高等学校计算机应用规划教材

嵌入式系统开发基础

——基于 ARM9 微处理器 C 语言程序设计(第二版)

侯殿有　编著

清华大学出版社

北　京

内 容 简 介

本书对 32 位精简指令系统嵌入式微处理器 S3C2410 的硬件系统和 C 语言驱动程序进行了详细的讲解，书中的源代码和实例程序对学习或从事嵌入式系统设计的读者都有很高的参考价值；另外在人机界面设计、系统初始化程序编写、仿真器设置和复杂工程项目构建等方面给出了简化做法，使初学者能够轻松、快速地掌握嵌入式系统设计方法。

本书以实用技术为主，内容通俗易懂，实例丰富，特别适合初学者和从事嵌入式系统设计工作的读者使用。

本书配套的电子课件、实验讲义、习题答案和部分工具软件可以到 http://www.tupwk.com. cn/downpage 网站下载。

图书在版编目(CIP)数据

嵌入式系统开发基础——基于 ARM9 微处理器 C 语言程序设计/侯殿有 编著. —2 版. —北京：清华大学出版社，2013.3

(高等学校计算机应用规划教材)

ISBN 978-7-302-31665-7

Ⅰ. ①嵌… Ⅱ. ①侯… Ⅲ. ①微型计算机—系统开发—高等学校—教材 ②C 语言—程序设计—高等学校—教材 Ⅳ. ①TP360.21 ②TP312

中国版本图书馆 CIP 数据核字(2013)第 042925 号

责任编辑：胡辰浩　袁建华
封面设计：牛静敏
版式设计：康　博
责任校对：蔡　娟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62796045

印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：** 17　　**字　　数：**392 千字
版　　次：2013 年 3 月第 1 版　　**印　　次：**2013 年 3 月第 1 次印刷
印　　数：1～4000
定　　价：30.00 元

产品编号：

第二版前言

嵌入式控制系统的教学现状

嵌入式控制系统的教学一般分为两个层次，首先完成以 MCS-51 为代表的 8 位单片机的教学，这在各个高校都获得了重视，大多数学校安排理论课 64 学时，实验课 32 学时，课时比较充足。在这个层次上，无论是讲授 C 语言程序设计或汇编语言程序设计，可供选择的教材都比较多。

第二个层次，也就是以 32 位 ARM 为代表的嵌入式控制系统的教学，许多学校都没有开设，这主要有以下 3 个原因：

一是缺乏师资。毕竟以 ARM 为代表的嵌入式控制系统设计是 20 世纪 90 年代才发展起来的新技术，它不仅包括高性能、功能丰富的硬件平台，而且软件开发的难度和嵌入式操作系统的应用，都对教师提出了更高的要求。

二是在课时安排上也有一定困难。这么复杂的软硬件系统，包括嵌入式操作系统，即使用 96(包括实验)学时，也不一定能讲深讲透。况且，整个教学计划中也没有这么多的时间。

第三个原因是没有合适的教材。特别是深入浅出、条理分明、适应本科生水平、课时比较合理的教材非常少。

为了克服上述困难，也为了满足教学需要，作者根据多年科研和教学经验编写了本书。

作者的想法是：在 32 位 ARM 为代表的嵌入式控制系统的教学中，不讲述带嵌入式操作系统的部分，而选择一种有代表性的 32 位单片机(类似 8 位机中的 MCS-51)，这里选择韩国三星 S3C2410 ARM9 单片机，在 ADS1.2 For Windows 集成开发环境中，用 C 语言完成嵌入式控制系统的开发工作。理论课内容安排 48 学时，实验课时间和内容由教师根据各校的时间和条件自行决定。

在 48 学时(16 周，每周三学时)内，集中将 S3C2410 的最基本硬件结构，软件资源学深学透，学会用 C 语言编写应用程序。在用 C 语言编写驱动程序时，尽量借助系统资源，参考例子程序、减少设计者的工作量。通过较短时间的学习，学生可以很快掌握嵌入式控制系统设计的方法，完成嵌入式控制系统的设计工作。

本书篇幅虽然不长，但程序源代码较多，对于从事嵌入式系统开发和学习来说是非常宝贵的资源，但是如果在课堂上讲解和分析这些代码，学时显然不够，建议教师主要讲解 S3C2410 的硬件资源和编程方法，具体程序代码留给学生课后慢慢消化理解。

教学实验平台介绍

有条件的学校，在完成理论课教学的同时，应安排一定的实验课，教学效果会更好。

作者接触的 ARM 9(SAMSUNG 2410)教学实验系统有：深圳英蓓特信息技术有限公司(http://www.embedinfo.com)的 Embest EDUKIT-Ⅱ/Ⅲ、北京博创科技集团(http://www.up-tech.com)的 UP-NETARM2410 教学实验系统、北京精仪达盛科技公司(http://www.techshine.com)

的 EL-ARM-830 教学实验系统，都有基于 ARM 9 系统资源的 C 语言实验程序例子，使用方便，可供选择。随书下载的实验讲义有两册：一是基于深圳英蓓特信息技术有限公司(http://www.embedinfo.com)的 Embest EDUKIT-Ⅱ/Ⅲ，实验时应配合 Embest EDUKIT-Ⅱ/Ⅲ教学实验系统平台，并安装 Embest IDE；二是基于北京精仪达盛科技公司(http://www.techshine.com)的 EL-ARM-830 教学实验系统，实验时应配合 EL-ARM-830 教学实验系统平台。两套实验系统程序的执行都要去掉目录中的中文目录并尽量缩短目录深度。

本书主要内容和教学方法，学习本书所需基础知识

第 1 章简单讲述嵌入式控制系统的定义、研究现状、研究方法。
第 2 章较详细地讲述基于 ARM 芯片的集成开发环境 ADS 1.2 的创建和使用。
第 3 章讲述 ARM 9 芯片 S3C2410 的片上资源和编程参考项目 2410test.mcp。
第 4 章讲述 S3C2410 的中断系统及编程。
第 5 章讲述 S3C2410 的 I/O 口和 I/O 口操作。
第 6 章讲述 S3C2410 的串口 URAT 及其编程。
第 7 章讲述 S3C2410 的 A/D 和 D/A 转换控制。
第 8 章讲述 ADC 和触摸屏控制。
第 9 章讲述 S3C2410 的实时时钟(RTC)和编程。
第 10 章讲述直接存储器存取(DMA)的工作原理及 S3C2410 的 DMA 控制器。
第 11 章讲述脉宽调制(PWM)的工作原理及 S3C2410 的 PWM 控制器。
第 12 章讲述看门狗(Watchdog)电路的工作原理及 S3C2410 的 Watchdog 控制。
第 13 章讲述双向二线制同步串行总线 I^2C 及 S3C2410 的 I^2C 控制电路。
第 14 章讲述数字音频信号(I^2S)和 S3C2410 的 I^2S 控制。
第 15 章对串行外设接口(SPI)进行了介绍。
第 16 章讲述 S3C2410 的人机界面设计。

以上各章内容除第 1~5 章外，其他各章内容基本独立，教师如果觉得在 48 学时内完成教学比较困难，除第 2、3、4、5 章和第 16 章做为重点建议必讲之外，其他各章可根据情况有选择地删节。

随书提供软件包一个，其中有本书的电子课件、S3C2410 使用手册、实验讲义、各章习题答案、ADS1.2、参考项目 2410test.mcp、通用字模提取程序和部分例子程序，可以在清华大学出版社网站(http://www.tupwk.com.cn/downpage)上免费下载。

第二版课件由孙颖馨老师在第一版基础上重新进行了制作，作者对她的工作表示感谢。

本书的特点是通过深入浅出的讲述，将基于 ARM 9 的嵌入式控制系统设计方法教给学生，使学生感到嵌入式控制系统设计简单易学，能够在最短的时间内入门。

学习本书时，学生至少要有 C 语言基础，如果有 MCS-51 单片机基础，学习本书就会更加轻松。

第二版与第一版区别

为了满足教学急需，第一版出书时间较紧，书中难免有错误或不足之处，在第二版中作者对书中内容进行了仔细斟琢研究，更正了已发现的错误。并根据多年教学经验和指导学生参加全国和省级“嵌入式”和“电子设计”大赛体会，删除了一些不适用的章节，增加了一部分新内容。

书中实验程序的注释是本书的重要内容，仔细阅读这些注释对于理解书中内容和练习编程非常重要。

为了使读者正确理解原程序，凡是原参考文献给出的注释，书中仍然保留英文，凡是作者给出的注释，用中文给出。

根据读者意见，为了使版面工整，方便阅读，注释采用分散方式对齐。

虽然做了很大努力，并请孙俊喜、才华两位教授对书中内容进行审核校对，但百密一疏，难免有考虑不周或错误之处，真诚欢迎读者多提宝贵意见和建议。我们的信箱是huchenhao@263.net，电话是010-62796045。

本书通用字模提取程序密码：194512125019。

侯殿有

2012年12月

目　　录

第1章　嵌入式控制系统简介

嵌入式控制系统设计是当前 IT 行业最热门的话题之一。什么是嵌入式控制系统？它们如何分类？各类系统的设计方法有什么不同？本章都将给出答案，并对嵌入式控制系统中最常用的 ARM 嵌入式微处理器做简要介绍。

1.1　单片机和嵌入式控制系统的定义和分类

在现有的不同文献中，嵌入式控制有着不同的定义，最常见的一种定义是：嵌入式系统是以应用为中心、以计算机技术为基础、软硬件可裁剪的，对功能、可靠性、成本、体积和功耗有严格要求的专用计算机系统。还有一种定义是：嵌入式系统就是一个具有特定功能或用途的计算机软硬件结合体。这些说法虽然在一定程度上对嵌入式进行了描述，但都不够全面或确切。

实际上，嵌入式控制系统是和单片机的产生和发展分不开的。本节将结合单片机对嵌入式系统进行定义，并对嵌入式控制系统的设计方法进行介绍。

1.1.1　单片机和嵌入式控制系统的定义

单片机就是在一片半导体硅片上集成了中央处理器单元(CPU)、存储器(RAM/ROM)和各种 I/O 接口的微型计算机。这样的一块集成电路芯片具有一台微型计算机的功能，因此被称为单片微型计算机，简称单片机。

单片机主要应用在测试和控制领域。由于单片机在使用时通常处于测试和控制领域的核心地位并嵌入其中，因此人们常把单片机称为嵌入式微控制器(Embedded Microcontroller Unit)，把嵌入某种微处理器或单片机的测试和控制系统称为嵌入式控制系统(Embedded Control System)。

嵌入式控制系统在航空航天、机械电子、家用电器等各个领域都有广泛的应用，其中家用电器领域是嵌入式控制系统最大的应用领域。MP3、MP4、数码相机、扫描仪、个人 PC、车载电视、DVD、PDA 等，到处都可以看到嵌入式控制系统的应用。

随着超大规模集成电路工艺和集成制造技术的不断完善，单片机的硬件集成度也在不断提高，已经出现了能满足各种不同需要、具有各种特殊功能的单片机。在 8 位单片机得到广泛应用的基础上，16 位单片机和 32 位单片机也应运而生，特别是以 ARM 技术为基础的 32 位精简指令集系统单片机(RISC Microprocessor)的出现，由于其性能优良、价格低廉，因此大有取代 16 位单片机而成为高档主流机型的趋势。

嵌入式控制系统由于其内核嵌入的微处理器不同，在应用上大致可分为两个层次，在系统简单、要求不高以及成本低的应用领域，大多采用以 MCS-51 为代表的 8 位单片机。

随着嵌入式控制系统与 Internet 的逐步结合，PDA、手机、路由器、调制解调器等复杂的高端应用，对嵌入式控制器提出了更高的要求，在少数高端应用领域以 ARM 技术为基础的 32 位精简指令系统单片机得到越来越多的青睐。嵌入式控制系统在高端应用领域又分为带嵌入式操作系统支持和不带嵌入式操作系统支持两种情况。

1.1.2　嵌入式控制系统的设计方法

作为嵌入式控制器的单片机，不管是 8 位、16 位还是 32 位，由于受其本身资源限制，其应用程序都不能在其自身上开发。开发其应用程序，都需要一台通用计算机，如常用的 IBM-PC 机或兼容机，Windows 95/98/2000 或 XP 操作系统，256MB 以上内存，1GB 以上硬盘存储空间(运行交叉编译环境 ADS1.2 最低配置)。这台通用计算机称为“宿主机”，嵌入式控制器的单片机称为“目标机”。应用程序在“宿主机”上开发，在“目标机”上运行。“目标机”和“宿主机”之间利用计算机并口通过一台叫“仿真器”的设备相连，程序可以从“宿主机”传到“目标机”，这也叫程序下载，也可以从“目标机”传到“宿主机”，这叫程序上传。应用程序通过“仿真器”的下载和上传，在“宿主机”上反复修改，这个过程叫做“调试”。调试好的应用程序，在“宿主机”上编译成“目标机”可以直接执行的机器码文件，通过一台叫“固化器”的设备下载并固化到“目标机”的程序存储器中，整个下载过程，称为烧片，也称为程序固化。

程序固化是单片机开发的最后一步，以后“宿主机”和“目标机”就可以分离，“宿主机”任务完成，“目标机”就可以独立执行嵌入式控制器的任务。

1.1.3　嵌入式控制系统各种设计方法的特点

1. 目标机上安装某种嵌入式操作系统

随着嵌入式系统的发展，应用程序变得越来越复杂，例如应用程序与 Internet 的结合、多线程、复杂的数据处理、高分辨率图形图像显示等，如果没有操作系统支持，应用程序的编写将变得非常困难。因此，人们在目标机上嵌入某种功能较强且占用内存较少的操作系统，用户程序在该操作系统支持下运行，这种操作系统叫做嵌入式操作系统。嵌入式操作系统有多种，如比较著名的 Windows CE、Linux、μC/OS-Ⅱ等。特别是 Linux 操作系统，由于其具有代码简练、功能强大、内核公开等优点，获得了广泛应用。

采用 Linux 操作系统来开发嵌入式系统，首先要在“宿主机”上建立 Linux 开发环境，这有两种作法：一是“宿主机”放弃原来的 Windows 操作系统，改装 Linux 操作系统，如安装 Linux Red Hat 9.0；二是在原来的 Windows 操作系统上安装一个虚拟机，在该虚拟机中安装 Linux 操作系统，如 Cygwin 1.5.10(可以从 http://www.cygwin.com 下载并安装最新版本)。

接着要根据应用程序的需要编写一个驱动程序，把该驱动程序和 Linux 操作系统一起编译，形成一个包含此驱动程序的 Linux 内核可执行文件 image，将此文件下载到“目标机”。今后，实现应用程序的功能，只需对内核中相应的函数进行调用即可。

由于“宿主机”和“目标机”之间文件的下载和上传是以文件形式进行的，所以在两个机器上都要有相应的文件管理系统，在“宿主机”上，可以使用 TFTP Server for Windows，

在“目标机”上则还要下载 Cramfs 文件管理系统。

为了实现上电时系统能自动按一定顺序启动，如系统的硬件初始化，包括时钟的设置、存储区的映射、设置堆栈指针、应用程序入口等，还必须有一个系统引导程序，即 Boot Loader，常用的 Boot Loader 是由韩国 Mizi 公司开发的 VIVI 软件，该软件特别适合 ARM 9 处理器，需要将 VIVI 下载到“目标机”上。

此外，还要为每一个项目的驱动程序和应用调试程序各编写一个工程管理文件 Makefile。

在 Linux 操作系统下，对应用程序和驱动程序的编辑和调试还需要一个交叉编译工具，要在 busybox 工具集中选择需要的部分进行编辑，形成可执行文件，下载到“目标机”上。

2. 目标机上不安装操作系统

在这种情况下，把 ARM9 只当成是 32 位单片机，使用 Code Warror IDE 对其进行开发，整个开发过程和开发 MCS-51 单片机一样，非常简单。

ADS(ARM Developer Suite)是 ARM 公司推出的新一代 ARM 开发工具，目前的最新版本是 ADS1.2。ADS 使用 Code Warror IDE 替代了旧的开发工具，使用 AXD 作为调试工具，现代集成开发环境的一些特点，如源文件编辑器、语法高亮和窗口驻留等功能都有体现。

ADS 使用并口通过 JTAG 仿真器与“目标机”相连，实现在线调试与仿真。

3. 两种设计方法的特点

带操作系统的嵌入式控制系统，在编制较复杂和高端应用程序时，例如上面提到的与 Internet 的结合、多线程、复杂的数据处理、高分辨率图形图像显示等，用户程序就会比较简单，但整个工程研制的时间开销不会少，因为要把很多时间放在对 Linux 操作系统的安装和熟悉上，虽然 Linux 操作系统是免费的，其内核可以根据用户需要进行剪裁，但要达到随意剪裁的水平，需要花费很多时间去熟悉和研究。此外，程序员还要学会驱动程序和 Makefile 文件的编写，特别是驱动程序，每一个设备都要有一个，它要和内核结合到一起，形成操作系统的一部分。也就是说，在开发嵌入式控制系统时，还要完成一部分操作系统的内核工作，难度较大，会花费很多时间。

系统在调试程序时，要占用“宿主机”较多资源，如使用并口连接 JTAG 仿真器、使用串口与“宿主机”通信、使用网口来传输文件。

如果在目标机上不安装嵌入式操作系统，把 ARM9 只当成是 32 位单片机来开发，那么，整个开发过程和开发 MCS-51 单片机一样，特别简单。这样就可以把主要时间放在对 ARM9 单片机软件和硬件的熟悉上，充分发挥 32 位单片机本身资源优势；把主要精力放在控制系统的稳定性和可靠性上，在较短时间内开发出高品质的嵌入式产品。

嵌入式控制系统大多具有小、巧、轻、灵、薄的特点，需要与 Internet 结合、多线程的系统等“高端应用”只占非常少的一部分，因此不采用嵌入式操作系统，也可以满足系统需要。

如果系统需要网络连接(连接 Internet 会使系统易受病毒攻击，导致系统稳定性下降，同时运行数据易泄密，因此工程上基本只使用局域网)，可以采用串行通信代替，点对点且距离不长，可以采用 232 标准，多点通信或距离较长，可以采用 485 标准。如遇多线程问题，可以采用多微处理器分级分布控制。

1.2　ARM 处理器简介

ARM 有 3 个含义：一是从事嵌入式微处理器开发的高科技公司的名字；二是代表一种低功耗、高性能的 32 位 RISC(精简指令系统)处理器的技术；三是代表一种微处理器产品。

本节介绍 ARM 微处理器系列的几种产品，从中可以看到 ARM 技术的发展和现状。

1.2.1　ARM 体系结构的发展

ARM 处理器是一种低功耗、高性能的 32 位 RISC(精简指令系统)处理器。本章将从其结构入手，分析目前流行的 ARM920T 硬件结构和编程。

ARM 处理器共有 31 个 32 位寄存器，其中 16 个可以在任何模式下看到。它的指令为简单的加载与存储指令(从内存加载某个值，执行完操作后再将其放回内存)。ARM 的特点有：所有的指令都带有条件；可以在加载数值的同时进行算术和移位操作。它可以在几种模式下操作，包括使用 SWI(软件中断)指令从用户模式进入系统模式。

ARM 处理器是一个综合体。ARM 公司自身并不制造微处理器，它们是由 ARM 的合作伙伴(Intel 或 LSI)制造的。ARM 还允许将其他处理器通过协处理器接口进行紧耦合。它还包括几种内存管理单元的变种，包括简单的内存保护到复杂的页面层次。

ARM 微处理器系列包括 ARM7、ARM9、ARM9E、ARM10E、SecurCore 等系列和 Intel 的 Xscale。其中，ARM7、ARM9、ARM9E 和 ARM10E 为 4 个通用处理器系列，每一个系列都提供一套相对独特的性能来满足不同应用领域的需求。SecurCore 系列专门为安全要求较高的应用而设计。

1. ARM7 系列微处理器

ARM7 系列微处理器是低功耗的 32 位 RISC 处理器，适用于对价位和功耗要求较高的消费类产品。ARM7 系列具有如下特点：

- 具有嵌入式 ICE—RT 逻辑，调试开发方便。
- 极低的功耗，适合对功耗要求较高的产品，如便携式产品。
- 能够提供 0.9 MIPS(MIPS，每秒百万条指令)/MHz 的三级流水线结构。
- 对操作系统的支持广泛，如 Windows CE、Linux、PalmOS(最流行的掌上电脑操作系统)等。
- 指令系统与 ARM9、ARM9E、ARM10E 系列兼容，便于用户的产品升级换代。
- 主频最高可达 130MHz，高速的运算处理能力可胜任绝大多数的复杂应用。

ARM7 系列微处理器主要应用于工业控制、Internet 设备、网络和调制解调器设备、移动电话等多种多媒体和嵌入式应用。

ARM7 系列微处理器包括如下几种类型的核：ARM7TDMI、ARM7TDMI-S、ARM720T、ARM7EJ。其中，ARM7TDMI 是目前使用最广泛的 32 位嵌入式 RISC 处理器，属低端 ARM 处理器核。TDMI 的基本含义如下：

- T 支持 16 位压缩指令集 Thumb

- D 支持片上 Debug
- M 内嵌硬件乘法器(Multiplier)
- I 嵌入式 ICE，支持片上断点和调试

2. ARM9 系列微处理器

ARM9 系列微处理器在高性能和低功耗方面有着非常突出的特点。具体如下：

- 5 级流水线结构，指令执行效率更高。
- 提供 1.1MIPS/MHz 的哈佛结构。
- 支持 32 位 ARM 指令集和 16 位 Thumb 指令集。
- 支持 32 位的高速 AMBA 总线接口。
- 全性能的 MMU，支持 WindowsCE、Linux、PalmOS 等多种主流嵌入式操作系统。
- MPU 支持实时操作系统。
- 支持数据 Cache(高速缓存)和指令 Cache，具有更高的指令和数据处理能力。

ARM9 系列微处理器主要应用于无线设备、仪器仪表、安全系统、机顶盒、高端打印机、数字照相机和数字摄像机等。

ARM9 系列微处理器包括 ARM920T、ARM922T 和 ARM940T 共 3 种类型，以适用于不同的应用场合。

3. ARM9E 系列微处理器

ARM9E 系列微处理器的主要特点如下：

- 支持 DSP 指令集，适用于需要高速数字信号处理的场合。
- 5 级流水线，指令执行效率更高。
- 支持 32 位 ARM 指令集和 16 位 Thumb 指令集。
- 支持 32 位的高速 AMBA 总线接口。
- 支持 VFP9 浮点处理协处理器。
- 全性能的 MMU，支持众多主流嵌入式操作系统。
- 支持数据 Cache 和指令 Cache，具有更高的处理能力。
- 主频最高可达 300MHz。

ARM9E 系列微处理器主要应用于下一代无线设备、数字消费品、成像设备、工业控制、存储设备和网络设备等领域。

ARM9E 系列微处理器包含 ARM926EJ-E、ARM946E-S 和 ARM966E-S 共 3 种类型，以适用于不同的应用场合。

4. ARM10E 系列微处理器

ARM10E 系列微处理器的主要特点如下：

- 支持 DSP 指令集，适用于需要高速数字信号处理的场合。
- 6 级流水线，指令执行效率更高。
- 支持 32 位 ARM 指令集和 16 位 Thumb 指令集。

- 支持 32 位的高速 AMBA 总线接口。
- 支持 VFP10 浮点处理协处理器。
- 全性能的 MMU，支持众多主流嵌入式操作系统。
- 支持数据 Cache 和指令 Cache，具有更高的处理能力。
- 主频最高可达 400MHz。
- 内嵌并行读/写操作部件。

ARM10E 系列微处理器主要应用于下一代无线设备、数字消费品、成像设备、工业控制、通信和信息系统等领域。

ARM10E 系列微处理器包括 ARM1020E、ARM1002E 和 ARM1026JE-S 共 3 种类型，以适用于不同的应用场合。

5. ARM920T

ARM920T 高缓存处理器是 ARM9 Thumb 系列中高性能的 32 位单片系统处理器。

ARM920TDMI 系列微处理器包含如下几种类型的内核。

- ARM9TDMI：只有内核。
- ARM940T：由内核、高速缓存和内存保护单元(MPU)组成。
- ARM920T：由内核、高速缓存和内存管理单元(MMU)组成。

ARM920T 提供完善的高性能 CPU 子系统，包括以下几个方面：

- ARM9TDMI RISC CPU。
- 16K 字节指令缓存与 16K 字节数据缓存。
- 指令与数据存储管理单元(MMU)。
- 写缓冲器。
- 高级微处理器总线架构(AMBA)总线接口。
- ETM(内置跟踪宏单元)接口。

ARM920T 中的 ARM9TDMI 内核可执行 32 位 ARM 及 16 位 Thumb 指令集。ARM9TDMI 处理器是哈佛结构，有包括取指、译码、执行、存储及写入的 5 级流水线。

ARM920T 处理器包括 CP14 和 CP15 两个协处理器。

- CP14：控制软件对调试通道的访问。
- CP15：系统控制处理器，提供 16 个额外寄存器来配置与控制缓存、MMU、系统保护、时钟模式以及其他系列选项。

ARM920T 处理器的主要特征如下：

- ARM9TDMI 内核，ARM v4T 架构。
- 两套指令集：ARM 高性能 32 位指令集和 Thumb 高代码密度 16 位指令集。
- 5 级流水线结构，即取指(F)、指令译码(D)、执行(E)、数据存储访问(M)和写寄存器(W)。
- 16K 字节数据缓存，16K 字节指令缓存。
- 写缓冲器：16 字的数据缓冲器。
- 标准的 ARMv4 存储器管理单元(MMU)：区域访问许可，允许以 1/4 页面大小对页面进行访问，16 个嵌入域，64 个输入指令 TLB 以及 64 个输入数据 TLB。

- 8 位、16 位、32 位的指令总线与数据总线。

6. SecurCore 系列微处理器

SecurCore(安全特性内核)系列微处理器除了具有 ARM 体系结构的各种主要特点外，在系统安全方面还具有如下特点：

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

SecurCore 系列微处理器主要应用于对安全性要求较高的产品及应用系统，如电子商务、电子政务、电子银行业务、网络和认证系统等领域。

SecurCore 系列微处理器包含 SecurCore SC100、SecurCore SC110、SecurCore SC200 和 SecurCour SC210 共 4 种类型，以适用于不同的应用场合。

7. Strong ARM 系列微处理器

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

8. ARM11 处理器的内核特点

ARM11 处理器是为了提高 MPU 处理能力而设计的。该系列主要有 ARM1136J，ARM1156T2 和 ARM1176JZ 共 3 个内核型号，RM11 处理器可以在 2.2mm^2 芯片面积和 0.24mW/MHz 下主频达到 500MHz。ARM11 处理器以众多消费产品市场为目标，推出了许多新技术，包括针对媒体处理的 SIMD(单指令多数据流)，用于提高安全性能的 TrustZone(安全区)技术，智能能源管理(IEM)，以及需要非常高的、可升级的、超过 2600 次 Dhrystone(逻辑运算性能测试)和 2.1 MIPS 的多处理技术。

上面对几个 ARM 处理器内核做了简单介绍。可以看到，随着处理器内核技术的发展，处理器的速度越来越快，其主要得益于 ARM 流水线的技术发展。

ARM1176JZF-S 可综合处理包括数字电视、机顶盒、游戏机以及手机在内的消费及无线产品。这一处理器采用了 ARM Jazelle® (Java 加速技术)、ARM TrustZone®技术(专门针对开放式操作系统例如 Symbian OS、Linux 和 Windows CE 的消费产品提供安全性能的关键技术)以及一个矢量浮点(VFP)协处理器(为嵌入式 3D 图像提供强大的加速功能)。

9. DSP 功能

DSP(digital signal processor，数字信号处理)是一种独特的微处理器，是以数字信号来处理大量信息的器件。其工作原理是接收模拟信号，并将其转换为 0 或 1 的数字信号，再对数字信号进行修改、删除、强化，并在其他系统芯片中把数字数据解译回模拟数据或实际环境格式。它不仅具有可编程性，而且其实时运行速度可以达每秒数以千万条复杂指令程序，远

远超过通用微处理器，是数字化电子世界中日益重要的计算机芯片。

目前，有很多应用要求多处理器的配置(多个 ARM 内核，或 ARM+DSP 的组合)，ARM11 处理器从设计开始就注重更容易地与其他处理器共享数据，以及从非 ARM 的处理器上移植软件。此外，ARM 还开发了基于 ARM11 系列的多处理器系统——MPCORE(由 2~4 个 ARM11 内核组成)。

1.2.2　ARM 体系结构的存储器格式

ARM 体系结构中字长的概念如下：

- 字(Word)，在 ARM 体系结构中，字的长度为 32 位，而在 8 /16 位处理器体系结构中，字的长度一般为 16 位。
- 半字(Half Word)，在 ARM 体系结构中，半字的长度为 16 位，与 8 /16 位处理器体系结构中字的长度一致。
- 字节(Byte)，在 ARM 体系结构和 8 /16 位处理器体系结构中，字节的长度均为 8 位。

指令长度可以是 32 位(ARM 状态下)，也可以是 16 位(Thumb 状态下)。

ARM920T 支持字节(8 位)、半字(16 位)、字(32 位)3 种数据类型，其中，字需要 4 字节对齐，半字需要 2 字节对齐。

ARM920T 体系结构将存储器看成是从零地址开始的字节的线性组合。从 0 字节到 3 字节放置第 1 个存储的字数据，从第 4 个字节到第 7 个字节放置第 2 个存储的字数据，依次排列。

作为 32 位的微处理器，ARM920T 体系结构所支持的最大寻址空间为 4GB(2^{32} 字节)。

ARM920T 体系结构支持两种方法存储字数据：即大端(Big Endian)格式和小端(Little Endian)格式。在大端格式中，字数据的高字节存储在低字节单元中，而字数据的低字节则存放在高地址单元中，如图 1-1 所示。在小端存储格式中，低地址单元存放的是字数据的低字节，高地址单元中，存放的是数据的高字节，如图 1-2 所示。

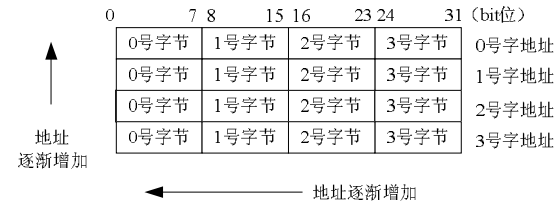

图 1-1　大端格式存储字数据

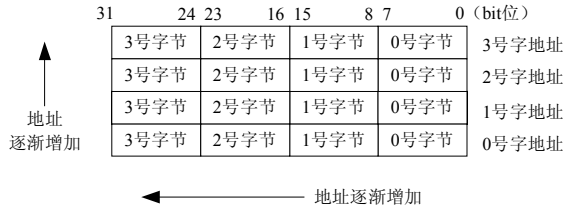

图 1-2　小端格式存储字数据

在基于 ARM920T 内核的嵌入式系统中，常用小端存储格式来存储字数据。

1.3　习题

1. 简述嵌入式控制系统的定义、嵌入式控制系统的分类。
2. 简述嵌入式控制系统各种设计方法的特点。
3. ARM 体系结构中的字、半字、字节的长度各是多少？
4. ARM 系统产品有几大类？每类的特点和应用场合分别是什么？
5. ARM 状态下指令长度是多少位？Thumb 状态下指令长度是多少位？
6. 什么是大端模式？什么是小端模式？在 ARM920T 内核的系统中，常采用哪种模式？

第2章　ADS1.2开发环境创建与简介

上一章讲过，在进行嵌入式控制系统的开发之前，必须要创建一个开发环境。之前在学习 MCS-51 单片机时使用的 Keil-C51 或 WEVE 6000 就是 51 单片机的开发环境。本章将主要介绍如何创建 ARM 单片机的开发环境 ADS1.2，以使读者掌握开发环境的使用、程序的调试、程序的固化和 ARM C 语言的基本规则。

2.1　ADS1.2 开发环境创建

本节介绍 ADS1.2 的技术特点以及安装 ADS1.2 的操作。

2.1.1　ADS1.2 概述

上一章讲过，作为嵌入式控制器的单片机，由于受其本身资源限制，其应用程序都不能在其自身上开发。开发其应用程序需要一台通用计算机，这台通用计算机称为“宿主机”，在“宿主机”上要安装有集成开发环境。

ADS，全称为 ARM developer Suite，就是 ARM 集成开发环境，它主要包括编译器、链接器、调试器、C 和 C++库等，是 ARM 公司推出的新一代 ARM 集成开发工具。其最新版本是 ADS1.2，该版本支持包括 Windows 和 Linux 在内的多种操作环境。ADS1.2 的组成如下。

1. 编译器

ADS 提供多种编译器，以支持 ARM 和 Thumb(在 ARM 体系中数据和指令采用 16 位字长)指令的编译。主要有以下 5 种编译器

- armcc：是 ARM C 编译器。
- tcc：是 Thumb C 编译器。
- armcpp：是 ARM C++编译器。
- tcpp：是 Thumb C++编译器。
- arm asm：是 ARM 和 Thumb 的汇编语言编译器。

2. 链接器

armlink 是 ARM 链接器。该命令既可以将编译得到的一个或多个目标文件和相关的一个或多个库文件进行链接，生成一个可执行文件，也可以将多个目标文件链接成一个目标文件，以供进一步的链接。

3. 符号调试器

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

4. fromELF

将 ELF 格式的文件转换为各种格式的输出文件，包括 BIN(二进制)格式映像文件、Motorola 32 位 S 格式映像文件、Intel 32 位格式映像文件和 Verilog 十六进制文件。FromELF 命令也能够为输入映像文件产生文本信息，例如代码和数据长度。

5. armar

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

6. CodeWarrior

CodeWarrior 集成开发环境(IDE)为管理和开发项目提供了简单多样化的图形用户界面，用户可以使用 ADS 的 CodeWarriorIDE 为 ARM 和 Thumb 处理器开发用 C、C++或者 ARM 汇编语言编写的程序代码。后续章节会经常使用 CodeWarrior 集成开发环境(IDE)来开发 C 语言程序。

7. 调试器

ADS 中含有 3 个调试器，即 AXD、Armsd 和 ADW/ADU。

在 ARM 体系中，可以选择多种调试方式，如 Multi-ICE(Multi-processor In-Circuit Emulator)、ARMulator 或 Angel。

- Multi-ICE 是一个独立的产品，是 ARM 公司自己的 JTAG 在线仿真器，而非 ADS 提供。
- ARMulator 是一个 ARM 指令集仿真器，集成在 ARM 的调试器 AXD 中，提供对 ARM 处理器的指令集的仿真，为 ARM 和 Thumb 提供精确的模拟。用户可以在硬件尚未做好的情况下开发程序代码，利用模拟器方式进行调试。
- Angel 是 ARM 公司常驻在目标机 Flash 中的监控程序，只需通过 RS-232C 串口与 PC 主机相连，就可以对基于 ARM 架构处理器的目标机进行监控器方式的调试。

8. C 和 C++库

ADS 提供了 ANSI C 库函数和 C++库函数，支持被编译的 C 和 C++代码。用户可以把 C 库中的与目标相关的函数作为自己应用程序中的一部分，重新进行代码的实现。这就为用户带来了极大的方便。

2.1.2　ADS1.2 的安装

在 ADS1.2 的安装盘中运行 setup.exe，安装 ARM Developer Suite v1.2，出现如图 2-1 和图 2-2 所示的对话框，同意许可协议，选择默认安装路径(C:\Program Files\ARM\vADS1.2)和

典型安装模式，单击 Next 按钮进入下一步，一直单击 Next 按钮，开始安装，如图 2-3 所示。

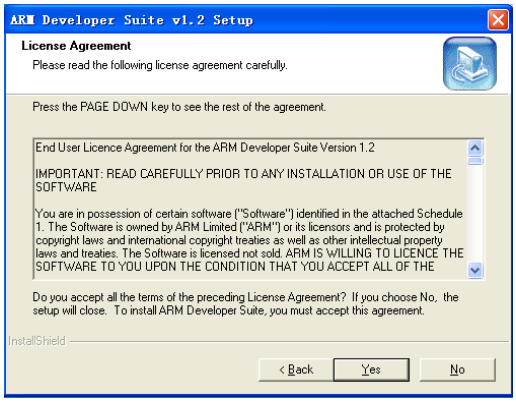

图 2-1　同意许可协议

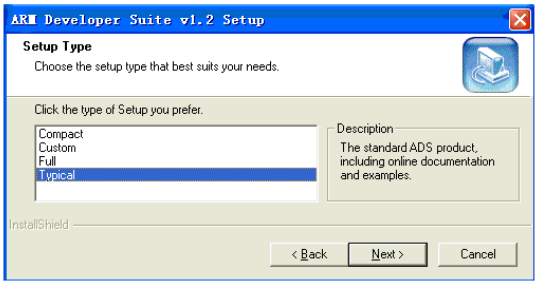

图 2-2　选择安装类型

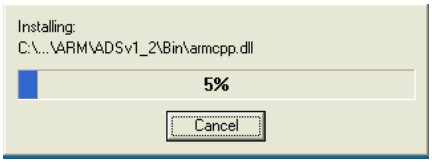

图 2-3　开始安装

安装结束，安装许可文件(Install License)，这一步可根据安装向导进行，单击“下一步”按钮，会出现如图 2-4 和图 2-5 所示的对话框。

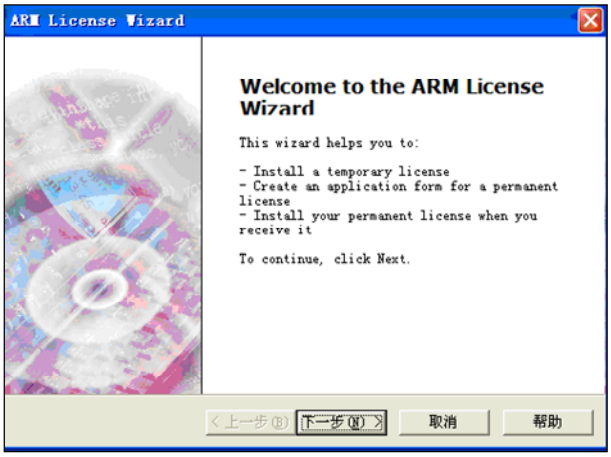

图 2-4　根据安装向导安装许可文件

在图 2-5 所示的对话框中单击 Browse 按钮查看许可文件，在 C:\Program Files\ARM\ADSV1_2\license\中选择 license.dat 文件并打开，单击“下一步”按钮，如图 2-6 所示，即可完成 ADS1.2 的安装。

最后，程序还要注册，注册文件在 C:\Program Files\ARM\ADSV1_2 文件夹中，单击注册文件，即完成程序注册，如图 2-7 所示。

安装并注册成功后，CodeWarrior 集成开发环境(IDE)就可以使用了。为了方便，可以在

桌面上创建一个快捷方式，在 C:\Program File\ARM\ADSv1-2\Bin 文件夹中有一个快捷方式图标，如图 2-8 所示，将其发送到桌面即可。

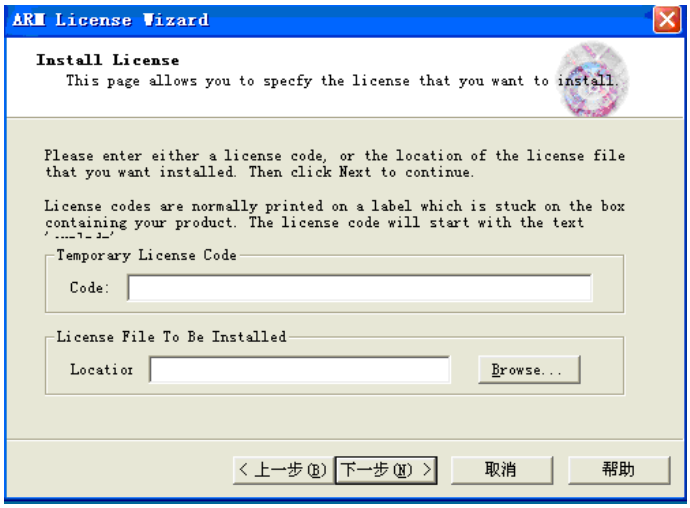

图 2-5　浏览许可文件

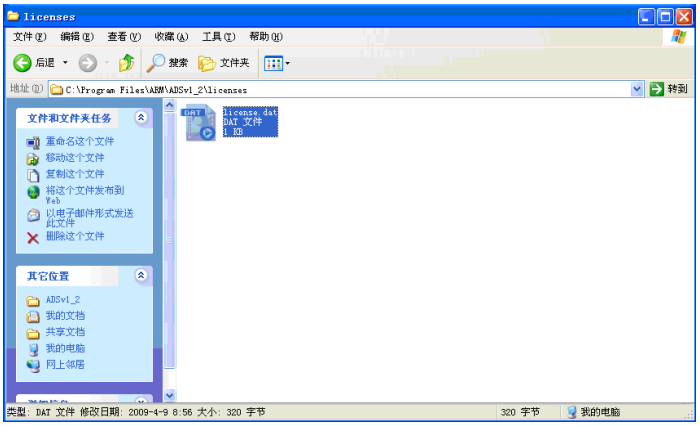

图 2-6　选择许可文件

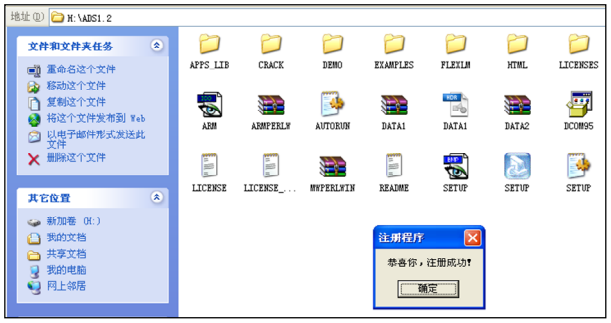

图 2-7　程序注册

图 2-8　IDE 快捷方式

打开计算机，双击 IDE 快捷方式图标，即可进入 CodeWarrior 集成开发环境，然后打开一个例子项目 bmw.mcp，如图 2-9 所示。

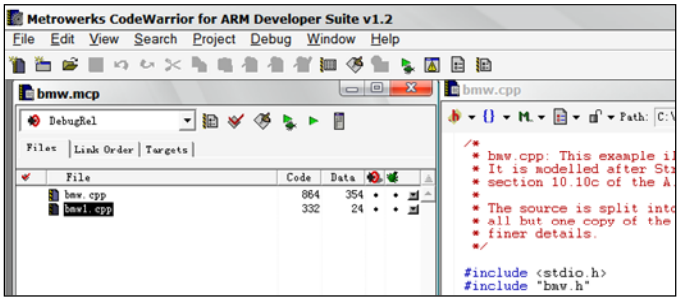

图 2-9　CodeWarrior 集成开发环境

2.2　ADS 集成开发环境的使用

与 MCS-51 单片机的开发环境 KeilC 一样，ADS 对用户的程序进行项目管理，一个 ADS 项目中可以包括汇编语言程序、C/C++语言程序、C 语言头文件、库文件等，这些文件还可以以文件夹的形式加入项目，本节将介绍 ADS 集成开发环境的使用。

2.2.1　建立一个新工程

运行 ADS1.2 集成开发环境(CodeWarrior for ARM Developer Suite)，选择 File|New 命令，打开 New 对话框，在 New 对话框中，打开 Project 选项卡，其中共有 7 项，ARM Executable Image 是 ARM 的通用模板，选中它即可生成 ARM 的执行文件，如图 2-10 所示。

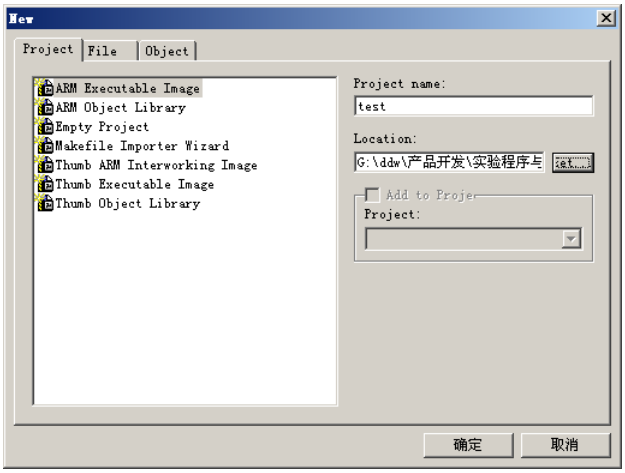

图 2-10　建立一个新工程

在 Project name 文本框中输入项目的名称，在 Location 文本框中输入其存放的位置，单击“确定”按钮保存项目。系统会在项目名称后面自动加上 ADS 项目后缀.mcp 后保存。

2.2.2　开发环境设置

(1) 在新建的工程中选择 Debug 版本，如图 2-11 所示，选择 Edit|Debug Settings 命令对

Debug 版本进行参数设置。

(2) 在如图 2-12 所示中，单击 Debug Setting 按钮，打开如图 2-13 所示对话框，选中 Target Settings 选项，在 Post-linker 下拉列表中选择 ARM fromELF 项，单击 OK 按钮。

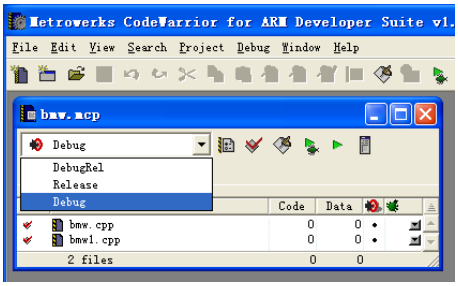

图 2-11　选择 Debug 版本

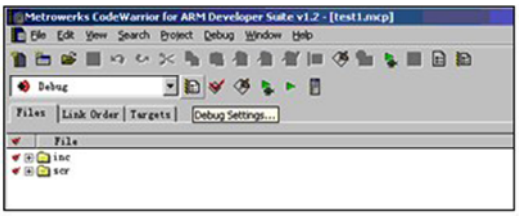

图 2-12　选择 Debug Setting

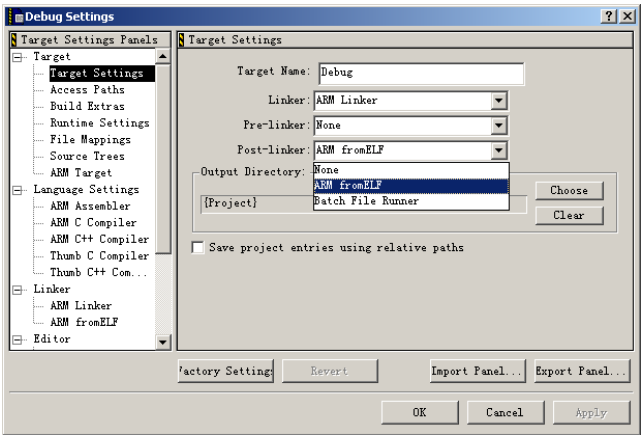

图 2-13　选择 Target Settings

(3) 在如图 2-14 所示的对话框中，单击 ARM Assembler，在 Architecture or Processor 下拉列表中选择 ARM920T。这是项目选择的 CPU 类型。

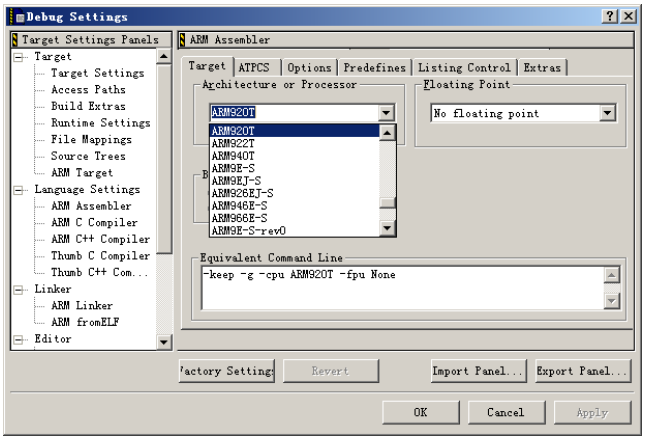

图 2-14　选择要编译的 CPU 类型

(4) 在如图 2-15 所示中，单击 ARM C Compliler，在 Architecture or Processor 下拉列表中也选择 ARM920T。这是 C 语言要编译的 CPU 核。

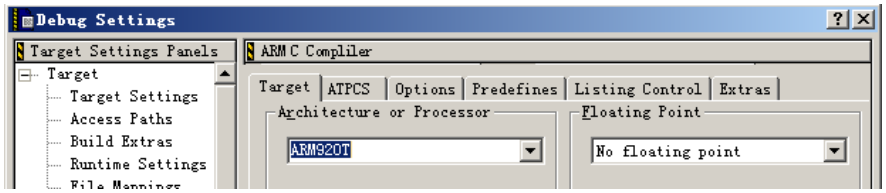

图 2-15　选择要编译的 CPU 核

(5) 在如图 2-16 所示中，单击 ARM Linker，在 Output 选项卡中设定程序的代码段地址，以及数据使用的地址。在 RO Base 文本框中输入程序代码存放的起始地址，在 RW Base 文本框中输入程序数据存放的起始地址。RW Base 地址必须是 SDRAM 的地址。

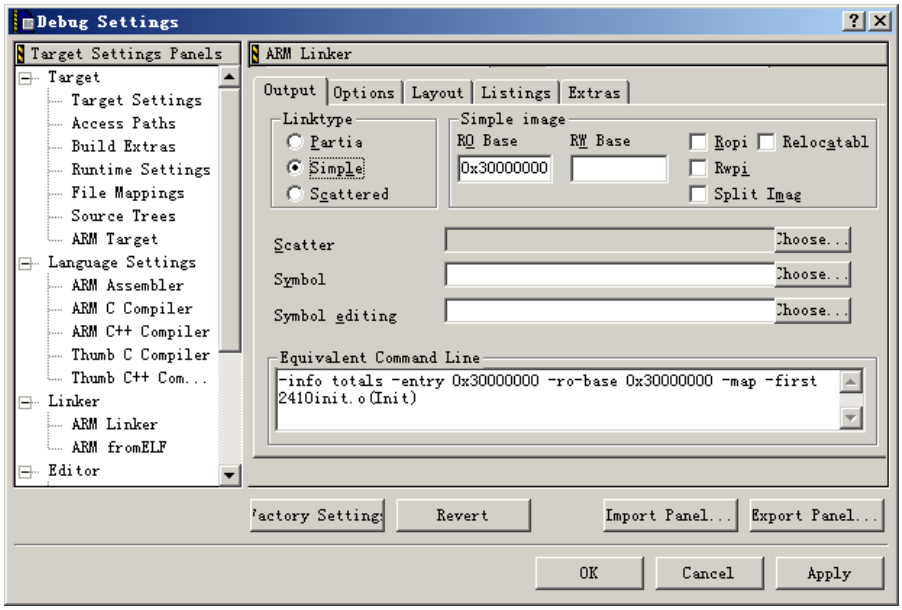

图 2-16　输入程序代码存放的起始地址

如图 2-17 所示，在 Options 选项卡 Image entry point 文本框中输入程序代码的入口地址，其他保持不变，如果是在 SDRAM 中运行，则可在 0x30000000~0x33ffffff 中选值，这是 64M SDRAM 的地址，但是这里用的是起始地址，所以必须把用户的程序空间给留出来，并且还要留出足够的程序使用的数据空间，而且还必须是 4 字节对齐的地址(ARM 状态)。通常入口点 Image entry point 为 0x30000000，RO_Base 也为 0x30000000。

如图 2-18 所示，在 Layout 选项卡的 Place at beginning of image 选项组内，需要输入项目的入口程序的目标文件名，例如，整个工程项目的入口程序是 2410init.s，那么应该在 Object/Symbol 文本框中输入其目标文件名 2410init.o，在 Section 文本框中输入程序入口的起始段标号。它的作用是通知编译器，整个项目的运行是从该段开始的。

(6) 在如图 2-19 所示对话框中，单击左栏的 ARM fromELF 选项，在 Output file name 文本框中设置输出文件名*.bin，前缀名可以自己取，在 Output format 下拉列表中选择 Plain binary，这是设置要下载到 flash 中的二进制文件。如图 2-19 中使用的是 test.bin。

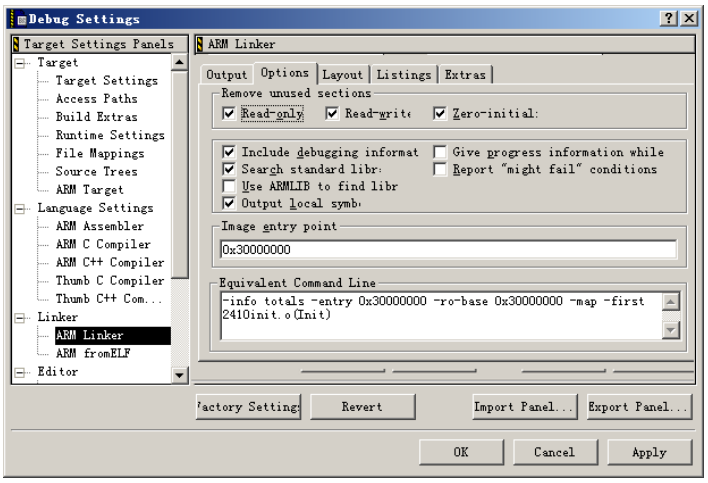

图 2-17　输入程序代码的入口地址

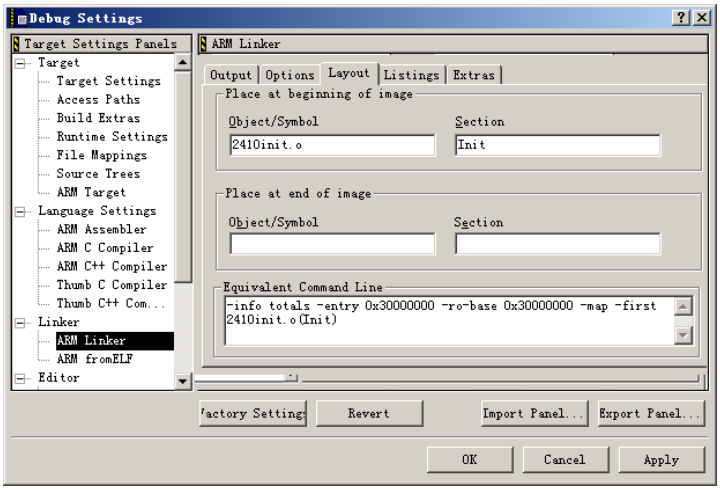

图 2-18　输入项目的入口

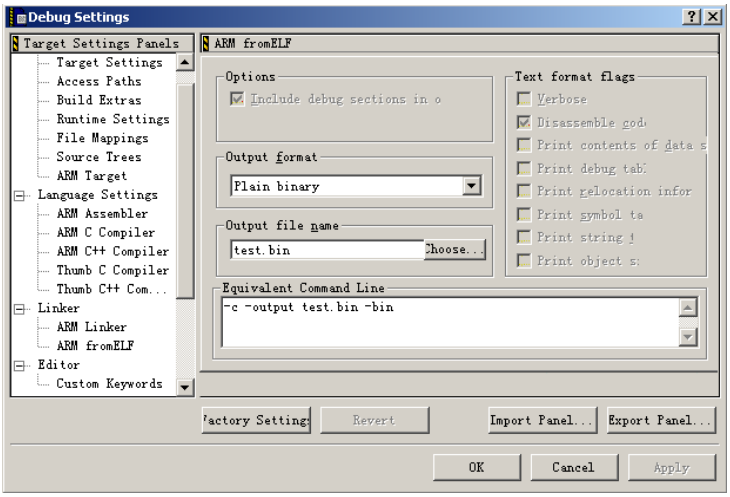

图 2-19　设置输出文件名

(7) 到此，ADS1.2 中的基本设置已经完成，可以将该新建的空项目文件作为模板保存起来。首先，要将该项目工程文件取一个合适的名字，如 S3C2410 ARM.mcp 等，然后，在 ADS1.2 软件的安装目录下新建一个合适的模板目录名，如 S3C2410 ARM Executable Image，再将刚刚设置完的 S3c2410 ARM.mcp 项目文件保存到该目录下即可。

(8) 新建项目工程后，可以选择 Project|Add Files 命令把和工程有关的文件加入，ADS1.2 不能自动进行文件分类，用户必须通过 Project|Create Group 命令来创建文件夹，然后把加入的文件选中，移入文件夹。或者将鼠标指针放在文件添加区，右击弹出快捷菜单，如图 2-20 所示。

在程序较多时，按类把文件归入几个文件夹，然后按文件夹加入项目，可以简化程序结构，方便调试。

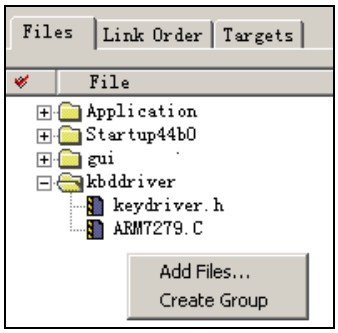

图 2-20　工程所有相关的文件加入项目

先选择 Add Files 命令，加入文件，再选择 Create Group 命令，创建文件夹，然后把文件移入文件夹内。读者可根据自己习惯，选择 Edit|Preference 命令，更改文本编辑的颜色、字体大小、形状、变量、函数的颜色等设置，如图 2-21 所示。

对不同的编辑内容选择不同的颜色，可以达到阅读和调试方便的效果。

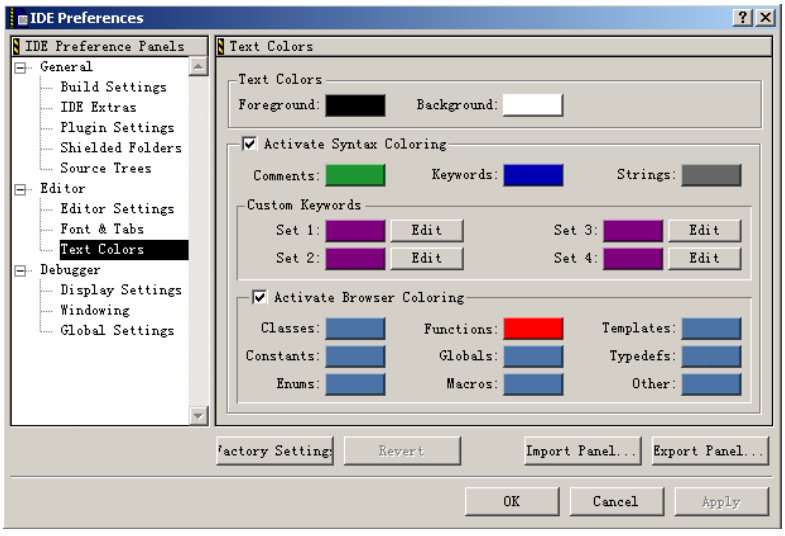

图 2-21　更改文本编辑的颜色

2.2.3　其他开发环境介绍

IAR(瑞典爱亚软件技术咨询公司)Embedded Workbench for ARM 是 IAR Systems 公司为 ARM 微处理器开发的一个集成开发环境，下面简称 IAR EWARM。与其他 ARM 开发环境相比，IAR EWARM 具有入门容易、使用方便和代码紧凑等特点。

IAR Systems 公司目前推出的最新版本是 IAR Embedded Workbench for ARM Version 4.42，并提供了一个 32K 代码限制学习版或 30 天时间限制的免费评估版，可以到 IAR 公司的网站 http://www.iar.com/ewarm 下载。

IAR EWARM 中包含一个全软件的模拟程序(simulator)。用户不需要任何硬件支持就可以模拟各种 ARM 内核、外部设备甚至中断的软件运行环境。从中可以了解和评估 IAR EWARM 的功能和使用方法。

IAR Embedded Workbench for ARM Version 4.42 是一个针对 ARM 处理器的集成开发环境，包含项目管理器、编辑器、编译连接工具和支持 RTOS(嵌入式实时控制系统)的调试工具，在该环境下可以使用 C/C++和汇编语言方便地开发嵌入式应用程序。

2.3　用 AXD 进行代码仿真、调试

项目建立并加入相应的文件后，“目标机”和“宿主机”通过 JTAG 仿真器进行连接，然后用 AXD 进行代码仿真、调试。本节将介绍用 AXD 进行代码仿真、调试的具体方法和步骤。

2.3.1　AXD 简介

前面讲过，ADS 对用户程序是以项目为单位进行管理的，在一个项目中可以包含后缀为.c 的 C 语言源文件、后缀为.cpp 的 C++源文件、后缀为.h 的 C 语言头文件、后缀为.s 的汇编语言源文件、后缀为.o 的目标码文件、后缀为.lib 的库文件等。这些文件可以单个加入项目，也可以文件夹的形式分类集体加入。

项目建立并加入相应文件之后，以项目为单位进行编译。在 Code Warrior IDE 开发环境中双击 Make 按钮，如图 2-22 所示，如果没有错误，会生成一个后缀为.axf 的计算机可执行文件，如果有错误则要修改源程序，再次 Make，直到没有错误为止。一般出现警告提示可以忽略。

在同一界面双击 Debug 按钮，如图 2-23 所示，进入 AXD，AXD 界面如图 2-24 所示。

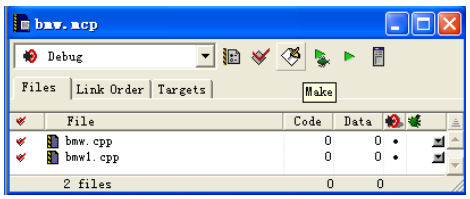

图 2-22　Make 界面

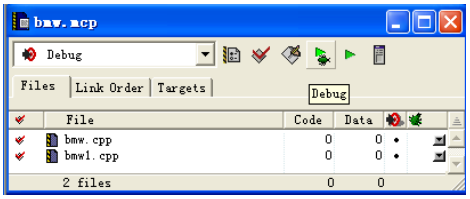

图 2-23　进入 AXD

AXD(ARM extended Debugger)是 ADS 软件中独立于 Code Warrior IDE 的图形软件，要使用 AXD 必须先生成包含调试信息的程序，即由 Code Warrior for ARM Developer Suite 编译生成的含有调试信息的可执行 ELF 格式的映像文件(*.axf)。

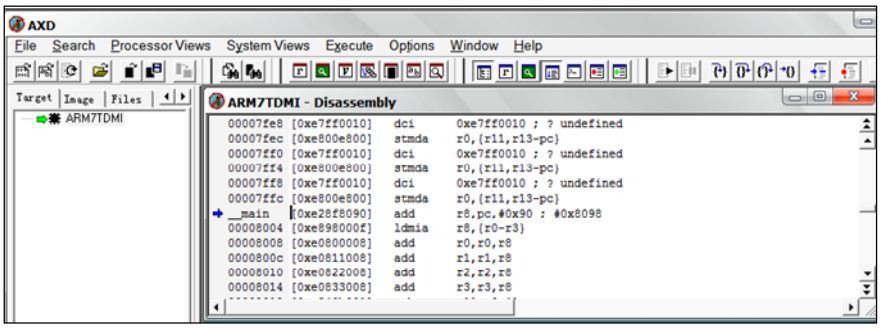

图 2-24　AXD 界面

1. 在 AXD 中打开调试文件

在 Code Warrior for ARM Developer Suite 界面中，单击 Debugger 进入 AXD 调试界面。选择 File | Load image 命令，打开 Load image 对话框，找到要装入的.axf 映像文件，单击“打开”按钮，即可把映像文件装载到目标内存中，如图 2-25 所示。

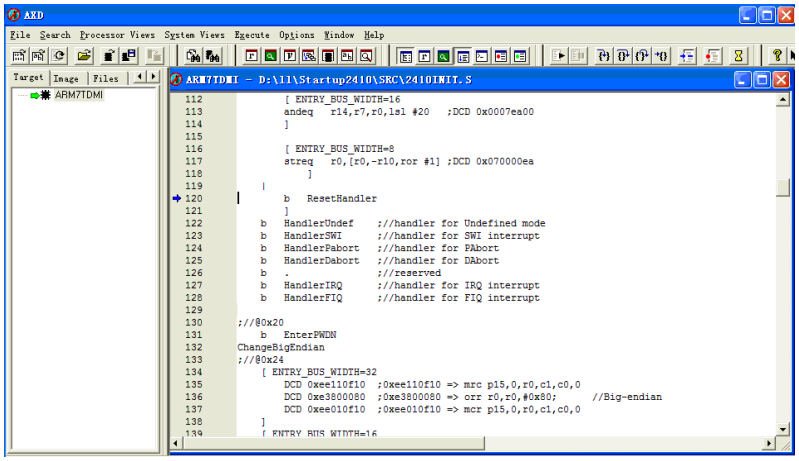

图 2-25　装入 axf 映像文件

利用 Execute 菜单中的子菜单对可执行映像文件进行调试，各选项的含义如下：

- 选择 Go 子菜单或按 F5 键，将全速运行代码。
- 选择 Stop 子菜单或按 Shift+F5 键，将停止运行代码。
- 选择 Step In 子菜单或按 F8 键，将以单步形式执行代码，若遇到函数，则进入函数内执行。
- 选择 Step 子菜单或按 F10 键，将以单步形式执行代码，若遇到函数，则把函数看成一条语句单步执行。
- 选择 Step Out 子菜单或按 Shift+F8 键，在 Step In 单步执行代码进入函数内后，若选择该子菜单，则可以从函数中跳出返回到上一级程序执行。

- 选择 Run To cursor 子菜单或按 F7 键，以全速运行到光标处停下。
- 选择 Show Execution Context 子菜单，可显示执行的内容。
- 选择 Delete All Breakpoint 子菜单，将清除所有的断点。

2. 查看存储器、寄存器、变量内容

利用 AXD Processor Views 和 System Views 菜单中的子菜单选项可以查看寄存器、变量值，还可以查看某个内存单元的数值等。各子菜单的含义如下：

- 选择 Registers 子菜单或按 Ctrl+R 键，可以查看或修改目标板处理器中寄存器中的值。
- 选择 Watch 子菜单或按 Ctrl+E 键，可以对处理器设置观察点，观察点可以是寄存器、地址等， 但不能修改。特别注意，Processor Views 菜单下的 Watch 只能观察处理器，而 System Views 菜单下的 Watch 或按 Alt+E 键时，可以对目标板上的任何资源建立观察，可增加或删除观察点。
- 选择 Variables 菜单或按 Ctrl+E 键，可以查看或修改当前可执行的映像文件(程序)中的变量值，这些变量可以是局部变量、全局变量、类变量。可增加或删除查看或修改的变量。
- 选择 Memory 子菜单或按 Ctrl+M 键，可以查看或修改存储器中的值。

在程序执行之前，可以先查看两个宏变量 IOPMOD 和 IOPDATA 的当前值。方法是：选择 AXD 的 Processor Views｜Memory 命令或按 Ctrl+M 键，查看或修改存储器中的值，如图 2-26 所示。

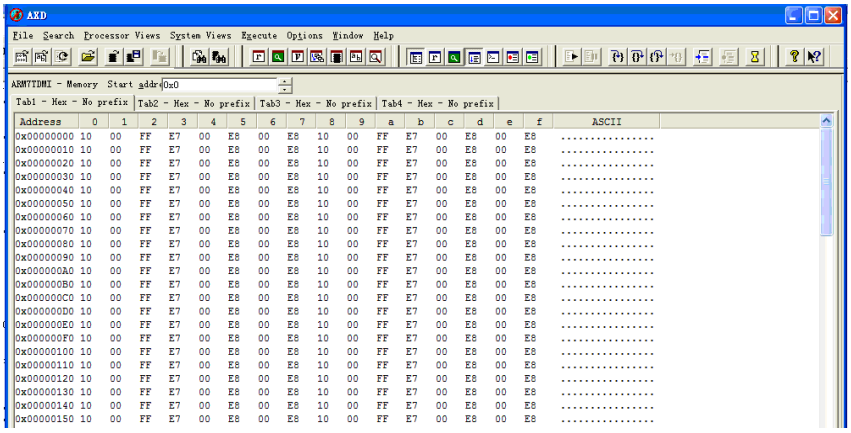

图 2-26　查看或修改存储器中的值

3. 查看或修改存储器地址

在 Memory Start address 文本框中，用户可以根据要查看或修改的存储器地址输入起始地址，在下面的表格中会列出连续的 64 个地址。因为 I/O 模式控制寄存器和 I/O 数据控制寄存器都是 32 位的控制寄存器，所以从 0x00000000 开始的连续 4 个地址空间存放的是 I/O 模式控制寄存器的值，从图中可以读出该控制寄存器的值，对于数据控制寄存器的内容，注意因为用的是小端模式，所以读数据时注意高地址中存放的高字节，低地址存放的低字节。

4. 断点设置、查看

在调试程序时，经常要设置断点，即在程序的某处设置断点，当程序执行到断点处即可停下，这时开发人员可以通过前面的方法查看寄存器、存储器或变量的值，以判定程序是否正常。设置断点的方法是将光标移到需设置断点处，使用快捷键 F9 在此处设置断点。

查看断点的方法是：选择 System Views｜breakpoint view 命令或按 Alt+K 组合键，在断点状态对话框中右击，利用弹出的快捷菜单可增加或删除断点。按 F5 键，程序将运行到断点，如果要进入函数内查看是如何运行的，可以选择 Execute｜Step In 命令或按下 F8 键，进入到子函数内部进行单步程序的调试。

2.3.2　JTAG 概述

在使用 AXD 对项目进行调试时，整个软件调试工作都是在“宿主机”上进行的，但程序的运行是在“目标机”上实现的。“宿主机”和“目标机”之间通过计算机并口使用 JTAG 仿真器相连接，实现信息的上传和下载。

JTAG 是 Joint Test Action Group(联合测试行动小组)的简称，由于 IEEE 1149.1 标准是由 JTAG 这个组织最初提出，最终由 IEEE 批准并且标准化的。所以 IEEE 1149.1 标准一般也俗称 JTAG 调试标准。

JTAG 标准主要用于芯片内部测试及对系统进行仿真、调试。JTAG 技术是一种嵌入式调试技术，它在芯片内部封装了专门的测试电路 TAP(Test Access Port 测试访问口)，通过专用的 JTAG 测试工具对内部节点进行测试。目前，大多数比较复杂的器件都支持 JTAG 协议，如 ARM、DSP、FPGA 器件等。标准的 JTAG 接口是 4 线：TMS、TCK、TDI、TDO，分别为测试模式选择、测试时钟、测试数据输入和测试数据输出。JTAG 测试允许多个器件通过 JTAG 接口串联在一起，形成一个 JTAG 链，能实现对多个器件分别测试。JTAG 接口还常用于实现 ISP(In-System Programmable 在线系统可编程)功能，如对 FLASH 器件进行编程等。

在 JTAG 调试中，边界扫描(Boundary-Scan)是一个很重要的概念。边界扫描技术的基本思想是在靠近芯片的输入输出管脚上增加一个移位寄存器单元。因为这些移位寄存器单元都分布在芯片的边界上，所以被称为边界扫描寄存器(Boundary-Scan Register Cell)。

当芯片处于调试状态时，这些边界扫描寄存器可以将芯片和外围的输入输出隔离开来。通过这些边界扫描寄存器单元，可以实现对芯片输入输出信号的观察和控制。如果需要捕获芯片的某个管脚上的输出，首先需要把该管脚上的输出装载到边界扫描链的寄存器单元中去，然后通过 TDO 输出，这样就可以从 TDO 上得到相应管脚上的输出信号。如果要在芯片的某个管脚上加载一个特定的信号，则首先需要通过 TDI 把期望的信号移位到与相应管脚相连的边界扫描链的寄存器单元中去，然后将该寄存器单元的值加载到相应的芯片管脚。

由于在正常的运行状态下，这些边界扫描寄存器对芯片是透明的，所以正常的运行不会受到任何影响。这样，边界扫描寄存器就提供了一个便捷的方式，用于观测和控制所需调试的芯片。另外，芯片输入输出管脚上的边界扫描(移位)寄存器单元可以相互连接起来，在芯片的周围形成一个边界扫描链(Boundary-Scan Chain)。一般的芯片都会提供几条独立的边界扫描链，用来实现完整的测试功能。边界扫描链可以串行地输入与输出，通过相应的时钟信号

和控制信号，可以方便地观察和控制处于调试状态下的芯片。

JTAG 仿真器需要设备驱动程序驱动，如对于教学实验系统(EDUKIT-Ⅲ)，JTAG 仿真器的驱动程序为两个动态链接库：EasyICEArm9Plus.dll、EasyICEArm7Plus.dll。把这两个文件复制到 C:\EmbestIDE\Bin\Device\路径下即可正常使用。

2.3.3　Nor 和 Nand Flash 的区别和使用

程序调试结束后，要将其可执行文件烧写(或称固化)到目标机中的某种 Flash 中运行。Flash 也叫非易失快闪存储器。

Nor 和 Nand 是现在市场上两种主要的非易失闪存技术。Intel 公司于 1988 年首先开发出 Nor Flash 技术。这项技术的开发和投放市场彻底地改变了原先由 EPROM 和 EEPROM 一统天下的局面。紧接着，1989 年东芝公司发表了 Nand Flash 结构，强调降低每比特的成本，提供更高的性能，并且像磁盘一样可以通过接口轻松升级。在具有 Nand Flash 接口的系统中，Nand Flash 存储器可以替代 Nor Flash 存储器使用。许多业内人士也搞不清楚 Nand 闪存技术相对于 Nor 技术的优越之处，因为大多数情况下，闪存只是用来存储少量的代码，这时 Nor 闪存更适合一些。而 Nand 则是高数据存储密度的理想解决方案。

Nor 的特点是 XIP(eXecute In Place 芯片内执行)特性，这样，应用程序可以直接在 Flash 闪存内运行，而不必再把代码读到系统 RAM 中。Nor 的传输效率很高，在 1~4MB 的小容量时具有很高的成本效益，但是很低的写入和擦除速度大大影响了它的性能。

Nand 结构能提供极高的单元密度，可以达到高存储密度，并且写入和擦除的速度也很快。应用 Nand 的困难在于 Flash 的管理和需要特殊的系统接口。

1. 性能比较

Flash 闪存是非易失存储器，可以对称为块的存储器单元块进行擦写和再编程。由于任何 Flash 器件的写入操作只能在空或已擦除的单元内进行，所以大多数情况下，在进行写入操作之前必须先进行擦除。Nand 器件执行擦除操作是十分简单的，而 Nor 则要求在进行写入前先要将目标块内所有的位都写为 0。

由于擦除 Nor 器件时是以 64KB~128KB 的块进行的，执行一个写入/擦除操作的时间为 5s，相应地，擦除 Nand 器件是以 8KB~32KB 的块进行的，执行相同的操作最多只需要 4ms。执行擦除时块尺寸的不同进一步拉大了 Nor 和 Nand 之间的性能差距，统计表明，对于给定的一套写入操作，尤其是更新小文件时，在基于 Nor 的单元中需要进行更多的擦除操作。这样，当选择存储解决方案时，设计师必须权衡一下如下各项因素：

- Nor 的读速度比 Nand 稍快一些。
- Nand 的写入速度比 Nor 快很多，Nand 的 4ms 擦除速度远比 Nor 的 5s 快。
- 大多数写入操作需要先进行擦除操作。
- Nand 的擦除单元更小，相应的擦除电路更少。

2. 容量和成本

Nand Flash 的单元尺寸几乎是 Nor 器件的一半，由于生产过程更为简单，Nand 结构可以

在给定的模具尺寸内提供更高的容量，也就相应地降低了价格。在 Nand 闪存中，每个块的最大擦写次数是一百万次，而 Nor 的擦写次数则是十万次。

Nor Flash 占据了容量为 1MB~16MB 闪存市场的大部分，而 Nand Flash 只是用在 8MB~128MB 的产品当中，这也说明 Nor 主要应用在代码存储介质中，Nand 则适合于数据存储。Nand 在 Compact Flash、Secure Digital、PC Cards 和 MMC 存储卡市场上所占份额最大。

3. 接口差别

Nor Flash 带有 SRAM 接口，有足够的地址引脚来寻址，可以很容易地存取其内部的每一个字节。基于 Nor 的闪存使用非常方便，可以像其他存储器那样连接，并可以在上面直接运行代码。

Nand 器件使用复杂的 I/O 口来串行存取数据，各个产品或厂商的方法可能各不相同。8 个引脚用来传送控制、地址和数据信息。Nand 的读写操作采用 512 字节的块，这一点与硬盘管理操作类似，显然，基于 Nand 的存储器就可以取代硬盘或其他块设备。

在使用 Nand 器件时，必须先写入驱动程序，才能继续执行其他操作。向 Nand 器件写入信息需要相当的技巧，因为设计师决不能向坏块写入，这就意味着在 Nand 器件上自始至终都必须进行虚拟映射。

S3C2410X 微处理器支持 Nand Flash 接口，大大方便了在嵌入式系统设计中的应用。鉴于两种存储器各自的优缺点，在 S3C2410X 嵌入式系统中，对 Nor Flash 和 Nand Flash 电路都进行了设计，以方便用户使用。

2.3.4　烧写 Flash

上一节讲到，程序调试结束，要将其可执行文件烧写(或称固化)到目标机的 Flash 中运行，这个过程要通过一个专门的下载软件来进行，该软件有多款，它们的安装和使用请参考具体的设备。

2.3.5　程序的运行

程序固化到 Flash 中后，运行前往往将其复制到 SDRAM 中，这样可以提高运行速度，作者在工作中，编写了一段将程序从 Flash 中复制到 SDRAM 中的 C 语言模块，实践证据可用，可供参考，其中 ARM9init(void)程序略。

```
//-------------------------------------------------------------------------------------------------------------
//     主程序
//-------------------------------------------------------------------------------------------------------------
#include   " def.h "
#include    " 2410addr.h "
#include    " 2410lib.h "
#define ARM_ADDR 0X30000000;    // 定义 SDRAM 地址
void (*run)(void );                          //定义函数指针
void ARM9init(void);                         //系统初始化程序
void CopyFromFlashToRAM(U32 * FlashAddr,U32 *ArmAddr,U32 ul);
```

```
void copy(void);
void main(void)
  {
    run=(void(*)(void)) ARM_ADDR;
    ARM9init();
copy();
    run();
}
//-------------------------------------------------------------------------------------------------------------------
//     复制程序
//-------------------------------------------------------------------------------------------------------------------
CopyFromFlashToRAM(U32 * pulFlashAddr,U32 *pulArmAddr,U32 ul)
{
U32 *pulSource=pulFlashAddr;
U32 *pulDest=pulArmAddr;
U32 i;
ul/=4;
for(i=0; i<ul; i++)
{
*pulDest++=*pulSource++;
}
}
//-------------------------------------------------------------------------------------------------------------------
//     调复制程序
//-------------------------------------------------------------------------------------------------------------------
copy(void)
{
U32*p1;
U32*p2;
P1=(U32 * )0x00200000;
P2=(U32 * )0x30000000;
CopyFromFlashToRAM(p1,p2,0x20000);
}
```

2.4　ARM C 语言程序的基本规则和系统初始化程序

一般使用 C 语言来开发嵌入式项目，ARM C 语言程序的基本规则和普通 C 语言程序有一定的区别，本节将介绍 ARM C 语言程序的基本规则和系统初始化程序的编写。

系统初始化程序的编写和项目调试的仿真器设置是初学者遇到的比较困难的问题，本节将帮助初学者绕过这些困难，顺利完成开发工作。

2.4.1　ARM 使用 C 语言编程基本规则

在应用系统的程序设计中，如果所有的编程任务均由汇编语言来完成，其工作量巨大，并且不易移植。由于 ARM 的程序执行速度较高，存储器的存储速度和存储量也很高，因此，C 语言的特点得以充分发挥，从而使应用程序的开发时间大为缩短，代码的移植十分方便，程序的重复使用率提高，程序架构清晰易懂，管理也较为容易。因此，C 语言在 ARM 编程中具有重要地位。

在 ARM 程序的开发中，需要大量读写硬件寄存器，尽量缩短程序的执行时间，因此，部分初始化代码一般使用汇编语言来编写，如 ARM 的启动代码，ARM 的操作系统的移植代码等，除此之外，绝大多数代码可以使用 C 语言来完成。

C 语言使用的是标准的 C 语言，ARM 的开发环境实际上就是嵌入了一个 C 语言的集成开发环境，只不过这个开发环境和 ARM 的硬件紧密相关。

在使用 C 语言时，有时要用到和汇编语言的混合编程。当汇编代码较为简洁时，则可使用直接内嵌汇编的方法，否则，使用将汇编代码以文件的形式加入项目当中，通过 ATPCS(ARM/Thumb Procedure Call Standard)的规定与 C 程序相互调用与访问。ATPCS 就是 ARM、Thumb 的过程调用标准，它规定了一些子程序间调用的基本规则。如寄存器的使用规则、堆栈的使用规则、参数的传递规则等。

在 C 程序和 ARM 的汇编程序之间相互调用必须遵守 ATPCS。而使用 ADS 的 C 语言编译器编译的 C 语言子程序满足用户指定的 ATPCS 的规则。但是，对于汇编语言来说，完全要依赖用户来保证各个子程序遵循 ATPCS 的规则。具体来讲，汇编语言的子程序应满足以下 3 个条件：

- 在子程序编写时，必须遵守相应的 ATPCS 规则；
- 堆栈的使用要遵守相应的 ATPCS 规则；
- 在汇编编译器中使用 atpcs 选项。

基本的 ATPCS 规定，详情请见相关的 PDF 文档，下面做一下简单说明。

1. 汇编程序调用 C 程序

- 汇编程序的设置要遵循 ATPCS 规则，保证程序调用时参数能正确传递。
- 在汇编程序中使用 IMPORT 伪指令声明将要调用的 C 程序函数。
- 在调用 C 程序时，要正确设置入口参数，然后使用 BL 调用。

2. C 程序调用汇编程序

- 汇编程序的设置要遵循 ATPCS 规则，保证程序调用时参数能正确传递。
- 在汇编程序中使用 EXPORT 伪指令声明本子程序，使其他程序可以调用此子程序。
- 在 C 语言中使用 extern 关键字声明外部函数(声明要调用的汇编子程序)。

在 C 语言的环境内开发应用程序，一般需要一个汇编的启动程序。从汇编的启动程序跳到 C 语言下的主程序，然后执行 C 程序。在 C 语言环境下读写硬件的寄存器，一般是通过宏调用。在每个项目文件的 Startup2410/INC 目录下都有一个名为 2410addr.h 的头文件，该文

件中定义了所有关于 2410 的硬件寄存器的宏。对宏的读写，就能操作 2410 的硬件，具体的编程规则同标准 C 语言。

2.4.2　初始化程序和开发环境设置

基于 ARM 芯片的应用系统，多数为复杂的片上系统，在系统中，多数硬件模块都是可配置的，需要由软件来预先设置其需要的工作状态，因此，在用户的应用程序之前，需要由一段专门的代码来完成对系统基本的初始化工作。由于此类代码直接面对处理器内核和硬件控制器进行编程，故一般均用汇编语言实现。

系统的基本初始化内容一般包括：

- 分配中断向量表
- 初始化存储器系统
- 初始化各工作模式的堆栈
- 初始化有特殊要求的硬件模块
- 初始化用户程序的执行环境
- 切换处理器的工作模式

此外，还要对项目的交叉编译环境进行设置，这其中包括处理器设置、仿真器设置和调试设置等 20 几个大项，近 100 个小项，如何保证各项都设置正确，这些都是需要解决的。初次学习 ARM 程序设计，有哪些硬件模块需要预先设置其需要的工作状态，如何设置？初始化程序代码直接面对处理器内核和硬件控制器进行编程，故一般用汇编语言实现。

初学者可能对 ARM 的汇编语言不熟，为了使初次学习嵌入式系统设计的读者绕开这些难点，尽快掌握 S3C2410 32 位单片机的 C 语言程序设计，完成具体的设计项目，建议读者在设计一个具体的嵌入式系统时参照已有的项目例子进行，因为在这些例子中，系统的初始化和交叉编译环境设置都已经完成了，默认使用这些设置，可以把主要精力放在提高系统的稳定性和可靠性上。这也是工程上常用的设计方法。

在随书下载的资料中有一个 2410test.mcp 项目，其中包含了 S3C2410 的所有硬件驱动程序，仔细阅读这些程序对今后的编程有非常重要的参考价值。

此外，还可以参考网上或各教学实验系统或其他资源的例子，模仿这些例子编程可以达到事半功倍的效果。

例如，要开发一个液晶(LCD)显示项目，液晶显示项目是嵌入式系统人机界面设计中最重要的项目，但也是较难的项目，如果不参考类似项目而是自己来做，那么项目的初始化程序和开发环境设置可能很难。

现在打开一个作者在工作中编写的例子项目 lcd.mcp，如图 2-27 所示，在项目窗口中可以看到，项目有 5 个文件夹组，分别是 lcddrv、start2410、Application、Gui、newh。

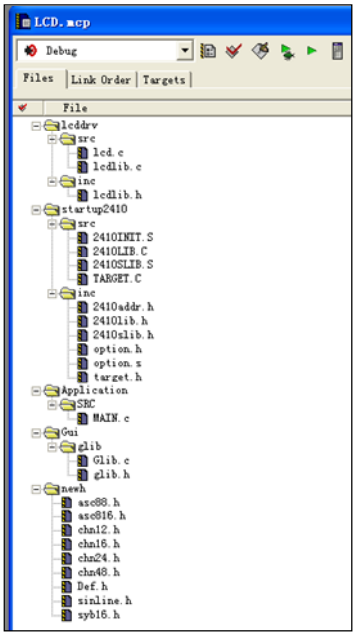

图 2-27　lcd.mcp 结构

初始化程序在文件夹组 startup2410 中，其中包括 3 个汇编语言程序 2410INIT.S、OPTION.S 和 2410SLIB.S，两个 C 语言程序 2410LIB.C 和 TARGET.C，4 个 C 语言头文件 2410addr.h、2410lib.h、2410slib.h 和 option.h。

现在先不去研究这些程序，把 startup2410 文件夹保留，只修改 Application 文件夹中的 MAIN.C，把项目所需要的新函数加到其中一个文件夹，或新建一个文件夹加到项目中，默认项目的开发环境和初始化程序设置，修改 newh 文件夹中所需显示的汉字字模的头文件，完成 LCD 显示项目的开发就很容易了。

2.5　习题

1. 下载随书资料，在用户的计算机中安装 ADS1.2，并建立一个新项目。
2. 在 AXD 中打开一个例子项目。
3. 利用 Execute 菜单中的子菜单对可执行映像文件进行如下调试试验：

(1) 选择 Go 子菜单或按 F5 键，全速运行代码。
(2) 选择 Stop 子菜单或按 Shift+F5 键，停止运行代码。
(3) 选择 Step In 子菜单或按 F8 键，单步执行代码。
(4) 选择 Step 子菜单或按 F10 键，单步执行代码。
(5) 选择 Step Out 子菜单或按 Shift+F8 键，从函数中跳出返回到上一级程序执行。
(6) 选择 Run To cursor 子菜单或按 F7 键，全速运行到光标处停下。
(7) 选择 Show Execution Context 子菜单，显示执行的内容。

4. 如何设置断点，如何清除断点。如何查看存储器、寄存器、变量内容？

第3章　ARM9微处理器S3C2410资源

本章介绍 ARM9 内核 32 位精简指令系统微处理器 S3C2410 的片上资源，包括 AMBA 总线、AHB总线、APB总线、处理器体系结构、处理器管理系统、处理器存储结构、时钟和电源管理等。片上资源的定义和使用，特别是头文件 2410addr.h 和参考项目 2410test.mcp 非常重要，对今后的编程有很大的帮助。

3.1　S3C2410 处理器介绍

本节介绍 S3C2410 处理器的体系结构、特点和应用领域，AMBA、AHB、APB 总线特点及应用，存储器存储空间映射等。

S3C2410 微处理器是一款由 SAMSUNG 公司为手持设备设计的低功耗、高度集成的基于 ARM920T 核的微处理器。为了降低系统总成本和减少外围器件，这款芯片中还集成了如下部件：16KB 指令 Cache、16KB 数据 Cache、MMU、外部存储器控制器、LCD 控制器(STN 和 TFT)、NAND Flash 控制器、4 个 DMA 通道、3 个 UART 通道、1 个 I^2C 总线控制器、1 个 I^2S 总线控制器，以及 4 个 PWM 定时器和 1 个内部定时器、通用 I/O 口、实时时钟、8 通道 10 位 ADC 和触摸屏接口、USB 主接口、USB 从接口、SD/MMC 卡接口等。现在它广泛应用于 PDA、移动通信、路由器、工业控制等领域，其内部结构如图 3-1 所示。

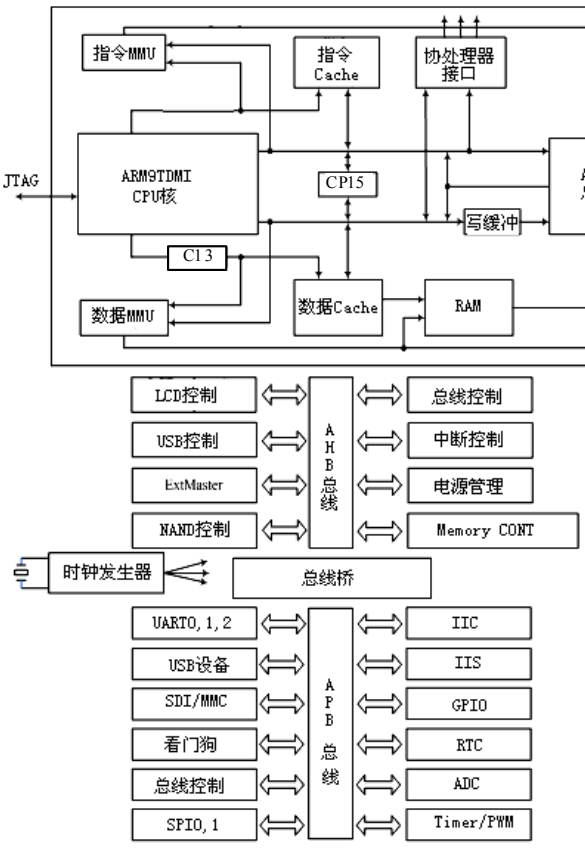

图 3-1　S3C2410X 结构框图

为了提高系统的运行速度，减少能量损失，S3C2410 微处理器把片上器件按器件频率、使用频度分成 3 个模块，各个模块通

过各自的总线连接，模块之间采用一种叫总线桥的结构过渡。下面简单介绍各类总线的特点。

3.1.1　AMBA、AHB、APB 总线特点

随着深亚微米工艺技术的日益成熟，集成电路芯片的规模越来越大。数字 IC 从基于时序驱动的设计方法，发展到了基于 IP(Internet Protocol，互联网协议)复用的设计方法，并在片上系统(SoS)设计中得到了广泛应用。在基于 IP 复用的 SoC 设计中，片上总线设计是最关键的问题。为此，业界出现了很多片上总线标准。其中，由 ARM 公司推出的 AMBA 片上总线受到了广大 IP 开发商和 SoC 系统集成者的青睐，已经成为一种流行的标准片上结构。AMBA 规范主要包括 AHB(Advanced High performance Bus)系统总线和 APB(Advanced Peripheral Bus)外围总线。

AMBA 2.0 规范包括 4 个部分：AHB、ASB、APB 和 Test Methodology。AHB 的相互连接采用了传统的带有主模块和从模块的共享总线，接口与互连功能分离，这对芯片上模块之间的互连具有重要意义。AMBA 已不仅是一种总线，更是一种带有接口模块的互连体系。下面将简要介绍比较重要的 AHB 和 APB 总线。

大多数挂在总线上的模块(包括处理器)只是单一属性的功能模块：主模块或者从模块。主模块是向从模块发出读写操作的模块，如 CPU、DSP 等；从模块则是接受命令并做出反应的模块，如片上的 RAM、AHB/APB 桥等。另外，还有一些模块同时具有两种属性，例如直接存储器存取(DMA)在被编程时是从模块，但在系统传输数据时必须是主模块。

如果总线上存在多个主模块，就需要仲裁器来决定如何控制各种主模块对总线的访问。虽然仲裁规范是 AMBA 总线规范中的一部分，但具体使用的算法由 RTL(Real Time Language，实时语言)设计工程师决定，其中两个最常用的算法是固定优先级算法和循环制算法。

AHB 总线上最多可以有 16 个主模块和任意多个从模块，如果主模块的数目大于 16，则需再加一层结构(具体参阅 ARM 公司推出的 Multi-layer AHB 规范)。APB 桥既是 APB 总线上唯一的主模块，也是 AHB 系统总线上的从模块。其主要功能是锁存来自 AHB 系统总线的地址、数据和控制信号，并提供二级译码以产生 APB 外围设备的选择信号，从而实现 AHB 协议到 APB 协议的转换。

AHB 主要用于高性能模块(如 CPU、DMA 和 DSP 等)之间的连接，作为 SoC 的片上系统总线，它包括如下特性：单个时钟边沿操作；非三态的实现方式；支持突发传输；支持分段传输；支持多个主控制器；可配置 32~128 位总线宽度；支持字节、半字节和字的传输。

APB 主要用于低带宽的周边外设之间的连接，例如 UART、USB 等，它的总线架构不像 AHB 那样支持多个主模块，在 APB 里面唯一的主模块就是 APB 桥。其特性包括：两个时钟周期传输；无需等待周期和回应信号；控制逻辑简单，只有 4 个控制信号。

3.1.2　S3C2410 处理器体系结构

- ARM920T 核，16 位/32 位 RISC 结构和 ARM 精简指令集。
- ARM MMU，支持 Windows CE、Linux 等操作系统。
- 指令 Cache、数据 Cache、写缓冲。

- 支持 ARM 调试结构，片上 ICE 支持 JTAG 调试方式。

3.1.3　S3C2410 处理器管理系统

- 支持大端(Big Endian)/小端(Little Endian)模式。
- 地址空间为每个内存块 128MB(一共 1GB)，每个内存块支持 8/16/32 位数据总线编程。
- 8 个内存块：6 个用于 ROM、SRAM 和其他，两个用于 ROM/SRAM/SDRAM。
- 1 个起始地址和大小可编程的内存块(Bank7)。
- 7 个起始地址固定的内存块(Bank0~Bank6)。
- 所有内存块可编程寻址周期。
- 支持 SDRAM 自动刷新模式。
- 支持多种类型 ROM 启动，包括 NOR/NAND Flash、EEPROM 等。

3.1.4　S3C2410 处理器存储器映射

S3C2410 的存储空间映射如图 3-2 所示。

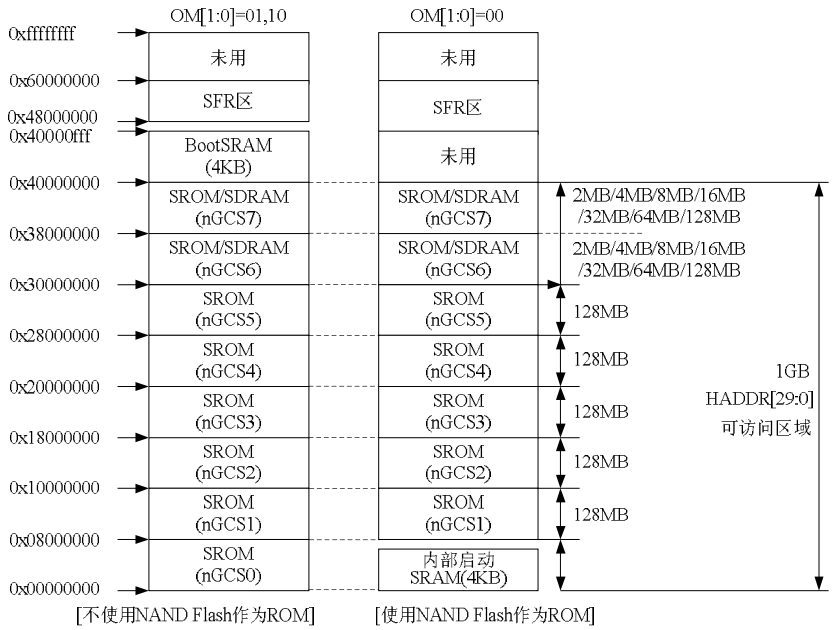

图 3-2　S3C2410 存储区地址映射

3.1.5　S3C2410 处理器时钟和电源管理

S3C2410 处理器时钟和电源管理非常复杂，这主要是为了最大限度地降低功耗。

1. 时钟

S3C2410 处理器的主时钟由外部晶振(Crystal)或者外部时钟(EXTCLK)提供，经电源和时钟管理模块选择后可以提供两种时钟信号，分别是 MPLL 和 UPLL。其中，MPLL 又分频为 CPU 使用的 FCLK、AHB 总线使用的 HCLK 和 APB 总线使用的 PCLK；UPLL 分频为 48MHZ，

供 USB 设备使用。

2. 时钟源选择

S3C2410 处理器的主时钟是由外部晶振(Crystal)提供还是由外部时钟(EXTCLK)提供是通过 S3C2410 引脚 OM[3:2] 由用户确定的，具体如表 3-1 所示。

表 3-1　时钟源选择

OM[3:2]	MPLL 状态	UPLL 状态	主时钟源	USB 时钟源
00	On	On	Crystal	Crystal
01	On	On	Crystal	EXTCLK
10	On	On	EXTCLK	Crystal
11	On	On	EXTCLK	EXTCLK

S3C2410 引脚的 OM[3:2]=00 时，晶体 Crystal 为 MPLL CLK 和 UPLL CLK 提供时钟源；OM[3:2]=01 时，晶体 Crystal 为 MPLL CLK 提供时钟源，EXTCLK 为 UPLL CLK 提供时钟源；OM[3:2]=10 时，EXTCLK 为 MPLL CLK 提供时钟源，晶体 Crystal 为 UPLL CLK 提供时钟源；OM[3:2]=11 时，EXTCLK 为 MPLL CLK 和 UPLL CLK 提供时钟源。

S3C2410 处理器时钟和电源管理框图如图 3-3 所示。

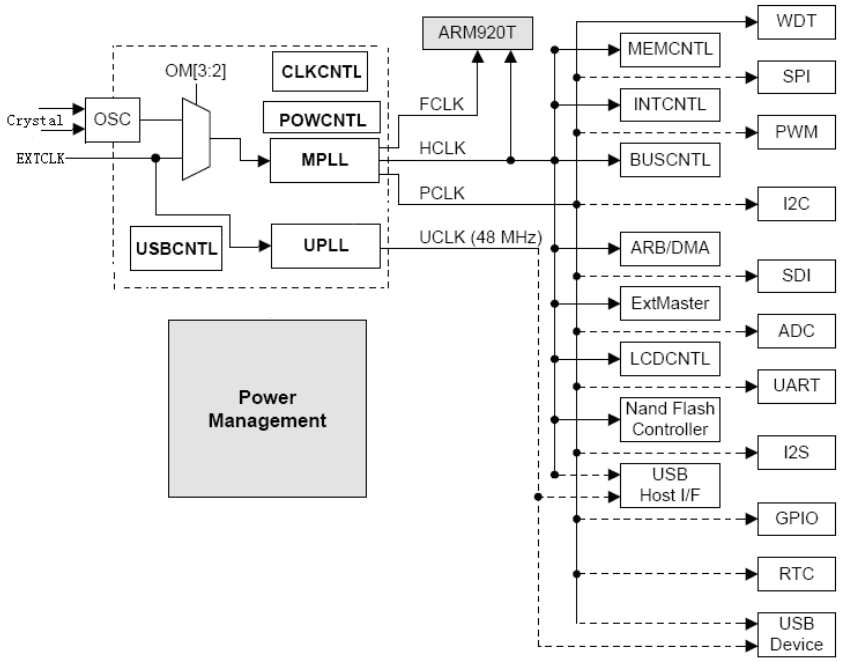

图 3-3　S3C2410 时钟和电源管理框图

3. 时钟控制逻辑

S3C2410 支持 HCLK、FCLK 和 PCLK 的频率按比率选择，其比率是通过时钟分频寄存器 CLKDIV 中的 HDIVN 和 PDIVN 进行控制的，如表 3-2 所示。

表 3-2　分频设定表

HDIVN	PDIVN	FCLK	HCLK	PCLK	Divide Ratio
0	0	FCLK	FCLK	FCLK	1:1:1 Default
0	1	FCLK	FCLK	FCLK/2	1:1:2
1	0	FCLK	FCLK/2	FCLK/2	1:2:2
1	1	FCLK	FCLK/2	FCLK/4	1:2:4 recommendded

4. 电源管理

S3C2410 电源管理模块通过 4 种模式有效地控制功耗：即正常(Normal)模式、省电(Slow)模式、空闲(Idle)模式和断电(Power-off)模式。

- Normal 模式：为 CPU 和所有的外设提供电源，所有的外设开启时，该模式下的功耗最大。这种模式允许用户通过软件控制外设，可以断开提供给外设的时钟以降低功耗。
- Slow 模式：采用外部时钟生产 FCLK 的方式，此时电源的功耗取决于外部时钟。
- Idle 模式：断开 FCLK 与 CPU 内核的连接，外设保持正常，该模式下的任何中断都可唤醒 CPU。
- Power-off 模式：断开内部电源，只给内部的唤醒逻辑供电。一般模式下需要两个电源，一个提供给唤醒逻辑，另一个提供给 CPU 和内部逻辑，而在 Power-off 模式下，后一个电源关闭。该模式可以通过 EINT[15:0]和 RTC 唤醒。

5. 时钟和电源管理寄存器

S3C2410 通过相应的控制寄存器实现对时钟和电源的管理，相关寄存器使用如表 3-3 所示。

表 3-3　时钟和电源管理寄存器

寄存器	地址	读写	说明	复位值
LOCKTIME	0X4C000000	R/W	PLL 锁定寄存器	0X00FFFFFF
MPLLCON	0X4C000004	R/W	MPLL 配置寄存器	0X0005C080
UPLLCON	0X4C000008	R/W	UPLL 配置寄存器	0X00028080
CLKCON	0X4C00000C	R/W	时钟信号生成制	0X7FFF0000
CLKSLOW	0X4C000010	R/W	SLOW 时钟控制	0X00000004
CLKDIVN	0X4C000014	R/W	分频控制	0X00000000

3.2　S3C2410 处理器片上资源的定义和使用

在 MCS-51 单片机开发系统中，所有的系统资源都在 reg51.h 头文件中定义，在编程时将 reg51.h 头文件引入程序，在程序中就可以很方便地使用这些系统资源了。

和开发 MCS-51 单片机一样，S3C2410 在头文件 2410addr.h 中定义了 S3C2410 的所有硬件资源，在编写 S3C2410 的驱动程序时必须引用这个头文件。

2410addr.h 将系统所有的资源进行了宏定义，宏的名称就是所定义的寄存器的名字前面加一个小写的“r”，方便记忆。

2410addr.h 的内容包括：Memory control、USB Host、INTERRUPT、DMA、CLOCK & POWER MANAGEMENT、LCD CONTROLLER、NAND flash、UART、PWM TIMER、USB DEVICE、WATCHDOG TIMER、IIC、IIS、I/O PORT、RTC、ADC、SPI、ISR、SD Interface 等，近 20 类。

在开发的项目中，引入 2410addr.h 头文件以后，就可以使用这些宏定义来对寄存器进行操作了，这些寄存器的用法后面会陆续介绍。更详细的内容可参见随书提供的软件包参考程序 24120test 中的 2410addr.h。

3.3　参考软件资源 2410test.mcp

在随书提供的资料中，有一个 2410test.mcp 项目，其中包括几乎所有 S3C2410 硬件驱动的 C 语言例子和头文件，仔细阅读这些程序对编程有很大的参考价值。

项目的主要代码如下：

```
//-------------------------------------------------------------------------------------------------------------------
//          引入所有实验所需头文件
//-------------------------------------------------------------------------------------------------------------------
#include <stdlib.h>
#include <string.h>
#include "def.h"
#include "option.h"
#include "2410addr.h"
#include "2410lib.h"
#include "2410slib.h"
#include "2410etc.h"
#include "2410IIC.h"
#include "2410iis.h"
#include "2410int.h"
#include "2410RTC.h"
#include "2410swi.h"
#include "timer.h"
#include "adc.h"
#include "dma.h"
#include "dma2.h"
#include "eint.h"
#include "extdma.h"
#include "k9s1208.h"
```

```
#include "mmu.h"
#include "nwait.h"
#include "sdi.h"
#include "stone.h"
#include "ts_auto.h"
#include "ts_sep.h"
#include "usbfifo.h"
#include "IrDA.h"
#include "lcd.h"
#include "lcdlib.h"
#include "glib.h"
#include "palette.h"
#include "spi.h"
#include "uart0.h"
#include "uart1.h"
#include "uart2.h"
#include "etc.h"
#include "flash.h"
#include "idle.h"
#include "pd6710.h"
#include "pll.h"
#include "power.h"
#include "pwr_c.h"
#include "stop.h"
//-------------------------------------------------------------------------------------------------------------------
//   定义一个二维的指针数组，数组中第一列是函数指针，第二列是函数功能
//-------------------------------------------------------------------------------------------------------------------
void * function[][2]=
{
//ADC, TSP
    (void *)Test_Adc,                   "ADC                    ",
    (void *)Test_DMA_Adc,               "ADC with DMA           ",
    (void *)Ts_Sep,                     "ADC TSP Seperate       ",
    (void *)Ts_Auto,                    "ADC TSP Auto           ",
//DMA
    (void *)Test_DMA,                   "DMA M2M                ",
    (void *)Test_DMAWorst,              "DMA Worst Test         ",
    (void *)Test_Dma0Xdreq,             "External DMA           ",
//EINT
    (void *)Test_Eint,                  "External Interrupt     ",
//IIC
    (void *)Test_Iic,                   "IIC(KS24C080)INT       ",
    (void *)Test_Iic2,                  "IIC(KS24C080)POL       ",
//IIS
```

```
        (void *)Record_Iis,                          "Reco IIS UDA1341       ",
        (void *)Test_Iis,                            "Play IIS UDA1341       ",
    //Interrupt
        (void *)Test_Fiq,                            "FIQ Interrupt          ",
        (void *)Change_IntPriorities,                "Change INT Priority    ",
    //IrDA
        (void *)Test_IrDA_Rx,                        "UART2 IrDA Rx          ",
        (void *)Test_IrDA_Tx,                        "UART2 IrDA Tx          ",
    //LCD
        (void *)Test_Lcd_Stn_1Bit,                   "STN 1Bit               ",
        (void *)Test_Lcd_Stn_2Bit,                   "STN 2Bit               ",
        (void *)Test_Lcd_Stn_4Bit,                   "STN 4Bit               ",
        (void *)Test_Lcd_Cstn_8Bit,                  "CSTN  8Bit             ",
        (void *)Test_Lcd_Cstn_8Bit_On,               "CSTN  8Bit On          ",
        (void *)Test_Lcd_Cstn_12Bit,                 "CSTN 12Bit             ",
        (void *)Test_Lcd_Tft_8Bit_240320,            "TFT240320  8Bit        ",
        (void *)Test_Lcd_Tft_8Bit_240320_On,         "TFT240320  8Bit On     ",
        (void *)Test_Lcd_Tft_16Bit_240320,           "TFT240320 16Bit        ",
        (void *)Test_Lcd_Tft_1Bit_640480,            "TFT640480  1Bit        ",
        (void *)Test_Lcd_Tft_8Bit_640480,            "TFT640480  8Bit        ",
        (void *)Test_Lcd_Tft_16Bit_640480,           "TFT640480 16Bit        ",
        (void *)Test_Lcd_Tft_8Bit_640480_Bswp,       "TFT640480 BSWP         ",
        (void *)Test_Lcd_Tft_8Bit_640480_Palette,    "TFT640480 Palette      ",
        (void *)Test_Lcd_Tft_16Bit_640480_Hwswp     ,"TFT640480 HWSWP        ",
    //Memory
    //MPLL
        (void *)Test_PLL,                            "MPLL Change            ",
        (void *)ChangePLL,                           "MPLL MPS Change        ",
        (void *)Test_PllOnOff,                       "MPLL On/Off            ",
    //PMS
        (void *)Test_SlowMode,                       "PMS Slow               ",
        (void *)Test_HoldMode,                       "PMS Hold               ",
        (void *)Test_IdleMode,                       "PMS Idle               ",
        (void *)Test_MMUIdleMode,                    "PMS Idle(MMU)          ",
        (void *)Test_IdleModeHard,                   "PMS Idle Hard          ",
        (void *)Test_InitSDRAM,                      "PMS SDRAM Init         ",
        (void *)Test_StopMode,                       "PMS STOP               ",
        (void *)Test_PowerOffMode,                   "PMS Power-Off STOP     ",
        (void *)Test_PowerOffMode_100Hz,             "PMS Power-Off 100Hz    ",
        (void *)MeasurePowerConsumption,             "PMS Measure Power      ",
    //RTC
        (void *)Test_Rtc_Alarm,                      "RTC Alarm              ",
        (void *)Display_Rtc,                         "RTC Display            ",
        (void *)RndRst_Rtc,                          "RTC Round Reset        ",
```

```
        (void *)Test_Rtc_Tick,                  "RTC Tick                ",
//SDI
        (void *)Test_SDI,                       "SDI Write/Read          ",
//SPI
        (void*) Test_Spi_MS_int,                "SPI0 RxTx Int           ",
        (void *)Test_Spi_MS_poll,               "SPI0 RxTx POLL          ",
        (void *)Test_Spi_M_Tx_DMA1,             "SPI0 Master Tx DMA1     ",
        (void *)Test_Spi_S_Rx_DMA1,             "SPI0 Slave Rx DMA1      ",
        (void *)Test_Spi_M_Rx_DMA1,             "SPI0 Master Rx DMA1     ",
        (void *)Test_Spi_S_Tx_DMA1,             "SPI0 Slave Tx DMA1      ",
        (void *)Test_Spi_M_Int,                 "SPI0 Master RxTx INT    ",
        (void *)Test_Spi_S_Int,                 "SPI0 Slave RxTx INT     ",
//Timer
        (void *)Test_TimerInt,                  "Timer Interrupt         ",
        (void *)Test_Timer,                     "Timer Tout              ",
//UART
        (void *)Test_Uart0_Int,                 "UART0 Rx/Tx Int         ",
        (void *)Test_Uart0_Dma,                 "UART0 Rx/Tx DMA         ",
        (void *)Test_Uart0_Fifo,                "UART0 Rx/Tx FIFO        ",
        (void *)Test_Uart0_AfcTx,               "UART0 AFC Tx            ",
        (void *)Test_Uart0_AfcRx,               "UART0 AFC Rx            ",
        (void *)Test_Uart1_Int,                 "UART1 Rx/Tx Int         ",
        (void *)Test_Uart1_Dma,                 "UART1 Rx/Tx DMA         ",
        (void *)Test_Uart1_Fifo,                "UART1 Rx/Tx FIFO        ",
        (void *)Test_Uart1_AfcTx,               "UART1 AFC Tx            ",
        (void *)Test_Uart1_AfcRx,               "UART1 AFC Rx            ",
        (void *)Test_Uart2_Int,                 "UART2 Rx/Tx Int         ",
        (void *)Test_Uart2_Dma,                 "UART2 Rx/Tx DMA         ",
        (void *)Test_Uart2_Fifo,                "UART2 Rx/Tx FIFO        ",
//USB
        (void *)Test_USBFIFO,                   "USB FIFO Test           ",
//WDT
        (void *)Test_WDT_IntReq,                "WDT INT Request         ",
//ETC
        (void *)Test_XBREQ,                     "External Bus Reqest     ",
        (void *)Test_NonalignedAccess,          "NonAlgined Access       ",
        (void *)Test_PD6710,                    "PC Card (PD6710)        ",
        (void *)ReadPageMode,                   "Read Page Mode          ",
        (void *)Test_SwiIrq,                    "SWI                     ",
        (void *)Test_WaitPin,                   "External Wait           ",
        (void *)Test_ISram,                     "Stone Test              ",
        (void *)Test_NecInterrupt,              "ETC NEC Int             ",
        (void *)Test_BattFaultInterrupt,        "nBATT_FAULT int         ",
//NAND, NOR Flash
```

```
    (void *)K9S1208_PrintBadBlockNum,    "NAND View Bad Block   ",
    (void *)K9S1208_PrintBlock,          "NAND View Page       ",
    (void *)K9S1208_Program,             "NAND Write           ",
    (void *)TestECC,                     "NAND ECC             ",
    (void *)ProgramFlash,                "NOR Flash Program    ",
    0,0
};
//-------------------------------------------------------------------------------------------------------------
// 主程序
//-------------------------------------------------------------------------------------------------------------
void main(void)
{
    int i;
    MMU_Init();                                  //内存管理初始化
    ChangeClockDivider(1,1);                     // 定义 FCLK、HCLK、PCLK 比例
                                                 //4:2:1
    ChangeMPllValue(0xa1,0x3,0x1);               // FCLK=202.8MHz
    Port_Init();                                 //I/O 口初始化
    Isr_Init();                                  //中断初始化
    Uart_Init(0,115200) ;                        //选时钟 PCLK，波特率 115200
    Uart_Select(0);                              //选串口 0
    Check_PowerOffWakeUp();                      //唤醒电源进入正常工作状态
      while(1)
    {
       i = 0;
       while(1)
       {
          Uart_Printf("%2d:%s",i,function[i][1]);  //在超级终端上显示主菜单
          i++;
          if((int)(function[i][0])==0)             //显示结束跳出
          {
             Uart_Printf("\n");
             break;
          }
          if((i%4)==0)
          Uart_Printf("\n");                        //每行显示 4 项
       }
       Uart_Printf("\nSelect the function to test : ");   //提示：选择某项实验
       i = Uart_GetIntNum();                        //读实验项目号放 i 中
       Uart_Printf("\n");                           //超级终端上显示内容回车换行
       rGPGCON = (rGPGCON & 0xfffffcff) | (1<<8);   //GPG4 做输出
       rGPGDAT = (rGPGDAT & 0xffef) | (1<<4);       // GPG4 输出 1 控制 LCD 显示开
         if(i>=0 && (i<(sizeof(function)/8)) )
//指针数组 function 中每行二列指针，每指针占 4 字节，总实验项数是 sizeof(function)/8
       ( (void (*)(void)) (function[i][0]) )();
```

```
//将 function[i][0]转化为函数的指针并执行该函数，参数为 0。
    }
}
```

读者对 2410test.mcp 项目应熟悉，该项目提供了 S3C2410 所有硬件资源的驱动程序，对编程有很大的帮助。

3.4　几个常用的输入/输出函数

嵌入式控制系统项目大多是软硬件结合的工程，程序需要反复调试才能最后完成，在调试过程中，“目标机”上运行的部分结果、变量、一些提示要通过串行口在“宿主机”上的超级终端上显示，通过键盘给“目标机”的数据、命令也要通过串行口下载到“目标机”。

在 2410test.mcp 项目中有一个头文件 2410lib.h，项目所需要的这些输入/输出函数都包括在其中，经常用到的几个函数如下：

```
//-------------------------------------------------------------------------
//      串口初始化，确定串口使用的时钟频率和波特率
//-------------------------------------------------------------------------
static int whichUart=0;
void Uart_Init(int pclk,int baud)
{
    int i;

    if(pclk == 0)
    pclk    = PCLK;
    rUFCON0 = 0x0;      //UART channel 0 FIFO control register, FIFO disable
    rUFCON1 = 0x0;      //UART channel 1 FIFO control register, FIFO disable
    rUFCON2 = 0x0;      //UART channel 2 FIFO control register, FIFO disable
    rUMCON0 = 0x0;      //UART chaneel 0 MODEM control register, AFC disable
    rUMCON1 = 0x0;      //UART chaneel 1 MODEM control register, AFC disable

//UART0
    rULCON0 = 0x3;      //Line control register : Normal,No parity,1 stop,8 bits
    rUCON0  = 0x245;    // Control register
    rUBRDIV0=( (int)(pclk/16./baud+0.5) -1 );     //Baud rate divisior register 0

//UART1
    rULCON1 = 0x3;
    rUCON1  = 0x245;
    rUBRDIV1=( (int)(pclk/16./baud) -1 );
//UART2
    rULCON2 = 0x3;
```

```
    rUCON2    = 0x245;
    rUBRDIV2=( (int)(pclk/16./baud) -1 );
    for(i=0;i<100;i++);
}

//-------------------------------------------------------------------------
//   串行通道选择，S3c2410 有 3 个串行通道，返回通道号
//-------------------------------------------------------------------------
void Uart_Select(int ch)
{
    whichUart = ch;
}

//-------------------------------------------------------------------------
//   等串行通道发送缓冲器空
//-------------------------------------------------------------------------
void Uart_TxEmpty(int ch)
{
    if(ch==0)
        while(!(rUTRSTAT0 & 0x4)); //Wait until tx shifter is empty.

    else if(ch==1)
        while(!(rUTRSTAT1 & 0x4)); //Wait until tx shifter is empty.

    else if(ch==2)
        while(!(rUTRSTAT2 & 0x4)); //Wait until tx shifter is empty.
}

//-------------------------------------------------------------------------
//   从串行通道接收一个字节数据或一个字符，没有数据时会一直等。
//-------------------------------------------------------------------------
char Uart_Getch(void)
{
    if(whichUart==0)
    {
        while(!(rUTRSTAT0 & 0x1)); //Receive data ready
        return RdURXH0();
    }
    else if(whichUart==1)
    {
        while(!(rUTRSTAT1 & 0x1)); //Receive data ready
        return RdURXH1();
    }
    else if(whichUart==2)
    {
        while(!(rUTRSTAT2 & 0x1)); //Receive data ready
        return RdURXH2();
```

```
    }
}

//-------------------------------------------------------------------------
//    从串行通道接收一个字节数据或一个字符，没有数据时会返回 0
//-------------------------------------------------------------------------
char Uart_GetKey(void)
{
    if(whichUart==0)
    {
        if(rUTRSTAT0 & 0x1)       //Receive data ready
            return RdURXH0();
        else
            return 0;
    }
    else if(whichUart==1)
    {
        if(rUTRSTAT1 & 0x1)       //Receive data ready
            return RdURXH1();
        else
            return 0;
    }
    else if(whichUart==2)
    {
        if(rUTRSTAT2 & 0x1)       //Receive data ready
            return RdURXH2();
        else
            return 0;
    }
}

//-------------------------------------------------------------------------
//     按格式在超级终端上显示，意义同 C 语言中 Printf(char *fmt,...)
//-------------------------------------------------------------------------
void Uart_Printf(char *fmt,...) // ...表示可变参数(多个 fmt 格式参数组成一个列表)
                                //不限个数和类型，fmt 格式只有字符串和%格式两种
{
    va_list ap;                 //定义指向可变参数列表的指针
    char string[256];

    va_start(ap,fmt);           //将第一个可变参数的地址付给 ap，ap 指向可变参数列表的开始

    vsprintf(string,fmt,ap);  //将参数 ap 指向的可变参数一起转换成 fmt 格式字符串，放//string 数组
                                中，其作用同 sprint()，只是参数类型不同
    Uart_SendString(string);  //把格式化字符串从开发板串口送出去
    va_end(ap);                 //恢复系统堆栈指针
```

```
//-------------------------------------------------------------------------
//    看门狗电路启动
//-------------------------------------------------------------------------
void Timer_Start(int divider)   //0:16us,1:32us 2:64us 3:128us
{
    rWTCON = ((PCLK/1000000-1)<<8)|(divider<<3);   //Watch-dog timer control register
    rWTDAT = 0xffff;   //Watch-dog timer data register
    rWTCNT = 0xffff;   //Watch-dog count register

      //Watch-dog timer enable & interrupt   disable
//    rWTCON = rWTCON |(1<<5) & !(1<<2);
    rWTCON = rWTCON | (1<<5) | ~(1<<2);
}
//-------------------------------------------------------------------------
//    看门狗电路停止
//-------------------------------------------------------------------------
int Timer_Stop(void)
{
    rWTCON = ((PCLK/1000000-1)<<8);
    return (0xffff - rWTCNT);
}
//-------------------------------------------------------------------------
//    MPLL 时钟配置
//-------------------------------------------------------------------------
void ChangeMPllValue(int mdiv,int pdiv,int sdiv)
{
    rMPLLCON = (mdiv<<12) | (pdiv<<4) | sdiv;
}


//-------------------------------------------------------------------------
//    FCLK:HCLK:PCLK 比例配置
//-------------------------------------------------------------------------
void ChangeClockDivider(int hdivn,int pdivn)
{
     // hdivn,pdivn FCLK:HCLK:PCLK
     //        0,0              1:1:1
     //        0,1              1:1:2
     //        1,0              1:2:2
     //        1,1              1:2:4
    rCLKDIVN = (hdivn<<1) | pdivn;

    if(hdivn)
        MMU_SetAsyncBusMode();
    else
        MMU_SetFastBusMode();
```

```
}
//-------------------------------------------------------------------------
//      UPLL 时钟配置
//-------------------------------------------------------------------------
void ChangeUPllValue(int mdiv,int pdiv,int sdiv)
{
    rUPLLCON = (mdiv<<12) | (pdiv<<4) | sdiv;
}
```

3.5　DEF.H 头文件

为了简化程序中数据类型的书写，在 2410test.mcp 中定义了一个头文件，具体如下：

```
//-------------------------------------------------------------------------
//    Def.h
//-------------------------------------------------------------------------
#ifndef __DEF_H__
#define __DEF_H__
#define U32 unsigned int
#define U16 unsigned short
#define S32 int
#define S16 short int
#define U8   unsigned char
#define S8   char
#define TRUE      1
#define FALSE     0
#endif /*__DEF_H__*/
```

3.6　习题

1. 下载随书提供软件包，打开 2410addr.h 头文件，仔细阅读程序，回答以下问题：

(1) 2410addr.h 头文件中，寄存器大约有多少类？每个类的定义有什么特点？

(2) 2410addr.h 头文件中，中断向量(PISR_XX)有多少个?有几个中断共用一个中断向量?从字面理解每一个中断向量处理什么中断。

(3) 中断挂起寄存器(PENDING)的相应位等于 1，表示当前正在响应的中断是什么中断，在 2410addr.h 头文件中，中断挂起寄存器(PENDING)的相应位是如何定义的？

(4) 在 S3C2410 中，中断屏蔽分二级管理，在 2410addr.h 头文件中，总中断屏蔽 BIT_ ALLMSK 和子中断屏蔽 BIT_SUB_ALLMSK 是如何定义的?

(5) 从字面简单了解各类常用寄存器，如：LCD CONTROLLER、UART、PWM TIMER、I/O PORT、ADC、SPI、ISR 等。

2. 下载随书提供软件包，打开 2410test.mcp 文件，熟悉以下内容：

(1) 2410test.mcp 可以完成多少项硬件驱动试验？从字面简单了解各项试验的目的。

(2) 2410test.mcp 除 2410addr.h 外还包含头文件：def.h、option.h、2410lib.h。今后编程经常用到它们，打开这些头文件，熟悉其中的内容。