen Test Example

Please touch LCD's left up corner: left=0554, up=0370

Please touch LCD's right bottom corner: right=0325, bottom=0129
```

X=0081, Y=0132

X=0108, Y=0120

确定坐标范围后，用户可以在触摸屏的有效范围内按下触摸屏，超级终端将会输出触摸屏的坐标值：

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

5.3.7 实验参考程序

1. 初始化程序

由于实验中使用到 S3C44B0X 处理器的 LCD 控制器、串行口控制器，所以初始化部分包括对 LCD 控制器和串行口的初始化。处理器 A/D 转换器可以在工作时初始化。

2. 触摸屏初始化

```

/*****
* name:      touchscreen_init
* func:      initialize TouchScreen
* para:      none
* ret:       none
* modify:
* comment:
*****/

void touchscreen_init(void)
{
#ifdef S3CEV40
// TSPX(GPC1_Q4(-)) TSPY(GPC3_Q3(-)) TSMY(GPC0_Q2(-)) TSMX(GPC2_Q1(+))
//      1          1          0          1
rPCONC = (rPCONC & 0xfffff00) | 0x55;
rPUPC  = (rPUPE  & 0xff0);           // Pull up
rPDATC = (rPDATC & 0xff0 ) | 0xe;    // should be enabled
#else
// S3CEV40
// TSPX(GPE4_Q4(+)) TSPY(GPE5_Q3(-)) TSMY(GPE6_Q2(+)) TSMX(GPE7_Q1(-))
//      0          1          1          0
rPCONE = (rPCONE & 0x300ff) | 0x5500;
rPUPE  = (rPUPE & 0xF);
rPDATE = 0xb8;
#endif
delay(100);

// set interrupt
#ifdef S3CEV40
rPUPG = (rPUPG & 0xFE) | 0x1;
pISR_EINT0=(int)touchscreen_int;           // set interrupt handler
rEXTINT = (rEXTINT & 0x7FFFFFFF0) | 0x2;    // falling edge trigger

```

```

    rI_ISPC |= BIT_EINT0;                                // clear pending_bit
    rINTMSK =~(BIT_GLOBAL|BIT_EINT0);
#else
    pISR_EINT2=(int)touchscreen_int;                    // set interrupt handler
    rEXTINT = (rEXTINT & 0x7FFF0FF) | 0x200;            // falling edge trigger
    rI_ISPC |= BIT_EINT2;                                // clear pending_bit
    rINTMSK =~(BIT_GLOBAL|BIT_EINT2);
#endif
    rCLKCON = (rCLKCON & 0x6FFF) | 0x1000;              // enable clock
    rADCPSR = 24;                                        // A/D prescaler
}

```

3 中断服务程序

```

/*****
* name:      touchscreen_int
* func:      TouchScreen interrupt handler function
* para:      none
* ret:       none
* modify:
* comment:
*****/

void touchscreen_int(void)
{
    UINT32T  unPointX[5], unPointY[6];
    UINT32T  unPosX, unPosY;
    rINTMSK |=BIT_EINT0;
    int      i;
    delay(500);
#ifdef S3CEV40
    // <X-Position Read>
    // TSPX(GPC1_Q4(+)) TSPY(GPC3_Q3(-)) TSMY(GPC0_Q2(+)) TSMX(GPC2_Q1(-))
    //      0          1          1          0
    rPDATC = (rPDATC & 0xfff0 ) | 0x9;
    rADCCON= 0x0014;                                // AIN5
#else
    // TSPX(GPE4_Q4(+)) TSPY(GPE5_Q3(-)) TSMY(GPE6_Q2(+)) TSMX(GPE7_Q1(-))
    //      0          1          1          0
    rPDATE =0x68;
    rADCCON=0x1<<2;                                // AIN1
#endif

    delay(100);                                     // delay to set up the next channel
    for(i=0; i<5; i++)
    {
        rADCCON |= 0x1;                                // Start X-position A/D
    }
}

```

```

conversion

    while(rADCCON & 0x1 == 1);                // Check if AD conversion starts
    while((rADCCON & 0x40) == 0);            // Check end of AD conversion
    unPointX[i] = (0x3ff&rADCSTAT);
}

// read X-position average value
unPosX = (unPointX[0]+unPointX[1]+unPointX[2]+unPointX[3]+unPointX[4])/5;
f_unPosX = unPosX;

#ifdef S3CEV40
    // <Y-Position Read>
    // TSPX(GPC1_Q4(-)) TSPY(GPC3_Q3(+)) TSMY(GPC0_Q2(-)) TSMX(GPC2_Q1(+))
    //      1          0          0          1
    rPDATC = (rPDATC & 0xfff0) | 0x6;
    rADCCON= 0x001C;                          // AIN70
#else
    // TSPX(GPE4_Q4(-)) TSPY(GPE5_Q3(+)) TSMY(GPE6_Q2(-)) TSMX(GPE7_Q1(+))
    //      1          0          0          1
    rPDATE = 0x98;
    rADCCON=0x0<<2;                          // AIN0
#endif
    delay(100);                               // delay to set up the next channel
    for(i=0; i<5; i++)
    {
        rADCCON |= 0x1;                      // Start Y-position conversion
        while(rADCCON & 0x1 == 1);            // Check if AD conversion starts
        while((rADCCON & 0x40) == 0);        // Check end of AD conversion
        unPointY[i] = (0x3ff&rADCSTAT);
    }
    // read Y-position average value
    unPosY = (unPointY[0]+unPointY[1]+unPointY[2]+unPointY[3]+unPointY[4])/5;
    f_unPosY = unPosY;

#ifdef S3CEV40
    rPDATC = (rPDATC & 0xfff0) | 0xe;        // should be enabled
#else
    rPDATE = 0xb8;                          // should be enabled
#endif
    delay(1000);

    f_unTouched = 1;
    delay(1000);
    rI_ISPC |= BIT_EINT0;                   // clear pending_bit
    rINTMSK =~(BIT_GLOBAL|BIT_EINT0);
#ifdef S3CEV40

```



```

        rI_ISPC |= BIT_EINT0;                                // clear pending_bit
    #else
        rI_ISPC |= BIT_EINT2;                                // clear pending_bit
    #endif
}

// <Y-Position Read>
// TSPX(GPC1_Q4(-)) TSPY(GPC3_Q3(+)) TSMY(GPC0_Q2(-)) TSMX(GPC2_Q1(+))
//      1          0          0          1
rPDATC = (rPDATC & 0xfff0 ) | 0x6;
rADCCON= 0x001C;                                             // AIN70
delay(100);                                                  // delay to set up the next channel
for(i=0; i<5; i++)
{
    rADCCON |= 0x1;                                           // Start Y-position conversion
    while(rADCCON & 0x1 == 1);                                // Check if AD conversion starts
    while((rADCCON & 0x40) == 0);                             // Check end of AD conversion
    unPointY[i] = (0x3ff&rADCDAT);
}
// read Y-position average value
unPosY = (unPointY[0]+unPointY[1]+unPointY[2]+unPointY[3]+unPointY[4])/5;
f_unPosY = unPosY;

uart_printf("X=%04d Y=%04d\n", unPosX, unPosY);

rPDATC = (rPDATC & 0xfff0 ) | 0xe;                           // should be enabled
delay(1000);

f_unTouched = 1;

    rI_ISPC |= BIT_EINT0;                                    // clear pending_bit
}

```

5.3.8 练习题

结合液晶显示控制实验，编写程序获取用户输入的 4 个坐标位置，并在液晶上画出由用户输入坐标组成的矩形。

第六章 通信与音频接口实验

6.1 IIC 串行通信实验

6.1.1 实验目的

- 通过实验掌握 IIC 串行数据通信协议的使用。
- 掌握 EEPROM 器件的读写访问方法。
- 通过实验掌握 S3C44B0X 处理器的 IIC 控制器的使用。

6.1.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

6.1.3 实验内容

编写程序对实验板上 EEPROM 器件 AT24C04 进行读写访问。

- 写入 EEPROM 某一地址，再从该地址读出；
- 把读出内容和写入内容进行比较，检测 S3C44B0X 处理器通过 IIC 接口，是否可以正常读写 EEPROM 器件 AT24C04。

6.1.4 实验原理

1. IIC 接口以及 EEPROM

IIC 总线为同步串行数据传输总线，其标准总线传输速率为 100kb/s，增强总线可达 400kb/s。总线驱动能力为 400pF。

IIC 总线可构成多主和主从系统。在多主系统结构中，系统通过硬件或软件仲裁获得总线控制使用权。应用系统中 IIC 总线多采用主从结构，即总线上只有一个主控节点，总线上的其它设备都作为从设备。IIC 总线上的设备寻址由器件地址接线决定，并且通过访问地址最低位来控制读写方向。

目前，通用存储器芯片多为 EEPROM，其常用的协议主要有两线串行连接协议（IIC）和三线串行连接协议。带 IIC 总线接口的 EEPROM 有许多型号，其中 AT24CXX 系列使用十分普遍。产品包括 AT2401/02/04/08/16 等，其容量（字节数 x 页）分别为 128x8/256x8/512x8/1024x8/2048x8，适用于 2V~5V 的低电压的操作。具有低功耗和高可靠性等优点。

AT24 系列存储器芯片采用 CMOS 工艺制造，内置有高压泵，可在单电压供电条件下工作。其标准封装为 8 脚 DIP 封装形式，如图 6-1。

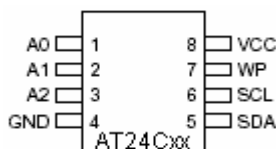


图 6-1

各引脚的功能说明如下：

SCL 串行时钟。遵循 ISO/IEC7816 同步协议；漏极开路，需接上拉电阻

在该引脚的上升沿，系统将数据输入到每个 EEPROM 器件，在下降沿输出。

SDA 串行数据线；漏极开路，需接上拉电阻

双向串行数据线，漏极开路，可与其他开路器件“线或”。

A0、A1、A2 器件/页面寻址地址输入端

在 AT24C01/02 中，引脚被硬连接；其他 AT24Cxx 均可接寻址地址线。

WP 读写保护

接低电平时可对整片空间进行读写；高电平时不能读写受保护区。

Vcc/GND 一般输入+5V 的工作电压

2. IIC 总线的读写控制逻辑

开始条件 (START_C)

在开始条件下，当 SCL 为高电平时，SDA 由高转为低。

停止条件 (STOP_C)

在停止条件下，当 SCL 为高电平时，SDA 由低转为高。

确认信号 (ACK)

在接收方应答下，每收到一个字节后将 SDA 电平拉低。

数据传送 (Read/Write)

IIC 总线启动或应答后 SCL 高电平期间数据串行传送；低电平期间为数据准备，并允许 SDA 线上数据电平变换。总线以字节 (8bit) 为单位传送数据，且高有效位 (MSB) 在前。IIC 数据传送时序如图 6-2 所示：

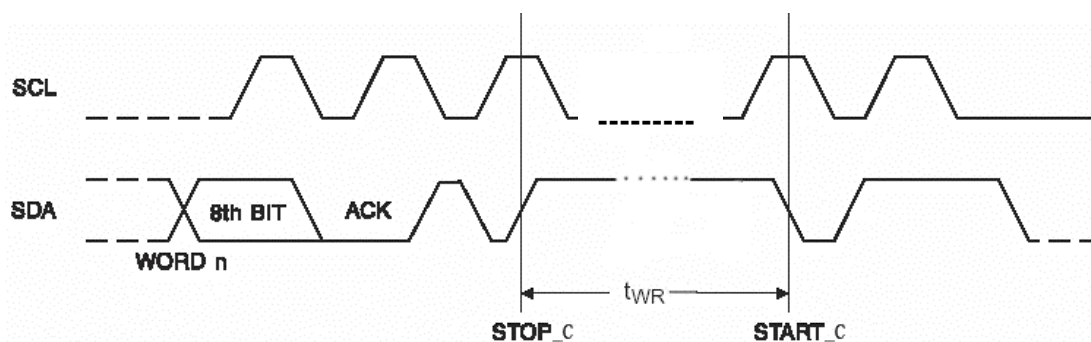


图 6-2 IIC 总线信号的时序

3. EEPROM 读写操作

1) AT24C04 结构与应用简述

AT24C04 由输入缓冲器和 EEPROM 阵列组成。由于 EEPROM 的半导体工艺特性写入时间为 5-10ms，如果从外部直接写入 EEPROM，每写一个字节都要等候 5-10ms，成批数据写入时则要等候更长的时间。具有 SRAM 输入缓冲器的 EEPROM 器件，其写入操作变成对 SRAM 缓冲器的装载，装载完后启动一个自动写入逻辑将缓冲器中的全部数据一次写入 EEPROM 阵列中。对缓冲器的输入称为页写，缓冲器的容量称为页写字节数。AT24C04 的页写字节数为 8，占用最低 3 位地址。写入不超过页写字节数时，对 EEPROM 器件的写入操作与对 SRAM 的写入操作相同；若超过页写字节数时，应等候 5-10ms 后再启动一次写操作。

由于 EEPROM 器件缓冲区容量较小（只占据最低 3 位），且不具备溢出进位检测功能，所以，从非零地址写入 8 个字节数或从零地址写入超过 8 个字节数会形成地址翻卷，导致写入出错。

2) 设备地址 (DADDR)

AT24C04XX 的器件地址是 1010。

3) AT24CXX 的数据操作格式

在 IIC 总线中对 AT24C04 内部存储单元读写，除了要给出器件的设备地址（DADDR）外还须指定读写的页面地址（PADDR），两者组成操作地址（OPADDR）如下：

1010 A2 A1- R/W （-为无效）

Embest ARM 教学系统中引脚 A2A1A0 为 000，因此系统可寻址 AT24C04 全部页面共 4KB 字节。按照 AT24C04 器件手册读写地址（ADDR=1010 A2 A1- R/W）中的数据操作格式如下：

写入操作格式

写任意地址 ADDR_W

START_C OPADDR_W ACK ADDR_W ACK data ACK STOP_C

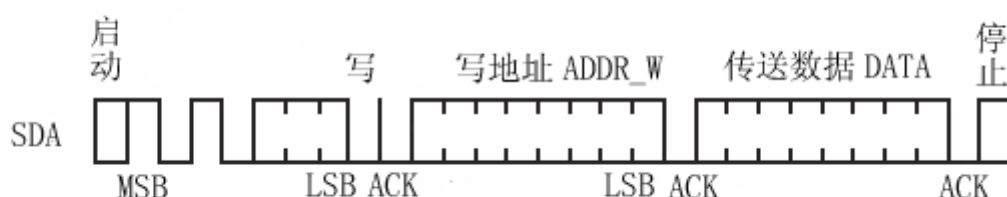


图 6-3 任意写一个字节

从地址 ADDR_W 起连续写入 n 个字节（同一页面）

START_C OPADDR_W ACK ADDR_W ACK data1 ACK data2 ACK ... dataN ACK STOP_C

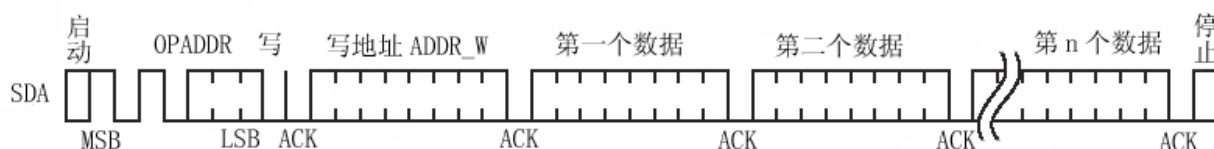


图 6-4 写 n 个字节

读出操作格式

读任意地址 ADDR_R

START_C OPADDR_W ACK ADDR_R ACK OPADDR_R ACK data STOP_C

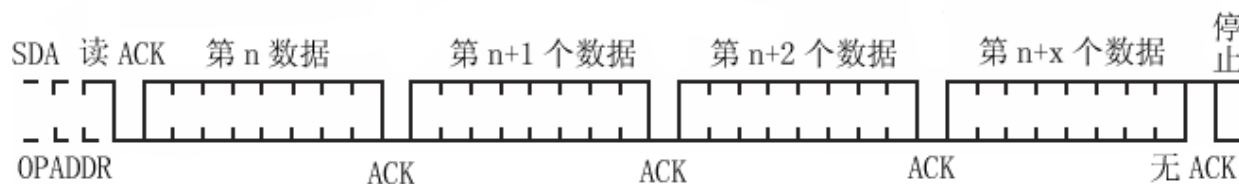


图 6-5 任意读一个字节

从地址 ADDR_R 起连续读出 n 个字节（同一页面）

START_C OPADDR_R ACK data1 ACK data2 ACK ... dataN ACK STOP_C



图 6-6 读 n 个字节

在读任意地址操作中除了发送读地址外还要发送页面地址（PADDR），因此在连续读出 n 个字节操作前要进行一个字节 PADDR 写入操作，然后重新启动读操作；注意读操作完后没有 ACK。

4. S3C44BOX 处理器 IIC 接口

1) S3C44BOX IIC 接口

S3C44BOX 处理器为用户进行应用设计提供了支持多主总线的 IIC 接口。处理器提供符合 IIC 协议的设备连接的双向数据线 IICSDA 和 IIC_SCL，在 IIC_SCL 高电平期间，IICSDA 的下降沿启动上升沿停止。S3C44BOX 处理器可以支持主发送、主接收、从发送、从接收四种工作模式。在主发送模式下，处理器通过 IIC 接口与外部串行器件进行数据传送，需要使用到如下寄存器：

IIC 总线控制寄存器 IICCON

| | [7] | [6] | [5] | [4] | [3:0] |
|--------|------------|---------------|-----------------|---------|----------------|
| IICCON | ACK Enable | Tx CLK select | Tx/Rx Interrupt | INT_PND | Tx Clock Value |

ACK Enable ---- 0: 禁止产生 ACK; 1: 允许产生 ACK 信号。

Tx CLK select ---- 0: IICCLK = fMCLK /16; 1: IICCLK = fMCLK /512。

Tx/Rx Interrupt ---- 0: 禁止 Tx/Rx 中断; 1: 允许 Tx/Rx 中断。

INT_PND ---- 写 0: 清除中断标志并重新启动 IIC 总线写操作;

---- 读 1: 中断标志置位。

Tx Clock Value ---- IIC 发送加载初始数据，决定了发送频率；

IIC 总线状态寄存器 IICSTAT

| | [7:6] | [5] | [4] | [3] | [2] | [1] | [0] |
|---------|--------|--------|-----|-----|-----|-----|-----|
| IICSTAT | Mode_S | Cond_S | SOE | ASF | ASS | AZS | LRB |

Mode_S ---- 00: 从接收 10: 主接收

01: 从发送 11: 主发送

Cond_S ---- 写 0: 产生 STOP_C 信号 读 0: IIC 总线空闲;

---- 写 1: 产生 START_C 信号 读 1: IIC 总线忙。

SOE ---- 0: 禁止 Tx/Rx 信号传输 1: 允许 Tx/Rx 信号传输。

ASF ---- 0: IIC 总线仲裁成功 1: 仲裁不成功, IIC 总线不能工作。

ASS ---- 作为从设备: 为 0: 检测到 START_C 或 STOP_C 信号;
为 1: 接收到地址。

AZS ---- 作为从设备时: 为 0: 收到 START_C 或 STOP_C 信号;
为 1: IIC 总线上的地址为 0。

LRB ---- 接收到的最低数据位: 为 0: 收到 ACK 信号;
为 1: 没有接收到 ACK 信号。

IIC 总线地址寄存器 IICADD

[7:0]

| | |
|--------|---------|
| IICADD | SlvADDR |
|--------|---------|

SlvADDR ---- 7:1 是从设备的设备地址和页面地址; 0 位是读写控制 (0: 写; 1: 读);
当 SOE=0 时可对 SlvADDR 进行读写。

IIC 总线发送接收移位寄存器 IICDS

[7:0]

| | |
|-------|----------|
| IICDS | ShitDATA |
|-------|----------|

ShitDATA ---- 7:0 存放 IIC 总线要移位传输或接收的数据

当 SOE=1 时可对 ShitDATA 进行读写

内部控制逻辑框图, 如图 6-7 所示:

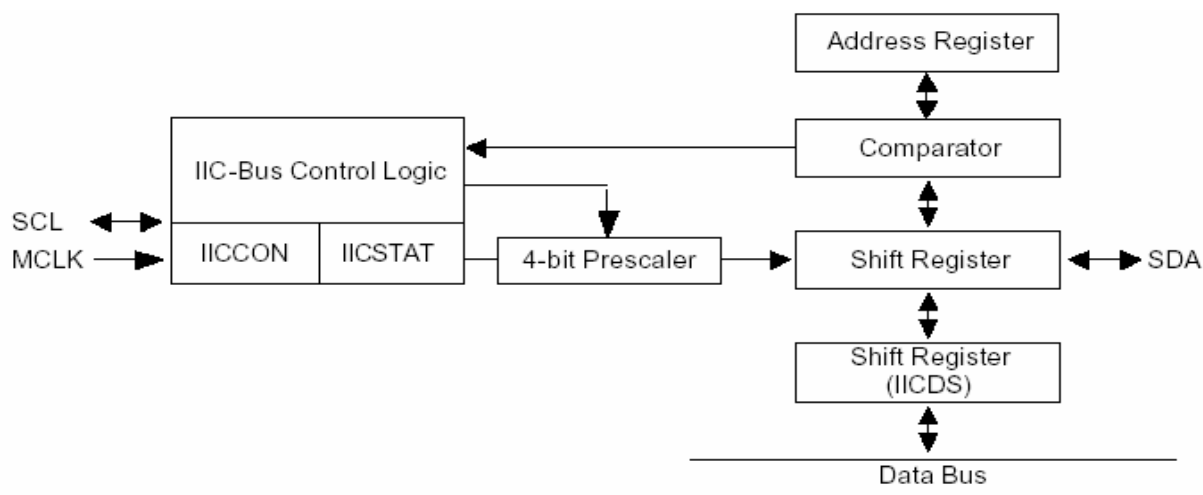


图 6-7 IIC 控制器逻辑框图

2) 使用 S3C44B0X IIC 总线读写方法

单字节写操作 (R/W=0)

Addr: 设备、页面及访问地址

| | | | | | |
|---------|--------------|-----|-------------|-----|--------|
| START_C | Addr(7bit) W | ACK | DATA(1Byte) | ACK | STOP_C |
|---------|--------------|-----|-------------|-----|--------|

同一页面的多字节写操作 (R/W=0)

OPADDR: 设备及页面地址 (高 7 位)

| | | | | | | |
|---------|----------------|-----|------|-------------|-----|--------|
| START_C | OPADDR(7bit) W | ACK | Addr | DATA(nByte) | ACK | STOP_C |
|---------|----------------|-----|------|-------------|-----|--------|

单字节读串行存储器件 (R/W=1)

Addr: 设备、页面及访问地址

| | | | | | |
|---------|--------------|-----|-------------|-----|--------|
| START_C | Addr(7bit) R | ACK | DATA(1Byte) | ACK | STOP_C |
|---------|--------------|-----|-------------|-----|--------|

同一页面的多字节读操作 (R/W=1)

Addr: 设备、页面及访问地址

| | | | | | | | | | |
|---------|-------|-----|------|-----|-------|-----|-------------|-----|--------|
| START_C | P & R | ACK | Addr | ACK | P & R | ACK | DATA(nByte) | ACK | STOP_C |
|---------|-------|-----|------|-----|-------|-----|-------------|-----|--------|

P & R = OPADDR_R=1010xxx (字节高 7 位) R ↑ 重新启动读操作

6.1.5 实验设计

1. 程序设计

程序设计流程图

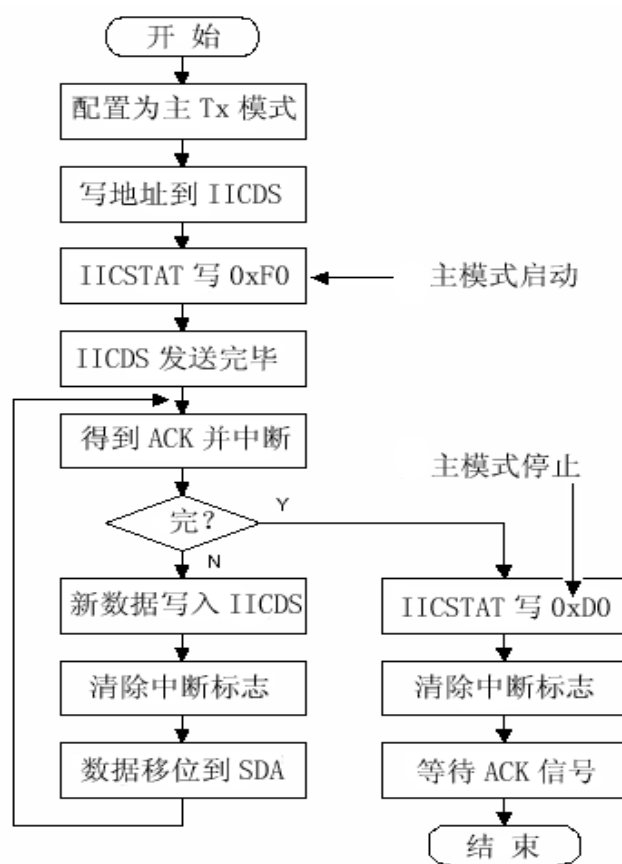


图 6-8 IIC 程序设计流程图 (S3C44B0X)

2. 电路设计

EduKit-III 实验平台中，使用 S3C44B0X 处理器内置的 IIC 控制器作为 IIC 通信主设备，AT24C04 EEPROM 为从设备。电路设计如图 6-9 所示：

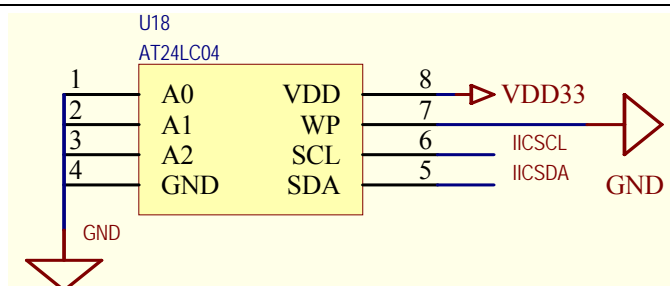


图 6-9 AT24C04 EEPROM 控制电路

6.1.6 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接到目标板上，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上，打开实验例程目录 6.1_iic_test 子目录下的 iic_test.Uv2 例程，编译链接工程，直到链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏“”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏“”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用！！！这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Run 或 F5 键全速运行程序，或者单步调试程序；
- 6) 结合实验内容和实验原理部分，掌握在 S3C44B0 处理器中使用 IIC 接口访问 EEPROM 存储空间的编程方法。

4. 观察实验结果

在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...
IIC operate Test Example
IIC Test using AT24C04...
```


Write char 0-f into AT24C04

Read 16 bytes from AT24C04

0 1 2 3 4 5 6 7 8 9 a b c d e f

5. 理解和掌握实验后，完成实验练习题。

6.1.7 实验参考程序

1. 初始化及测试主程序

```

/*****
* name:      iic_test
* func:      test iic
* para:      none
* ret:       none
* modify:
* comment:
*****/

void iic_test(void)
{
    UINT8T      szData[16];
    unsigned int  i;

    uart_printf("IIC Test using AT24C04...\n");
    uart_printf("Write char 0-f into AT24C04\n");

    f_nGetACK = 0;

    // Enable interrupt
    rINTMOD = 0x0;
    rINTCON = 0x1;
    rINTMSK &= ~(BIT_GLOBAL|BIT_IIC);
    pISR_IIC = (unsigned)iic_int;

    // Initialize iic
    rIICADD = 0x10;                // S3C44B0X slave address
    rIICCON = 0xaf;                // Enable ACK, interrupt, IICCLK=MCLK/16, Enable
ACK//64Mhz/16/(15+1) = 257Khz
    rIICSTAT = 0x10;                // Enable TX/RX

    // Write 0 - 16 to 24C04
    for(i=0; i<16; i++)
        write_24c040(0xa0, i, i);

    // Clear array
    for(i=0; i<16; i++)

```

```

        szData[i]=0;

        // Read 16 byte from 24C04
        for(i=0; i<16; i++)
            read_24c040(0xa0, i, &(szData[i]));

        // Printf read data
        uart_printf("Read 16 bytes from AT24C04\n");
        for(i=0; i<16; i++)
        {
            uart_printf("%2x ", szData[i]);
        }
        uart_printf("\n");
    }
}

```

2. 中断服务程序

```

/*****
* name:      iic_int
* func:      IIC interrupt handler
* para:      none
* ret:       none
* modify:
* comment:
*****/

void __irq iic_int(void)
{
    rl_ISPC=BIT_IIC;
    f_nGetACK = 1;
}

```

3. IIC 写 AT24C04 程序

```

/*****
* name:      write_24c040
* func:      write data to 24C040
* para:      unSlaveAddr --- input, chip slave address
*            unAddr      --- input, data address
*            ucData      --- input, data value
* ret:       none
* modify:
* comment:
*****/

void write_24c040(UINT32T unSlaveAddr,UINT32T unAddr,UINT8T ucData)
{

```

```

f_nGetACK = 0;

/* Send control byte */
rIICDS = unSlaveAddr;           // 0xa0
rIICSTAT = 0xf0;                // Master Tx,Start

while(f_nGetACK == 0);          // Wait ACK
f_nGetACK = 0;

/* Send address */
rIICDS = unAddr;
rIICCON = 0xaf;                 // Resumes IIC operation.

while(f_nGetACK == 0);          // Wait ACK
f_nGetACK = 0;

/* Send data */
rIICDS = ucData;
rIICCON = 0xaf;                 // Resumes IIC operation.

while(f_nGetACK == 0);          // Wait ACK
f_nGetACK = 0;

/* End send */
rIICSTAT = 0xd0;                // Stop Master Tx condition
rIICCON = 0xaf;                 // Resumes IIC operation.
delay(5);                       // Wait until stop condition is in effect.
}

```

4. IIC 读 AT24C04 程序

```

/*****
* name:      read_24c040
* func:      read data from 24C040
* para:      unSlaveAddr --- input, chip slave address
*            unAddr      --- input, data address
*            pData       --- output, data pointer
* ret:       none
* modify:
* comment:
*****/

void read_24c040(UINT32T unSlaveAddr,UINT32T unAddr,UINT8T *pData)
{
    char cRecvByte;

    f_nGetACK = 0;

```

```

/* Send control byte */
rIICDS = unSlaveAddr;           // 0xa0
rIICSTAT = 0xf0;                // Master Tx,Start

while(f_nGetACK == 0);          // Wait ACK
f_nGetACK = 0;

/* Send address */
rIICDS = unAddr;
rIICCON = 0xaf;                 // Resumes IIC operation.

while(f_nGetACK == 0);          // Wait ACK
f_nGetACK = 0;

/* Send control byte */
rIICDS = unSlaveAddr;           // 0xa0
rIICSTAT = 0xb0;                // Master Rx,Start
rIICCON = 0xaf;                 // Resumes IIC operation.

while(f_nGetACK == 0);          // Wait ACK
f_nGetACK = 0;

/* Get data */
cRecvByte = rIICDS;
rIICCON = 0x2f;
delay(1);

/* Get data */
cRecvByte = rIICDS;

/* End receive */
rIICSTAT = 0x90;                // Stop Master Rx condition
rIICCON = 0xaf;                 // Resumes IIC operation.
delay(5);                       // Wait until stop condtion is in effect.

*pData = cRecvByte;
}

/* Get data */
cRecvByte = rIICDS;
rIICCON = 0x2f;
delay(1);

/* Get data */
cRecvByte = rIICDS;

```

```

/* End receive */
rIICSTAT = 0x90;           // Stop Master Rx condition
rIICCON = 0xaf;            // Resumes IIC operation.
delay(5);                  // Wait until stop condition is in effect.

*pData = cRecvByte;        // store the read in content
}

```

6.1.8 练习题

编写程序往 AT24C04 芯片上存储某天日期字符串，再读出，并通过串口或液晶屏输出。

6.2 以太网通讯实验

6.2.1 实验目的

- 通过实验了解以太网通讯原理和驱动程序开发方法。
- 通过实验掌握 IP 网络协议和网络应用程序开发方法。

6.2.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机，以太网集线器(Hub，可选)。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

6.2.3 实验内容

熟悉以太网控制器 CS8900A，在内部以太局域网基于 TFTP/IP 协议,下载代码到目标板上。

6.2.4 实验原理

1. 以太网通讯原理

以太网是由 Xeros 公司开发的一种基带局域网碰撞检测（CSMA/CD）机制，使用同轴电缆作为传输介质，数据传输速率达到 10M；使用双绞线作为传输介质，数据传输速率达到 100M/1000M。现在普遍遵从 IEEE802.3 规范。

- 结构

以太网结构示意图如下：

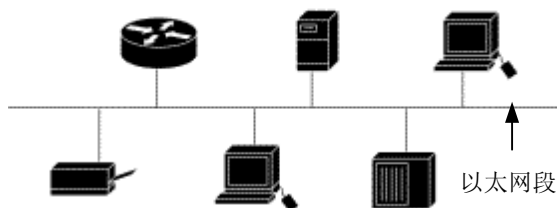


图 6-10 以太网结构示意图

● 类型

以太网/IEEE 802.3-采用同轴电缆作为网络媒体，传输速率达到 10Mbps；

100Mbps 以太网-又称为快速以太网，采用双绞线作为网络媒体，传输速率达到 100Mbps；

1000Mbps 以太网-又称为千兆以太网，采用光缆或双绞线作为网络媒体。

● 工作原理

以太网的传输方法，也就是以太网的介质访问控制（MAC）技术称为载波监听多路存取和冲突检测（CSMA / CD），下面我们分步来说明其原理：

1. 载波监听：当你所在的网站（计算机）要向另一个网站发送信息时，先监听网络信道上有无信息正在传输，信道是否空闲。

2. 信道忙碌：如果发现网络信道正忙，则等待，直到发现网络信道空闲为止。

3. 信道空闲：如果发现网络信道空闲，则向网上发送信息。由于整个网络信道为共享总线结构，网上所有网站都能够收到你所发出的信息，所以网站向网络信道发送信息也称为“广播”。但只有你想要发送数据的网站识别和接收这些信息。

4. 冲突检测：网站发送信息的同时，还要监听网络信道，检测是否有另一台网站同时在发送信息。如果有，两个网站发送的信息会产生碰撞，即产生冲突，从而使数据信息包被破坏。

5. 遇忙停发：如果发送信息的网站检测到网上的冲突，则立即停止该网络信息的发送，并向网上发送一个“冲突”信号，让其它网站也发现该冲突，从而摒弃可能一直在接收受损的信息包。

6. 多路存取：如果发送信息的网站因“碰撞冲突”而停止发送，就需等待一段时间，再回到第一步，重新开始载波监听和发送，直到数据成功发送为止。

所有共享型以太网上的网站，都是经过上述六步步骤，进行数据传输的。

由于 CSMA / CD 介质访问控制法规定在同一时间里，只能有一个网站发送信息，其它网站只能收听和等待，否则就会产生“碰撞”。所以当共享型网络用户增加时，每个网站在发送信息时产生“碰撞”的概率增大，当网络用户增加到一定数目后，网站发送信息产生的“碰撞”会越来越多，想发送信息的网站不断地进行：监听—Λ 发送—Λ 碰撞—Λ 停止发送—Λ 等待—Λ 再监听—Λ 再发送……

● 以太网/IEEE 802.3 帧的结构

下图所示为以太网/IEEE 802.3 帧的基本组成。

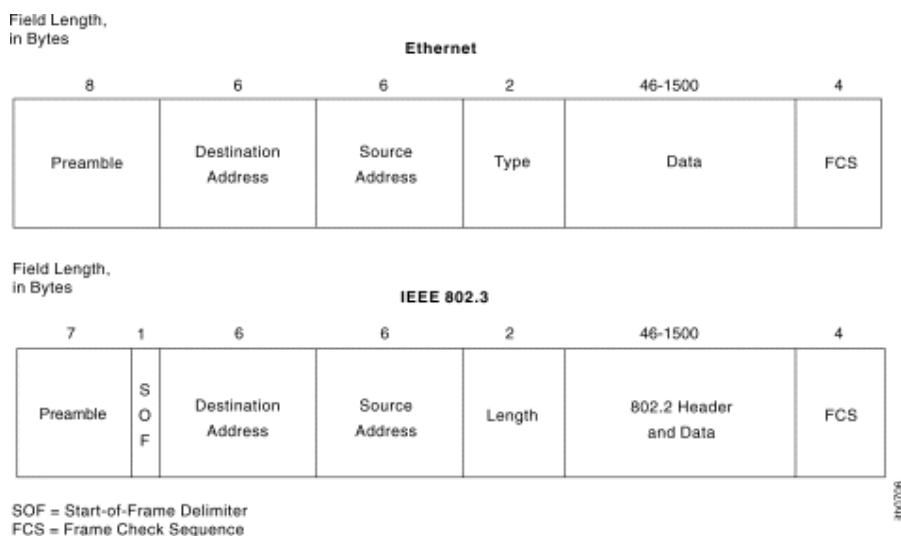


图 6-11 以太网/IEEE 802.3 帧的基本组成

如图所示，以太网和 IEEE 802.3 帧的基本结构如下：

前导码：由 0、1 间隔代码组成，可以通知目标站作好接收准备。IEEE 802.3 帧的前导码占用 7 个字节，紧随其后的是长度为 1 个字节的帧首定界符（SOF）。以太网帧把 SOF 包含在了前导码当中，因此，前导码的长度扩大为 8 个字节。

帧首定界符（SOF）：IEEE 802.3 帧中的定界字节，以两个连续的代码 1 结尾，表示一帧的实际开始。

目标和源地址：表示发送和接收帧的工作站的地址，各占据 6 个字节。其中，目标地址可以是单址，也可以是多点传送或广播地址。

类型（以太网）：占用 2 个字节，指定接收数据的高层协议。

长度（IEEE 802.3）：表示紧随其后的以字节为单位的数据段的长度。

数据（以太网）：在经过物理层和逻辑链路层的处理之后，包含在帧中的数据将被传递给在类型段中指定的高层协议。虽然以太网版本 2 中并没有明确作出补齐规定，但是以太网帧中数据段的长度最小应当不低于 46 个字节。

数据（IEEE 802.3）：IEEE 802.3 帧在数据段中对接收数据的上层协议进行规定。如果数据段长度过小，使帧的总长度无法达到 64 个字节的最小值，那么相应软件将会自动填充数据段，以确保整个帧的长度不低于 64 个字节。

帧校验序列（FSC）：该序列包含长度为 4 个字节的循环冗余校验值（CRC），由发送设备计算产生，在接收方被重新计算以确定帧在传送过程中是否被损坏。

● 以太网驱动程序开发方法

以太网驱动程序是针对实验板上的以太网网络接口芯片 CS8900A 编程，正确初始化芯片，并提供数据输入输出和控制接口给高层网络协议使用。

CS8900A 是由美国 CIRRUS LOGIC 公司生产的以太网控制器，由于其优良的性能、低功耗及低兼的价格，使其在市场上 10Mbps 嵌入式网络应用中占有相当的比例。

CS8900A 主要性能为：

- （1）符号 Ethernet II 与 IEEE802.3（10Base5、10Base2、10BaseT）标准；
- （2）全双工，收发可同时达到 10Mbps 的速率；
- （3）内置 SRAM，用于收发缓冲，降低对主处理器的速度要求；
- （4）支持 16 位数据总线，4 个中断申请线以及三个 DMA 请求线；
- （5）8 个 I/O 基地址，16 位内部寄存器，IO Base 或 Memory Map 方式访问；
- （6）支持 UTP、AUI、BNC 自动检测，还支持对 10BaseT 拓扑结构的自动极性修正；
- （7）LED 指示网络激活和连接状态；
- （8）100 脚的 LQFP 封装，缩小了 PCB 尺寸。

CS8900A 的应用原理图如下所示：

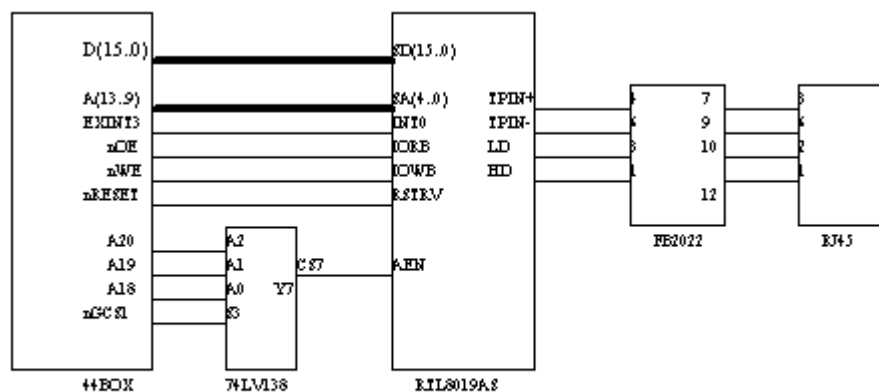


图 6-12 RTL8019AS 的应用原理图

cs8900a.c 文件是 CS8900A 的驱动程序，函数功能如下：

CS_Init() -- CS8900A 初始化。初始化步骤为：1、检测 CS8900A 芯片是否存在，然后软件复位 CS8900A；2、如果使用 Memory Map 方式访问 CS8900A 芯片内部寄存，便设置 CS8900A 内部寄存基地址（默认为 IO 方式访问）；3、设置 CS8900A 的 MAC 地址；4、关闭事件中断（本例子使用查询方式，如果使用中断方式，则添加中断服务程序再打开 CS8900A 中断允许位）；5、配置 CS8900A 10BT，然后允许 CS8900A 接收和发送，接收发送 64-1518 字节的网络帧及网络广播帧。

CS_Close() -- 关闭 CS8900A 芯片数据收发功能，及关闭中断请求。

CS_Reset() -- 复位 CS8900A 芯片。

CS_Identification() -- 获得 CS8900A 芯片 ID 和修订版本号。

CS_TransmitPacket() -- 数据包输出。将要发送的网络数据包从网口发送出去。发送数据包时，先把发送命令写到发送命令寄存器，把发送长度写到发送长度寄存器，然后等待 CS8900A 内部总线状态寄存器发送就绪位置位，便将数据包的数据顺序写到数据端口寄存器（16 位宽，一次两个字节）。

CS_ReceivePacket() -- 数据包接收。查询数据接收事件寄存器，若有数据帧接收就绪，读取接收状态寄存器（与接收事件寄存器内容一致，忽略之），及读取接收长度寄存器，得到数据帧的长度，然后从数据端口寄存器顺序读取数据（16 位宽，一次两个字节）。

2. IP 网络协议原理

TCP/IP 协议是一组包括 TCP (Transmission Control Protocol) 协议和 IP (Internet Protocol) 协议，UDP (User Datagram Protocol) 协议、ICMP (Internet Control Message Protocol) 协议和其他一些协议的协议组。

TCP / IP 的历史可以追溯至 70 年代中期，最早由斯坦福大学的两名研究人员于 1973 年提出，当时 ARPA (Advanced Research Project Agency, 高级研究计划局) 为了实现异种网之间的互连与互通，大力资助网间网技术的研究开发，于 1977 年到 1979 年间推出较完整的与目前形式一样的 TCP / IP 体系结构和协议规范。1980 年前后，DARPA (国防部高级研究计划局) 开始将 ARPANET 上的所有机器转向 TCP / IP 协议，并以 ARPANET 为主干建立 Internet。在 1985 年，美国国家科学基金会 (NSF, National Scientific Foundation) 开始涉足 TCP / IP 的研究和开发，并逐渐成为极为重要的角色。国家科学基金会资助建立了全球性的 Internet 网并采用 TCP / IP 为其传输协议。

● 结构

TCP/IP 协议采用分层结构，共分为四层，每一层独立完成指定功能，如下图所示：

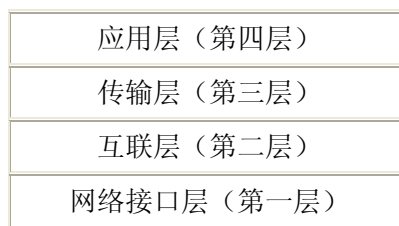


图 6-13 TCP/IP 协议层次

网络接口层：负责接收和发送物理帧，它定义了将数据组成正确帧的规程和在网络中传输帧的规程，帧是指一串数据，它是数据在网络中传输的单位。网络接口层将帧放在网上，或从网上把帧取下来。

互联层：负责相邻结点之间的通信，本层定义了互联网中传输的“信息包”格式，以及从一个节点通过一个或多个路由器运行必要的路由算法到最终目标的“信息包”转发机制。主要协议有 IP、ARP、ICMP、IGMP 等。

传输层：负责起点到终点的通信，为两个用户进程之间建立、管理和拆除有效的端到端连接。主要协议有 TCP、UDP 等。

应用层：它定义了应用程序使用互联网的规程。应用程序通过这一层访问网络，主要遵从 BSD 网络应用接口规范。主要协议有 SMTP、FTP、TELNET、HTTP 等。

● 主要协议介绍

1. IP

网际协议 IP 是 TCP/IP 的心脏，也是网络层中最重要的协议。

IP 层接收由更低层（网络接口层例如以太网设备驱动程序）发来的数据包，并把该数据包发送到更高层---TCP 或 UDP 层；相反，IP 层也把从 TCP 或 UDP 层接收来的数据包传送到更低层。IP 数据包是不可靠的，因为 IP 并没有做任何事情来确认数据包是按顺序发送的或者没有被破坏。IP 数据包中含有发送它的主机的地址（源地址）和接收它的主机的地址（目的地址）。

IP 是一个无连接的协议，主要就是负责在主机间寻址并为数据包设定路由，在交换数据前它并不建立会话。因为它不保证正确传递，另一方面，数据在被收到时，IP 不需要收到确认，所以它是不可靠的。如果 IP 目标地址为本地地址，IP 将数据包直接传给那个主机；如果目标地址为远程地址的话，IP 在本地的路由表中查找远程主机的路由（看起来好象我们平时拨 114 一样）。如果找到一个路由，IP 用它传送数据包。如果没找到，就会将数据包发送到源主机的缺省网关，也称之为路由器。

当前 IP 协议有 IPv4 和 IPv6 两个版本，IPv4 正被广泛使用，IPv6 是下一代高速互联网的基础协议。下面这个表是 IPv4 的数据包格式：

| 0 | 4 | 8 | 16 | 32 |
|------|------|------|----------------|----|
| 版本 | 首部长度 | 服务类型 | 数据包总长 | |
| 标识 | | | DF MF 碎片偏移 | |
| 生存时间 | | 协议 | 首部校验和 | |
| | | | 源 IP 地址 | |
| | | | 目的 IP 地址 | |
| | | | 选项 | |
| | | | ===== | |
| | | | 数据 | |

下面我们看一看 IP 协议头的结构定义：

```

struct ip_header
{
    UINT    ip_v:4;           /* 协议版本 */
    UINT    ip_hl:4;          /* 协议头长度 */
    UINT8   ip_tos;           /* 服务类型 */
    UINT16  ip_len;           /* 数据包长度 */
    UINT16  ip_id;            /* 协议标识 */
    UINT16  ip_off;           /* 分段偏移域 */
    UINT8   ip_ttl;           /* 生存时间 */
    UINT8   ip_p;             /* IP 数据包的高层协议 */
    UINT16  ip_sum;           /* checksum */
    struct in_addr ip_src, ip_dst; /* 源和目的 IP 地址 */
};

```

ip_v: IP 协议的版本号, IPv4 为 4, IPv6 为 6。

ip_hl: IP 包首部长度,这个值以 4 字节为单位.IP 协议首部的固定长度为 20 个字节,如果 IP 包没有选项,那么这个值为 5。

ip_tos: 服务类型,说明提供的优先权。

ip_len: 说明 IP 数据的长度.以字节为单位。

ip_id: 标识这个 IP 数据包。

ip_off: 碎片偏移, 这和上面 ID 一起用来重组碎片的。

ip_ttl: 生存时间, 每经过一个路由时减一,直到为 0 时被抛弃。

ip_p: 协议, 表示创建这个 IP 数据包的高层协议.如 TCP,UDP 协议。

ip_sum: 首部校验和, 提供对首部数据的校验。

ip_src,ip_dst: 发送者和接收者的 IP 地址。

关于 IP 协议的详细情况,请参考 RFC791 文档。

IP 地址实际上是采用 IP 网间网层通过上层软件完成“统一”网络物理地址的方法,这种方法使用统一的地址格式,在统一管理下分配给主机。Internet 网上不同的主机有不同的 IP 地址,在 IPv4 协议中,每个主机的 IP 地址都是由 32 比特,即 4 个字节组成的。为了便于用户阅读和理解,通常采用“点分十进制表示方法”表示,每个字节为一部分,中间用点号分隔开来。如 211.154.134.93 就是嵌入开发网 WEB 服务器的 IP 地址。每个 IP 地址又可分为两部分。网络号表示网络规模的大小,主机号表示网络中主机的地址编号。按照网络规模的大小,IP 地址可以分为 A、B、C、D、E 五类,其中 A、B、C 类是三种主要的类型地址,D 类专供多目传送用的多目地址,E 类用于扩展备用地址。

2. TCP

如果 IP 数据包中有已经封好的 TCP 数据包,那么 IP 将把它们向‘上’传送到 TCP 层。TCP 将包排序并进行错误检查,同时实现虚电路间的连接。TCP 数据包中包括序号和确认,所以未按照顺序收到的包可以被排序,而损坏的包可以被重传。

TCP 将它的信息送到更高层的应用程序,例如 Telnet 的服务程序和客户程序。应用程序轮流将信息送回 TCP 层,TCP 层便将它们向下传送到 IP 层,设备驱动程序和物理介质,最后到接收方。下面这个表是 TCP 协议的数据包头格式:

| 0 | 4 | 8 | 10 | 16 | 24 | 32 |
|------|---|----|----|------|------|----|
| 源端口 | | | | 目的端口 | | |
| 序列号 | | | | | | |
| 确认号 | | | | | | |
| 首部长度 | | 保留 | | 窗口 | | |
| | | | | | | |
| | | | | | | |
| 校验和 | | | | 紧急指针 | | |
| 选项 | | | | | 填充字节 | |

关于 TCP 协议的详细情况,请查看 RFC793 文档。

TCP 对话通过三次握手来初始化。三次握手的目的是使数据段的发送和接收同步;告诉其它主机其一次可接收的数据量,并建立虚连接。

我们来看看这三次握手的简单过程:

(1) 初始化主机通过一个同步标志置位的数据段发出会话请求。

(2) 接收主机通过发回具有以下项目的数据段表示回复:同步标志置位、即将发送的数据段的起始字节的序号、应答并带有将收到的下一个数据段的字节序号。

(3) 请求主机再回送一个数据段,并带有确认序号和确认号。

3. UDP

UDP 与 TCP 位于同一层,但对于数据包的顺序错误或重发不处理。因此,UDP 不被应用于那些使用虚电路的面向连接的服务,UDP 主要用于那些面向查询---应答的服务,例如 NFS。相对于 FTP 或 Telnet,这些服务需要交换的信息量较小。使用 UDP 的服务包括 NTP(网落时间协议)和 DNS(DNS 也使用 TCP)。

UDP 协议的数据包头格式为:

| | | |
|---|----|----|
| 0 | 16 | 32 |
|---|----|----|

| | |
|-----------|-----------|
| UDP 源端口 | UDP 目的端口 |
| UDP 数据报长度 | UDP 数据报校验 |

关于 UDP 协议的详细情况,请参考 RFC768 文档。

UDP 协议适用于无须应答并且通常一次只传送少量数据的应用软件。

4. ICMP

ICMP 与 IP 位于同一层，它被用来传送 IP 的控制信息。它主要是用来提供有关通向目的地址的路径信息。ICMP 的'Redirect'信息通知主机通向其他系统的更准确的路径，而'Unreachable'信息则指出路径有问题。另外，如果路径不可用了，ICMP 可以使 TCP 连接'体面地'终止。PING 是最常用的基于 ICMP 的服务。

关于 ICMP 协议的详细情况可以查看 RFC792 文档。

5. ARP

要在网络上通信，主机就必须知道对方主机的硬件地址(我们不是老遇到网卡的物理地址嘛)。地址解析就是将主机 IP 地址映射为硬件地址的过程。地址解析协议 ARP 用于获得在同一物理网络中的主机的硬件地址。

解释本地网络 IP 地址过程:

(1) 当一台主机要与别的主机通信时，初始化 ARP 请求。当该 IP 断定 IP 地址是本地时，源主机在 ARP 缓存中查找目标主机的硬件地址。

(2) 要是找不到映射的话，ARP 建立一个请求，源主机 IP 地址和硬件地址会被包括在请求中，该请求通过广播，使所有本地主机均能接收并处理。

(3) 本地网上的每个主机都收到广播并寻找相符的 IP 地址。

(4) 当目标主机断定请求中的 IP 地址与自己的相符时，直接发送一个 ARP 答复，将自己的硬件地址传给源主机。以源主机的 IP 地址和硬件地址更新它的 ARP 缓存。源主机收到回答后便建立起了通信。

关于 ARP 协议的详细情况可以查看 RFC826 文档。

6. TFTP 协议

TFTP 是一个传输文件的简单协议，一种简化的 TCP/IP 文件传输协议，它基于 UDP 协议而实现，支持用户从远程主机接收或向远程主机发送文件。此协议设计的时候是进行小文件传输的。因此它不具备通常的 FTP 的许多功能，它只能从文件服务器上获得或写入文件，不能列出目录，不进行认证，它传输 8 位数据。

因为 TFTP 使用 UDP，而 UDP 使用 IP，IP 还可以使用其它本地通信方法。因此一个 TFTP 包中会有以下几段：本地媒介头，IP 头，数据报头，TFTP 头，剩下的就是 TFTP 数据了。TFTP 在 IP 头中不指定任何数据，但是它使用 UDP 中的源和目标端口以及包长度域。由 TFTP 使用的包标记（TID）在这里被用做端口，因此 TID 必须介于 0 到 65,535 之间。

初始连接时候需要发出 WRQ (请求写入远程系统) 或 RRQ (请求读取远程系统), 收到一个确定应答, 一个确定的可以写出的包或应该读取的第一块数据。通常确认包包括要确认的包的包号, 每个数据包都与一个块号相对应, 块号从 1 开始而且是连续的。因此对于写入请求的确定是一个比较特殊的情况, 因此它的包的包号是 0。如果收到的包是一个错误的包, 则这个请求被拒绝。创建连接时, 通信双方随机选择一个 TID, 因为是随机选择的, 因此两次选择同一个 ID 的可能性就很小了。每个包包括两个 TID, 发送者 ID 和接收者 ID。在第一次请求的时候它会将请求发到 TID 69, 也就是服务

器的 69 端口上。应答时，服务器使用一个选择好的 TID 作为源 TID，并用上一个包中的 TID 作为目的 ID 进行发送。这两个被选择的 ID 在随后的通信中会被一直使用。

此时连接建立，第一个数据包以序列号 1 从主机开始发出。以后两台主机要保证以开始时确定的 TID 进行通信。如果源 ID 与原来确定的 ID 不一样，这个包会被认为发送到了错误的地址而被抛弃。

关于 TFTP 协议的详细情况可以查看 RFC783 文档。

● 网络应用程序开发方法

进行网络应用程序开发有两种方法：一是采用 BSD Socket 标准接口，程序移植能力强；二是采用专用接口直接调用对应的传输层接口，效率较高。

1. BSD Socket 接口编程方法

Socket（套接字）是通过标准的文件描述符和其它程序通讯的一个方法。每一个套接字都用一个半相关描述：{ 协议，本地地址、本地端口 } 来表示；一个完整的套接字则用一个相关描述：{ 协议，本地地址、本地端口、远程地址、远程端口 }，每一个套接字都有一个本地的由操作系统分配的唯一套接字号。

Socket 接口有三种类型：

（1）流式 Socket（SOCK_STREAM）

流式套接字提供可靠的、面向连接的通信流；它使用 TCP 协议，从而保证了数据传输的正确性和顺序性。

（2）数据报 Socket（SOCK_DGRAM）

数据报套接字定义了一种无连接的服务，数据通过相互独立的报文进行传输，是无序的，并且不保证可靠、无差错。它使用数据报协议 UDP

（3）原始 Socket

原始套接字允许对底层协议如 IP 或 ICMP 直接访问，它功能强大但使用较为不便，主要用于一些协议的开发。

Socket 编程原理图如下：

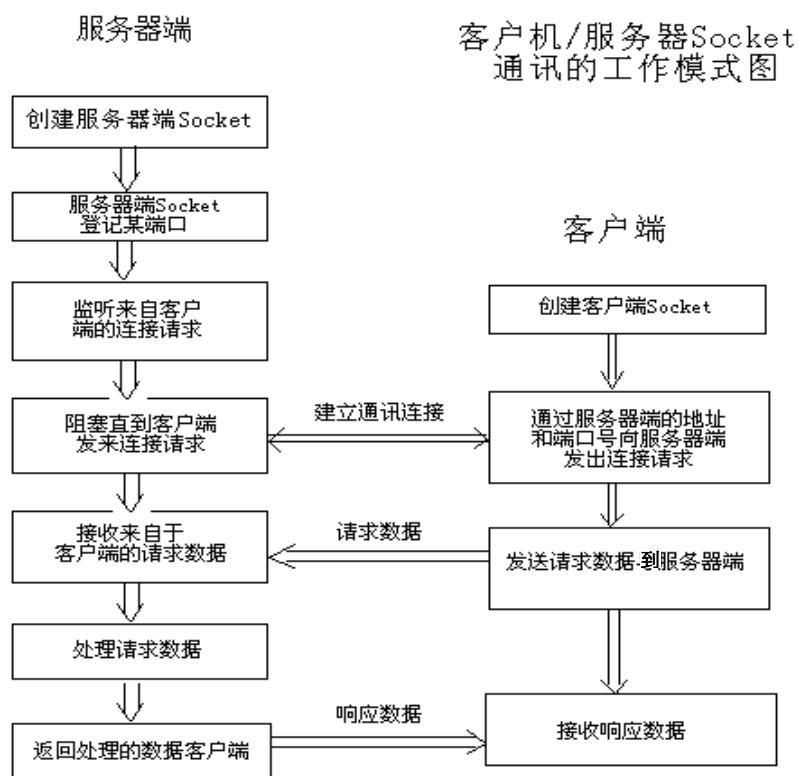


图 6-14 Socket 编程原理图

常用的 Socket 接口函数有:

socket() —— 创建套接字

bind() —— 指定本地地址

connect() —— 连接目标套接字

accept() —— 等待套接字连接

listen() —— 监听连接

send() —— 发送数据

recv() —— 接收数据

select() —— 输入/输出多路复用

closesocket() —— 关闭套接字

一个标准的服务器端接收数据范例程序如下:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
```

```
#define MYPORT 4950 /* the port users will be sending to */
```

```

#define MAXBUFLEN 100

void main()
{
    int sockfd;
    struct sockaddr_in my_addr; /* my address information */
    struct sockaddr_in their_addr; /* connector's address information */
    int addr_len, numbytes;
    char buf[MAXBUFLEN];

    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1 )
    {
        perror("socket");
        exit(1);
    }
    my_addr.sin_family = AF_INET; /* host byte order */
    my_addr.sin_port = htons(MYPORT); /* short, network byte order */
    my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
    bzero(&(my_addr.sin_zero), /* zero the rest of the struct */
        sizeof(my_addr.sin_zero));

    if( bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1 )
    {
        perror("bind");
        exit(1);
    }
    addr_len = sizeof(struct sockaddr);
    if ( (numbytes=recvfrom(sockfd, buf, MAXBUFLEN, 0, \
        (struct sockaddr *)&their_addr, &addr_len)) == -1 )
    {
        perror("recvfrom");
        exit(1);
    }
    printf("got packet from %s\n",inet_ntoa(their_addr.sin_addr));
    printf("packet is %d bytes long\n",numbytes);
    buf[numbytes] = '\0';
    printf("packet contains \"%s\"\n",buf);
    close(sockfd);
}

```

2. 传输层专有接口编程方法

网络协议都可以直接提供专有函数接口给上层或者跨层调用，用户可以调用每个协议代码中特有的接口实现快速数据传递。

实验板的网络协议包提供的就是 TFTP 协议的专用接口，应用程序可以通过它接收用户从主机上使用 TFTP 传递过来的数据。主要接口函数有：

TftpRecv(int* len) -- 接收用户数据，网络协议包自动完成连接过程，并从网络获取用户传递过来的数据包，每次接收最大长度由 len 指定，返回前改写成实际接收到的数据长度，函数返回数据首地址指针，若返回值为 NULL，表示接收故障。

MakeAnswer() -- 返回应答信号，每当用户处理完一个数据包后，需要调用此函数给对方一个确认信号，使得数据传输得以继续。

3. 关于 CS8000A

➤ CS8900A 特点介绍

CS8900A 是由美国 CIRRUS LOGIC 公司生产的以太网控制器，由于其优良的性能、低功耗及低廉的价格，使其在市场上 10Mbps 嵌入式网络应用中占有相当的比例。

CS8900A 主要性能为：

- (1) 符合 Ethernet II 与 IEEE802.3 (10Base5、10Base2、10BaseT) 标准；
- (2) 全双工，收发可同时达到 10Mbps 的速率；
- (3) 内置 SRAM，用于收发缓冲，降低对主处理器的速度要求；
- (4) 支持 16 位数据总线，4 个中断申请线以及三个 DMA 请求线；
- (5) 8 个 I/O 基地址，16 位内部寄存器，IO Base 或 Memory Map 方式访问；
- (6) 支持 UTP、AUI、BNC 自动检测，还支持对 10BaseT 拓扑结构的自动极性修正；
- (7) LED 指示网络激活和连接状态；
- (8) 100 脚的 LQFP 封装，缩小了 PCB 尺寸。

➤ CS8900A 引脚分布

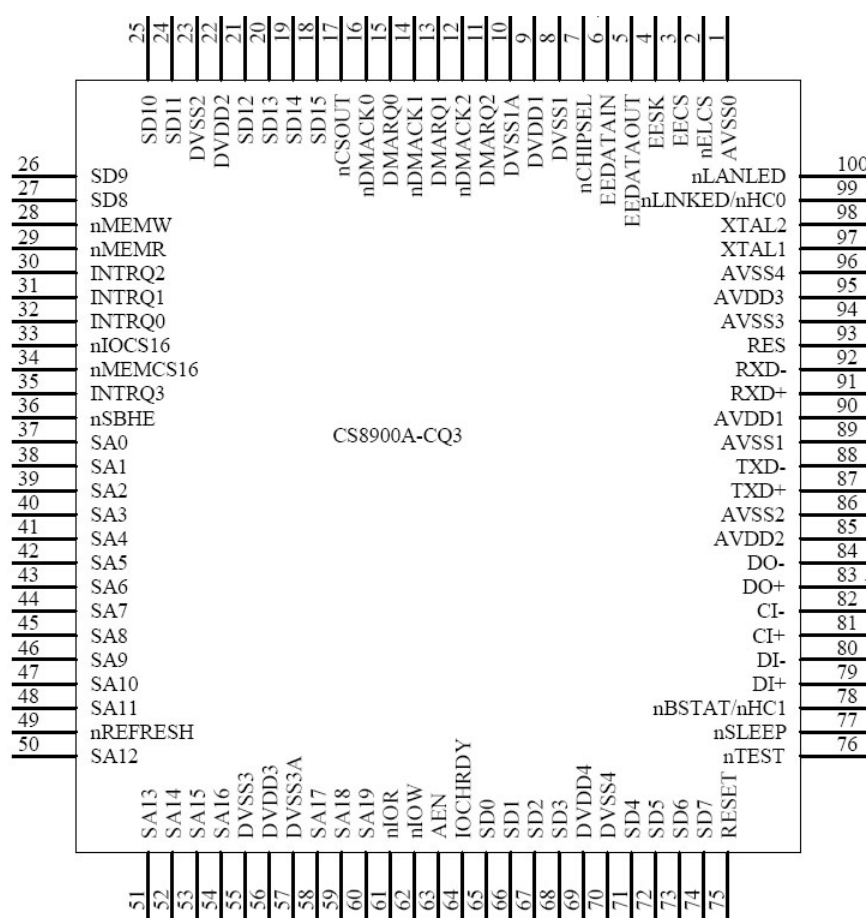


图 6-2-4 CS8900A 的引脚分布图

➤ 复位与初始化过程

引起 CS8900A 复位的因素有很多种,有人为的需要,也有意外产生的复位。如外部复位信号(在 RESET 引脚加至少 400ns 的高电平)引起复位,上电自动复位,下电复位(电压低于 2.5V),EEPROM 校验失败引起复位以及软件复位等。复位之后,CS8900A 需要重新进行配置。

每次复位之后(除 EEPROM 校验失败引起复位以外),CS8900A 都会检查 EEDataIn 引脚,判断是否有外部的 EEPROM 存在。如果 EEDI 是高电平,则说明 EEPROM 存在,CS8900A 会自动将 EEPROM 中的数据加载到内部寄存器中(Embest EduKit-III 实验平台未采用外接 EEPROM 的方法,具体的配置过程可以参考 CS8900A 的硬件手册);如果 EEDI 为低电平,则 EEPROM 不存在,CS8900A 会按照下表所示进行默认的配置。

表 6-2-1 CS8900A 的默认配置

| PacketPage 地址 | 寄存器内容 | 寄存器描述 |
|---------------|------------------------|----------------|
| 0020h | 0300h | I/O 基地址 |
| 0022h | XXXX XXXX XXXX X100 | 中断号 |
| 0024h | XXXX XXXX XXXX XX11 | DMA 通道号 |
| 0026h | 0000h | DMA 基址到待发送帧的偏移 |
| 0028h | X000h | DMA 帧数 |
| 002Ah | 0000h | DMA 字节数 |
| 002Ch | XXX0 0000h | 内存基地址 |
| 0030h | XXX0 0000h | 引导 PROM 基地址 |
| 0034h | XXX0 0000h | 引导 PROM 地址掩码 |
| 0102h | 0003h | 寄存器 3—RxCFG |
| 0104h | 0005h | 寄存器 5—RxCTL |
| 0106h | 0007h | 寄存器 7—TxCFG |
| 0108h | 0009h | 寄存器 9—TxCMD |
| 010Ah | 000Bh | 寄存器 B—BufCFG |
| 010Ch | Undefined | 保留 |
| 010Eh | Undefined | 保留 |
| 0110h | Undefined | 保留 |
| 0112h | 00013h | 寄存器 13—LineCTL |
| 0114h | 0015h | 寄存器 15—SelfCTL |
| 0116h | 0017h | 寄存器 17—BusCTL |

| | | |
|-------|-------|-----------------|
| 0118h | 0019h | 寄存器 19--TestCTL |
|-------|-------|-----------------|

➤ PACKETPAGE 结构

CS8900A 的结构的核心是提供高效访问方法的内部寄存器和缓冲内存。

PacketPage 是 CS8900A 中集成的 RAM。它可以用作接收帧和待发送帧的缓冲区，除此之外还有一些其他的用途。PacketPage 为内存或 I/O 空间提供了一种统一的访问控制方法，减轻了 CPU 的负担，降低了软件开发的难度。此外，它还提供一系列灵活的配置选项，允许开发者根据特定的系统需求设计自己的以太网模块。PacketPage 中可供用户操作的部分可以划分为以下几个部分：

| | |
|--------------|----------|
| 0000h——0045h | 总线接口寄存器 |
| 0100h——013Fh | 状态与控制寄存器 |
| 0140h——014Fh | 发送初始化寄存器 |
| 0150h——015Dh | 地址过滤寄存器 |
| 0400h | 接收帧地址 |
| 0A00h | 待发送帧地址 |

（具体地址及定义见实验参考程序 1）

4. CS8000A 工作模式

CS8900A 有两种工作模式，一种是 I/O 访问方式，一种是内存访问方式。网卡芯片复位后默认工作方式为 I/O 连接，I/O 端口基址为 300H，下面对它的几个主要工作寄存器进行介绍（寄存器后括号内的数字为寄存器地址相对基址 300H 的偏移量）。

LINECTL (0112H) LINECTL 决定 CS8900 的基本配置和物理接口。在本系统中，设置初始值为 00d3H，选择物理接口为 10BASE-T，并使能设备的发送和接收控制位。

RXCTL (0104H) RXCTL 控制 CS8900 接收特定数据报。设置 RXTCL 的初始值为 0d05H，接收网络上的广播或者目标地址同本地物理地址相同的正确数据报。

RXCFG (0102H) RXCFG 控制 CS8900 接收到特定数据报后会引发接收中断。RXCFG 可设置为 0103H，这样当收到一个正确数据报后，CS8900 会产生一个接收中断。

BUSCT (0116H) BUSCT 可控制芯片的 I/O 接口的一些操作。设置初始值为 8017H，打开 CS8900 的中断总控制位。

ISQ (0120H) ISQ 是网卡芯片的中断状态寄存器，内部映射接收中断状态寄存器和发送中断状态寄存器的内容。

PORT0 (0000H) 发送和接收数据时，CPU 通过 PORT0 传递数据。

TXCMD (0004H) 发送控制寄存器，如果写入数据 00C0H，那么网卡芯片在全部数据写入后开始发送数据。

TXLENG (0006H) 发送数据长度寄存器，发送数据时，首先写入发送数据长度，然后将数据通过 PORT0 写入芯片。

以上为几个最主要的工作寄存器（为 16 位），CS8900 支持 8 位模式，当读或写 16 位数据时，低位字节对应偶地址，高位字节对应奇地址。例如，向 TXCMD 中写入 00C0H，则可将 00h 写入 305H，将 C0H 写入 304H。

系统工作时，应首先对网卡芯片进行初始化，即写寄存器 LINECTL、RXCTL、RCCFG、BUSCT。发数据时，写控制寄存器 TXCMD，并将发送数据长度写入 TXLENG，然后将数据依次写入 PORT0 口，如将第一个字节写入 300H，第二个字节写入 301H，第三个字节写入 300H，依此类推。网卡芯

片将数据组织为链路层类型并添加填充位和 CRC 校验送到网络同样，处理器查询 ISO 的数据，当有数据来到后，读取接收到的数据帧。

5. CS8900A 驱动程序设计

以太网驱动程序是针对实验板上的以太网接口芯片 CS8900A 编程，正确初始化芯片，并提供数据输入输出和控制接口给高层网络协议使用。

源码中，cs8900a.c 文件是 CS8900A 的驱动程序，函数功能如下：

CS_Init()——CS8900A 初始化。初始化步骤为：①检测 CS8900A 芯片是否存在，然后软件复位 CS8900A；②如果使用 Memory Map 方式访问 CS8900A 芯片内部寄存，就设置 CS8900A 内部寄存基地址（默认为 IO 方式访问）；③设置 CS8900A 的 MAC 地址；④关闭事件中断（本例子使用查询方式，如果使用中断方式，则添加中断服务程序再打开 CS8900A 中断允许位）；⑤配置 CS8900A 10BT，然后允许 CS8900A 接收和发送，接收发送 64-1518 字节的网络帧及网络广播帧（见实验参考程序 2）。

CS_Close()——关闭 CS8900A 芯片数据收发功能，及关闭中断请求。

CS_Reset()——复位 CS8900A 芯片。

CS_Identification()——获得 CS8900A 芯片 ID 和修订版本号。

CS_TransmitPacket ()——数据包输出。将要发送的网络数据包从网口发送出去。发送数据包时，先把发送命令写到发送命令寄存器，把发送长度写到发送长度寄存器，然后等待 CS8900A 内部总线状态寄存器发送就绪位置位，便将数据包的数据顺序写到数据端口寄存器（16 位宽，一次两个字节）。

CS_ReceivePacket ()——数据包接收。查询数据接收事件寄存器，若有数据帧接收就绪，读取接收状态寄存器（与接收事件寄存器内容一致，忽略之），及读取接收长度寄存器，得到数据帧的长度，然后从数据端口寄存器顺序读取数据（16 位宽，一次两个字节）。

6.2.5 实验操作步骤

1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接到目标板上，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 配置

1) 将 PC 机 IP 地址设置为 192.192.192.x（x 取值在 30-200 之间）。

2) 运行 PC 机 DOS 窗口或者“开始”系统按钮上的“运行”菜单，输入命令 command:

```
C:\DOCUME~1\SS>cd \
```

```
C:\>arp -s 192.192.192.200 00-06-98-01-7e-8f
```

```
C:\>arp -a
```



```
Interface: 192.192.192.36 --- 0x2
```

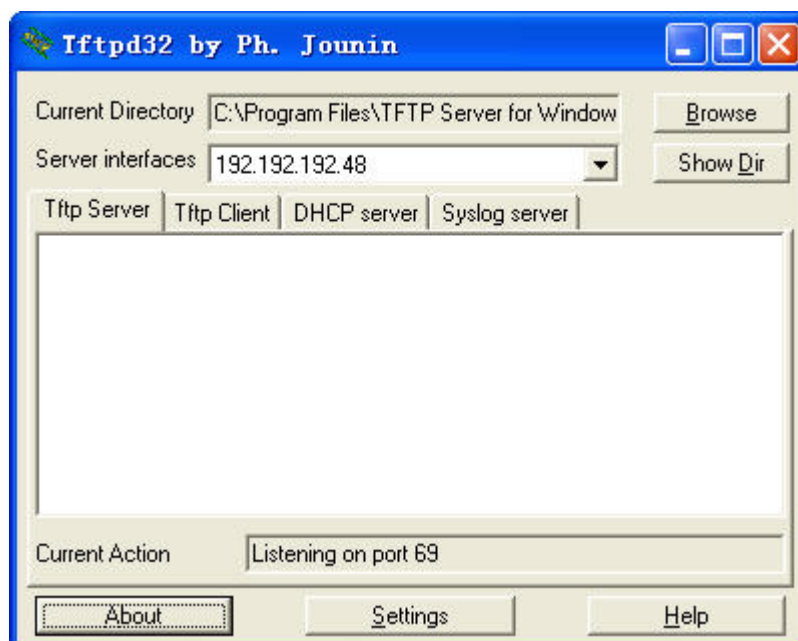
| Internet Address | Physical Address | Type |
|------------------|-------------------|--------|
| 192.192.192.200 | 00-06-98-01-7e-8f | static |

为 PC 机添加一个到目标板的地址解析。

3) 在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。使用教学系统提供的交叉网线连接开发板网口（NET1）和 PC 机；也可以使用直通网线，连接到与 PC 同一局域网的 HUB 上。

4. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上，打开实验例程目录 6.2_tftp_test 子目录下的 tftp_test.Uv2 例程，编译链接工程，直到链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用！！！这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Run 或 F5 键全速运行程序；
- 6) 运行光盘中附带的 DHCP Server 软件（Tools\tftpd32\tftpd32.exe），以动态分配 IP 地址（也可用网络中的 DHCP 服务器）：



在 PC 上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...
Ethernet TFTP client Test Example
```

```

Reset CS8900A successful,   Rev F.

S/s -- DHCP IP addr

D/d -- Default IP addr(192.192.192.200)

Y/y -- Input New IP addr

Press a key to continue ...

```

7) 在超级终端输入 d, 选择默认 IP, 界面如下:

```

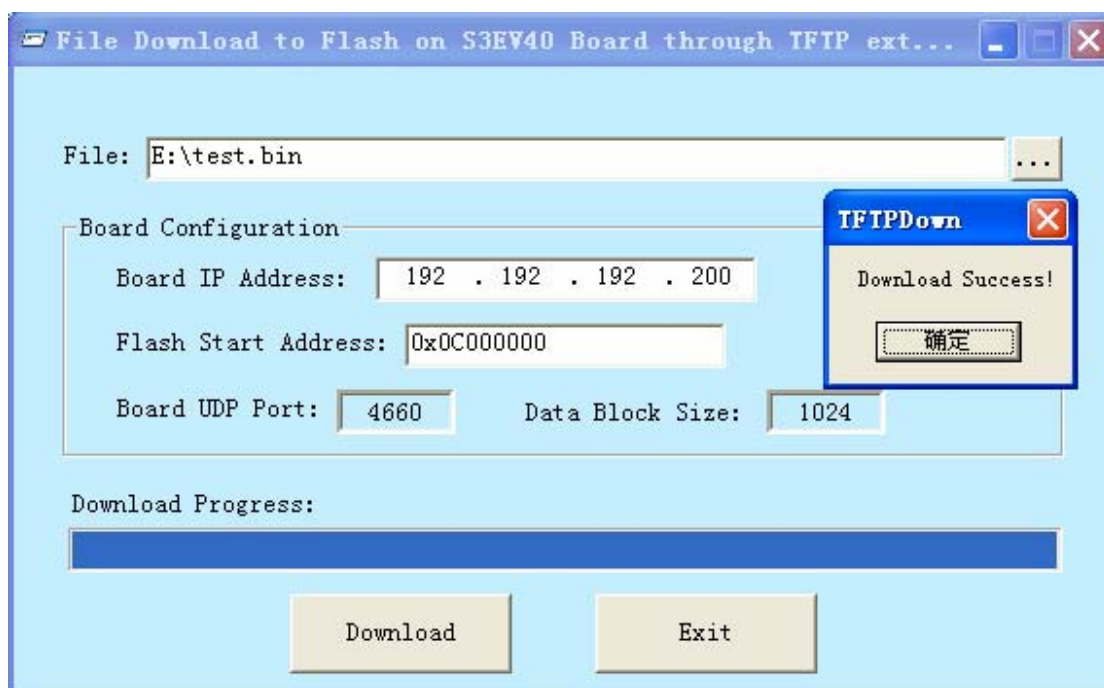
Press a key to continue ... d

Set local ip 192.192.192.200

Press any key to exit ...

```

8) 在 PC 上运行 TFTPDown.exe 程序 (在 CD1 根目录 Tools 下), 目标板地址输入 192.192.192.200, Flash Start Address 输入 0x0c000000, 然后选取想要下载的文件 (BIN 文件、ELF 文件等各种文件均可), 点击 Download, 程序开始通过 TFTP 协议下载文件到目标板 Flash 中, 成功或者出错都有提示对话框。



4. 观察实验结果

使用 MDK 停止目标板运行, 打开 Memory 窗口, 输入 0x0c000000, 然后检查 Flash 中的数据是否和下载的文件数据一致。

25. 完成实验练习题

理解和掌握实验后, 完成实验练习题。

6.2.6 实验参考程序

```

/*****
* name:      tftp_test
* func:      Tftp test
* para:      none
* ret:       none
* modify:
* comment:
*****/

void tftp_test()
{
    char* pData;
    unsigned long write_addr;
    char input_string[64];
    char tmp_ip[4] = {0,0,0,0};
    int  tmp,len,i,j,num=0;
    int  b10 =0; int b100 =0; int flag=0;

    NicInit();
    NetInit();

    uart_printf("\n S/s -- DHCP IP addr\n");
    uart_printf(" D/d -- Default IP addr(192.192.192.200)\n");
    uart_printf(" Y/y -- Input New IP addr\n");
    uart_printf(" Press a key to continue ... %c \n",i = uart_getch());
    switch(i)
    {
    case 'Y':
    case 'y':
        uart_printf(" Please input IP address(xxx.xxx.xxx.xxx) then press ENTER:");
        uart_getstring((char *)&input_string);

        for( i = 0;((i <16)&(input_string[i] != '\0')); i++)
            if(input_string[i] == '.') num +=1;

        if(num != 3) flag = 1;
        else
        {
            num = i - 1; j =0;
            for( i = num; i >= 0; i--)
            {
                if(input_string[i] != '.' )
                {
                    if((input_string[i] < '0' | input_string[i] > '9')) flag = 1;
                    else

```

```

        {

            tmp = (input_string[i] - 0x30);
            if (b100) { tmp *=100; b10 =0; }
            if (b10)  { tmp *= 10; b100 =1;}

            b10 = 1;
            if(tmp < 256) tmp_ip[j] += tmp; else local_ip = 0x4dc0c0c0;
        }

    }else { j++; b10 =0; b100 =0;}
}

if(!flag)
    local_ip = ((tmp_ip[0]<<24))+((tmp_ip[1]<<16))+((tmp_ip[2]<<8))+tmp_ip[3];
else
{
    uart_printf("\nIP address error (xxx.xxx.xxx.xxx)!\n");
    return;
}

break;

case 'S':
case 's':
    DhcpQuery();
    break;

case 'D':
case 'd':
default:
    local_ip = 0xc8c0c0c0;          // config local ip 192.192.192.200
    break;
}

uart_printf("\n Set local ip %d.%d.%d.%d\n",
            local_ip&0x000000FF,(local_ip&0x0000FF00)>>8,
            (local_ip&0x00FF0000)>>16,(local_ip&0xFF000000)>>24);
uart_printf("\nPress any key to exit ...\n");

for( ; ; )
{
    if( uart_getkey() )
        return;
}

```

```

pData = (char *)TftpRecv(&len);
if( (pData == 0) || (len <= 0) )
    continue;

write_addr = (pData[0])+(pData[1]<<8)+(pData[2]<<16)+(pData[3]<<24);
pData = pData + sizeof(long);
len -= 4;

if((*pData == 'E')&&*(pData+1) == 'N')&&*(pData+2) == 'D') //run
{
    MakeAnswer();
    return;
}
else if((*pData == 'R')&&*(pData+1) == 'U')&&*(pData+2) == 'N') //run
{
    MakeAnswer();

    run = (void *)write_addr;
    (*run)(); // jump the new address to run
}

if(write_addr >= 0x0c000000 && write_addr < 0x0c800000)
{
    memcpy((void *)write_addr, pData, len);
}
else if(FlashID(write_addr))
{
    int write_ptr, offset_ptr, write_len, sector_size;
    char *data_ptr;

    data_ptr = pData;
    write_ptr = write_addr;
    while(write_ptr < write_addr+len)
    {
        sector_size = FlashSectorBackup(write_ptr, sector_buf);
        offset_ptr = (write_ptr & ~(0 - sector_size));
        write_len = sector_size - offset_ptr;
        if(write_len > len - (write_ptr - write_addr))
            write_len = len - (write_ptr - write_addr);
        memcpy(&sector_buf[offset_ptr], data_ptr, write_len);
        FlashEraseSector(write_ptr);
        FlashProgram(write_ptr & (0 - sector_size),(unsigned short *) sector_buf, sector_size);
        data_ptr += write_len;
        write_ptr += write_len;
    }
}

```



```

        else
        {
            continue;
        }

        if(memcmp((void *)write_addr, pData, len) == 0)
        {
            MakeAnswer();
        }
    }
}

```

6.2.7 练习题

改写 tftp_test 例程，改变实验板 IP 地址，下载的时候改变 Flash 起始地址，重新进行实验，检查是否正确下载。

6.3 音频接口 IIS 实验

6.3.1 实验目的

- 掌握有关音频处理的基础知识。
- 通过实验了解 IIS 音频接口的工作原理。
- 通过实验掌握对处理器 S3C44B0 中 IIS 模块电路的控制方法。
- 通过实验掌握对常用 IIS 接口音频芯片的控制方法。

6.3.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

6.3.3 实验内容

编写程序播放一段由 wav 文件保存的录音。

6.3.4 实验原理

1. 数字音频基础

- 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的二进制数据再现原始声音，这个过程中使用的设备是模拟/数字转换器（Analog to Digital Converter，即 ADC），ADC 以每秒上万次的速率对声波进行采样，每次采样都记录下了原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（赫兹）。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保证了声音还原的质量。

人耳的听觉范围通常是 20Hz~20KHz，根据奈奎斯特（NYQUIST）采样定理，用两倍于一个正弦波的频率进行采样能够真实地还原该波形，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 16bit，44KHz，就是根据以上原理制定。

● 音频编码

脉冲编码调制 PCM（Pulse Code Modulation）编码的方法是对语音信号进行采样，然后对每个样值进行量化编码，在“采样频率和采样精度”中对语音量化和编码就是一个 PCM 编码过程。ITU-T 的 64kbit / s 语音编码标准 G.711 采用 PCM 编码方式，采样速率为 8KHz，每个样值用 8bit 非线性的 μ 律或 A 律进行编码，总速率为 64kbit / s。

CD 音频即是使用 PCM 编码格式，采样频率 44KHz，采样值使用 16bit 编码。

使用 PCM 编码的文件在 Windows 系统中保存的文件格式一般为大家熟悉的 wav 格式，实验中用到的就是一个采样 44.100KHz，16 位立体声文件 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 32kbit / s 语音编码标准 G.721 采用 ADPCM 编码方式，每个语音采样值相当于使用 4bit 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），RA（RealAudio），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. IIS 音频接口

IIS (Inter-IC Sound) 是一种串行总线设计技术，是 SONY、PHILIPS 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如便携 CD 机、数字音频处理器等。IIS 将音频数据和时钟信号分离，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。

IIS 总线仅处理音频数据，其它信号如控制信号等单独传送，基于减少引脚数目和布线简单的目的，IIS 总线只由三根串行线组成：时分复用的数据通道线，字选择线和时钟线。

- continuous serial clock (SCK);
- word select (WS);
- serial data (SD);

使用 IIS 技术设计的系统的连接配置参见图 6-15。

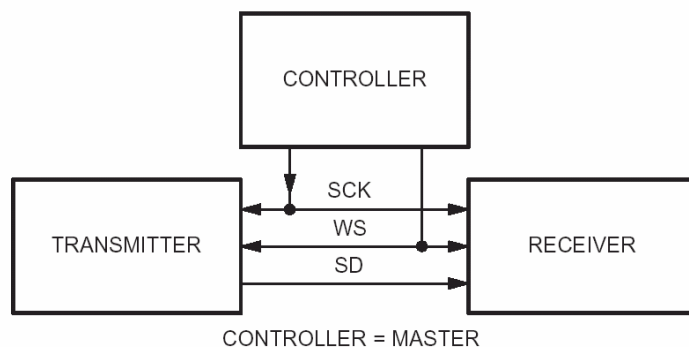


图 6-15 IIS 系统连接简单配置图

IIS 总线接口的基本时序参见图 6-16。

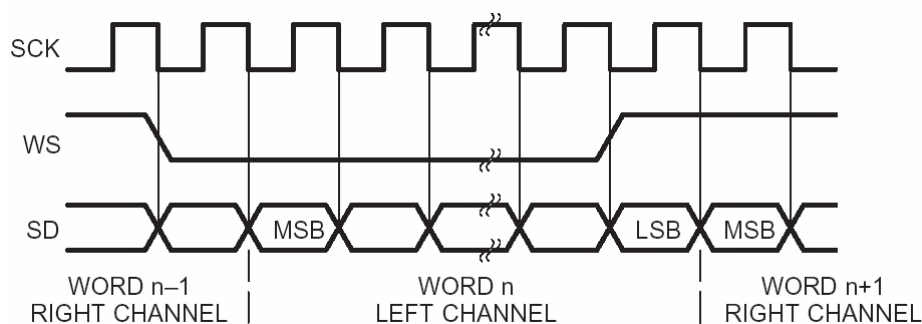


图 6-16 IIS 接口基本时序图

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，接收器和发送器将自动截取或填充。关于 IIS 总线的其它细节可参见 [《I2S bus specification》](#)。

在实验中，IIS 总线接口由处理器 S3C44B0 的 IIS 模块和音频芯片 UDA1341 硬件实现，我们关注的是正确的配置 IIS 模块和 UDA1341 芯片，音频数据的传输反而比较简单。

3. 电路设计原理

● S3C44B0 外围模块 IIS 说明

(1) 信号线

- 处理器中与 IIS 相关的信号线有五根：
- 串行数据输入 IISDI，对应 IIS 总线接口中的 SD 信号，方向为输入。
- 串行数据输出 IISDO，对应 IIS 总线接口中的 SD 信号，方向为输出。
- 左右通道选择 IISLRCK，对应 IIS 总线接口中的 WS 信号，即采样时钟。
- 串行位时钟 IISCLK，对应 IIS 总线接口中的 SCK 信号。

音频系统主时钟 CODECLK，一般为采样频率的 256 倍或 384 倍，符号为 256fs 或 384fs，其中 fs 为采样频率。CODECLK 通过处理器主时钟分频获得，可以通过在程序中设定分频寄存器获取，分频因子可以设为 1 到 16。CODECLK 与采样频率的对应表格见下表，实验中需要正确的选择 IISLRCK 和 CODECLK。

表 6-1 音频主时钟和采样频率的对应表

| IISLRCK (fs) | 8.000 KHz | 11.025 KHz | 16.000 KHz | 22.050 KHz | 32.000 KHz | 44.100 KHz | 48.000 KHz | 64.000 KHz | 88.200 KHz | 96.000 KHz |
|------------------|--------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| CODECLK (MHz) | 256fs | | | | | | | | | |
| | 2.0480 | 2.8224 | 4.0960 | 5.6448 | 8.1920 | 11.2896 | 12.2880 | 16.3840 | 22.5792 | 24.5760 |
| | 384fs | | | | | | | | | |
| | 3.0720 | 4.2336 | 6.1440 | 8.4672 | 12.2880 | 16.9344 | 18.4320 | 24.5760 | 33.8688 | 36.8640 |

需要注意的是，处理器主时钟可以通过配置锁相环寄存器进行调整，结合 CODECLK 的分频寄存器设置，可以获得所需要的 CODECLK。

(2) 寄存器

处理器中与 IIS 相关的寄存器有三个：

IIS 控制寄存器 IICON，通过该寄存器可以获取数据高速缓存 FIFO 的准备好状态，启动或停止发送和接收时的 DMA 请求，使能 IISLRCK、分频功能和 IIS 接口。

IIS 模式寄存器 IISMOD，该寄存器选择主/从、发送/接收模式，设置有效电平、通道数据位，选择 CODECLK 和 IISLRCK 频率。

IIS 分频寄存器 IISPSR。

(3) 数据传送

数据传送可以选择普通模式或者 DMA 模式，普通模式下，处理器根据 FIFO 的准备状态传送数据到 FIFO，处理器自动完成数据从 FIFO 到 IIS 总线的发送，FIFO 的准备状态通过 IIS 的 FIFO 控制寄存器 IISFCON 获取，数据直接写入 FIFO 寄存器 IISFIF。DMA 模式下，对 FIFO 的访问和控制完全由 DMA 控制器完成，DMA 控制器自动根据 FIFO 的状态发送或接收数据。

DMA 方式下数据的传送细节请参考处理器手册中 DMA 章节。

● 音频芯片 UDA1341TS 说明

电路中使用的音频芯片是 PHILIPS 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），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 和数码摄像机等。

UDA1341TS 提供两组音频输入信号线、一组音频信号输出线，一组 IIS 总线接口信号，一组 L3 总线。

IIS 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，当该芯片工作于微控制器输入模式使用的，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 S3C44BOX 中没有 L3 总线专用接口，电路中使用 I/O 口连接 L3 总线。L3 总线的接口时序和控制方式参见 UDA1341TS 手册。

● 电路连接

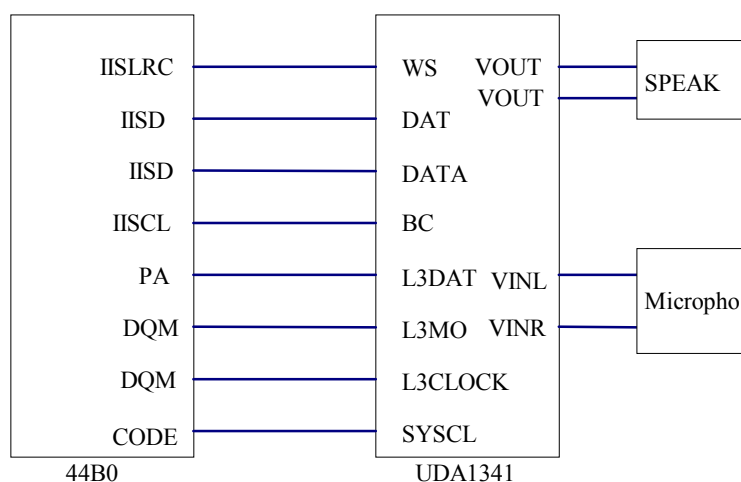


图 6-17 IIS 接口电路

6.3.5 实验步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接到目标板上，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上，打开实验例程目录 6.3_iis_test 子目录下的 iis_test.Uv2 例程，编译链接工程，直到链接工程成功；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（慎用！！！这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Run 或 F5 键全速运行程序，或者单步调试程序。

4. 观察实验结果

- 1). 在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...
IIS test example
Menu(press digital to select):
1: play wave file
2: record and play
```

- 2) 选择程序操作方式，选择 1 的话将会听到音乐，音乐播放完后会显示播放结束。

```
Play end!!!
```

- 3) 选择 2 的话将会进行录音，录音完成后，按任意键进行录音回放，播放完后，显示播放结束

```
Start recording....
End of record!!!
Press any key to play record data!!!
Play end!!!
```

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

6.3.6 实验参考程序

1. 环境及函数声明

```
/*-----*/
/*
                                function declare
*/
/*-----*/

void write_l3addr(UINT8T ucData);
void write_l3data(UINT8T ucData, int nHalt);
void init_1341(char cMode);
```

2. 初始化程序

```
/*-----*/
* name:      iis_init
* func:      Initialize IIS circuit
* para:      none
* ret:       none
* modify:
* comment:
*****/

void iis_init(void)
{
    rPCONE = (rPCONE&0xffff) | (2<<16);      // PE8:CODECLK
#ifdef S3CEV40
    rPCONC = (rPCONC&0xFFFFFFF0) | (0xFF); // PC0:IISLRCLK PC1:IISDO PC2:IISDI PC3:IISCLK
    rPCONF = (rPCONF&0x3ff);
#else
    rPCONC = (rPCONC&0xFFFFFFF0);
    rPCONF = (rPCONF&0x3ff) | (0x249000);    // PF5:IISLRCLK PF6:IISDO PF7:IISDI PF8:IISCLK
#endif
    f_nDMADone = 0;
    init_1341(PLAY);                        // initialize philips UDA1341 chip
}
*****

* name:      init_1341
* func:      Init philips 1341 chip
* para:      none
* ret:       none
* modify:
* comment:
*****/

void init_1341(char cMode)
{
    // Port Initialize
#ifdef S3CEV40
```

```

    rPCONA = (rPCONA&0x1ff);           // PA9:L3DATA
    rPCONB = (rPCONB&0x7CF);           // PB4:L3M, PB5:L3CLK
#else
    rPCONE = (rPCONE&0x303FF)|(0x25400); // PE7:L3MODE PE6:L3DATA PE5:L3CLK
#endif
    L3M_HIGH();                         // L3M=H(start condition),L3C=H(start condition)

    L3C_HIGH();

    write_l3addr(0x14+2);                // status (000101xx+10)
#ifdef FS441KHZ
    write_l3data(0x60,0);                // 0,1,10,000,0: reset,256fs,no DCfilter,iis
#else
    write_l3data(0x40,0);                // 0,1,00,000,0: reset,512fs,no DCfilter,iis
#endif

    write_l3addr(0x14+2);                // status (000101xx+10)
#ifdef FS441KHZ
    write_l3data(0x20,0);                // 0,0,10,000,0 no reset,256fs,no DCfilter,iis
#else
    write_l3data(0x00,0);                // 0,0,00,000,0 no reset,512fs,no DCfilter,iis
#endif

    // Set status register
    write_l3addr(0x14+2);                // status (000101xx+10)
    write_l3data(0x81,0);                // 1,0,0,0,0,0,11: OGS=0,IGS=0,ADC_NI,DAC_NI,sngl speed,AoffDon

    write_l3addr(0x14+0);                // data0 (000101xx+00)
    write_l3data(0x0A,0);

    // Record mode
    if(cMode)
    {
        write_l3addr(0x14+2);            // status (000101xx+10)
        write_l3data(0xa2,0);            // 1,0,1,0,0,0,10: OGS=0,IGS=1,ADC_NI,DAC_NI,sngl speed,AonDoff

        write_l3addr(0x14+0);            // data0 (000101xx+00)
        write_l3data(0xc2,0);            // 11000, 010 : DATA0, Extended addr(010)
        write_l3data(0x4d,0);            // 010, 011, 01 : DATA0, MS=9dB, Ch1=on Ch2=off,
    }
}

```

3. IIS 控制程序

```

/*****
* name:      iis_test
* func:      Test IIS circuit

```

```

* para:      none
* ret:       none
* modify:
* comment:
*****/

void iis_test(void)
{
    UINT8T uclnInput;

    iis_init();                                // initialize IIS

    uart_printf("Menu(press digital to select):\n");
    uart_printf("1: play wave file \n");
    uart_printf("2: record and play\n");
    do{
        uclnInput = uart_getch();
    }while((uclnInput != 0x31) && (uclnInput != 0x32));

    if(uclnInput == 0x31)

        iis_play_wave(1);

    if(uclnInput == 0x32)
        iis_record();

    iis_close();                                // close IIS
    uart_printf(" end.\n");
}
/*****
* name:      iis_play_wave
* func:      play wave data
* para:      nTimes  --  input, play times
* ret:       none
* modify:
* comment:
*****/

void iis_play_wave(int nTimes)
{
    unsigned char* pWavFile;
    int          nSoundLen;
    int          i;

    // enable interrupt
    rINTMOD = 0x0;
    rINTCON = 0x1;

```



```

// execute command "download t.wav 0xc300000" before running
//   pWavFile = (unsigned char*)0xC300000;
pWavFile = (unsigned char*)g_ucWave;

// initialize philips UDA1341 chip
init_1341(PLAY);

// set BDMA interrupt
pISR_BDMA0 = (unsigned)bdma0_done;
rINTMSK = ~(BIT_GLOBAL|BIT_BDMA0);

for(i=nTimes; i!=0; i--)
{
    // initialize variables
    f_nDMADone = 0;
    nSoundLen= 155956;

    rBDISRC0 = (1<<30)+(1<<28)+((int)(pWavFile));    // Half word,increment,pBufTmp
    rBDIDES0 = (1<<30)+(3<<28)+((int)rIISFIF);        // M2IO,Internal peripheral,IISFIF
    rBDICNT0 = (1<<30)+(1<<26)+(3<<22)+(0<<21)+(1<<20)+nSoundLen; //

IIS,Int,auto-reload,enable DMA
    rBDCON0  = 0x0<<2;

    // IIS Initialize
    rIISCON  = 0x22;                                // Tx DMA enable,Rx idle,prescaler
enable
    rIISMOD  = 0xC9;                                // Master,Tx,L-ch=low,iis,16bit
ch.,codeclk=256fs,Irck=32fs
    rIISPSR  = 0x22;                                // Prescaler_A/B enable, value=3
    rIISFCON = 0xF00;                                // Tx/Rx DMA,Tx/Rx FIFO

    rIISCON |= 0x1;                                  // enable IIS
    while( f_nDMADone == 0);                          // DMA end
    rIISCON = 0x0;                                    // IIS stop
    uart_printf(" Play end!!!\n");
}
}

/*****
* name:      iis_record
* func:      record wave
* para:      none
* ret:       none
* modify:
* comment:
*****/

```

```

void iis_record(void)
{

    unsigned char* pRecBuf;
    int          nSoundLen;
    int          i;

    // enable interrupt
    rINTMOD=0x0;
    rINTCON=0x1;

    //-----//
    //                      record                      //
    //-----//

    uart_printf("Start recording...\n");

    pRecBuf = (unsigned char *)0x0C400000; // for download
    for(i= (UINT32T)pRecBuf; i<((UINT32T)pRecBuf+REC_LEN+0x20000); i+=4)
    {
        *((volatile unsigned int*)i)=0x0;
    }

    init_1341(RECORD);

    // set BDMA interrupt
    f_nDMADone = 0;
    pISR_BDMA0 = (unsigned)bdma0_done;
    rINTMSK    = ~(BIT_GLOBAL|BIT_BDMA0);

    // BDMA0 Initialize
    rBDISRC0 = (1<<30)+(3<<28)+((int)rIISFIF);           // Half word,inc,pBufTmp
    rBDIDES0 = (2<<30)+(1<<28)+((int)pRecBuf);           // M2IO,fix,IISFIF
    rBDICNT0 = (1<<30)+(1<<26)+(3<<22)+(1<<21)+(1<<20)+REC_LEN;
    rBDCON0  = 0x0<<2;

    // IIS Initialize
    rIISCON = 0x1a;                                     // Rx DMA enable,Rx idle,prescaler
enable
    rIISMOD = 0x49;                                     //      Master,Tx,L-ch=low,iis,16bit
ch.,codeclk=256fs,lrcck=32fs
    rIISPSR = 0x22;                                     // Prescaler_A/B enable, value=3
    rIISFCON= 0x500;                                    // Tx/Rx DMA,Tx/Rx FIFO --> start
piling....
    rIISCON |=0x1;                                     // Rx start

    while(f_nDMADone == 0);                            // DMA end

```

```

    rINTMSK |= BIT_BDMA0;
    delay(10);
    rIISCON = 0x0; // IIS stop
    rBDICNT0 = 0x0; // BDMA stop

    uart_printf("End of record!!!\n");
    uart_printf("Press any key to play record data!!!\n");
    while(!uart_getch());

    //-----//
    //                play                //
    //-----//
    // initialize philips UDA1341 chip
    init_1341(PLAY);

    // set BDMA interrupt
    pISR_BDMA0 = (unsigned)bdma0_done;
    rINTMSK = ~(BIT_GLOBAL|BIT_BDMA0);

    // initialize variables
    f_nDMADone = 0;
    nSoundLen = REC_LEN;

    rBDISRC0 = (1<<30)+(1<<28)+((int)(pRecBuf)); // Half word,increment,pBufTmp
    rBDIDES0 = (1<<30)+(3<<28)+((int)rIISFIF); // M2IO,Internal peripheral,IISFIF
    rBDICNT0 = (1<<30)+(1<<26)+(3<<22)+(0<<21)+(1<<20)+nSoundLen; //
    IIS,Int,auto-reload,enable DMA
    rBDCON0 = 0x0<<2;

    // IIS Initialize
    rIISCON = 0x22; // Tx DMA enable,Rx idle,prescaler
enable
    rIISMOD = 0xC9; // Master,Tx,L-ch=low,iis,16bit
ch.,codeclk=256fs,lrc=32fs
    rIISPSR = 0x22; // Prescaler_A/B enable, value=3
    rIISFCON = 0xF00; // Tx/Rx DMA,Tx/Rx FIFO

    rIISCON |= 0x1; // enable IIS
    while(f_nDMADone == 0); // DMA end
    rIISCON = 0x0; // IIS stop

    uart_printf(" Play end!!!\n");
}

```

6.3.7 练习题

编写程序实现通过按钮调整音量的大小。

6.4 USB 接口实验

6.4.1 实验目的

- 了解 USB 接口基本原理。
- 掌握通过 USB 接口与 PC 通讯的编程技术。

6.4.2 实验设备

- 硬件: Embest EduKit-III 实验平台, ULINK USB-JTAG 仿真器套件, PC 机。
- 软件: μ Vision IDE for ARM 集成开发环境, Windows 98/2000/NT/XP。

6.4.3 实验内容

编写 USB 通信程序程序, 基于已有的 USB 驱动程序接口, 完成与 PC 端的 USB 测试程序之间的数据接收与发送。

6.4.4 实验原理

1. USB 基础

(1) 定义

通用串行总线协议 USB (Universal Serial Bus) 是由 Intel、Compaq、Microsoft 等公司联合提出的一种新的串行总线标准, 主要用于 PC 机与外围设备的互联。1994 年 11 月发布第一个草案, 1996 年 2 月发布第一个规范版本 1.0, 2000 年 4 月发布高速模式版本 2.0, 对应的设备传输速度也从 1.5Mb/s 的低速和 12Mb/s 的全速提高到如今的 480Mb/s 的高速。

其主要特点是:

支持即插即用。允许外设主机和其它外设工作时进行连接配置使用及移除。

传输速度快。USB 支持三种设备传输速率: 低速设备 1.5 Mb/s、中速设备 12 Mb/s 和高速设备 480 Mb/s。

连接方便。USB 可以通过串行连接或者使用集线器 Hub 连接 127 个 USB 设备, 从而以一个串行通道取代 PC 上其他 I/O 端口如串行口、并行口等, 使 PC 与外设之间的连接更容易。

独立供电。USB 接口提供了内置电源。

低成本。USB 使用一个 4 针插头作为标准插头, 通过这个标准插头, 采用菊花链形式可以把多达 127 个的 USB 外设连接起来, 所有的外设通过协议来共享 USB 的带宽。

(2) 组成

USB 规范中将 USB 分为五个部分: 控制器、控制器驱动程序、USB 芯片驱动程序、USB 设备以及针对不同 USB 设备的客户驱动程序。

控制器 (Host Controller), 主要负责执行由控制器驱动程序发出的命令, 如位于 PC 主板的 USB 控制芯片。

控制器驱动程序 (Host Controller Driver), 在控制器与 USB 设备之间建立通信信道, 一般由操作系统或控制器厂商提供。

USB 芯片驱动程序 (USB Driver), 提供对 USB 芯片的支持, 设备上的固件 (Firmware)。

USB 设备(USB Device)，包括与 PC 相连的 USB 外围设备。

设备驱动程序(Client Driver Software)，驱动 USB 设备的程序，一般由 USB 设备制造商提供。

(3) 传输方式

针对设备对系统资源需求的不同，在 USB 规范中规定了四种不同的数据传输方式：

同步传输 (Isochronous)，该方式用来联接需要连续传输数据，且对数据的正确性要求不高而对时间极为敏感的外部设备，如麦克风、喇叭以及电话等。同步传输方式以固定的传输速率，连续不断地在主机与 USB 设备之间传输数据，在传送数据发生错误时，USB 并不处理这些错误，而是继续传送新的数据。同步传输方式的发送方和接收方都必须保证传输速率的匹配，不然会造成数据的丢失。

中断传输 (Interrupt)，该方式用来传送数据量较小，但需要及时处理，以达到实时效果的设备，此方式主要用在偶然需要少量数据通信，但服务时间受限制的键盘、鼠标以及操纵杆等设备上。

控制传输 (Control)，该方式用来处理主机到 USB 设备的数据传输，包括设备控制指令、设备状态查询及确认命令，当 USB 设备收到这些数据和命令后，将依据先进先出的原则处理到达的数据。主要用于主机把命令传给设备、及设备把状态返回给主机。任何一个 USB 设备都必须支持一个与控制类型相对应的端点 0。

批量传输 (Bulk)，该方式不能保证传输的速率，但可保证数据的可靠性，当出现错误时，会要求发送方重发。通常打印机、扫描仪和数字相机以这种方式与主机联接。

(4) 关键词

USB 主机(Host)

USB 主机控制总线上所有的 USB 设备和所有集线器的数据通信过程，一个 USB 系统中只有一个 USB 主机，USB 主机检测 USB 设备的连接和断开、管理主机和设备之间的标准控制管道、管理主机和设备之间的数据流、收集设备的状态和统计总线的活动、控制和管理主机控制器与设备之间的电气接口，每一毫秒产生一帧数据，同时对总线上的错误进行管理和恢复。

USB 设备 (Device)

通过总线与 USB 主机相连的称为 USB 设备。USB 设备接收 USB 总线上的所有数据包，根据数据包的地址域来判断是否接收；接收后通过响应 USB 主机的数据包与 USB 主机进行数据传输。

端点 (Endpoint)

端点是位于 USB 设备中与 USB 主机进行通信的基本单元。每个设备允许有多个端点，主机只能通过端点与设备进行通讯，各个端点由设备地址和端点号确定在 USB 系统中唯一的地址。每个端点都包含一些属性：传输方式、总线访问频率、带宽、端点号、数据包的最大容量等。除控制端点 0 外的其他端点必须在设备配置后才能生效，控制端点 0 通常用于设备初始化参数。USB 芯片中，每个端点实际上就是一个一定大小的数据缓冲区。

管道 (Pipe)

管道是 USB 设备和 USB 主机之间数据通信的逻辑通道，一个 USB 管道对应一个设备端点，各端点通过自己的管道与主机通信。所有设备都支持对应端点 0 的控制管道，通过控制管道主机可以获取 USB 设备的信息，包括：设备类型、电源管理、配置、端点描述等。

2. USB 设备开发

USB 设备开发包括硬件电路设计和软件设计二部分内容，其中软件部分又包括 USB 芯片驱动程序和应用程序二部分。

USB 设备在硬件上通过 USB 芯片实现，USB 芯片负责：

管理和实现 USB 物理层差分信号；

通过配置和管理寄存器初始化设备。

提供连接的端点；

电源管理；

通过寄存器管理端点；

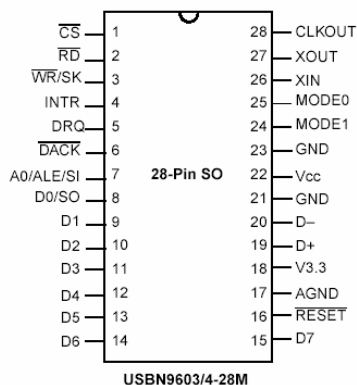
USB 芯片驱动程序基于以上硬件资源实现 USB 的功能。

USB 芯片提供多个标准的端点，每个端点都支持单一的总线传输方式。端点 0 支持控制传输，其他端点支持同步传输、批量传输或中断传输中的任意一种。管理和使用这些端点，实际上就是通过操作相应的控制寄存器、状态寄存器、中断寄存器和数据寄存器来实现。其中，控制寄存器用于设置端点的工作模式、启用端点的功能等；状态寄存器用于查询端点的当前状态；中断寄存器则用于设置端点的中断触发和响应功能；数据寄存器则是设备与主机交换数据用的缓冲区。

✧ 电路设计原理

Embest EduKit-III USB 接口模块采用美国国家半导体公司的 USBN9603 USB 控制器，该控制器是全速 USB 节点器件，完全兼容 USB1.0，USB1.1 通信规范。

USBN9603/4-28M 芯片引脚图如下：



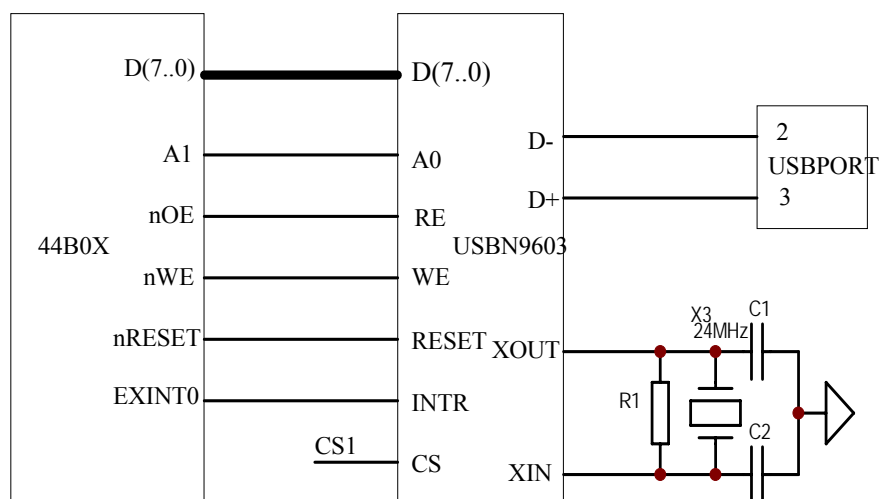
USBN9603 与 MCU 的接口模式分为两种：

- ◆ 8 位并行总线模式（Parallel Interface），使用并行总线方式时又可选择复用（Multiplexed）或非复用（Non-Multiplexed）模式，其中地址/数据线的复用方式电路设计稍显复杂。
- ◆ 微总线模式（MICROWIRE interface）。

以上模式的选择由管脚 MODE0，MODE1 决定。

在使用复用的 8 位并行总线模式下，USBN9603 支持与 MCU 之间的增强型 DMA 方式传输，使用 DMA 方式传输使 MCU 和 USBN9603 之间交换数据的速度成倍提高，最终可以显著提高 PC 与 USB 的通讯速度。

USBN9603 在 Embest EduKit-III 评估板与 CPU 连接图如下：



EduKit-III 的电路设计中采用的是非复用的 8 位并行总线模式，该模式中没有使用 DMA 方式，因此 DACK 接高电平。CPU 通过译码器生成的片选信号 CS1 对 USB 控制器进行选通，USBN9603 通过 EXINT1 对 CPU 发出中断请求。

✧ 设备驱动程序设计

USB 读写

Embest EduKit-III 的 USB 控制器 USBN9603 用户寄存器有两个，分别为只写的内部地址寄存器，与可读写的寄存器，内部地址寄存器的地址为 0x02000002，数据寄存器地址为 0x02000000。

对 USB 控制器进行读操作（包括读 USB 内部寄存器及数据）时，第一步是设置 USB 6bits 宽的内部地址寄存器，指明将从 USB 某个内部地址读一个字节，第二步是从数据寄存器读出 8bits 宽的数据。

对 USB 进行写操作类似读操作，第一步同样是设置 USB 的内部地址寄存器，指明将要写一个字节数据到 USB 内部某个地址中去。

USB 中断

Embest EduKit-III 的 USB 控制器中断请求引脚连接 S3C44B0X 外部中断引脚 EXINT1，对应的中断向量为 1，初始化 USB 中断的步骤是：

使 EINT1 中断使能。

安装 USB 中断服务程序入口到中断向量中去。

初始化 IO 端口 G 组控制器 PCONG，PUPG 指明 EXINT1 是作为中断输入引脚使用。

设置外部中断寄存器 EXTINT，指明触发中断方式。

初始化 USB

初始化 USB 需要使用 USB 读写函数对 USB 控制器内部的控制寄存器进行设置。需要设置的 USB 控制寄存器如下：

首先，通过设置主控制寄存器 MCNTRL 软件复位位（SRST），复位 USB 控制器。

设置主控制寄存器 MCNTRL，电压调整位（VGE），及中断输出（INTOC）位，以禁止中断输出。

写时钟寄存器 CCONF，设置 USB 控制器工作频率。

初始化功能地址寄存器 FAR (Function Address Register), 及 EPC0 寄存器 (Endpoint 0 Control Register), 端点号 0 为双向端点, 作控制使用。

设置中断掩码寄存器, 有主掩码寄存器(MAMSK), 无应答事件寄存器(NAKMSK), 发送事件寄存器(TXMSK), 接收事件寄存器(RXMSK), Alternate 事件寄存器(ALTMSK)。

最后允许 USB 控制器中信号输出, 使控制器附加到 USB 总线上。

USB 中断服务例程

中断服务程序处理 USB 控制器产生的中断, 它将数据从 USB 内部 FIFO 读出, 并建立正确的事件标志, 以通知主循环程序处理。

基本步骤如下:

从主事件寄存器 (MAEV) 读出产生中断的事件。

根据主事件寄存器某位状态判别事件, 接着读取相应的事件寄存器: 接收事件寄存器 (RXEV), 或发送事件寄存器 (TXEV), 或无应答事件寄存器 (NAKEV), 或 Alternate 事件寄存器 (ALTEV)。

进一步判别事件寄存器的某位状态, 根据具体事件, 分别做相应的操作。

通道 0(端点 0)用于控制传输, 在驱动程序中调用 rxevent0(), txevent0()处理端点 0 的事件。

通道 1 中由 Txevent1 () 处理端点 1 (单向发送) 的事件, rxevent1 () 处理端点 2 (单向接收) 的事件。

通道 2 中由 Txevent2 () 处理端点 3 (单向发送) 的事件, rxevent2 () 处理端点 4 (单向接收) 的事件。通道 3 中由 Txevent3 () 处理端点 5 的事件, rxevent3()处理端点 6 的事件。

6.4.5 实验操作步骤


1. 准备实验环境


使用 ULINK USB-JTAG 仿真器连接到目标板上, 使用 Embest EduKit-III 实验板自带的串口线, 连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序 (波特率 115200、1 位停止位、无校验位、无硬件流控制); 或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下 (如果已经拷贝, 可跳过此步骤);
- 2) 使用 ULINK USB-JTAG 仿真器连接到目标板上, 打开实验例程目录 6.4_usb_test 子目录下的 usb_test.Uv2 例程, 编译链接工程, 直到链接工程成功;
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境(工程默认已经配置正确), 点击工具栏 “

项或点击工具栏 “
 - 4) 如果需要将程序烧写固化到 Flash 中, 仅需要更改分散加载文件即可 (**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序, 建议实验中不操作**)。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件, 重新编译工程, 点击 MDK 的 Flash 菜单, 选择 Download 烧写调试代码到目标系统的 Nor Flash 中, 重启实验板, 实验板将会运行烧写到 Nor Flash 中的代码;

5) 安装 USB 设备驱动程序。

运行 USB 例程后，Windows 弹出发现新硬件的提示对话框，按照安装向导安装驱动程序 Embest EduKit-III USB Driver，驱动程序安装文件位于 6.4_usb_test\Driver 目录。

6) 运行 USB 数据传送演示软件。

运行位于 6.4_usb_test 目录下 demo.exe，出现如下窗口：



7) 发送和接收数据。

在传送数据窗口里输入要发送的数据，点发送按钮，在接收数据窗口里，显示目标板 USB 控制器返回的数据。

8) 在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...

USB9603 Test Example

Receive data...

Receive OK.
```

6.4.6 实验参考程序

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

【功能说明】9603 写控制,根据具体硬件结构修改

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

```
void write_usb(unsigned char addr,unsigned char dat)
```

```
{
```

```
  (*(volatile unsigned char *)0x02000002)=addr;
```

```

(* (volatile unsigned char *)0x02000000)=dat;
}
/*****

【功能说明】9603 读控制,根据具体硬件结构修改
*****/

unsigned char read_usb(unsigned char addr)
{
(* (volatile unsigned char *)0x02000002)=addr;
return (* (volatile unsigned char *)0x02000000);
}
/*****

【功能说明】中断初始化
*****/

void Isr_Init(void)
{
rINTMOD &= ~BIT_EINT1;           // EINT1 interrupt
rINTMSK &= ~BIT_GLOBAL;         // GLOBAL valid
pISR_EINT1 = (int)Eint1Isr;
rPCONG = rPCONG | (3 << 2);      // GPG1 设为中断引脚
rPUPG = rPUPG & 0xFD;           // GPG1 上拉电阻有效
rEXTINT=0x00;                   // 低电平触发
rINTMSK=rINTMSK|BIT_EINT1;      // 关外部中断
}
/*****

【功能说明】初始化 USBN9603
*****/

void Init_9603(void)
{
status_GETDESC=0;
usb_cfg=0;
/*give a soft reset, then set ints to push pull, active hi or lo*/
write_usb(MCNTL,SRST);
while(read_usb(MCNTL) & SRST);
write_usb(MCNTL,VGE+INT_H_P);
/*initialize the clock generator *****/
write_usb(CCONF,CODIS+0x0c);
/*set default address, enable EP0 only *****/
write_usb(FAR,AD_EN+0);          // FAR=FUNCTION ADDRESS REGISTER
write_usb(EPC0,0x00);
/*set up interrupt masks *****/
write_usb(NAKMSK,NAK_O0);        /*NAK evnts*/
write_usb(TXMSK,TXFIFO0+TXFIFO1+TXFIFO2+TXFIFO3); /*TX events*/
write_usb(RXMSK,RXFIFO0+RXFIFO1+RXFIFO2+RXFIFO3); /*RX events*/
write_usb(ALTMSK,SD3+RESET_A);   /*ALT evnts*/
write_usb(MAMSK,INTR_E+RX_EV+NAK+TX_EV+ALT);
/*enable the receiver and go operational *****/

```

```

    FLUSHTX0;                                /*flush TX0 and disable */
    write_usb(RXC0,RX_EN);                    /*enable the receiver */
    write_usb(NFSR,OPR_ST);                   /*go operational */
    write_usb(MCNTL,VGE+INT_L_P+NAT);        /*set NODE ATTACH */
    delay(100);
}
/*****
【功能说明】处理 USB 中断, 配置设备, 传输数据
*****/
void __irq Eint1Isr(void)
{
    rI_ISPC = BIT_EINT1;                      /* 清中断 */
    evnt=read_usb(MAEV);                      /*check the events */
    //Uart_Printf(0, "\n evnt=%x", evnt);
    if(evnt&RX_EV)
    {
        evnt=read_usb(RXEV);                  /*check the RX events */
        if (evnt&RXFIF00)
            rxevent_0();/*endpoint 0 */
        else if(evnt&RXFIF01)
            rxevent_1();/*endpoint 2 */
        else if(evnt&RXFIF02)
            rxevent_2();/*endpoint 4 */
        else if(evnt&RXFIF03)
            rxevent_3();/*endpoint 6 */
    }
    else if(evnt&TX_EV)
    {
        evnt=read_usb(TXEV);                  /*check the TX events */
        if (evnt&TXFIF00)
            txevent_0();/*endpoint 0 */
        else if(evnt&TXFIF01)
            txevent_1();/*endpoint 1 */
        else if(evnt&TXFIF02)
            txevent_2();/*endpoint 3 */
        else if(evnt&TXFIF03)
            txevent_3();/*endpoint 5 */
    }
    else if(evnt&ALT)
        usb_alt();                          /*alternate event? */
    /*NAKs can come so fast and furious (especially with OHCI hosts)*/
    /*that they MUST have a lower priority than the other events. If*/
    /*they did not, the other events could get starved out. */
    else if(evnt&NAK)
    {
        evnt=read_usb(NAKEV);                /*check the NAK events */
    }
}

```

```

        if (evnt&NAK_00) nak0();          /*endpoint 0          */
        //else if (evnt&NAK_01) onak1();  /*endpoint 2          */
        //else if (evnt&NAK_02) onak2();  /*endpoint 4          */
        //else if (evnt&NAK_I3) inak3();  /*endpoint 5          */
    }
}

```

```

/*****

```

【功能说明】 This subroutine handles TX events for FIFO3 (endpoint 5)

```

*****/

```

```

void txevent_3(void)

```

```

{
    txstat=read_usb(TXS3);          /*get transmitter status */
    //Uart_Printf(0,"\\n txstat=%x",txstat);
    if (txstat & ACK_STAT)
    {
        /*-----
        * previous data packet from current ep was received successfully by host
        *-----*/
        FLUSHTX3;
        //Flag = TRUE;
    }
    else
    {
        /*-----
        * there is no ACK
        * retransmit the previous data packet
        *-----*/
        if(dtapid_TGL3PID)
            write_usb(TXC3,TX_TOGL+TX_LAST+TX_EN+RFF);
        else write_usb(TXC3,TX_LAST+TX_EN+RFF);
        dtapid_TGL3PID=!dtapid_TGL3PID;
        // write_usb(TXC3,TX_LAST+TX_EN+RFF);
    }
}

```

```

/*****

```

【功能说明】 This subroutine handles RX events for FIFO3 (endpoint 6)

```

*****/

```

```

void rxevent_3(void)

```

```

{
    U8 i,bytes_count;
    // U8 j = 0;
    rxstat=read_usb(RXS3);          /*get receiver status */
    if(rxstat&SETUP_R)
    {
    }
}

```

```

else if(rxstat&RX_ERR)
{
    FLUSHRX3;                /*flush RX2 and disable */
    write_usb(RXC3,RX_EN);    /*re-enable the receiver */
}
else
{
    do{
        bytes_count = read_usb(RXS3)&0x0f;
        (*(volatile unsigned char *)0x02000002)=RXD3;
        for(i = 0; i< bytes_count; i++)
        {
            COMbuf[2][COMfront[2]++] = (*(volatile unsigned char *)0x02000000);
            if(COMfront[2] > 64)
            {
                COMfront[2] = 0;
                if(COMfront[2] == COMtail[2])
                {
                    COMtail[2]++;
                    if(COMtail[2] > 64)
                        COMtail[2] = 0;
                }
            }
        }
    }while(bytes_count == 0x0f);

    FLUSHRX3;                /*flush RX2 and disable */
    write_usb(RXC3,RX_EN);    /*re-enable the receiver */
}
}

```

6.4.7 实验练习题

改写系统主循环函数 Main(), 将接收的数据重发两遍回 PC 主机端, 观察 PC 端测试程序的接收数据窗口输出的变化。

第七章 基础应用实验

7.1 A/D 转换实验

7.1.1 实验目的

- 通过试验掌握模数转换（A/D）的原理。
- 了解模拟输入通道中采样保持的原理和作用。
- 掌握 S3C44B0X 处理器的 A/D 转换功能。

7.1.2 实验设备

- 硬件：Embest EduKit-III 实验平台，ULINK USB-JTAG 仿真器套件，PC 机。
- 软件：μVision IDE for ARM 集成开发环境，Windows 98/2000/NT/XP。

7.1.3 实验内容

- 了解采样保持器与 A/D 转换器的接口电路。
- 设计分压电路，利用 S3C44B0X 集成的 A/D 模块，把分压值转换为数字信号，并观察转换结果。

7.1.4 实验原理

1. A/D 转换器（ADC）

随着数字技术，特别是计算机技术的飞速发展与普及，在现代控制、通信及检测领域中，对信号的处理广泛采用了数字计算机技术。由于系统的实际处理对象往往都是一些模拟量（如温度、压力、位移、图像等），要使计算机或数字仪表能识别和处理这些信号，必须首先将这些模拟信号转换成数字信号，这就必须用到 A/D 转换器。

A/D 转换器的类型、工作原理和主要性能指标请参照触摸屏试验部分。

2. A/D 转换的一般步骤

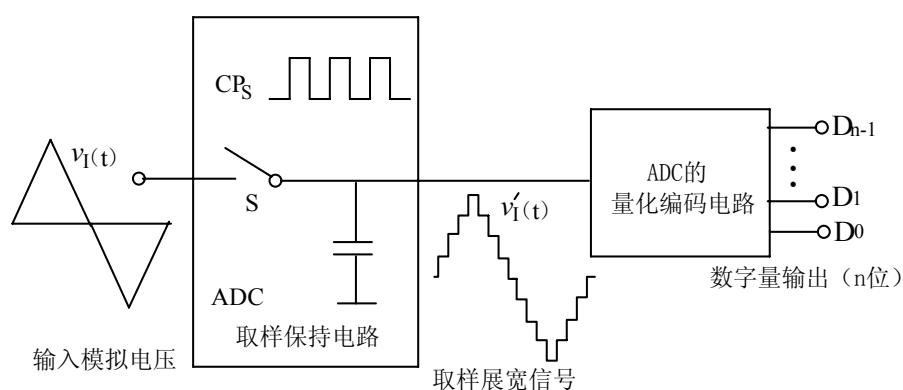


图 7-1 模拟量到数字量的转换过程

模拟信号进行 A/D 转换的时候，从启动转换到转换结束输出数字量，需要一定的转换时间，在这个转换时间内，模拟信号要基本保持不变。否则转换精度没有保证，特别当输入信号频率较高时，会造成很大的转换误差。要防止这中误差的产生，必须在 A/D 转换开始时将输入信号的电平保持住，而在 A/D 转换结束后，又能跟踪输入信号的变化。因此，一般的 A/D 转换过程是通过取样、保持、量化

和编码这四个步骤完成的。一般取样和保持主要由采样保持器来完成，而量化编码就由 A/D 转换器完成。

3. 采样保持器

1) 采样保持器原理

采样保持器是一种具有信号输入，信号输出的以及由外部指令控制的模拟门电路。主要由模拟开关 K、电容 C 和缓冲放大器 A 组成，一般结构如下：

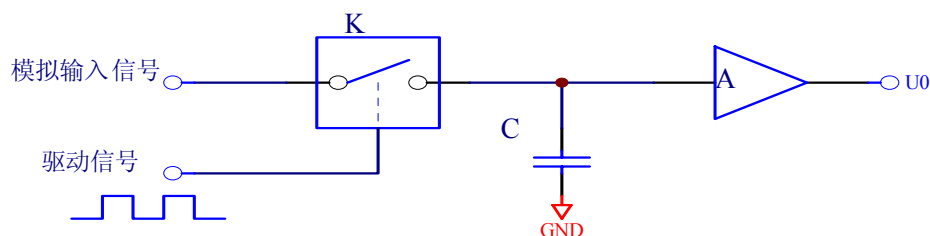


图 7-2 采样保持电路

采样保持器工作时有两种状态，分别是采样状态和保持状态，采样时输出跟随着输入变化，保持时输出不改变。工作过程如下图：

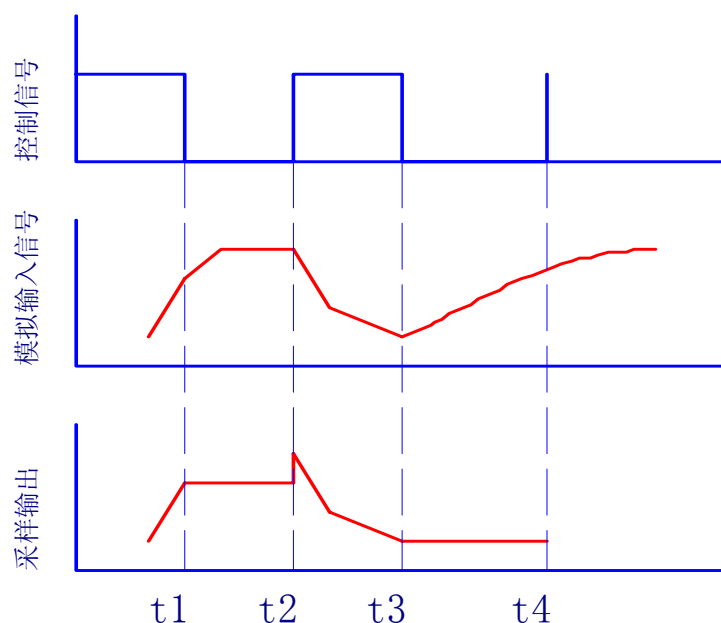


图 7-3 采样保持工作原理示意图

在 t_1 时刻之前，驱动信号为高电平，模拟开关 K 闭合，模拟输入信号给电容充电，电容上的电压跟随着输入电压的变化而变化，输出跟随输入变化，这个时期为采样期。过了 t_1 时刻，控制信号为低电平，模拟开关 K 打开，输入与输出断开，此时电容 C 上面的电压将保持 K 断开瞬间的电压，输出也将等于电容上的电压保持不变，这个时期为保持期。当控制信号的下一个高电平到来时，将会进入下一个采样阶段，然后是保持阶段。

在采样过程中为了正确无误的采样到输入信号，采样频率必须符合采样定律。即： $f_s \geq 2f_{i\max}$ 式中 f_s 为采样频率， $f_{i\max}$ 为输入信号 I 的最高频率分量的频率。

2) 采样保持器的性能参数

- 孔径时间 t_{AP}

孔径时间 t_{AP} 是指保持指令给出瞬间到模拟开关有效切断所经历的时间。由于孔径时间的存在，采样保持的输出值与希望的输出值有一定的误差，这个误差为孔径误差，如果保持指令与 A/D 开始转换指令同时发出，将因为孔径时间的存在，转换值将不是保持值。为了保证转换精度，最好在发出保持指令后延迟一段时间，等输出稳定以后再启动 A/D 转换模块。

- 孔径不定 Δt_{AP}

孔径不定 Δt_{AP} 是指孔径时间的变化，孔径时间只是采样时间的延迟，如果每次延迟相同的话，对总的采样结果精确性不会有影响，如改变孔径时间，则对精度有影响。

- 捕捉时间 t_{Ac}

捕捉时间是指当采样保持器从保持状态转到采样状态时，采样保持器的输出从保持状态的值变到当前的输入值所需的时间。它包括逻辑输入开关的动作时间、保持电容的充电时间、放大器的设定时间等。捕捉时间不影响采样精度，但对采样频率的提高有影响。

- 保持电压的下降

当采样 / 保持器处在保持状态时，由于保持电容器 C 的漏电流使保持电压值下降，下降值随保持时间增大而增加，所以，往往用保持电压的下降率来表示，即为

$$\frac{\Delta U}{\Delta T} = IC$$

式中 I 为保持容 C 的漏电流。

这里只介绍，采样保持器的几个重要参数，不过采样还有其他的一些参数，例如馈送、电荷转移偏差和跟踪到保持的偏差的参数。

目前采样保持器大多是集成在一块芯片上，芯片内部不包含保持电容，保持电容是由用户根据需要外接到芯片上。常用的采样保持器有 AD582、LF198 等。

3) AD 转换跟采样保持器的关系

在数据采集系统中，采样保持器用来对输入 A / D 转换器的模拟信号进行采集和保持，以确保 A / D 转换的精度。要保证 A / D 转换的精度，就必须确保 A / D 转换过程中输入的模拟信号的变化量不得大于 $LSB/2$ 。在数据采集系统中，如果模拟信号不经过采样保持器而直接输入 A / D 转换器，那么，系统允许该模拟信号的变化率就得降低。

一个 n 位的 A / D 转换器能表示的最大数字是 2^N ，设它的满量程电压为 FSR ，则它的“量化单位”或最小有效位 $LSB = FSR/2^N$ 。如果在转换时间 t_{CONV} 内，正弦信号电压的最大变化不超过 $LSB/2$ 所代表的电压，则在 $Um=FSR$ 条件下，数据采集系统可采集的最高信号频率为：

$$f_{\max} = \frac{1}{2^{n+1} \pi t_{conv}}$$

加采样 / 保持器后，这样就变成在 $\Delta t = t_{AP}$ 内，即在采样 / 保持器的孔径时间内讨论系统可采集模拟信号的最高频率。仍考虑对正弦信号采样，则在 n 位 A / D 转换器前加上采样 / 保持器后，系统可采集的信号最高频率为：

$$f_{\max} = \frac{1}{2^{n+1} \pi t_{AP}}$$

4. S3C44BOX 处理器的 A/D 转换

处理器内部集成了采用近似比较算法（计数式）的 8 路 10 位 ADC，集成零比较器，内部产生比较时钟信号；支持软件使能休眠模式，以减少电源损耗。其主要特性：

- 精度（Resolution）：10-bit
- 微分线性误差（Differential Linearity Error）：± 1 LSB
- 积分线性误差（Integral Linearity Error）：± 2 LSB (Max. ± 3 LSB)
- 最大转换速率（Maximum Conversion Rate）：100 KSPS
- 输入电压（Input voltage range）：0-2.5V
- 输入带宽（Input bandwidth）：0-100 Hz（无采保持电路 S/H(sample&hold)）
- 低功耗（Low Power Consumption）

● A/D 功能框图

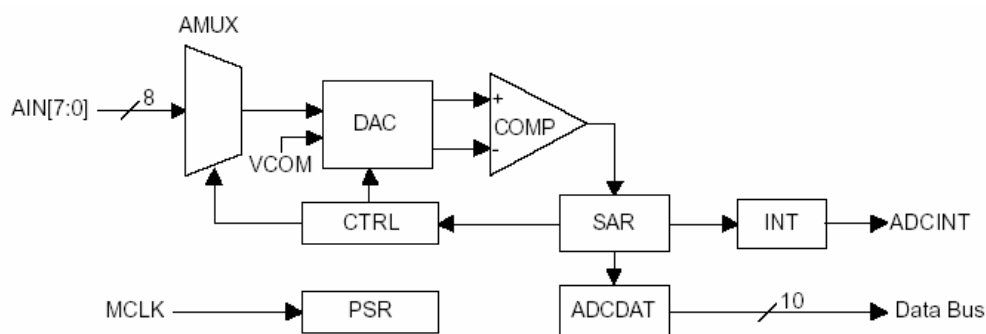


图 7-3 S3C44BOX 处理器的 ADC 框图

VCOM 是外部比较电压输入，实验系统中只需要接滤波电容到地，即比较范围为 5-0V。处理器的 AFREFB、AREFT 引脚亦需要接滤波电容到地处理。

● S3C44BOX 处理器 A/D 转换器的使用

1) 寄存器组

处理器集成的 ADC 只使用到三个寄存器，即 ADC 控制寄存器（ADCCON）、ADC 数据寄存器（ADCCON）、ADC 预装比例因子寄存器（ADCCON）。

ADC 控制寄存器（ADCCON）

| | 6 | 5 | 4:2 | 1 | 0 |
|--------|------|-------|-----------|-------------|--------------|
| ADCCON | FLAG | SLEEP | IN_SELECT | READ_ START | ENABLE_START |

FLAG ---- 0: A/D 转换正在进行；1: A/D 转换结束

SLEEP ---- 0: 正常状态；1: Sleep 模式

IN_SELECT ---- 选择转换通道[4:2]

000 = AIN0 001 = AIN1 010 = AIN2 011 = AIN3

100 = AIN4 101 = AIN5 110 = AIN6 111 = AIN7

READ_START ---- 0: 禁止 Start-by-read 1: 允许 Start-by-read

ENABLE_START ---- 0: A/D 转换器不工作 1: A/D 转换器开始工作

ADC 预装比例因子寄存器 (ADCPSR)

7:0

| | |
|--------|-----------|
| ADCPSR | PRESCALER |
|--------|-----------|

PRESCALER ---- 比例因子。该数据决定转换时间的长短，数据越大转换时间就越长

ADC 数据寄存器 (ADCDAT)

9:0

| | |
|--------|--------|
| ADCDAT | ADCDAT |
|--------|--------|

ADCPSR ---- A/D 转换数据值

2) A/D 转换的转换时间计算

例如系统时钟为 66MHz，PRESCALER=20；所有 10 位转换时间为：

$66 \text{ MHz} / 2(20+1) / 16 = 95.238 \text{ KHz} = 10.5 \text{ us}$ 16 是指 10 位转换所需最少周期数

3) A/D 转换器使用注意：

- ① ADC 的模拟信号输入通道没有采样保持电路，使用时可以设置较大的 ADCPSR 值，以减少输入通道因信号输出电阻过大而产生的信号电压。
- ② ADC 的转换频率在 0~100Hz
- ③ 通道切换时，应保证至少 15us 的间隔
- ④ ADC 从 SLEEP 模式退出，通道信号应保持 10ms 以使 ADC 参考电压稳定
- ⑤ Start-by-read 可使用 DMA 传送转换数据

7.1.5 实验设计

1. 电路设计

● 采样保持接口电路

在 S3C44B0X 中 A/D 模块有 8 个模拟输入通道，通道的切换可以由内部的定时器完成。如果要进行 8 个通道连续变化的信号的转换，还必须在 8 个通道全部加采样保持器，采样保持的接口电路如下图 7-5。模拟输入信号为需要转换的信号，驱动控制信号可以通过编程利用 ARM 里面的 timer 产生，也可以通过 I/O 口来控制，输出信号直接接到 A/D 模块中的输入通道。

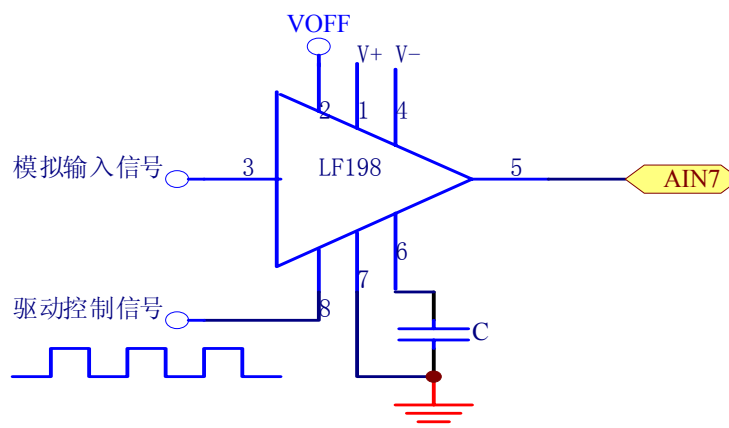


图 7-5 采样保持电路

● 分压电路

分压电路比较简单，为了保证电压转换时是稳定的，可以直接调节可变电阻得到稳定的电压值。

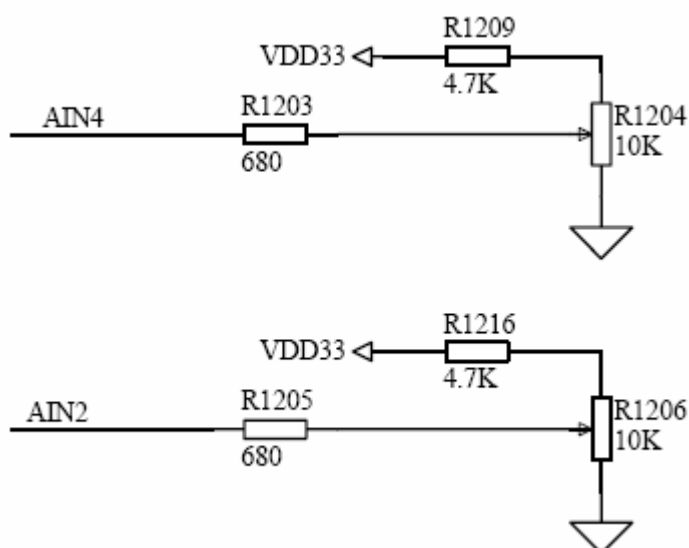


图 7-5 分压电路

2. 软件程序设计

试验主要是对 S3C44BOX 中的 A/D 模块进行操作，所以软件程序也主要是对 A/D 模块中的寄存器进行操作，其中包括对 ADC 控制寄存器（ADCCON）、ADC 数据寄存器（ADCDAT）和 ADC 预装比例因子寄存器（ADCPSR）的读写操作。同时为了观察转换结果，可以通过串口在超级终端里面观察。

7.1.6 实验操作步骤



1. 准备实验环境

使用 ULINK USB-JTAG 仿真器连接目标板，使用 Embest EduKit-III 实验板附带的串口线，连接实验板上的 UART0 和 PC 机的串口。

2. 串口接收设置

在 PC 机上运行 windows 自带的超级终端串口通信程序（波特率 115200、1 位停止位、无校验位、无硬件流控制）；或者使用其它串口通信程序。

3. 打开实验例程

- 1) 拷贝实验平台附带光盘 CD1\CD1_Basic_070615\Software 文件夹到 RealView MDK 软件的安装目录的 Keil\ARM\Boards\Embest\目录下（如果已经拷贝，可跳过此步骤）；
- 2) 使用 μ Vision IDE for ARM 通过 ULINK USB-JTAG 仿真器连接实验板，打开实验例程目录 7.1_adc_test 子目录下的 adc_test.Uv2 例程，编译链接工程；
- 3) 根据 ReadMe 目录下的 ReadMeCommon.txt 及 readme.txt 文件配置集成开发环境（工程默认已经配置正确），点击工具栏 “”，在 Option for Target 对话框的 Linker 页中选择 RuninRAM.sct 分散加载文件，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或点击工具栏 “”，下载工程生成的.axf 文件到目标板的 RAM 中调试运行；
- 4) 如果需要将程序烧写固化到 Flash 中，仅需要更改分散加载文件即可（**慎用!!! 这一步的操作将会破坏 Flash 中原有固化程序，建议实验中不操作**）。在 Option for Target 对话框的 Linker 页中选择 RuninFlash.sct 分散加载文件，重新编译工程，点击 MDK 的 Flash 菜单，选择 Download 烧写调试代码到目标系统的 Nor Flash 中，重启实验板，实验板将会运行烧写到 Nor Flash 中的代码；
- 5) 点击 Debug 菜单的 Go 或 F5 键运行程序。

4. 观察实验结果

- 1). 在 PC 机上观察超级终端程序主窗口，可以看到如下界面：

```
boot success...
S3C44B0x ADC Conversion Test Example
Input ADCPSR value (1 - 255):
```

- 2). 输入预装比例因子，该数据决定转换时间的长短，数据越大转换时间就越长，输入范围是 0~255，然后按回车键,将可以观察到试验结果，改变可调电阻的值将观察到不同的转换值。

```
Input ADCPSR value (1 - 255):20
ADC conv. freq.=95238Hz
Please press 'Enter' to convert
A2=0x3ff
```

5. 完成实验练习题

理解和掌握实验后，完成实验练习题。

7.1.7 实验参考程序

1. 环境及函数声明

```
/*-----*/
/*          function declare          */
/*-----*/
```

```
void adc_test (void);
```

2. 初始化程序

```

/*****
* name:      Main
* func:      c code entry
* para:      none
* ret:       none
* modify:
* comment:
*****/

void Main(void)
{
    sys_init();                // Initial 44B0X's Interrupt,Port and UART

    //user interface

    uart_printf("\n S3C44B0x ADC Conversion Test Example\n");

    adc_test();                // Test start
}

```

3. A/D 转换程序

```

/*****
* name:      adc_test
* func:      test adc
* para:      none
* ret:       none
* modify:
* comment:
*****/

void adc_test(void)
{
    int nAdcpsr;

    // Initialize adc
    rCLKCON = 0x7ff8;
    rADCCON = 0x1 | (0<<2);        // Enable ADC
    delay(100);                    // delay for 10ms for ADC reference voltage stabilization.

    //set ADCPSR value
    uart_printf( "Input ADCPSR value (1 - 255):" );
    nAdcpsr = uart_getintnum();
}

```

```
rADCPSR = nAdcpsr;
uart_printf( "ADC conversion freq. = %dHz\n", (int)(MCLK/(2.*(nAdcpsr+1.))/16.) );

//AIN2 AD conversion
uart_printf("\nPlease press 'Enter' to convert\n");
while( uart_getch() != '\n' )
{
    rADCCON = 0x1 | (0x2<<2);           // AIN2
    while( !(rADCCON&0x40) );           // wait  conversion finish
    uart_printf( "Ain2 = 0x%03x\n", rADCDAT );
}
}
```

7.1.8 练习题

参考试验程序，采用分压电路，对第四通道的输入电压进行循环采样，循环周期为 100ms，并求出采样数据的平均值。

附录 售后服务与技术支持

深圳市英蓓特信息技术有限公司承诺为我们的客户提供相关技术支持。如果您在使用我公司产品的时候，遇到任何问题，可以通过下列途径与我们客户服务部的技术支持工程师联系：

- **英蓓特公司网站**

关于英蓓特公司产品的最新最准确的信息（包括公司产品信息以及相关资料），您可以通过以下网址得到：<http://www.embedinfo.com>。

- **技术论坛**

英蓓特公司提供两个主力论坛供我们的广大客户以及业界的工程师相互交流和学习。

ARM开发论坛：

讨论 ARM 技术、ARM 系列芯片、ARM 开发工具、ARM 嵌入式处理器的开发、ARM 应用的论坛。

Embest IDE用户论坛：

Embest IDE 用户技术交流、技术支持的论坛。

- **邮件**

用户可以通过邮件地址 support@embedinfo.com 直接与我们的客户服务工程师联系。

- **电话**

用户可以在工作时间拨打我们的客户服务热线电话+**86-755-25631365/25635626**。

- **传真**

用户如有相关资料需要传真，您可以使用这个号码 **86-755-25616057**。

警告：

请务必注意静电的防护。超过任何最大承受值，均会对产品产生永久损害。同时，不推荐在临界状态使用产品。