# 嵌入式系统设计与应用[共5篇]

来源：网络 作者：独酌月影 更新时间：2024-08-03

*第一篇：嵌入式系统设计与应用第一章：嵌入式系统定义：以应用为中心，以计算机技术为基础，软硬件可裁剪，适应应用系统对功能，可靠性，成本，体积，功耗严格要求的专用计算机系统。（一切非PC计算机系统）嵌入式系统特点：“专用”计算机系统，运行环境...*

**第一篇：嵌入式系统设计与应用**

第一章：

嵌入式系统定义：以应用为中心，以计算机技术为基础，软硬件可裁剪，适应应用系统对功能，可靠性，成本，体积，功耗严格要求的专用计算机系统。（一切非PC计算机系统）嵌入式系统特点：“专用”计算机系统，运行环境差异很大，比通用PC系统资源少，功耗低，体积小，集成度高，成本低，具有完整的系统测试和可靠性评估体系，具有较长的生命周期，需要专用开发工具和方法进行设计，包含专用调试电路，多学科知识集成系统。嵌入式系统应用范围：汽车，工业控制，通信设备，消费电子，商业终端，航空航天，军事需求。

嵌入式系统的基本开发流程：系统定义与需求分析阶段，方案设计阶段，详细设计阶段，软、硬件集成测试阶段，系统功能性测试及可靠性测试阶段。

系统定义与需求分析阶段：对系统需求进行分析，制定系统的设计依据。方案设计阶段：确定系统初步设计方案并形成设计描述文档。详细设计阶段：完善初步方案，对方案实施详细设计。

软硬件集成测试阶段：对系统软硬件进行综合测试，验证系统设计功能。

系统功能性能测试及可靠性测试测试：对系统功能，性能，可靠性进行综合测评。

对于使用操作系统的嵌入式系统来说,嵌入式系统软件结构一般包含4个层面：板级支持包层，实时操作系统(RTOS)层，应用程序接口(API)层，应用程序层。有些资料将应用程序接口API归属于OS层，按3层划分的应用程序控制系统的运作和行为；操作系统与硬件无关，不同的嵌入式操作系统其组成结构也不尽相同 嵌入式操作系统种类繁多,大体分为两种：商用型和免费型

商用型：VxWorks，Windows CE，pSoS，Palm OS，OS-9，LynxOS，QNX和LYNX 免费型：Linux和uC/OS—II uC/OS—II具有执行效率高，占用空间小，可移植性及扩展性强，实施性能优良，稳定性和可靠性良好等特点。其内核采用微内核结构，将基本功能(如进程管理，存储管理，中断处理)放在内核中，留给用户一个标准API函数，并根据各个任务的优先级分配CPU时间。交叉开发环境：交叉开发是指一个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调试的开发方式，它通常采用宿主机/目标机模式。

第二章：

RISC是精简指令集

精简指令集体系结构的优点：硬连线的指令译码逻辑，便于流水线执行，大多数RISC指令为单周期执行。

精简指令集处理器的优点：处理器关心面积小，开发时间缩短，开发成本降低，容易实现高性能，低成本的处理器。

精简指令集体系结构缺点：与CISC相比，通常RISC的代码密度低；RISC不能执行x86代码；RISC给优化编译程序带来了困难

ＡＲＭ设计采用的RISC技术特征主要有：Load/Store体系结构，固定的32位指令，3地址指令格式。

ARM7TDM名称具体含义：ARM7：32位ARM体系结构4T版本；T：Thumb16位压缩指令集；D：支持片上Debug，使处理器能够停止以响应调试请求；M：增强型Multiplier，与前代相比具有较高的性能且产生64位的结果。I：EmbeddedICE硬件以支持片上断点和观察点。ARM7 3级流水线：（取指级，译码级，执行级）ARM9TDMI 流水线操作：（取指，译码，执行，缓冲/数据，回写）5级 ARM处理器核可工作两种状态：ARM状态和Thumb状态

从ARM进入Thumb状态，当操作数寄存器Rm的状态位bit[0]为1时，执行“BX Rm”指令进入Thumb状态

从Thumb进入ARM状态，当操作数寄存器Rm的状态位bit[0]为0时，执行“BX Rm”指令进入ARM状态

ARM处理器工作模式（共7种）：除用户模式外的其他六种模式称为特权模式。特权模式：主要处理异常和监控调用（有时也称为软件中断），他们可以自由地访问系统资源和切换模式

ARM处理器总共有37个寄存器，均为32位 ARM状态下的通用寄存器分为3类： 未分组寄存器:R0~~R7（为公用寄存器）

分组寄存器：R8~~R14

R13通常用于堆栈指针SP

R14用做子程序链接寄存器

程序计数器：R15(PC)

用做程序计数器

ARM程序状态寄存器中

条件码标志（N Z C V）

N——在结果是带符号的二进制补码的情况下，结果为负，N=1 否则为0 Z——结果为0 Z=1 否则为0 C——针对加法：产生进位

C=1 否则为0

针对减法：产生借位

C=0 否则为1

针对有移位操作的非加减法指令

C为移位操作中最后移出位的值

对于其他指令

C通常不变

V——对于加减法指令

操作数和结果为带符号的整数时，产生溢出

V=1 否则为0

对于其他指令

V通常不发生变化 ARM的异常中断响应过程： 一：将CPSR的内容保存到将要执行的异常中断对应的SPSR中，以实现对处理器当前状态，中断屏蔽字以及各条件标志位的保存。二：设置当前状态寄存CPSR中的相应位：

设置CPSR模式控制位CPSR[4:0]，使处理器进入相应的执行模式

当进入Reset或FIQ模式时，还要设置中断标志位（CPSR[6]=1）禁止FIQ中断，否则其值不变

设置中断标志位（CPSR[7]=1），禁止IRQ中断

三：将寄存器LR-设置成异常返回地址 return link（对于某些异常中断，保存在LR-的值可能与实际返回的地址有一个常数的偏移量），使异常处理程序执行完后能正确返回原程序

四：给程序计数器PC强制赋值，使程序从相应的向量地址开始执行中断处理程序。

非向量中断和中断向量的区别和联系

异常中断的优先级：复位（最高），数据异常中断，FIQ，IRQ，取值指异常中止，SWI未定义指令

ARM支持的数据类型（6种）：8位有符号和无符号字节

16位有符号和无符号半字，以2字节的边界定位

32位有符号和无符号半字，以4字节的边界定位

ARM存储器组织：以字节为单位寻址的存储器中有“小端”和“大端”两种方式存储字 小端格式：较高的有效字节存放在较高的存储器地址，较低的有效字节存放在较低的存储器地址

大端格式：较高的有效字节存放在较低的存储器地址，较低的有效字节存放在较高的存储器地址

ARM处理器能方便地配置为其中任何一种存储器方式，但他们的缺省设置为小端格式（71页有题）

ARM7TDM内核的重要特性：53页最上面

第三章：

指令分类中基本指令格式

S

可选后缀，若指定S，则根据指令执行结果更新CPSR中的条件码 ARM寻址方式

立即寻址有选择题

寄存器间接寻址：ARM的数据传送指令都是基于寄存器间接寻址，即通过Load/Store完成对数据的传送操作

103页举例

可能为考题

伪操作

是ARM汇编语言程序里的一些特殊指令助记符，它的作用主要是为完成汇编程序做各种准备，在源程序进行汇编时由汇编程序处理，而不是在计算机运行期间由机器执行 ARM嵌入式系统程序设计可以分为ARM汇编语言程序设计、嵌入式C语言程序设计以及C语言与汇编语言的混合编程。

ARM汇编程序中

AREA指示符定义本程序段位代码段

即申请一个定义段 161页程序

可能考

嵌入式C语言程序设计中修饰符：interrupt、near、far、huge Interrupt在函数修饰为中断函数，没有输入和输出参数 第三章课后习题见李向妮笔记

第四章

DMA

I2C

I2S 基于S3C44B0X的最小系统设计：

嵌入式最小系统是指保证嵌入式微处理器可靠工作所必需的基本电路组成的系统，通常包括处理器单元、时钟单元、复位单元、、存储器单元、供电电源和调试接口。

基于ARM的嵌入式最小系统基本组成包括：基于ARM核的微处理器、电源电路、复位电路、时钟电路、存储器电路（FLASH和SDRAM）、UART接口电路和JTAG调试接口

第五章：

uC/OS—II采用的抢占式内核是一个真正的实时操作系统

uC/OS—II基本特点：源码开放；可移植性；可裁剪；抢占式内核；可扩展的多任务；可确定的执行时间；中断管理；稳定性和可靠性

uC/OS—II的文件结构（与内核功能相关的文件）：任务管理；同步通信；内存管理；时间管理

uC/OS—II任务及其运行状态：

任务是一个简单的程序，对应于实际应用中的一个逻辑功能。对uC/OS—II来说,任务是系统运行的基本单元，系统以任务为单元分配内存资源和处理时间，每个任务都有自己独立的寄存器和栈空间。

任务看起来就像一个无限循环永不返回的函数，但是不同于函数的是，它有一套自己的内存空间，运行时完全占用处理器资源，在任意确定的时刻都处于休眠、就绪、运行、挂起以及中断服务这五种状态之一 图见书上337

第六章

uCLinux与标准Linux的最大区别就在于内存管理。标准Linux是针对有MMU的处理器设计的

uCLinux不使用虚拟内存管理技术，采用的是实存储器管理策略，也就是说uCLinux系统对内存的访问是直接的

uCLinux与标准Linux系统在进程的创建

进程的执行

进程的终止

上有着显著不同 基于uCLinux操作系统的应用开发环境一般是由目标系统硬件开发板和宿主PC机所构成。目标硬件开发板用于运行操作系统和系统应用软件，而目标板所用到的操作系统的内核编译、电子词典应用程序的开发和调试则需要通过宿主PC机来完成。目标板用来进行内核编译

PC机用来进行调试

移植就是使一个实时操作系统能够在某个微处理器平台上或者微控制器上运行。uCLinux移植包括3个层次的移植: 处理器结构层次移植、芯片层次移植、板级移植。

移植思路：开发环境确定以后，首先，要为uCLinux设计一个BootLoader，通过BootLoader来初始化硬件，引导uCLinux运行。BootLoader的设计可以在ads中或者Linux中实现。其次，针对硬件环境和设计的BootLoader修改uClinux内核。最后，在交叉编译环境下配置、编译、链接uClinux，下载编译得到的印象文件到FLASH，通过BootLoader来启动uCLinux。如果参数默认或者无参数，则先执行BootLoader，否则BootLoader第二位，参数执行为第一位。

**第二篇：基于嵌入式系统的图形界面应用设计范文**

目录 嵌入式系统概述...................................................................................................................4

1.1 嵌入式系统简介.......................................................................................................4 1.2 嵌入式系统的组成...................................................................................................5 1.3 本课题的背景和意义...............................................................................................5 2 硬件平台及ARM体系结构................................................................................................7

2.1 处理器选择...............................................................................................................7 2.2 ARM体系结构............................................................................................................7 3 软件平台...............................................................................................................................9

3.1嵌入式操作系统选择................................................................................................9 3.2 交叉编译环境的建立.............................................................................................10 3.2.1 上位机的软硬件配置.........................................................................................10 3.2.2 硬件连接与调试.............................................................................................11 3.2.3 配置TFTP及NFS服务...................................................................................13 3.2.4 安装交叉编译工具.........................................................................................15 4 移植 Bootloader.................................................................................................................16 4.1 Bootloader 概述.................................................................................................16 U-boot 简介...................................................................................................................17 4.2.1 U-boot 的获取.................................................................................................17 4.2.2 U-boot 目录结构.............................................................................................17 U-boot 的启动过程及工作原理...................................................................................18 4.3.1 启动模式介绍...................................................................................................18 4.3.2 启动阶段1分析...............................................................................................19 4.3.3 启动阶段2分析...............................................................................................20 U-boot的移植过程........................................................................................................20 4.4.1 准备工作...........................................................................................................20 4.4.2 添加支持 NAND Flash 启动功能...................................................................21 4.4.3 添加 NAND Flash 读写功能.......................................................................22 4.4.4 修改 U-boot环境变量保存方式....................................................................22 4.4.5 加入 NAND Flash 闪存型号支持...............................................................23 4.5 U-boot 的烧写及测试.........................................................................................23 4.6 设置U-boot环境变量..................................................................................................24 5 Linux 内核的移植.............................................................................................................25 Linux 内核的结构.........................................................................................................25 Linux 启动过程简析.....................................................................................................26 Linux内核的移植过程..................................................................................................26 5.3.1 选择参考板.......................................................................................................26 5.3.2 修改 NAND Flash 分区信息...........................................................................26 5.3.3 关闭 ECC 校验.................................................................................................27 5.4 CS8900a网卡的移植过程....................................................................................28 5.4.1 修改硬件地址映射...........................................................................................28 5.4.2 添加 CS8900A 内核编译项.............................................................................28 5.5 Linux 内核的剪裁配置.......................................................................................29 5.5.1 使用配置菜单...........................................................................................................29 5.5.2 基本配置选项...................................................................................................30 5.5.3 驱动程序配置选项...................................................................................................31 5.5.4 保存配置文件...................................................................................................31 5.5.5 编译 Linux 内核.............................................................................................31 5.6 内核的下载及启动...............................................................................................32 5.6.1 将引导信息加入内核映像...............................................................................32 5.6.2 内核映像的下载及运行...................................................................................32 6 建立根文件系统.................................................................................................................33 6.1 根文件系统概述...................................................................................................33 6.1.1 根文件系统简介...........................................................................................33 6.1.2 NFS 文件系统与Cramfs文件系统...................................................................33 6.2 建立Linux根文件系统目录...............................................................................33 7 心得体会.............................................................................................................................34 基于嵌入式系统的图形界面应用设计 嵌入式系统概述

1.1 嵌入式系统简介

嵌入式系统是一种以应用为目的，软硬件可裁减，适应应用系统对功能、成本、体积、功耗严格要求的专用计算机系统。随着计算机的发展和应用的普及，嵌入式系统取得了迅猛的发展，嵌入式产品已经渗透到社会经济、军事、交通、通信等相关行业，而且深入到家电、娱乐等各个领域，掀起了一场数字化革命。嵌入式图形用户界面系统是嵌入式系统的一个重要组成部分，它将极大的促进嵌入式技术的发展和应用。

嵌入式系统是当前最热门、最有发展前途的IT应用技术之一。一方面，嵌入式系统广泛的应用于智能家电、手持终端、工业控制等专用设备上，通常这些设备的硬件资源(如处理器、存储器等)非常有限，并且对成本很敏感，有时对实时响应要求很高；另一方面，随着计算机技术的发展，越来越多的嵌入式系统设备需要良好的人机交互界面，这需要一个高性能、高可靠、占用系统资源少的用户图形界面的支持。为了适应嵌入式设备对人机交互界面的要求，本论文对基于嵌入式Linux图形用户界面的设计实现进行了研究。

一方面，嵌入式硬件性能不断提升，使得嵌入式设备上运行精美的图形用户界面成为可能；另一方面，嵌入式手持式消费电子产品的普及，例如PDA(个人数字助理)、智能手机、PMP(便携式多媒体播放器)，一个完善的嵌入式图形用户界面成为不可缺少的组成部分，嵌入式GUI(图形用户界面：Graphical User Interface)为嵌入式系统提供了一种应用于特殊场合的人机交互接口。

纵观国际相关产业在图形用户界面方面的发展现状，许多国际知名公司早已认识到GUI在嵌入式产品方面产生的强大增值功能，以及带动的巨大市场价值，因此在公司内部成立了专门从事GUI研究与设计的部门。

图形用户界面(GUI)是一种结合计算机科学、美学、心理学、行为学，以及商业领域需求分析的人机系统工程。这种面向用户的系统工程设计目的是优化产品性能，使操作更人性化，减轻使用者的认知负担，使其更适合用户的操作需要，直接提升产品的市场竞争力。

图形用户界面的广泛流行是当今计算机技术的重大成就之一，它极大的方便了非专业用户的使用，可以通过窗口、菜单方便的进行操作。一个图形用户界面通常由三个基本层次组成，也就是显示模型、窗口模型和用户模型。用户模型包含了显示和交互的主要特征，因此用户图形界面有时也仅指用户模型。

然而，对于嵌入式系统来说，由于其固有的体积、功耗以及价格的限制，使得传统的图形用户界面并不能直接应用于嵌入式系统[41，在嵌入式系统上实现GUI是一个具有挑战性的课题。总的来说，嵌入式GUI要求简单、直观、可靠、占用资源小且反应速度快，以适应系统硬件资源有限的条件嘲。另外，由于嵌入式系统硬件本身的特殊性，嵌入式GUI应具备高度可移植性与可裁减性，以适应不同的硬件条件和使用需求。具体的嵌入式GUI一般具备如下特点： 1.体积小

2.运行时耗用系统资源小

3.上层接口与硬件无关，高度可移植 4.高可靠性

5.在某些应用场合应具备实时性

目前，嵌入式GUI的开发正处于起步阶段，有许多技术难题尚待解决，因此对嵌入式图形系统的研究成为嵌入式系统研究及发展中的一个重要内容。

可以预见，随着电子、计算机等行业的高速发展，嵌入式系统将以它专用化，效率高的特性深入实际应用的各个领域，因而开发与改进嵌入式图形用户界面有着长远的意义。

1.2 嵌入式系统的组成

嵌入式系统一般可以分为四个部分：嵌入式处理器、嵌入式外围设备、嵌入式操作系统和嵌入式应用软件，如图1-1所示。

图1-1 嵌入式系统的组成

1.3 本课题的背景和意义

嵌入式导航计算机是飞机，车辆，导弹和船舶等运载体上的重要设备，主要任务是按照原定的计划和任务，以要求的精度，在一定时间内将载体引 导至目的地。嵌入式导航计算机主要分为两部分：硬件电路，嵌入式操作系 统。本课题的目的就是针对其硬件环境，搭建起一个高效、稳定的嵌入式操 作系统的平台。它具有通用操作系统的基本特点，能够有效管理复杂的系统 资源；能够快速的处理大量的信息；能够提供库函数、驱动程序、工具集以 及部分应用程序。在这个系统平台上可以运行导航程序，接受传感器的数据 经处理后得到任务所需要的信息，从而实施导航任务。

嵌入式Linux有着嵌入式导航计算机操作系统需要的很多特色：支持多 任务处理、中断处理及任务间通信，性能稳定，剪裁性好，开发与使用都很 方便。因此，本设计选用嵌入式Linux作为嵌入式导航计算机的操作系统，这对于实现导航计算机的高效率、低功耗具有现实意义。

图1-2 嵌入式导航计算机硬件平台结构图

1.4 本课题的主要工作和研究内容

本课题的最重目标是为嵌入式导航计算机移植Linux操作系统。通过参 阅大量文献，学习嵌入式Linux系统和 ARM 体系微处理芯片S3C2410的相 关知识，研究启动下载程序 Bootloader 和 Linux内核的基本工作原理，并且 搭建交叉编译平台，重点是移植Bootloader和内核，以及制作根文件系统。具体工作内容包括：

（1）学习Linux操作系统的知识。

（2）了解 ARM的体系结构和S3C2410芯片硬件结构。（3）完成交叉编译环境的建立。（4）修改并移植U-boot 1.2.0。

（5）修改和裁剪Linux 2.6.24.4内核，移植网卡驱动程序。（6）制作根文件系统。（7）编写应用程序进行测试。

（8）将内核和根文件系统部署到开发板。2 硬件平台及ARM体系结构

2.1 处理器选择

本设计的处理器选择高性能、低功耗的ARM9微处理器Samsung S3c2410。S3c2410 是著名半导体公司 Samsung 推出的一款 32 位 RISC 处理器。S3c2410的内核基于 ARM920T，带有MMU功能，主频高达203MHz，可以支持 Linux、WinCE 等主流嵌入式操作系统。同时它还采用了一种叫做Advanced Microcontroller Bus Architecture（AMBA）的新型总线结构。

此外S3c2410还集成了以下片上功能：

（1）16KB指令 Cache和16KB的数据Cache；（2）LCD控制器（支持STN和TFT）；

（3）4通道DMA；

（4）3通道UART；

（5）2通道USB；

（6）4路PWM和 1个内部时钟控制器；

（7）117个通用IO，24路外部中断；

（8）16位看门狗定时器；（9）RTC（实时时钟）；

（10）1通道IIC/IIS控制器；

（11）NAND Flash控制器；

（12）PLL数字锁相环。

S3c2410 将系统的存储空间分为 8 组（bank），每组大小是 128MB，共1GB。Bank0 到 Bank6 都采用固定 Bank 起始寻址，用于 ROM 或 SRAM。Bank7具有可编程的 Bank的起始地址和大小，用于ROM、RAM或SDRAM。S3c2410还支持从NAND Flash启动，NAND Flash具有容量大、比NOR Flash价格低等特点。系统采用NAND Flash与SDRAM相结合的方式，可以获得非常高的性价比。

2.2 ARM体系结构

ARM微处理器基本架构：

ARM9微处理器采用RISC体系结构：优先选取使用频最高的简单指令，避免复杂指令

RISC体系结构应具有如下特点：

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

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

5.所有的指令都可根据前面的执行结果决定是否被执行，从而提高指令的执行效率。

6.可用加载/存储指令批量传输数据，以提高数据的传输效率。7.可在一条数据处理指令中同时完成逻辑处理和移位处理。8.在循环处理中使用地址的自动增减来提高运行效率。ARM微处理器的寄存器结构：

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

2.31个通用寄存器，包括程序计数器（PC指针），均为32位的寄存器。3.6个状态寄存器，用以标识CPU的工作状态及程序的运行状态，均为32位，目前只使用了其中的一部分。

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

ARM微处理器的指令结构：

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

ARM9系列微处理器具有以下特点： 1.5级整数流水线，指令执行效率更高。2.提供1.1MIPS/MHz的哈佛结构。

3.支持32位ARM指令集和16位Thumb指令集。4.支持32位的高速AMBA总线接口。

5.全性能的MMU，支持Windows CE、Linux、Palm OS等多种主流嵌入式操作系统。

6.MPU支持实时操作系统。

7.支持数据Cache和指令Cache，具有更高的指令和数据处理能力。大小都为16K。8.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。

9.ARM9系列微处理器包含ARM920T、ARM922T和ARM940T三种类型，以适用于不同的应用场合。软件平台

3.1嵌入式操作系统选择

本设计的嵌入式操作系统选择嵌入式Linux操作系统。

嵌入式操作系统是一种支持嵌入式系统应用的操作系统软件，它是嵌入式系统(包括硬、软件系统)极为重要的组成部分，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等Browser。

一般情况下，嵌入式操作系统可以分为两类，一类是面向控制、通信等领域的实时操作系统，如 WindRiver 公司的 VxWorks、ISI 的 pSOS、QNX系统软件公司的QNX、ATI的 Nucleus等；另一类是面向消费电子产品的非实时操作系统，这类产品包括个人数字助理(PDA)、移动电话、机顶盒等。嵌入式 Linux 操作系统

Linux 的嵌入式改造主要围绕体积和实时性展开，目前已经有很多公司在进行这方面的工作，其中包括 RT-Linux，uClinux，Embedix，Xlinux，MidoriLinux和红旗嵌入式 Linux等等。

与目前市场上的众多商业的实时系统相比，嵌入式Linux除具有内核稳定，功能强大，支持多种硬件平台，兼容性好的优势外，还拥有以下的特点：(1)完全开放源代码

嵌入式 Linux完全开放其源代码，这使得修改，裁剪 Linux成为可能，开发者可以根据实际需要优化操作系统代码，降低整个系统的开销与能耗。(2)成本低

GPL协议保证了源自Linux的嵌入式Linux也是开放源代码的自由软件。而大多数嵌入式Linux使用的开发工具也是遵守GPL协议的，同样也可以免费获得。

(3)丰富的实用软件支持

Linux 提供了大量的实用程序和各种应用软件。这些软件的正确性和有效性都经过了实际检验，可以根据需要合理利用他们迅速构建嵌入式应用的软件环境。这样可以极大地减小嵌入式软件开发的时间和费用，提高系统可靠性。而商用的实时操作系统也试图提供各种常用软件工具包，但其数量是无法和Linux操作系统匹敌的。由此可见，选择嵌入式Linux操作系统，就有了丰富的资源保障，在节省成本的同时，提高了开发效率。

3.2 交叉编译环境的建立

采用交叉开发环境（Cross Development Environment）是嵌入式应用软

件开发时的一个显著特点，通常在通用计算机上编写程序，然后通过交叉编 译生成目标平台上可运行的二进制代码格式，最后再下载到目标平台上的特 定位置运行，交叉开发环境是指编译、链接和调试嵌入式应用软件的环境，它与运行嵌入式应用软件的环境有所不同，通常采用主机/目标及模式。交叉开发模型如图2-1所示：

图3-1 交叉开发模型

3.2.1 上位机的软硬件配置

硬件：

本课题用到一台通用PC机和一台笔记本电脑，其硬件配置如下：

PC机：

CPU：P IV 2.0G RAM：256MB 串口：RS-232 并口：25针母头 笔记本电脑： CPU：PM 705 RAM：768MB 网卡：10/100MBps自适应网卡 软件：

PC机的操作系统为Windows XP，装有DNW串口调试工具以及SJF2410 三星Flash烧写工具。前者用于串口调试，后者用于烧写Bootloader。

笔记本电脑的操作系统为 Ubuntu7.10，装有 GCC 等编译工具以及arm-linux-gcc交叉编译工具，并开启TFTP和 NFS服务。用于 Linux内核编 译和软件开发，并作为TFTP服务器和NFS主机。其中，Ubuntu7.10 是 Linux 的桌面发行版之一，是当今最为流行的桌面Linux 系统。使用 Linux 操作系统及其自带的工具，是目前最权威的嵌入式Linux系统开发方式，但是许多操作都是基于命令行的，所以需要扎实的Linux基础知识。

在 Ubuntu 中建立 arm用户，专门用于 ARM 开发。在 home 目录中建立下列几个子目录：

Boot：用于存放bootloader相关程序。Kernel：用于存放 Linux内核源码。FS：用于存放根文件系统相关的程序。Program：用于存放用户程序。

3.2.2 硬件连接与调试

硬件连接方式：

图3-2 硬件连接图

（1）开发板串口UART0通过交叉串口线与PC主机的 COM1口相连。

（2）开发板的JTAG口通过20PIN排线与SUPER JTAG调试头相连，再通过25PIN并口线连接到主机的LPT1口。

（3）开发板的网卡接口通过以太网线连接到路由器的LAN1口。

（4）笔记本的网卡接口通过以太网线连接到路由器的LAN2口。

（5）路由器的 WAN口连接到INTERNET。串口调试：

在本课题嵌入式系统中的目标开发板，采用串口调试的方法，即把串口当作目标开发板的显示终端，无论是打印输出，还是管理配置输入，都使用串口，这就需要主机系统装有串口调试工具。PC机中安装有DNW串口调试工具，在使用DNW之前，应当对PC机的串口进行设置。方法如下：

（1）XP系统中，右键单击“我的电脑”，选择“属性”。

（2）选择“硬件”——“设备管理器。

（3）选择“端口”——“串口（COM1）”，打开的对话框按图2-3设置。

图3-3 串口属性设置

运行 DNW 工具，选择菜单“Configuration”——“Options”，按图 2-4所示进行设置。

图 3-4

DNW 设置

每次使用DNW之前，应当设置DNW连接到串口。点击菜单中的 “Serial Port”——“Connect”，当DNW的标题栏出现[COM1,115200bps]的提示后，表明已经连接好，此时才可以使用DNW工具。

3.2.3 配置TFTP及NFS服务

TFTP 服务简介：

TFTP（Trivial File Transfer Protocol）协议即简单文件传输协议，是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。TFTP承载在UDP上，提供不可靠的数据流传输服务，不提供存取授权与认证机制，使用超时重传方式来保证数据的到达。

TFTP 服务在 Linux 系统中有客户端和服务器两个软件包。配置 TFTP服务，必须都安装好。

TFTP 服务安装与配置：

（1）Ubuntu中安装tftp工具只需在终端中键入命令： $ sudo apt-get install tftp tftpd 其中，前者是客户端，后者是服务器。

（2）Ubuntu是debian类的系统，默认是没有安装 inetd的，安装命令如下： $ sudo apt-get install netkit-inetd（3）在home目录里建立tftpboot 文件夹，命令如下： $ cd ~ $ sudo mkdir tftpboot $ sudo chmod 777 tftpboot 其中，参数 777 的意义是：根管理员、组和其他用户对 tftpboot 文件夹 的权限均为“可读、可写、可以执行”（4）修改/etc/inetd.conf，添加如下语句：

tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /home/arm/tftpboot 目的是指定 tftp 服务的根目录为/home/arm/tftpboot，修改/etc/inetd.conf 文件后应当重启 inted进程，命令如下： $ sudo /etc/init.d/inetd reload（5）重启 inted 进程后，配置即可生效，在 tftpboot 中建立文件 test 后，用

下列命令可以进行测试： $ cd ~ $ tftp 127.0.0.1 Tftp> get test 若可下载test 文件，则证明TFTP服务配置正确。NFS 服务简介：

NFS 就是 Network File System 的缩写，最早之前是由 Sun 这家公司所开发的。最大的功能就是可以透过网络，让不同的机器、不同的操作系统、可以彼此分享个别的档案(share files)。所以，可以简单的将它看做是一个文件服务器(file server)。通过 NFS 服务器可以让开发板将网络远端的 NFS 主机分享的目录，挂载到开发板当中，在开发板看起来，那个远端主机的目录就好像是自己的根目录一样，可以方便的进行开发调试。NFS 服务安装与配置：

（1）Ubuntu上默认是没有安装NFS服务器的，首先要安装NFS服务程序： $ sudo apt-get install nfs-kernel-server 在安装nfs-kernel-server时，apt 会自动安装nfs-common和portmap。这样，宿主机就相当于NFS Server。（2）配置/etc/exports：

NFS 挂载目录及权限由/etc/exports 文件定义。本课题要将 home 目录中 的/home/zp/share 目录让 192.168.0.\*的 IP 共享, 则在该文件末尾添加下列语句：

/home/arm/FS/myrootfs 192.168.0.2/10(rw,sync,no\_root\_squash)配置参数说明： rw：具有可擦写的权限。

sync：文件同步写入到内存和硬盘当中。

no\_root\_squash：若登陆共享目录的使用者是 root 的话，则他的权限将被限 制为匿名使用者，通常他的UID和GID都会变为nobody。（3）本地测试NFS：

输入以下命令可以将NFS根目录挂载到本地的/mnt 目录中： $ sudo mount 192.168.0.2:/home/arm/FS/myroot /mnt 此时/mnt 中的内容应当与NFS根目录中的内容一致。

3.2.4 安装交叉编译工具

交叉编译简介：

所谓交叉编译，简单的说，就是在一个平台上生成另一个平台上的可执行代码，比如在 PC平台上（X86 CPU）编译出能运行在以 ARM 为内核的CPU平台上的程序，一般选择GNU开发工具 gcc。GNU的开发工具都是免费的，遵循 GPL协议，任何人可以从网上获取。GNU 提供的编译工具包括汇编器as、c编译器gcc、c++编译器g++、连接器ld和二进制转换工具objcopy。出于兼容性和稳定性考虑，本课题选择目前比较稳定的版本 Cross-3.3.2 和Cross-3.4.1。

交叉编译器的安装及配置：

（1）获取arm-linux交叉编译工具：

登陆arm-linux项目组的FTP服务器：

ftp.arm.linux.org.uk/pub/armlinux/toolchain/

下载cross-3.3.2.tar.bz2和cross-3.4.1.tar.bz2。

（2）通过下列命令可以安装arm-linux交叉编译工具：

$ cp cross-3.4.1.tar.bz2 /

$ cd /

$ tar jxvf cross-3.4.1.tar.bz2

这样，交叉编译工具就被安装到了/usr/local/arm/3.4.1中。用同样的方法 可以安装cross-3.3.2版的交叉编译工具。

（3）设置环境变量：

修改home目录下的profile文件，加入如下代码，指明交叉编译工具的 目录。

# User specific environment and startup programs

export TARGET=arm-linux

export PRJROOT=/home/arm

export

PATH=$PATH:$HOME/bin:$PREFIX/bin:/usr/local/arm/3.4.1/bin:/sbin:/usr/ sbin:/usr/local/sbin

测试交叉编译器：

可以通过一个简单的程序测试安装好的交叉编译工具，看其能否正常工作。编写一个 hello.c源文件，通过以下命令进行编译，编译后生成名为Hello的可执行文件，通过 file 命令可以查看文件的类型。当显示以下信息是表明交叉编译工具正常安装了，通过编译生成了ARM体系可执行的文件。注意，通过该交叉编译器编译的可执行文件只能在 ARM 体系下执行，不能在基于X86的普通PC上执行。

$ arm-linux-gcc –o Hello hello.c

$ file Hello

Hello:ELF 32-bit LSB executable ,ARM, version 1(ARM), for GNU/Linux 2.4.3, dynamically linked(uses shared libs), not stripped 4 移植 Bootloader 4.1 Bootloader 概述

简单地说，Bootloader就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。

通常，Bootloader是严重地依赖于硬件而实现的，特别是在嵌入式世界。因此，在嵌入式世界里建立一个通用的Bootloader几乎是不可能的，不同处理器构架都有不同的 Bootloader。Bootloader 不但依赖于 CPU 的体系结构，而且依赖于嵌入式板级设备的配置。对于不同的嵌入式板而言，即使它们使用同一种处理器，要想让运行在一块板子上的Bootloader运行在另一块板子上，一般也要修改其源代码。

目前常用的Bootloader程序有以下几种： U-boot、VIVI、Blob和RedBoot。其中，U-boot 功能丰富，且对于ARM体系支持良好，事实上，它已成为ARM平台上标准Bootloader。因此，本课题选用U-boot 作为移植对象。U-boot 简介

U-boot 是德国 DENX 小组的开发用于多种嵌入式 CPU 的 Bootloader 程序，U-boot 不仅仅支持嵌入式Linux系统的引导，还支NetBSD，VxWorks，QNX，ARTOS，LynxOS 等嵌入式操作系统。U-boot 除了支持 ARM 系列的处理器外，还能支持 MIPS、x86、PowerPC、NIOS、XScale 等诸多常用系列的处理器。

4.2.1 U-boot 的获取

U-boot 的源码可以从sourceforge网站下载，网址为：

http://sourceforge.net/project/u-boot。

下载的文件为u-boot-1.2.0.tar.bz2，用以下命令将其解压。

$ tar jcvf u-boot-1.2.0.tar.bz2 /home/arm/boot/ 4.2.2 U-boot 目录结构

解压后，在 U-boot 顶层目录下有 18 个子目录，分别存放和管理不同的源码。这些目录中所要存放的文件有其规则，可以分为3类，如表3-1所示：

第一类目录与处理器体系结构或开发板硬件直接相关。第二类目录是一些通用的函数或者驱动程序。第三类目录是U-boot 的应用程序、工具或者文档。

表4-1 U-boot顶层目录下部分目录的存放规则 U-boot 的启动过程及工作原理

4.3.1 启动模式介绍

Bootloader 都包含两种不同的操作模式：“启动加载”模式和“下载”模式，这种区别仅对于开发人员才有意义。但从最终用户的角度看，Boot Loader 的作用就是用来加载操作系统，而并不存在所谓的启动加载模式与下载工作模式的区别。

启动加载（Bootloading）模式：这种模式也称为“自主”（Autonomous）模式。也即 Bootloader 从目标机上的某个固态存储设备上将操作系统加载 到 RAM中运行，整个过程并没有用户的介入。这种模式是 Boot Loader 的

正常工作模式，因此在嵌入式产品发布的时侯，Bootloader 显然必须工作在这种模式下。

下载（Downloading）模式：在这种模式下，目标机上的 Boot Loader 将通过串口连接或网络连接等通信手段从主机（Host）下载文件，比如：下载内核映像和根文件系统映像等。从主机下载的文件通常首先被 Bootloader保存到目标机的 RAM 中，然后再被 Bootloader 写到目标机上的 FLASH 类固态存储设备中。Bootloader 的这种模式通常在第一次安装内核与根文件系统时被使用； 此外，以后的系统更新也会使Bootloader的这种工作模式。工作于这种模式下的 Bootloader 通常都会向它的终端用户提供一个简单的命令行接口。

U-boot 这样功能强大的Bootloader 同时支持这两种工作模式，而且允许用户在这两种工作模式之间进行切换。

大多数Bootloader都分为阶段1(stage1)和阶段2(stage2)两大部分，U-boot也不例外。依赖于CPU体系结构的代码（如CPU初始化代码等）通常都放在阶段1中且通常用汇编语言实现，而阶段 2 则通常用C语言来实现，这样可以实现复杂的功能，而且有更好的可读性和移植性。

图 4-1 U-boot启动代码流程图

4.3.2 启动阶段1分析

如果 S3C2410 被配置成从 NAND 闪存启动，上电后，S3C2410 的 NAND 闪存控制器会自动把 NAND 闪存中的前 4K 数据搬移到内部 RAM中，并把 0x00000000 设置为内部 RAM 的起始地址，CPU 从内部 RAM 的0x00000000 位置开始启动。因此要把最核心的启动程序放在 NAND 闪存的前4K中。由于NAND闪存控制器从NAND闪存中搬移到内部RAM的代码是有限的，所以，在启动代码的前 4K里，必须完成 S3C2410 的核心配置，并把启动代码的剩余部分搬到 RAM 中运行。这前4K完成的主要工作就是 U-boot 启动的第一个阶段(stage1)。

U-boot 的stage1代码通常放在start.s文件中，它用汇编语言写成。此阶段要完成的主要工作如下：

（1）设置异常向量，当发生异常时，执行 cpu/arm920t/interrupts.c 中定义的中断处理函数。

（2）设置CPU的模式为SVC（管理模式，操作系统使用的保护模式）（3）关闭看门狗。（4）禁掉所有中断。

（5）设置 cpu 频率，默认频率比为 FCLK:HCLK:PCLK = 1:2:4，默认FCLK的值为120 Mhz，该值为S3C2410手册的推荐值。（6）调用cpu初始化函数cpu\_init\_crit。其中一个功能是设置CP15寄存 器，失效指令(I)Cache和数据(D)Cache后，禁止MMU与 Cache。（7）重定向，将 NAND Flash代码复制到 RAM，其中有以下两个个步 骤：

（a）通过copy\_myself子程序，把数据从Nand Flash中拷贝到RAM。

（b）配置栈空间，配置代码段的开始地址、动态内存区长度、全局数据 大小以及分配IRQ 和FRQ的栈空间。

（8）BSS（Block Started by Symbol）段清零。（9）进入C代码：

ldr pc, \_start\_armboot \_start\_armboot:.word start\_armboot 其中 start\_armboot 是 U-boot 运行的第一个 C 程序，在 lib\_arm/board.c 文件中定义。随后进入第二阶段。

4.3.3 启动阶段2分析

lib\_arm/board.c 中的 start armboot 是 C 语言开始的函数，也是整个启动代码中C语言的主函数，同时还是整个 U-boot(armboot）的主函数，该函数主要完成如下操作：（1）定义初始化函数表。

（2）NAND Flash初始化，利用 nand\_init()函数实现。（3）环境变量初始化，利用env\_relocate()函数实现。（4）外围设备初始化，利用 devices\_init()函数实现。（5）使能中断，利用enable\_interrupts()函数实现。（6）初始化网络设备。

（7）进入U－boot 的命令循环，接受用户输入的命令，执行相应的工作。

U-boot的移植过程

移植U-boot 的主要工作就是添加开发板硬件相关的文件、配置选项，然后进行编译。

4.4.1 准备工作

（1）建立开发板编译项，在顶层Makefile中加入如下两行： LJD2410\_config : unconfig @$(MKCONFIG)$(@:\_config=)arm arm920t LJD2410 NULL s3c24x0 各项意义如下：

arm：CPU 的架构(ARCH)arm920t：CPU 的类型(CPU),其对应于 cpu/arm920t 子目录。LJD2410：开发板的型号(BOARD),对应于 board/crane2410 目录。NULL：开发者/或经销商(vender)。s3c24x0：片上系统(SOC)。

（2）在 board 子目录中建立 LJD2410开发板目录： $ cp rf board/smdk2410 board/LJD2410 $ cd board/LJD2410 $ mv smdk2410.c LJD2410.c（3）在 include/configs/中建立配置头文件：

$ cp include/configs/smdk2410.h include/configs/LJD2410.h（4）测试编译能否成功： $ make distclean $ make LJD2410\_config $ make CROSS\_COMPILE=arm-linux-如果编译成功，证明已经建立好了LJD2410的编译项，但是还要进行进一步的修改，因为现在的代码是完全拷贝 smdk2410 开发板的，还不能工作在LJD2410板上。接下来要按照 LJD2410板的硬件配置来进一步移植。（5）调整SDRAM的刷新率，修改 lowlevel\_init.S： #define REFCNT 1268 在smdk2410.c中调整 HCLK为 100MHz： /\*Fout = 200MHz \*/ #define M\_MDIV 0x5C #define M\_PDIV 0x4 #define M\_SDIV 0x0 4.4.2 添加支持 NAND Flash 启动功能

由于U-boot 不支持从NAND Flash启动，所以将程序复制到RAM里面去需要新加代码实现，一般通过 copy\_myself 函数实现。这可以参考 VIVI的copy\_muself代码将其添加到Start.S中，详见附录 A-1。

在Start.S中调用了nand\_reak\_ll函数，该函数用于NAND Flash读操作，在U-boot 中没有定义，需要新加代码实现，该函数的实现可以参考VIVI源代码。将VIVI/s3c2410/nand\_read.c 复制到LJD2410目录内即可。

由于使用了新的 Flash 读函数，在编译时需要重新链接，修改 LJD2410目录中的Makefile文件，将原先的OBJS := myboard.o flash.o 改为：OBJS := myboard.o nand\_read.o。

S3c2410处理器带有NAND Flash控制器，但是U-boot 没有定义其寄存器地址，修改 include/s3c2410.h文件，加入如下代码： #define oNFCONF 0x00 #define oNFCMD 0x04 #define oNFADDR 0x08 #define oNFDATA 0x0C #define oNFSTAT 0x10 #define oNFECC 0x14 4.4.3 添加 NAND Flash 读写功能

U-boot 运行至第二阶段进入 start\_armboot()函数。其中 nand\_init()函数是对 NAND Flash 的最初初始化函数。其调用与CFG\_NAND\_LEGACY 宏有关，如果没定义

CFG\_NAND\_LEGACY 这个宏，就按照 start\_armboot()调用 drivers/nand/nand.c 中的 nand\_init 函数(该函数在 1.2.0 已经被实现)默认规定，但还有个 board\_nand\_init()函数没实现，需自己添加。如果定义CFG\_NAND\_LEGACY，就不使用默认的nand\_init，而调用自己写的nand\_init函数了，本课题选择第二种方式。

在/drivers/nand\_legacy/nand\_legacy.c 中添加 NAND Flash 初始化函数nand\_init，详见附录 A-2。

可以看到 nand\_init()调用 NF\_Init()函数，使能 NAND Flash 控制器和 NAND Flash；调用 NF\_Reset()函数置位，NF\_WaitRB()查询 NAND Flash 的状态，最后再调用 nand\_probe((ulong)nand)函数探测 NAND Flash。

在 include/configs/smdk2410.h 文件的后半部原先有 Flash 的参数，删除它，并加入NAND Flash的参数，并且开启一些命令宏。

4.4.4 修改 U-boot环境变量保存方式

由于本课题使用NAND Flash作为外存储器，所以U-boot 的参数存储函数应当进行适当的修改。

在/common/env\_common.c里添加default\_env函数，此函数的作用是对环境变量保存方式的简单初始化。这个文件中还定义了U-boot 保存环境变量的底层函数。其中/\* Environment not changable \*/行下面的部分应当用 default\_env 函数代替。这样，就可以在 U-boot 命令行中实现对环境变量的设置与保存。文件 /common/env\_nand.c 中 包 含 了 Flash 擦 写 函 数，结合 CFG\_NAND\_LEGACY这个宏，添加代码实现 NAND Flash的擦写功能。初 始化环境仍用 default\_env函数替换。

4.4.5 加入 NAND Flash 闪存型号支持

在/include/linux/mtd/

nand\_ids.h

中

对

nand\_flash\_dev nand\_flash\_ids结构体的赋值进行修改，加入下列代码：

{“Samsung K9F1208U0B”, NAND\_MFR\_SAMSUNG, 0x76, 26, 0, 3, 0x4000, 0}, 这样，U-boot 就可以正确识别此款NAND Flash芯片。

4.5 U-boot 的烧写及测试

若开发板中没有任何程序，则不能启动，需要先将 U-boot 烧写到 Flash中。常用的烧写方法有如下几种：（1）将Flash取下，用编程器烧写。（2）通过串口线烧写。（3）通过JTAG调试接口烧写。

本课题采用第三种方法。通过JTAG接口烧写的优点是操作简单，但是烧写速度较慢，总体来说是一种非常经济实用的方法。具体操作如下：（1）连接好开发板和PC主机，主机安装并口设备驱动程序。

（2）将 u-boot.bin 拷贝至 sjf2410 目录下，用以下命令运行 sjf2410：

sjf2410 /f:u-boot.bin（3）sjf2410程序启动后，会有三个选项，依次为：

（a）选择Flash芯片型号，（b）选择程序类型，（c）选择烧写起始地址。

本课题全部选择“0”即可。

（4）烧写完毕后选择“2”退出sjf2410。

烧写完成后，断开JTAG线，PC机运行DNW串口调试软件。重启开发板后，DNW中会输出以下信息，表明U-boot可以正常启动：

其中，“LJD2410 >”即系统提示符，在此可以输入 U-boot 的命令并执行。U-boot 提供了几十个常用的命令，通过这些命令，可以对开发板进行调试，可以引导Linux内核，还可以擦写 Flash 完成系统部署等功能。

输入“help”命令，可以看到U-boot 当前的所有命令列表，如表3-2所示，每一条命令后面是简单的说明。

表4-2 U-boot中几个常用命令及其说明

4.6 设置U-boot环境变量

U-boot的环境变量存储在NAND Flash中U-boot程序映像后面的128Kb字节中，这部分被称为“变量区”。

本课题中，设置U-boot 环境变量共有两种方法：（1）在板级头文件LJD2410.h中定义有相关的环境变量宏。

这类的宏名称中以“CONFIG\_”开头，区别于以“CFG\_”开头的内部变量宏。以开发板的IP地址为例，LJD2410.h中有如下代码：

#define CONFIG\_IPADDR 192.168.0.10 这种方法定义简便，但是每次更改环境变量必须重新编译、烧写U-boot程序，操作复杂，不方便调试。（2）使用命令设置环境变量。

这种方法得益于先前所做的移植工作，优点是操作简便，可以在线设置，重启开发板即可生效。还是以设置开发板 IP 地址为例，U-boot 提示符下输入以下命令：

LJD2410 > setenv ipaddr 192.168.0.10 LJD2410 > saveenv 系统显示：

Saving Environment to NAND...Erasing Nand...Writing to Nand...done 表明新设置的环境变量已保存至Flash中的变量区。5 Linux 内核的移植

Linux 内核的结构

在对Linux内核移植之前，首先要明确内核源码的基本组织情况，只有了解了各目录级代码的功能才能准确找到需要修改和改进的地方。

Linux 内核主要由 5 个子系统组成：进程调度、内存管理、虚拟文件系 统、网络接口、进程间通信。

Linux内核源码中几个主要的目录说明如下：

（1）/arch包含了所有硬件结构特定的内核代码。

Linux 系统能支持如此多平台的部分原因是因为内核把原程序代码清晰的划分为体系结构无关部分和体系结构相关部分。对于任何平台，都必须包含以下几个目录：

（a）boot:包括启动内核所使用的部分或全部平台特有代码。

（b）kernel:存放支持体系结构特有的(如信号处理和SMP)特征的实现。

（c）

lib:存放高速体系结构特有的(如strlen和 memcpy)通用函数的实现。

（d）mm:存放体系结构特有的内存管理程序的实现。

（e）math-emu:模拟 FPU 的代码。对于 ARM 处理器来说，此目录用mach-xxx代替。

（2）/drivers包含了内核中所有的设备驱动程序。

（3）/fs包含了所有的文件系统的代码。

（4）/include包含了建立内核代码时所需的大部分库文件。

该目录也包含了不同平台需要的库文件。比如，asm-arm是 arm平台需要的库文件。

（5）/init 包含了内核的初始化代码，内核从此处工作。这是研究核心如何工作的好起点。

（6）/ipc包含了进程间通信代码。

（7）/kernel包含了主内核代码。

（8）/mm包含了所有内存管理代码。

（9）/net 包含了和网络相关的代码。

（10）/documents包含了内核源码各个部分的说明文件。

通常，在每个目录下，都有一个 Kconfig 文件和一个Makefile文件，这两个文件都是编译时使用的辅助文件，仔细阅读这两个文件对弄清各个文件之间的联系和依托关系很有帮助；而且在有的目录下还有 Readme 文件，它是对该目录下的文件的一些说明，同样有利于我们对内核源码的理解。

显然，移植工作的重点就是移植arch目录下的文件。Linux 启动过程简析

Linux 内核启动就是引导内核映像启动的过程。典型的内核映像是zImage，包含自引导程序和压缩的vmlinux两部分。

启动过程从内核映像入口开始执行，解压 vmlinux并转到虚拟地址空间；再调用统一的内核启动函数 start\_kernel()，完成一系列基本初始化；随后启动一个叫做 init 的内核线程，完成挂载文件系统、初始化设备驱动和启动用户空间 init 进程等工作。

Linux内核的移植过程

5.3.1 选择参考板

内核的移植工作主要是修改跟硬件平台相关的代码，一般不涉及 Linux内核通用程序。移植的难度也取决于两种硬件平台的差异。Linux 对于特定硬件平台的软件叫做BSP（Board Support Package）.Linux 内核已经支持了各种体系的多种开发板，我们很容易从中找到与本课题类似的目标板，参考该目标板并做一定的修改，即可完成移植工作。选择参考板的原则如下：

（1）参考板与开发板具有相同的处理器，至少类似的处理器；

（2）参考板与开发板具有相同的外围接口电路，至少基本接口相同；

（3）Linux内核已经支持参考板，至少有非官方的补丁或者BSP；

（4）参考板Linux设备驱动工作正常，至少已经驱动基本接口。

根据以上原则，本课题选择SMDK2410作为参考板。修改顶层Makefile文件，指定体系结构和编译器地址：

ARCH := arm

CROSS\_COMPILE := /usr/local/arm/3.4.1/bin/arm-linux-

5.3.2 修改 NAND Flash 分区信息

本课题中，NAND Flash应按照功能分为 4个分区，如图4-1所示：

图5-1 NAND Flash分区示意图 Linux 内核对于 Flash分区由 arch/arm/plat-s3c24xx/common-smdk.c 文件中的 mtd\_partition smdk\_default\_nand\_part 结构体定义，默认已经分为了8个区。按照图4-1的分区信息，修改该结构体为：

static struct mtd\_partition smdk\_default\_nand\_part[] = {

[0] = {

.name = “U-boot”，.size = 0x00100000，.offset = 0x0，}，[1] = {

.name = “Kernel”，.offset = 0x00100000，.size = 0x00300000，}，[2] = {

.name = “RootFS”，.offset = 0x00400000，.size = 0x02800000，}，[3] = {

.name = “User”，.offset = 0x02d00000，.size = 0x00f00000，}，同时还应根据CPU手册修改NAND Flash的读写时序：

static struct s3c2410\_platform\_nand smdk\_nand\_info = {

.tacls

= 0，.twrph0

= 30，.twrph1

= 0，};5.3.3 关闭 ECC 校验

本设计中，内核都是通过 U-boot 写到

Nand Flash 的，U-boot 通过的软件ECC算法产生ECC校验码，这与内核校验的ECC码不一样，而内核中的 ECC 码是由

S3C2410 中

Nand Flash 控制器产生的。所以，我们在这里选择禁止内核

ECC 校验.，具体操作如下： 文件drivers/mtd/nand/s3c2410.c中，找到s3c2410\_nand\_init\_chip()函数，将最后一行的

chip-->eccmode

= NAND\_ECC\_SOFT 改为：

chip-->eccmode

= NAND\_ECC\_NONE

5.4 CS8900a网卡的移植过程

本课题中使用的LJD2410开发板带有 CS8900A网卡芯片，并提供RJ-45网络接口。Linux内核中并没有为 ARM体系配置CS8900A的网卡驱动，需要自己添加。CS8900A的驱动文件有两个：CS8900A.h 和CS8900A.c，这两个文件可以由网络获得，将其拷贝至 drivers/net/arm 文件夹下，但这样并不能使驱动程序正常工作，还应对内核源文件做些修改。

5.4.1 修改硬件地址映射

（1）在/arch/arm/mach-s2410文件夹里建立文件smdk2410.h，添加如下代码：

#define pSMDK2410\_ETH\_IO

\_\_phys\_to\_pfn(0x19000000)

#define vSMDK2410\_ETH\_IO

0xE0000000

#define SMDK2410\_EHT\_IRQ

IRQ\_EINT9

这三个宏分别定义了网卡的物理地址、虚拟地址和占用的中断号。

（2）修改/arch/arm/mach-s2410/mach-smdk2410.c，添加如下代码：

#include

（3）在 map\_desc smdk2410\_iodesc[]结构题中添加CS8900A对于的 io 空间映射：

static struct map\_desc smdk2410\_iodesc[] \_\_initdata = {

{ vSMDK2410\_ETH\_IO , pSMDK2410\_ETH\_IO, SZ\_, MT\_DEVICE }, };

5.4.2 添加 CS8900A 内核编译项

Kconfig 文件是 Linux2.6 内核引入的配置文件，是内核配置选项的源文件。只有在这个文件里加入相应代码，才能在编译选项中出现菜单项。

在/drivers/net/arm/Kconfig中增加CS8900A的编译项代码：

config

ARM\_CS8900

tristate “CS8900 support”

depends on NET\_ETHERNET && ARM && ARCH\_SMDK2410

help …

最后应在/drivers/net/arm/Makefile 中添加：

obj-$(CONFIG\_ARM\_CS8900)

+= cs8900.o

以上工作完成后，新移植的CS8900A驱动就可以编译进内核里了。

5.5 Linux 内核的剪裁配置

配置内核选项是整个移植过程中很重要的一步，本设计使用SMDK2410作为参考开发板，所以可以参考内核中 SMDK2410 开发板的配置文件，通过以下命令将其复制到内核根文件夹下：

$ cp arch/arm/config/smdk2410\_defconfig.config 在此基础上，根据本课题的实际需求进行配置增减。

5.5.1 使用配置菜单

配置内核可以选择不同的配置界面，图形界面或者光标界面。由于光标菜单运行时不依赖于X11图形软件环境，可以运行在字符终端上，所以光标菜单界面比较通用。图4-2所示就是执行 make menuconfig出现的配置菜单。

在各级子菜单中，选择相应的配置时，有 3种选择，它们代表的含义分别如下：

Y—将该功能编译进内核。

N—不将该功能编译进内核。

M—将该功能编译成可以在需要时动态插入到内核中的模块。

图5-2 内核配置主菜单 内核配置原则是：将与内核其他部分关系较远且不经常且不经常使用的部分功能代码编译成可加载模块，有利于减少内核长度，减小内核消耗的内存，简化该功能相应的环境改变时对内核的影响；不需要的功能就不选；与内核关系紧密而且经常使用的部分功能代码直接编译到内核中。

5.5.2 基本配置选项

Linux内核的各个版本配置餐单各不相同，下面以本课题使用的2.6.24.4版为例，结合本课题的实际需求，简介下内核的基本配置选项。

（1）General setup：包含通用的一些配置选项，保持默认即可。

（2）Enable loadable module supple：包含支持动态模块的配置选项，保持默认。

（3）System Type：包含系统平台列表及其相关的配置，去掉SMDK2410以外所有开发板的支持、开启s3c2410 DMA支持。

（4）Bus support：包含各种总线配置选项，全部去掉。

（5）Kernel Features：包含内核特性相关选项，保持默认。

（6）Boot options：包含内核启动相关选项，其中内核启动参数设置为：

“noinitrd console=ttySAC0,115200 root=/dev/nfs init=linuxrc nfsroot=192.168.0.2:/home/arm/FS/myrootfs mem=64M

ip=192.168.0.10:192.168.0.2:192.168.0.1:255.255.255.0:LJD2410:eth0:off”，支持NFS文件系统。

（7）Floating point emulation：包含浮点数运算仿真功能，需要开启“NWFPE”选项。

（8）Userapace binary formats：包含支持的应用程序格式，仅保留“ELF”格式支持，去掉其它。

（9）Power management options：包含电源管理功能，保持默认。

（10）Networking：包含网络功能：需要开启基本功能选项。

（11）Device Drivers：包含设备驱动选项，下一小节将详细介绍。

（12）File systems：包含各种文件系统的支持选项，去掉“EX2”等选项，仅保留 ROM 文件系统支持，在“Miscellaneous filesystems”子菜单中近保留“cramfs”文件系统支持，并且开启NFS文件系统支持，去掉其它选项。

（13）Kernel hacking：包含各种内核调试选项，保持默认。

（14）Security options：包含安全性有关选项，保持默认。

（15）Cryptographic API：包含加密算法，保持默认。

（16）Library routines：包含几种压缩和校验函数，保持默认。5.5.3 驱动程序配置选项

几乎所有Linux的设备驱动都在“Device Drivers”菜单下，它对设备驱动程序加以归类，放在子菜单下。本课题对于设备驱动的裁剪较多，具体如下：

（1）MTD support：MTD设备驱动，应添选NAND Flash驱动支持。

（2）Network debice support：网络设备支持，在子菜单“Ethernet（10 or 100Mbits）”中可以看到CS8900A网卡的配置项，这正是4.4节工作的结果。

（3）Real Time Clock：时钟驱动选项，应选上“Samsung S3C series SoC RTC”，这样系统时钟才能正常运行。

（4）由于嵌入式导航计算机只使用串口作为输入输出接口，所以应该剪裁掉那些无用的驱动，包括：并口、ATA及SATA驱动、RAID驱动、ISDN支持、输入设备驱动、多媒体设备支持、USB 支持以及MMC/SD卡支持。（5）其它驱动支持保持默认即可。

5.5.4 保存配置文件

内核配置主菜单中选择“Save an Alternate Configuration File”即可将目前的配置状态保存成文件。程序默认保存为“.config”，此文件位于内核根目录内，可以直接修改。

5.5.5 编译 Linux 内核

正式编译Linux内核之前，应当清理一下内核树，命令如下：

$ make mrproper

此命令会清除掉.config 文件，所以应当在配置内核之前做。

Linux 2.6 版本的编译已经简化，使用一个 make 命令就可以完成诸如建 立文件依赖、生成zImage、编译模块、安装模块等一系列功能。内核编译完 成后，将在/arch/arm/boot 目录中生成 image 和 zIamge 两个内核映像文件，其中 image 为正常大小的映像文件，而 zImage 为压缩后的映像文件。此时 编译好的可加载模块也被安装到预定位置，默认为/lib/modules。5.6 内核的下载及启动

5.6.1 将引导信息加入内核映像

U-boot 引导内核时需要检查一个 64byte 的头信息，其中包含了入口地址、映像类型等基本信息。这个引导头可以用 U-boot 附带的 mkimage 工具生成，命令如下：

$ mkimage-n \'linux-2.6.24\'-A arm-O linux-T kernel-C none-a 0x30008000-e 0x30008040-d zImage zImage.img 各个参数的含义：

-n：设置映像名

-A：设置体系信息

-O：设置操作系统信息

-T：设置映像类型

-c：压缩类型

-a：读入地址

-e：入口地址

-d：源映像文件

该命令生成的zImage.img文件就可以下载到开发板运行了。

5.6.2 内核映像的下载及运行

将上一小节中生成的zImage.img文件拷贝到主机tftpboot 文件夹内。启动开发板，进入U-boot 提示符。使用tftp命令将内核映像下载到开发板内存中：

LJD2410> tftp 0x30008000 zImage.img

TFTP from server 192.168.0.2;our IP is 192.168.0.10

Filename ‘zImage.img’

Load address : 0x30008000

Loading:

####

Done

其中 0x30008000 为指定的下载到内存的地址，zImge.img 就是带有引导头的内核映像。当内核下载完成后，可以通过bootm命令启动内核：

LJD2410> bootm 0x30008000 6 建立根文件系统

6.1 根文件系统概述

6.1.1 根文件系统简介

对于嵌入式操作系统而言，仅包含内核是不够的，还必须有文件系统的支持。跟文件系统（root filesystem）是 Linux系统的核心部分，包含系统使用的软件和库，以及无偶有用来为用户提供支持架构和用户使用的应用软件，并作为存储数据读写结果的区域。在Linux系统启动时，首先完成内核安装及环境初始化，最后会寻找一个文件系统作为根文件系统被加载。Linux系统中使用“/”来唯一表示根文件系统的安装路径。嵌入式系统中通常可以选择的根文件系统有:Romfs, CRAMFS, RAMFS,JFFS2, EXT2等，甚至还可以使用NFS(网络文件系统)作为根文件系统。

6.1.2 NFS 文件系统与Cramfs文件系统

NFS（Network File System）是由SUN公司发展，并于1984年推出的一种文件系统。它可以让开发者通过网络连接，使开发板可以直接挂载主机的某一个指定文件夹作为根文件系统。在嵌入式开发过程中，通常使用这种文件系统搭建交叉编译环境。

cramfs(Compressed ROM File System)是Linux创始人Linus Torvalds开发的一个适用于嵌入式系统的文件系统。cramfs是一个只读文件系统，采用了zlib压缩，压缩比一般可以达到1:2，但仍可以做到高效的随机读取。Linux系统中，通常把不需要经常修改的目录压缩存放，并在系统引导的时候再将压缩文件解开。因为 cramfs 不会影响系统读取文件的速度，而且是一个高度压缩的文件系统，因此本课题最终选用cramfs作为根文件系统部署到开发板。

6.2 建立Linux根文件系统目录

嵌入式Linux根文件系统必须包含一些必须的目录，比如设备目录/dev、命令目录/bin、库目录/lib等等。

本课题构建根文件系统的工作目录是 myrootfs，通过下列命令可以在myrootfs中创建所需的子目录：

$ mkdir bin dev etc lib proc sbin sys usr $ mkdir usr/bin usr/lib usr/sbin lib/modules $ mkdir mnt tmp var $ chmod 1777 tmp $ mkdir var/lib var/lock var/log var/run var/tmp $ chmod 1777 var/tmp $ mkdir home root boot 这样，一个基本的根文件系统就建立起来了，但是各个目录都是空的，缺少各种程序和命令工具，需要进一步完善。心得体会

本课题的目标是为基于ARM9处理器的导航计算机移植Linux操作系统。研究过程中，使用了 LJD2410 型开发板，此开发板的处理器是基于 ARM920T的 Samsung S3c2410，能够满足嵌入式导航计算机的硬件需求。本课题所做的工作简要总结如下：

首先，本文对嵌入式系统、嵌入式Linux操作系统和ARM体系处理器做了简单介绍，并且分析了嵌入式导航计算机的操作系统需求。

其次，介绍了交叉开发环境的建立。本课题两台主机连接开发板的方法，主机分别安装不同的操作系统，在开发过程中完成不同的工作。通过 TFTP和NFS等网络服务，实现高效连接，有利于提高开发效率。这部分是整个课题的基础，之后的所有工作都是在这个基础上完成的。

第三，本文重点介绍了 Linux 系统的移植过程。Linux 系统移植包括三个方面：启动加载程序（Bootloader）的移植，Linux 内核的移植和根文件系统的建立。本课题选用功能强大的 U-boot 作为启动加载程序，通过对其源代码进行修改，使其可以正常运行于开发板，并且实现下载、烧写等功能。内核则采用了2024年 4月发布的2.6.24.4版本，移植了网卡驱动，并针对课题需求，进行了修改和裁剪，使得内核加载更快，运行更稳定。根文件系统选用了Cramfs文件系统，这种文件系统采用压缩格式，存储空间需求小，但是不影响读取速度，非常适合与嵌入式Linux系统。这三个方面的工作有前后继承关系，但是又有一定独立性，移植过程中应多调试，多实验。

最后，简单介绍了系统部署的方法。将Linux 内核和根文件系统部署到开发板后，开发板就可以脱离交叉开发环境而独立运行，最终达到设计需求。

本课题充分利用前人积累的经验，结合最新的软件版本进行移植工作。在移植过程中遇到了许多困难和问题，主要靠查阅文献和自己的试探性试验来研究问题，通过多次的实践，最终得到明确的解决方法。虽然移植后的Linux系统可以正常运行在开发板上，能满足设计需求。但由于时间仓促，许多问题没有深入研究，难免会出现一定的疏漏和瑕疵，需要我在今后的学习中不断努力，加以改进。

**第三篇：嵌入式系统应用与学习心得体会**

μC/OS嵌入式实时操作系统的应用和学习心得

μC/OS-II 是一种基于优先级的抢占式多任务实时操作系统，包含了实时内核、任务管理、时间管理、任务间通信同步(信号量，邮箱，消息 队列)和内存管理等功能。它可以使各个任务独立工作，互不干涉，很容易实现准时而且无误执行，使实时应用程序的设计和扩展变得容易，使应用程序的设计过程大为减化。它是一个完整的、可移植、可固化、可裁剪的抢占式实时多任务内核。μC/OS-II绝大部分的代码是用ANSII的C语言编写的，包含一小部分汇编代码，使之可供不同架构的微处理器使用。至今，从8位到64位，μC/OS-II已在超过40种不同架构上的微处理器上运行。μC/OS-II已经在世界范围内得到广泛应用，包括很多领域，如手机、路由器、集线器、不间断电源、飞行器、医疗设备及工业控制上。实际上，μC/OS-II已经通过了非常严格的测试，并且得到了美国航空管 理局的认证，可以用在飞行器上。这说明μC/OS-II是稳定可靠的，可用于与人性命攸关的安全紧要系统。除此以外，μC/OS-II 的鲜明特点就是源码公开，便于移植和维护。

我们在学习μC/OS嵌入式实时操作系统时用的教科书是任哲编著的 《嵌入式实时操作系统μC/OS-Ⅱ原理及应用》，这本书介绍了嵌入式实时操作系统μC/OS-Ⅱ内核的任务管理和调度、系统时钟和节拍服务、时间管理、中断、任务的通信和同步、内存的简单管理原理，同时给出了实例。最后，还介绍了μC/OS-Ⅱ的移植方法。

在多任务系统中，内核负责管理各个任务，或者说为每个任务分配CPU时间，并且负责任务之间的通讯。内核提供的基本服务是任务切换。之所以使用实时内核可以大大简化应用系统的设计，是因为实时内核允许将应用分成若干个任务，由实时内核来管理它们。内核本身也增加了应用程序的额外负荷，代码空间增加ROM的用量，内核本身的数据结构增加了RAM的用量。但更主要的是，每个任务要有自己的栈空间，这一块吃起内存来是相当厉害的。内核本身对CPU的占用时间一般在2到5个百分点之间。μC/OS-Ⅱ有一个精巧的内核调度算法,实时内核精小，执行效率高，算法巧妙,代码空间很少，具有如下特点：只支持基于优先级的抢占式调度算法，不支持时间片轮循；64个优先级,只能创建64个任务,用户只能创建56个任务；每个任务优先级都不相同；不支持优先级逆转；READY队列通过内存映射表实现快速查询；效率非常高；支持时钟节拍；支持信号量，消息队列，事件控制块，事件标志组，消息邮箱任务通讯机制；支持中断嵌套，中断嵌套层数可达255层，中断使用当前任务的堆栈保存上下文；每个任务有自己的堆栈，堆栈大小用户自己设定；支持动态修改任务优先级；任务TCB为静态数组，建立任务只是从中获得一个TCB，不用动态分配，释放内存；任务堆栈为用户静态或者动态创建，在任务创建外完成，任务创建本身不进行动态内存分配；任务的总个数（OS\_MAX\_TASKS）由用户决定；0优先级最高，63优先级最低；有一个优先级最低的空闲任务,在没有用户任务运行的时候运行。

任务的调度核心主要是在任务就绪表中查找具有最高优先级别的就绪任务及实现任务切换，即将找到的的最高优先级别的任务的“任务号”---> “当前任务的任务号OSPrioCur”；从OSTCBPrioTbl［］取出最高优先级别任务的任务控制块首地址---> “当前任务的任务控制块指针OSTCBCur”，最后调用OS\_TASK\_SW（）完成现场切换（即将CPU的寄存器信息保存到当前正运行任务的私栈中，然后从待运行的高优先级别的任务私栈中弹出原先保存寄存器信息到CPU）。

时钟节拍服务放到一个时钟节拍任务中完成，通过采用啥希散列表机制来管理延时任务，每次时钟节拍服务只需要处理极少数的延时任务，从而大大减少了时钟节拍服务花费的时间，提高了系统的实时性。

另外，在μC／OS系列RTOS中，时钟节拍服务除了会跟踪延时的任务，还会跟踪那些指定了超时时限的等待任务。也就是说，当指定的超时时限结束时，即使任务等待的事件没有发生，时钟节拍服务也会使该任务恢复运行。

uC/OS-II的时间管理是通过定时中断来实现的，该定时中断一般为10毫秒或100毫秒发生一次，时间频率取决于用户对硬件系统的定时器编程来实现。中断发生的时间间隔是固定不变的，该中断也成为一个时钟节拍。

uC/OS-II要求用户在定时中断的服务程序中，调用系统提供的与时钟节拍相关的系统函数，例如中断级的任务切换函数，系统时间函数。

uC/OS-II 中最多可以支持64 个任务，分别对应优先级0～63，其中0 为最高优先级。63为最低级，系统保留了4个最高优先级的任务和4个最低优先级的任务，所有用户可以使用的任务数有56个。还提供了任务管理的各种函数调用，包括创建任务，删除任务，改变任务的优先级，任务挂起和恢复等。

系统初始化时会自动产生两个任务：一个是空闲任务，它的优先级最低，该任务仅给一个整形变量做累加运算；另一个是系统任务，它的优先级为次低，该任务负责统计当前cpu的利用率。

想使用在MCU上使用μC/OS-II则必须满足以下几个条件：①处理器C 编译器支持可重入代码的生成；②用C 语言可以打开和关闭中断；③ 处理器支持中断，并能产生定时中断(中断频率通常设置在10～100 Hz 之间)；④ 处理器支持足够的RAM 空间，以满足多任务环境下设置任务堆栈的要求；⑤处理器有相应的指令,能将堆栈指针和其它CPU 寄存器读出和存储到堆栈或内存中。μC/OS-II代码中大部分是用C语言写的，但涉及到数据类型的重定义、堆栈结构的设计、任务切换时状态的保存和恢复等问题的大部分代码由于与处理器有关，是用汇编语言实现的。移植所要做的工作，就是在不同的处理器上用汇编语言来改写与处理器有关的代码及其他与处理器特性相关的部分。

在μC/OS-II移植过程中涉及以上问题的代码都包含在文件OS\_CPU.H、OS\_CPU\_C.C、OS\_CPU\_A.ASM中。因此移植的主要工作也在源代码的基础上围绕着这三个文件的改写展开。

总之，学习μC/OS嵌入式实时操作系统首先得先了解整个系统的结构,工作时的流程,比如任务1运行时如果有外部中断会发生什么(调用中断服务程序,把相应的任务加如就绪列,内核进行调度,就绪队列中的最高优先级的任务...)等，再对代码进行具体的研究会比较容易。

我们需熟知μC/OS嵌入式实时操作系统中的函数及其应用，例如引起任务调度的系统函数有OSStart（）uCOS/II启动 ；OSTaskSuspend（）挂起任务 ；OSTaskResume（）恢复任务 ；OSTimeDly()/OSTimeDlyHMSM()等待一个事件的发生等。只有在熟识系统运行的方式及可以引发其运行的函数的情况下我们才能较好的去运用它，用它来实现移植，从而达到我们的目的。

学习实时操作系统，任哲编著的 《嵌入式实时操作系统μC/OS-Ⅱ原理及应用》这本书相对来说比较适合初学者，它篇幅适中，通俗易懂，若能结合北京航空航天大学《ucosii操作系统讲义》就更加好了。

正所谓读书百遍，其义自现，通过对《北航》和《任哲》的资料不断反复地学习和阅读，我心中总算有些眉目了，我是先用别人的程序进行移植调试后才加入自己的代码，虽然开始都是实验---下载--失败，但随着我的不断努力，最终也是能够成功的，所以说学习这种技术，需要的是耐心，因为它需要我们一步一步慢慢加深理解，调试程序，有时一个符号打错了都会造成实验的失败，我们就要耐心仔细的在代码的海洋里寻找着这一根小针并把它去除改正。

**第四篇：嵌入式系统的主要应用**

嵌入式系统的主要应用

嵌入式系统是一种包括硬件和软件的完整的计算机系统，它的定义是：“嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可剪裁，适用于应用系统对功能、可靠性、成本、体积和功耗有严格要求的专用计算机系统。”嵌入式系统所用的计算机是嵌入到被控对象中的专用微处理器，但是功能比通用计算机专门化，具有通用计算机所不能具备的针对某个方面特别设计的、合适的运算速度、高可靠性和较低比较成本的专用计算机系统。

嵌入式系统的应用前景是非常广泛的，人们将会无时无处不接触到嵌入式产品，从家里的洗衣机、电冰箱，到作为交通工具的自行车、小汽车，到办公室里的远程会议系统等等。在家中、办公室、公共场所，人们可能会使用数十片甚至更多这样的嵌入式无线电芯片，将一些电子信息设备甚至电气设备构成无线网络；在车上、旅途中，人们利用这样的嵌入式无线电芯片可以实现远程办公、远程遥控，真正实现把网络随身携带。其应用领域可以包括：

1．交通管理：在车辆导航、流量控制、信息监测与汽车服务方面，嵌入式系统技术已经获得了广泛的应用，内嵌GPS模块，GSM模块的移动定位终端已经在各种运输行业获得了成功的使用。目前GPS设备已经从尖端产品进入了普通百姓的家庭，只需要几千元，就可以随时随地找到你的位置。2．家庭智能管理系统：水、电、煤气表的远程自动抄表，安全防火、防盗系统，其中嵌有的专用控制芯片将代替传统的人工检查，并实现更高，更准确和更安全的性能。

3.POS网络及电子商务：公共交通无接触智能卡发行系统，公共电话卡发行系统，自动售货机，各种智能ATM终端将全面走入人们的生活。

4.环境工程与自然：水文资料实时监测，防洪体系及水土质量监测、堤坝安全，地震监测网，实时气象信息网，水源和空气污染监测。在很多环境恶劣，地况复杂的地区，嵌入式系统将实现无人监测。

5.机器人：嵌入式芯片的发展将使机器人在微型化，高智能方面优势更加明显，同时会大幅度降低机器人的价格，使其在工业领域和服务领域获得更广泛的应用。

6.工业控制：相对于其他的领域，机电产品可以说是嵌入式系统应用最典型最广泛的领域之一。从最初的单片机到现在的工控机、SOC在各种机电产品中均有着巨大的市场。工业设备是机电产品中最大的一类，在目前的工业控制设备中，工控机的使用非常广泛，这些工控机一般采用的是工业级的处理器和各种设备，其中以X86的MPU最多。工控的要求往往较高，需要各种各样的设备接口，除了进行实时控制，还须将设备状态，传感器的信息等在显示屏上实时显示。这些要求8位的单片机是无法满足的，以前多数使用16位的处理器，随着处理器快速的发展，目前32位、64位的处理器逐渐替代了16位处理器，进一步提升了系统性能。采用PC104总线的系统，体积小，稳定可靠，受到了很多用户的青睐。不过这些工控机采用的往往是DOS或者Windows系统，虽然具有嵌入式的特点，却不能称作纯粹的嵌入式系统。另外在工业控制器和设备控制器方面，则是各种嵌入式处理器的天下。这些控制器往往采用16位以上的处理器，各种MCU，Arm、Mips、68K系列的处理器在控制器中占据核心地位。这些处理器上提供了丰富的接口总线资源，可以通过它们实现数据采集，数据处理，通讯以及显示（显示一般是连接LED或者LCD）。最近飞利浦和ARM共同推出32位RISC嵌入式控制器，适用于工业控制，采用最先进的0.18微米CMOS嵌入式闪存处理技术，操作电压可以低至1.2伏，它还能降低25%到30%的制造成本，在工业领域中对最终用户而言是一套极具成本效益的解决方案。美国TERN工业控制器基于Am188/186ES、i386EX、NEC V25、Am586(Elan SC520)，采用了SUPERTASK实时多任务内核，可应用于便携设备、无线控制设备、数据采集设备、工业控制与工业自动化设备以及其它需要控制处理的设备。

7.家电行业是嵌入式应用的另一大行业。现在只有按钮、开关的电器显然已经不能满足人们的日常需求，具有用户界面，能远程控制，智能管理的电器是未来的发展趋势。据IDG发布的统计数据表明，未来信息家电将会成长五至十倍。中国的传统家电厂商向信息家电过渡时，首先面临的挑战是核心操作系统软件开发工作。硬件方面，进行智能信息控制并不是很高的要求，目前绝大多数嵌入式处理器都可以满足硬件要求，真正的难点是如何使软件操作系统容量小、稳定性高且易于开发。Linux核心可以起到很好的桥梁作用，作为一个跨平台的操作系统，它可以支持二三十种CPU，而目前已有众多家电业的芯片都开始做Linux的平台移植工作。1999年就登录中国的微软“维纳斯”计划给了国人一个数字家庭的概念，引导各大家电厂商纷纷投入到这场革命中来，虽然最终未能获得成功，却使信息家电深入人心。如今各大厂商仍然在努力推出适用于新一代家电应用的芯片，英特尔公司已专为信息家电业研发了名为StrongARM的ARM CPU系列，这一系列CPU本身不象X86CPU需要整合不同的芯片组，它在一颗芯片中可以包括你所需要的各项功能，即硬件系统实现了SOC的概念。美商网虎公司已将全球最小的嵌入式操作系统——QUARK成功移植到StrongARM系列芯片上，这是第一次把Linux、图形界面和一些程序进行完整移植（QUARK的内核只有143K），它将为信息家电提供功能强大的核心操作系统。相信在不久的将来，数字智能家庭必将来到我们身边。

这些应用中，可以着重于在控制方面的应用。就远程家电控制而言，除了开发出支持TCP/IP的嵌入式系统之外，家电产品控制协议也需要制订和统一，这需要家电生产厂家来做。同样的道理，所有基于网络的远程控制器件都需要与嵌入式系统之间实现接口，然后再由嵌入式系统来控制并通过网络实现控制。所以，开发和探讨嵌入式系统有着十分重要的意义。

**第五篇：《嵌入式系统设计》教案**

嵌入式系统设计教案

课程总学时： 32 讲课学时： 24 实验学时： 8 授 课 人： 杨词慧

南昌航空大学信息工程学院

目录 嵌入式系统概述.......................................................................................................................1 1.1 嵌入式系统的基本概念...............................................................................................1 1.2 嵌入式系统的发展历史...............................................................................................2 1.3 嵌入式系统的体系结构...............................................................................................2 1.4 嵌入式处理器...............................................................................................................3 1.5 嵌入式操作系统...........................................................................................................4 1.6 嵌入式系统的应用及发展趋势...................................................................................7 ARM体系结构.........................................................................................................................8 2.1 ARM设计思想.............................................................................................................8 2.2 ARM体系结构分析.....................................................................................................9 2.3 ARM处理器系列.......................................................................................................11 2.4 ARM处理器模式.......................................................................................................15 2.5 ARM体系的异常处理...............................................................................................16 2.6 ARM内部寄存器.......................................................................................................18 2.7 ARM体系的存储系统...............................................................................................19 ARM指令系统及程序设计基础...........................................................................................21 3.1 ARM寻址方式...........................................................................................................21 3.2 ARM指令集...............................................................................................................24 3.3 Thumb指令集............................................................................................................33 3.4............................................................................................................................................35 3.5............................................................................................................................................35 3.6 嵌入式系统设计与开发过程.....................................................................................36 STM32微控制器...................................................................................................................37 4.1 STM32微控制器的性能指标....................................................................................37 2 3 4

I 嵌入式系统概述

教学目的：使学生对嵌入式系统的基本概念和体系结构、嵌入式处理器、嵌入式操作系统、嵌入式系统的历史、应用及发展趋势有一定的了解。

教学重点：嵌入式系统的基本概念、嵌入式处理器和嵌入式操作系统。教学难点：嵌入式系统的体系结构、嵌入式处理器。教学方法与教学手段：课堂讲授，多媒体教学。教学时间：2课时。教学内容

1.1 嵌入式系统的基本概念

(1)嵌入式系统的定义

先举例说明生活中的各种嵌入式系统设备，如iPhone、小米手机、洗衣机、电压力锅等。IEEE（国际电气和电子工程师协会）的定义：嵌入式系统是用于控制、监视或者辅助操作机器和设备的装置（Devices used to control, monitor, or assist the operation of equipment, machinery or plants）。

微机学会的定义：嵌入式系统是以嵌入式应用为目的的计算机系统，可分为系统级、板级和片级。a)系统级：各种类型的工控机、PC104等模块。b)板级：各种类型的带CPU的主板或OEM产品。c)片级：各种以单片机、DSP、微处理器为核心的产品。

一般定义：嵌入式系统是以应用为中心，以计算机技术为基础，软件硬件可裁剪，对功能、可靠性、成本、体积、功耗要求严格的专用计算机系统。

(2)嵌入式系统的特点 a)专用、软硬件可剪裁配置。b)低功耗、高可靠性、高稳定性。c)软件代码矮小精悍。d)代码可固化。e)实时性。f)弱交互性。

g)软件开发通常需要专门的开发工具、环境和方法。h)要求开发、设计人员具有较高的技能。i)具有较长的生命周期。(3)嵌入式系统的分类

按嵌入式微处理器的位数可分为：4位、8位、16位、32位和64位。按实时性能可分为：非实时系统和实时系统。

按软件结构可分为：嵌入式单线程系统和嵌入式事件驱动系统。

按应用领域可分为：信息家电类、消费电子类、医疗电子类、移动终端类、通信类、汽车电子类、工业控制类、航空电子类、军事电子类等。

1.2 嵌入式系统的发展历史

(1)以单芯片为核心的可编程控制器形成的系统

1971年11月，Intel推出Intel 4004。随后出现Intel 8080/8085、8086、Motorola的6800、68000，Zilog的Z80、Z8000。

以微处理器为核心构成的系统

单板机：Intel的iSBC系列、Zilog的MCB等

将计算机做在一个芯片上，大部分应用于专业性强的工业控制系统中，没有操作系统的支持，系统结构和功能相对单一，处理效率较低，存储容量较小。

(2)以嵌入式CPU为基础、以简单操作系统为核心的嵌入式系统

CPU种类繁多，通用性较弱；系统开销小，效率高；操作系统达到一定的兼容性和扩展性；应用软件较专业化，用户界面不够友好。

(3)以嵌入式操作系统为标志的嵌入式系统

嵌入式操作系统可运行于各种不同类型的微处理器上，兼容性好；操作系统内核小，效率高；具备文件和目录管理，支持多任务、网络应用，具备图形窗口和用户界面；有大量的应用程序接口API。

(4)以Internet为标志的嵌入式系统 嵌入式设备与Internet的结合。

1.3 嵌入式系统的体系结构

(1)体系结构

(2)硬件层

 嵌入式处理器：ARM、DSP、FPGA  存储器系统：ROM、FLASH、SDRAM  中断控制器、DMAC、定时器/计数器、UART、USB控制器、LCD控制器等  I/O接口：USB、I2C、SPI、CAN等

(3)中间层

 硬件抽象层(Hardware Abstraction Layer, HAL)。位于操作系统内核与硬件电路之间的接口层，隐藏硬件接口细节。

 板级支持包（Board Support Package, BSP）。为上层的驱动程序提供访问硬件设备寄存器的函数包。 设备驱动程序

1.4 嵌入式处理器

(1)嵌入式处理器的分类

中高端的嵌入式微处理器（Embedded Micro-Processor Unit, EMPU）低端的微控制器（Microcontroller Unit, MCU）DSP处理器（Digital Signal Processor, DSP）高度集成的片上系统（System on Chip, SoC）(2)嵌入式微处理器

由通用计算机中的CPU演变而来，只保留与嵌入式应用紧密相关的功能硬件，在工作温度、抗电磁干扰、可靠性、功耗等方面做了各种增强。

主要的嵌入式处理器类型：ARM、MIPS、PowerPC、68000系列等。特点

a)在设计中考虑低功耗。

b)采用可扩展的处理器结构。处理器内部留有很多扩展接口。c)具有很强的存储区保护功能。d)提供丰富的调试功能。

e)对实时任务具有很强的支持能力。(3)微控制器

俗称单片机，将整个计算机系统集成到一块芯片中。

WatchWatchdogdogGPIO/计数GPIOUARTUART定时定时/计数A/DA/DD/A器D/A器2CCANCANII2CSPISPIUSBUSBCPUCPUSRAMSRAM核心部分FlashFlash8位单片机示意图以一种微处理器为核心，芯片内部集成Flash、RAM、总线逻辑、定时器/计数器、I/O口、串行口、PWM、A/D、D/A等。

最早的单片机： 1976年，Intel的8048，Motorola的68HC05、Zilog公司的Z80.(4)DSP处理器

对系统结构和指令进行了特殊设计，使其适合DSP算法 高效乘累加运算、超标量操作、指令流水线 高效数据存取、硬件重复循环 确定性操作（程序执行时间可预测）应用场合：音视频编解码、数字滤波、FFT等(5)片上系统

将整个系统做在一个芯片上 优点

a)通过改变内部工作电压，降低芯片功耗 b)减少芯片对外的引脚数，简化制造过程

c)减少外围驱动接口单元及电路板之间的信号传递，加快微处理器数据处理的速度 d)内嵌线路可避免信号传递时所造成的系统杂讯 联发科推出28nm双核处理器MT6572 a)代号武松，基于Cortex-A7架构 b)主频为1.2GHz c)处理器上整合了Wi-Fi、FM收音机、GPS以及蓝牙四种功能 d)支持500万像素摄像头(6)嵌入式处理器的发展趋势 a)内部结构

SoC设计，与DSP、Flash、FPGA融合；性能更强，集成更多的功能部件；双核或多核结构 b)功耗更低 c)可靠性更高 d)支持ISP、ISD 1.5 嵌入式操作系统

(1)操作系统的概念及功能

操作系统。是一组计算机程序的集合，用来有效地控制和管理计算机的硬件和软件资源，并为用户提供方便的应用接口。

功能：处理器管理；存储器管理；设备管理；文件管理；用户接口(2)操作系统的分类 按程序调度的方法分为

 顺序操作系统。只含一个运行程序，独占CPU时间，顺序执行。如DOS系统。 分时操作系统。系统内同时有多道程序运行。如Unix系统。 实时操作系统。

从应用角度，嵌入式操作系统可分为  面向低端信息家电

 面向高端信息家电  面向个人通信终端  面向通信设备  面向汽车电子  面向工业控制

从实时性的角度，嵌入式操作系统可分为  具有强实时特点的嵌入式操作系统  具有弱实时特点的嵌入式操作系统  没有实时特点的嵌入式操作系统(3)实时操作系统（RTOS）

是具有实时性且能支持实时控制系统工作的操作系统，首要任务是调度一切可利用的资源来完成控制任务。

对现场不停监测，一旦有事件发生能立即处理。与通用OS的区别：实时性，代码尺寸小。一般包括以下几个重要组成部分：

 实时内核：任务管理、定时器管理、存储器管理、任务间通信与同步等。 网络组件  文件系统  图形用户界面

IEEE的Unix委员会规定了实时操作系统须具备以下几个特点：  支持异步事件的响应。 中断和调度任务的优先机制。 支持指令性计划占式调度。 支持同步。

(4)常见的嵌入式操作系统 嵌入式Linux  实时的嵌入式Linux：如RT-Linux、KURT-Linux等。RT-Linux将通常的Linux任务优先级设为最低。

 一般的嵌入式Linux：如μCLinux。 开源，内核小、效率高，可定制 μC/OS和μC/OS-II  μC/OS-II(MicroController Operating System)是由Jean J.Labrosse开发的实时操作系统内核。

 已被移植到Intel、ARM、Motorola等公司的81种不同的处理器上。

 Labrosse用一年时间开发了μC/OS实时操作系统； 1992年在《Embedded System Programming》上 发表介绍文章，并公布源代码；1993年写了《μC/OS, The Real-Time Kernel》；书及源码推动了μC/OS-II的发展。

 μC/OS-II只是一个实时操作系统的内核，全部核心代码只有8.3 KB。 只包含进程调度、时钟管理、内存管理和进程间的通信与同步等基本功能。Windows CE  多线程、完整优先权、多任务的32位嵌入式操作系统。 基本内核大小至少为200KB。VxWorks  美国WindRiver公司于1983年设计

 是目前嵌入式系统领域中使用最广泛、市场占有率最高的系统。 拥有良好的持续发展能力和高性能的内核及友好的用户开发环境。

 支持多种处理器，如ARM、x86、i960、SunSparc、MIPS RX000、PowerPC、StrongARM等。 以良好的可靠性和实时性，广泛应用在通信、军事、航空、航天等领域。

 应用案例：美国F-

16、FA-18战斗机，B-2隐形轰炸机，“爱国者”导弹，1997年4月在火星登陆的火星探测器。Palm OS  32位嵌入式操作系统，由3Com公司的Palm Computing部门开发。

 在掌上电脑和PDA市场上占有很大的市场份额，曾占据90%的PDA市场份额。 2024年04月29日惠普12亿美元收购Palm QNX  一个实时、可扩充的操作系统。

 部分遵循POSIX（可移植操作系统接口）相关标准。

 内核仅提供4种服务：进程调度、进程间通信、底层网络通信和中断处理。 内核非常小巧（QNX4.X约为12KB），运行速度极快。iOS  苹果推出，基于Darwin  最新：iOS8 Android  Google开发的基于Linux平台的开源手机操作系统 Delta OS  是电子科技大学实时系统教研室和北京科银京成技术有限公司联合研制并开发的全中文嵌入式操作系统。

 绝大部分代码由C语言编写。

 已成功应用于通信、网络、信息家电等多个应用领域。pSOS ISI公司研发的产品。

模块化、高性能、完全可扩展。

1.6 嵌入式系统的应用及发展趋势

(1)嵌入式系统的应用领域  消费电子领域  通信网络领域

 工业控制领域、机器人领域  交通管理与汽车电子领域  国防与航空航天领域  医疗仪器领域

(2)嵌入式系统的发展趋势

 小型化、智能化、网络化、可视化  多核技术的应用  低功耗、绿色环保

 云计算、可重构、虚拟化等技术被进一步应用到嵌入式系统中  嵌入式系统软件将逐渐PC化  融合趋势

 微控制器MCU与SoC的结合  微控制器MCU与DSP的结合  ARM与DSP的结合

 微控制器MCU与CPLD/FPGA的结合  安全性 ARM体系结构

教学目的：使学生对ARM设计思想、ARM处理器系列、ARM体系结构、ARM处理器模式、ARM内部寄存器、ARM体系的存储系统有一定的了解。

教学重点：ARM设计思想、ARM体系结构、ARM处理器模式、ARM内部寄存器。教学难点：ARM处理器模式、ARM内部寄存器。教学方法与教学手段：课堂讲授，多媒体教学。教学时间：4课时。教学内容

2.1 ARM设计思想

(1)RISC 传统的CISC(Complex Instruction Set Computing)指令集中，约20%指令占整个程序代码的80%。RISC(Reduced Instruction Set Computing)是一种设计思想，其目标是设计出一套能在高时钟频率下单周期执行、简单而有效的指令集。

RISC设计重点在于降低硬件执行指令的复杂度，而传统的CISC更侧重于硬件执行指令的功能性，使CISC指令变得复杂。

(2)RISC设计思想的实现

 指令集。减少了指令种类，指令只实现简单的功能，指令长度固定。

 流水线。指令的处理过程被拆分成几个更小的、能够被流水线并行执行的单元。

 寄存器。更多通用寄存器，可存数据和地址，可为所有数据操作提供快速的局部存储访问。 load-store结构。处理器只处理寄存器中数据，用load和store指令完成寄存器和外存间的数据传送

(3)ARM的设计思想  较小的核。降低功耗

 高的代码密度。考虑成本和物理尺寸限制

 较小的处理器内核管芯(Die)面积。留给外设电路的空间较大  硬件调试技术(4)ARM微处理器的特点

 体积小，功耗低，成本低，性能高；  支持Thumb/ARM指令集,兼容8/16位器件；

 大量使用寄存器，指令执行速度更快；  大多数数据操作都在寄存器中完成；  寻址方式灵活简单，执行效率高；  指令长度固定。

2.2 ARM体系结构分析

(1)包含典型 的RISC 体系结构特征

 统一寄存器文件加载/存储体系结构，数据处理操作只针对寄存器内容；  简单寻址模式，所有加载/存储地址只通过寄存器内容和指令字段确定。 指令长度固定，简化了指令译码。(2)还提供

 可组合使用转换与算术或逻辑运算指令  自动递增和自动递减寻址模式，可优化程序循环  加载存储多个指令以最大化数据吞吐量  几乎所有指令都采取条件执行的方式(3)普林斯顿结构和哈佛结构

普林斯顿结构：也称冯·诺伊曼结构，它将程序指令存储器和数据存储器合并在一起的存储结构。ARM7系列基于普林斯顿结构。

哈佛结构：将程序指令存储和数据存储分开的存储结构。ARM9系列之后都基于哈佛结构。

(4)流水线

ARM7的三级流水线在执行单元完成了大量的工作，执行单元的工作往往占用多个时钟周期，从而成为系统性能的瓶颈。

ARM9采用哈佛架构，避免了数据访问了取指的总路线冲突，采用五级流水线设计。

五级流水线技术把三级流水线中的执行单元进一步细化，减少了在每个时钟周期内必须完成的工

作量，进而允许使用较高的时钟频率。

无论三级流水线还是五级流水线，在以下情况下都会发生阻塞：  多周期指令、跳转分支指令  中断发生

 相邻指令之间的寄存器冲突：如果当前指令（A）的目的操作数寄存器和下一条指令（B）的源操作数寄存器一致，B指令就需要等A回写之后才能译码。

(5)ARM体系结构的版本

2.3 ARM处理器系列

(1)ARM处理器系列

 经典ARM处理器：ARM7、ARM9、ARM10、ARM11  Cortex-A系列处理器：Cortex-A15、Cortex-A9、Cortex-A8、Cortex-A7、Cortex-A5  Cortex-R系列处理器：Cortex-R7、Cortex-R5、Cortex-R4  Cortex-M系列处理器：Cortex-M4、Cortex-M3、Cortex-M1、Cortex-M0+、Cortex-M0、CMSDK、CMSIS  SecurCore处理器(2)ARM内核版本命名规则

(3)ARM7系列微处理器 a)主要特点

 最高主频：130MIPS(Million Instructions Per Second)；  功耗低；

 代码密度高，兼容16位的微处理器；  可得到广泛的操作系统和实时操作系统支持；  众多的开发工具，优秀的调试机制；  采用3级流水线结构；

 提供0.25μm、0.18μm和0.13μm的生产工艺。b)冯·诺伊曼结构：数据和指令使用同一条总线。

c)包括ARM7TDMI、ARM7TDMI-S、ARM7EJ-S和ARM720T四种类型，适用于不同的市场要求 d)ARM7TDMI  ARM公司最早为业界普遍认可并广泛应用的处理器核。 T：Thumb；D：Debug；M：Multiplier；I：Embedded ICE logic。e)ARM7TDMI-S  是ARM7TDMI的可综合(synthesizable)版本（软核）。

 ARM以“软”核的方式把ARM7TDMI核授权给处理器厂商，处理器厂商可进行修改和综合。这就是ARM7TDMI-S。

 综合出的整个核比“硬”核大50%，电源效率降低50%。f)ARM7EJ-S  是可综合的、带有增强型DSP（E变种）和Java加速功能（J变种）的32位RISC嵌入式处理器。

 主要用于数字音频播放器、带Java功能的无线手持设备、喷墨打印机、数码相机和PDA等方面。

g)ARM720T  专为使用Windows CE、Symbian OS操作系统平台设计。 主要用于数字音频播放器、喷墨打印机和数码相机等。(4)ARM9系列微处理器 a)主要特点

 5级整数流水线；

 单一的32位AMBA(Advanced Microcontroller Bus Architecture)总线接口；  MMU支持Windows CE、Symbian OS、Linux等；  支持实时操作系统，包括VxWorks；  统一的数据Cache和指令Cache；

 提供0.25μm、0.18μm和0.13μm的生产工艺。

b)包括ARM9TDMI、ARM920T、ARM940T和ARM9E四种类型。后三种含有Cache。

c)采用Harvard体系结构

 指令与数据分开存储。 采用指令快存 和数据快存。

d)在相同工艺条件下，ARM9TDMI的处理能力是ARM7TDMI的两倍。e)ARM920T 

Motorola MC9328MX1和Samsung S3C2410X处理器都采用ARM920T 核心。 主要应用于通信终端、3G基带和应用处理器、基于OS的平台设备、数码相机、音频/视频解码和机顶盒等。

f)ARM940T  与ARM920T相比，实现了一个更小的D-Cache、I-Cache和MPU。适于不需运行操作系统的平台。

g)ARM9E  使用单一的处理器核，提供微控制器、DSP、Java应用系统的解决方案；  DSP指令集；

 在0.13μm工艺下，主频可达300MIPS的性能；  集成实时跟踪调试功能；  可选的VFP9浮点处理协处理器；

 高性能的AHB(Advanced High performance Bus)。(5)ARM10系列微处理器 a)主要特点

 6级流水线；

 在典型的0.13μm工艺下，主频可达400MIPS的性能；  单一的32位AMBA 总线接口；

 MMU支持Windows CE、Symbian OS、Linux等；  统一的数据Cache和指令Cache；

 提供0.25μm、0.18μm和0.13μm的生产工艺；  并行读取/写入部件。

b)包括：ARM1020、ARM10200、ARM1020E、ARM1022E、ARM1026EJ-S。c)使用ARM10TDMI处理器核，采用ARMv5T结构。d)ARM10TDMI  在相同工艺条件下，处理能力是ARM9TDMI的两倍；

 采用提高时钟频率、6级流水线、转移预测逻辑、64位存储器和无阻塞的存/取逻辑等措施提升性能。

e)ARM10E。新节能模式，64位Load/Store体系，与ARM10TDMI相比具有的特点

 DSP指令集；

 可选的VFP10浮点处理协处理器；

 在实时控制和三维图像处理时，主频可达650MFLPS(百万次浮点运算每秒)。

(6)ARM11系列微处理器

 ARM1156T2-S内核、ARM1156T2F-S内核、ARM1176JZ-S内核和ARM11JZF-S内核  ARM1156T2-S内核和 ARM1156T2F-S内核  基于ARM v6指令集体系结构；

 是首批含有ARM Thumb-2内核技术的产品。(7)Cortex-A系列微处理器

 适于高计算要求、运行丰富操作系统及提供交互媒体和图形体验的应用领域。 支持传统 ARM、Thumb指令集和新的高性能紧凑型 Thumb-2 指令集。 移动互联网的支持

 低功率设计，支持 Adobe Flash 10.1  高性能 NEON 引擎，广泛支持媒体编解码器  高性能

 Cortex-A15：为新一代移动基础结构应用和无线基础结构应用提供高性能的解决方案。

 Cortex-A9。800 MHz1 GHz 的频率下，提供的性能超过 2024 DMIPS。

 Cortex-A5 低成本实现，在 400-800 MHz 的频率下，提供的性能超过 1200 DMIPS，是尺寸最小、功耗最低的 ARM 多核处理器。

 多核技术：Cortex-A15、Cortex-A5和Cortex-A9 处理器都支持 ARM 第二代多核技术。 高级扩展

 Thumb-2，提供最佳代码大小和性能。 TrustZone，安全扩展，提供可信计算。

 Jazelle 技术，提高执行环境（如 Java、.Net、MSIL、Python 和 Perl）速度。

(8)Cortex-R系列微处理器

 为具有严格的实时响应限制的深层嵌入式系统提供高性能计算解决方案。

 快速。以高时钟频率获得高处理性能。

 确定性。处理在所有场合都必须符合硬实时限制。 安全。系统必须可靠且可信。

 成本效益。在处理器及其内存系统中都具有竞争力的成本和功耗。 应用领域

 智能手机

 企业系统：硬盘驱动器、联网和打印  消费电子：机顶盒、数字电视和播放器  医疗行业、工业和汽车行业的可靠系统  功能集

(9)Cortex-M系列微处理器

 向上兼容的高能效、易于使用的处理器

 针对成本和功耗敏感的 MCU 和终端应用的混合信号设备进行过优化。 更低的功耗，更长的电池寿命。

 高密度指令集，更小的代码，更低的硅成本。

RISC 处理器内核高性能32 位CPU具有确定性的运算低延迟3 阶段管道Thumb-2 技术16/32 位指令的最佳混合小于8 位设备3 倍的代码大小对性能没有负面影响低功耗模式集成的睡眠状态支持多电源域基于架构的软件控制嵌套矢量中断控制器(NVIC)低延迟、低抖动中断响应不需要汇编编程以纯C 语言编写的中断服务例程工具和RTOS 支持广泛的第三方工具支持Cortex 微控制器软件接口标准(CMSIS)最大限度地增加软件成果重用CoreSight调试和跟踪JTAG 或2 针串行线调试(SWD)连接支持多处理器支持实时跟踪(10)SecurCore系列微处理器

 为安全要求较高应用设计。智能卡

 SIM、ID、银行业、付费电视、公共交通、电子政务

2.4 ARM处理器模式

(1)32位ARM处理器工作状态

32位ARM处理器有三种工作状态  ARM状态。对应32位ARM指令集  Thumb状态。对应16位Thumb指令集  Jazelle状态。对应8位的Jazelle指令集

用于在处理器指令层次对JAVA加速

只有进入特定的状态，相应的指令集才有效。CPSR的J(Jazelle)和T(Thumb)位反映程序的状态。Thumb-2  与现有 ARM 和 Thumb 解决方案向后兼容，同时扩展了 Thumb 指令集的可用功能；  使用少于 31% 的内存以降低系统成本；  提供比现有高密度代码高出 38% 的性能。(2)64位ARM处理器工作状态

ARMv8架构

两种主要执行状态：AArch64, AArch32 AArch64：引入了一套新的指令集“A64”专门用于64位处理 AArch32：兼容现有的32位ARM指令集 ARMv8架构支持三个主要指令集

A32(或 ARM)：32 位固定长度指令集

T32(Thumb)：以 16 位固定长度指令集的形式引入，在引入 Thumb-2 技术时增强为 16 位和 32 位混合长度

A64：提供与 ARM 和 Thumb 指令集类似功能的64位固定长度指令集(3)ARM处理器运行模式

 用户模式(usr)：ARM处理器正常的程序执行状态。 快速中断模式(fiq)：用于高速数据传输或通道处理  外部中断模式(irq)：用于通用的中断处理。 管理模式(svc)：操作系统使用的保护模式。

 数据访问终止模式(abt)：当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护。

 系统模式(sys)：运行具有特权的操作系统任务。

 未定义指令中止模式(und)：当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真。

运行模式可通过软件改变，也可通过外部中断或异常处理改变。用户模式之外的模式称为非用户模式或特权模式。除用户模式和系统模式之外的5种称为异常模式，常用于处理中断和异常、访问受保护的系统资源等情况。

2.5 ARM体系的异常处理

(1)ARM体系中3种控制程序执行流程的方式：  顺序执行  跳转分支指令

 异常中断：处理器暂时中断当前数据流的现象。(2)对异常的响应。ARM处理器执行完当前指令后：  进入与特定的异常相应的操作模式；

 将引起异常指令的下一条指令的地址保存到新模式的R14中；

 将CSPR原值保存到新模式的SPSR中；

 通过设置CSPR的第7位来禁止IRQ。如果为FIQ中断，则还要设置CSPR的第6位来禁止FIQ；

 给PC强制赋向量地址值。(3)中断向量表

指定了异常中断及其处理程序的对应关系，它通常存放在存储地址的低端。

大小为32字节，其中每个异常中断占据4字节空间，用于存放一个跳转指令或者一个向PC寄存器中赋值的指令。

(4)异常优先级：当几个异常中断同时发生时，就必须按照一定的次序来处理这些异常中断。(5)从异常返回

将连接寄存器LR的值减去相应偏移量后送到PC中； 将SPSR复制回CPSR中；

若在进入异常处理时设置了中断禁止位，则要清除。通过普通指令控制PC返回

软件中断的返回指令 MOVS R15, R14;将链接寄存器内容移入PC并转换模式 IRQ，FIQ和预取异常终止中断的返回指令 SUBS R15, R14, #4 数据终止异常的返回指令 SUBS R15, R14, #8;异常在导致异常的指令的下一条指令后产生

2.6 ARM内部寄存器

(1)寄存器结构

32位ARM有31个32位通用寄存器，6个状态寄存器。通用寄存器可用来保存数据和地址信息，用R为前缀加寄存器序号表示

15个通用寄存器（R0~R14）、一个或两个状态寄存器及程序计数器可在任意时间和处理器模式下被访问，有些处理器模式拥有自身独立的寄存器

(2)通用寄存器

分成三类：

R0~R7：未分组寄存器。每个未分组寄存器在所有的处理器模式下都表示同一个物理寄存器。R8~R14：分组寄存器。每个分组寄存器与一个用户模式的寄存器对应。R15：程序计数器PC。

分组寄存器R8~R14可分为两组：

R8~R12：每个寄存器对应两组不同的物理寄存器，一组是FIQ模式下的，记为R8\_fiq~R12\_fiq, 另一组是除FIQ模式外的：R8\_usr~R12\_usr。

R13~R14：分别对应6个不同的物理寄存器。用户模式和系统模式共用一个寄存器，另外5个对应其余5种。

R13 \_ R14 \_

可以是以下几种模式之一：usr、svc、abt、und、irq和fiq。

R13：被称为堆栈指针SP，但没有任何指令强制性使用R13作为堆栈指针

R14又被称为链接寄存器LR(Link Register)。当调用子程序时，返回地址被自动保存到R14。由于ARM采用了多级流水线技术，所以当正常读取PC值时，该值为当前指令地址值加8，或是加12。

2.7 ARM体系的存储系统

(1)地址空间

将存储器看作是从零地址开始的字节的线性组合 0-3字节：第1个存储的字数据 4-7字节：第1个存储的字数据 依次排列(2)存储器格式 大端格式

小端格式

(3)存储器访问对准

无论取指还是内存访问都以字、半字或字节对准访问 a)非对齐的指令预取操作

ARM状态：将一个非对齐地址写入PC，数据的第0位和第1位被忽略，PC的bit[1:0]为0 Thumb状态：数据的第0位被忽略，PC的bit[0]为0 b)非对齐地址内存的访问操作（LOAD/STORE操作）

执行结果不可预知

忽略字单元地址低两位的值，半字单元最低位的值（分别对应访问字和半字）

在LDR和SWP指令中，对存储器访问忽略造成地址不对齐的低地址位，然后使用这些低地址位控制装载数据的循环 ARM指令系统及程序设计基础

教学目的：使学生对ARM指令系统及程序设计的基础知识有一定的了解。教学重点：ARM指令系统。

教学难点：ARM程序设计的基础知识。教学方法与教学手段：课堂讲授，多媒体教学。教学时间：4课时。教学内容

3.1 ARM寻址方式

(1)寻址方式

处理器根据指令中给出的地址信息寻找物理地址的方式。寻找操作数或操作数地址的方式。

(2)立即寻址

也叫立即数寻址，操作数本身在指令中给出，该操作数被称为立即数。例如： ADD R0, R0, #1;R0R0+1 立即数须以“#”为前缀，对于十六进制表示的立即数，还需在“#”后加上“0x”或“&”。(3)寄存器寻址

操作数存在寄存器中。例如： ADD R0, R1, R2;R0R1+R2(4)寄存器间接寻址

以寄存器的值作为操作数地址。例如： ADD R0, R1, [R2]

;R0  R1+[R2](5)基址变址寻址

a)基址加偏移的寻址方式

将寄存器内容与指令中给出的地址偏移量相加，得到操作数的有效地址。如： LDR R0, [R1, #4];R0  [R1+4]

前变址：基址加变址作为操作数地址。

后变址：基址作为操作数的地址，传送后自动更新基址寄存器的值。

b)基址加索引的寻址方式

将基址寄存器的值与索引寄存器的值相加，形成操作数的有效地址。例如： LDR R0, [R1, R2]

;R0  [R1+R2] c)多寄存器寻址

一条指令可完成多个（最多16个）寄存器值的传送。例如：

LDMIA R0, {R1, R2, R4};R1  [R0] 22

;R2  [R0+4];R4  [R0+8](6)寄存器移位寻址

操作数为寄存器中数做相应的移位而得到 例如：

ARM中的移位或循环移位操作：  LSL：逻辑左移（Logical Shift Left） LSR：逻辑右移（Logical Shift Right）ADD

R0, R1, R2, LSL #3;R0R1+8×R2  ASR：算术右移（Arithmetic Shift Right）。移位过程中保持符号位不变，若源操作数为正数，则字的高端空出的位补0。若源操作数为负数，则字的高端空出的位补1。

 ROR：循环右移（Rotate Right）。从字的最低端移出的位依次填入字的高端空出的位。 RRX：扩展的循环右移（Rotate Right Extended）。操作数向右移一位，左侧空位由状态寄存器C位填充。当移位的类型为RRX时，无需指定移位的位数，其它的则须指定移位的位数。(7)相对寻址

以PC当前值作为基地址，指令中的地址标号作为位移量，两者相加后得到操作数的有效地址。例如：

BL

NEXT

MOV „„

PC, LR

;从子程序返回 NEXT

;跳转至子程序 NEXT „„

(8)堆栈寻址

a)堆栈：按先进后出(FILO)的方式工作，使用堆栈指针(Stack Pointer, SP)指示当前操作位置。b)根据栈指针的指向位置可将堆栈分为

 满堆栈：SP指向最后压入堆栈的数据。 空堆栈：SP指向下个将放入数据空位置

c)根据堆栈的生成方式可将堆栈分为

 递增堆栈(Ascending Stack)：堆栈由低地址向高地址生成。

 递减堆栈(Descending Stack)：由高地址向低地址生成。d)ARM支持四种类型堆栈工作方式

满递增堆栈：SP指向最后压入的数据，且由低地址向高地址生成。满递减堆栈：SP指向最后压入的数据，且由高地址向低地址生产。

空递增堆栈：SP指向下个将放入数据的空位置，且由低地址向高地址生成。空递减堆栈：SP指向下个将要放入数据的空位置，且由高地址向低地址生成。

3.2 ARM指令集

(1)ARM指令集分类 加载/存储指令

数据处理指令 分支指令

状态寄存器访问指令 异常/中断指令 协处理器指令(2)ARM指令的特点 所有指令都是32bit；

大多数指令都在单周期内完成； 所有指令都可以条件执行； load/store体系结构；

指令集可以通过协处理器扩展。(3)ARM指令的格式

(4)ARM指令的条件执行

所有ARM指令都可包含一个可选的条件码，只有当CPSR中条件标志位满足指定条件时，指令才会被执行。否则以NOP指令通过流水线。

(5)加载/存储指令

Load：将内存中数据装载到寄存器 Store：将寄存器中的数据存入内存 a)单寄存器传输指令

{} {B} Rd, addressing1 {} SB | H | SH Rd, addressing2 {} {B} Rd, LABEL LDR/STR指令：寄存器在前，地址在后 LDM/STM指令：地址在前，寄存器在后

LABEL：相对PC的寻址方式。编译器在汇编时，会将标号LABEL汇编成PC的偏移量存入该指令的立即数字段。

B: Byte；S: Sign；H: Half LDR/STR：读/写一个32bit字到/从一个32位寄存器，要求读/写地址字对齐。LDRB：内存8bit字节32bit寄存器；不要求地址对齐，寄存器高24位清零。STRB：寄存器低8位内存的某个地址；不要求地址对齐。

LDRH：16bit半字 32bit寄存器；要求地址半字对齐，寄存器的高16bit清零。STRH：寄存器低16bit内存；要求地址半字对齐。

LDRSH：有符号16bit半字32bit寄存器中；要求地址半字对齐，寄存器高16bit根据符号位扩展。

LDRSB：有符号8bit字节32bit寄存器中；不要求地址对齐，寄存器高24bit根据符号位扩展。

“数据”一列指的是这条指令所访问的存储单元

“基址寄存器”一列的内容是指执行指令之后“基址寄存器”的内容 b)多寄存器传输指令

{} Rn{!}, {^} LDM / STM：从由基址寄存器指示的一片连续存储器到寄存器列表所指示的多个寄存器之间传送数据。

!：表示执行完操作后将变化之后的地址值写入基址寄存器

^：对于LDM操作，如恢复的寄存器中含有PC(R15)寄存器，则指令执行的同时CPU自动将SPSR拷贝到CPSR中，例如：

LDMFD {R0-R12, LR, PC}^ 数据的传送发生在User用户模式下的寄存器，而非当前模式寄存器，例如：

LDMDB SP, {R0-LR}^ 例如：

STMIA R0!, {R1-R5};以R0为地址指针，将R1-R5保存到内存，指针向上移动

c)交换指令

SWP {B} {} Rd, Rm, [Rn] SWP：字交换；SWPB：字节交换 Rd  [Rn], [Rn]  Rm 例如：

SWP R1, R2, [R3];R1[R3] ,[R3] R2 SWP R1, R1, [R2];R1与[R2]内容互换(6)数据处理指令

a)数据传送指令

MOV {cond} {S} Rd, Operand 例如： MOV R1, R0 MOVEQ PC, R14

;将R14值传到PC b)数据取反传送指令 MVN {cond} {S} Rd, Operand 例如：

MVN R1, #2;将立即数2取反送至R1 c)算术运算指令

ADD、ADC ADD|ADC {cond} {S} Rd, Oper1, Oper2 例如： ADDS ADCS ADCS ADC R0, R4, R8 R1, R5, R9;加低端的字;带进位加第二字

R2, R6, R10;带进位加第三字 R3, R7, R11;带进位加第四字

可实现 128位加法 SUB：减法指令

SUB {cond} {S} Rd, Oper1, Oper2 SBC：带借位减法指令

SBC {cond} {S} Rd, Oper1, Oper2 例：SBCS R0, R1, R2;R0=R1–R2–C RSB：反向减法指令

RSB {cond} {S} Rd, Oper1, Oper2 操作数2减去操作数1，例如： RSB R0, R1, R2;R0 = R2 – R1 RSC：带借位的逆向减法指令 RSC {cond} {S} Rd, Oper1, Oper2 d)逻辑运算指令 AND：与

AND {cond} {S} Rd, Oper1, Oper2 ORR：或

ORR {cond} {S} Rd, Oper1, Oper2 EOR：异或

EOR {cond} {S} Rd, Oper1, Oper2 e)比较指令 CMP CMP {cond} Operand1, Operand2 CMN：反值比较 CMN {cond} Operand1, Operand2 将第一个数与第二个数的反值进行比较，相当于完成两个数相加。例如： CMN R1, R0;R1+R0，并设置CPSR f)测试指令 TST TST {cond} Operand1, Operand2

把两个操作数按位进行与运算，根据结果更新CPSR。

TEQ TEQ {cond} Operand1, Operand2 把两个操作数按位进行异或运算，根据结果更新CPSR g)乘法指令 MUL MUL {cond} {S} Rd, Oper1, Oper2 RdOperand1×Operand2。例如： MUL

R0, R1, R2 MLA MLA {cond} {S} Rd,Oper1,Oper2,Oper3 RdOperand1×Operand2+Operand3 例如：

MLA R0, R1, R2, R3;R0 = R1×R2+R3;R0 =R1×R2 UMULL：64位无符号数乘法指令 UMULL {cond} {S} Rd\_L, Rd\_H, Operand1, Operand2 Rd\_H:Rd\_L Operand1×Operand2 例如： UMULL R0, R1, R2, R3;R0 =(R2×R3)的低32位;R1 =(R2×R3)的高32位 UMLAL： 64位无符号数乘加指令 UMLAL {cond} {S} Rd\_L, Rd\_H, Operand1, Operand2 Rd\_H:Rd\_L+= Operand1×Operand2 例如：

UMLAL

R0, R1, R2, R3;R0 =(R2×R3)的低32位+R0;R1 =(R2×R3)的高32位+R1 SMULL：64位有符号数乘法指令 SMULL {cond} {S} Rn\_L, Rn\_H, Operand1, Operand2 Rd\_H:Rd\_L Operand1×Operand2 SMLAL： 64位有符号数乘加指令 SMLAL {cond} {S} Rn\_L, Rn\_H, Operand1, Operand2 Rd\_H:Rd\_L+= Operand1×Operand2(7)分支指令

a)B(分支指令)和 BL(带链接分支指令)B {cond} LABEL BL {cond} LABEL

允许向前或向后跳转最高32MB。

BL：带返回的分支指令，用于调用一个将其返回地址存入链接寄存器的函数。例如：

SUBPRG BL …… SUBPRG „„; MOV PC, LR;返回

b)BX(分支并可选地交换指令集)和BLX(带链接分支并可选地交换指令集)BX {cond} Rm BLX {cond} LABEL | Rm Rm：转移地址，bit[0]为0时，目标地址处为ARM指令，否则为Thumb指令。BX 和 BLX是唯一可使用的切换指令集的方法。

BX 和 BLX指令执行与B和BL指令相同的分支操作，并可从ARM指令集切换到THUMB指令集。

c)长跳转

通过向PC寄存器中写目标地址值，可实现在4GB地址空间中进行任意跳转。例如： MOV MOV LR, PC;保存返回地址

R15, #0x00110000;无条件转向0x110000(8)状态寄存器访问指令 a)MRS MRS {} Rd, 将状态寄存器的内容传送到通用寄存器中。例如： MRS b)MSR MSR {} \_, Rm MSR {} \_,# 表示下列情况之一： c：控制域，即PSR[7:0] PSR[32:24] CPSR|SPSR  通用寄存器。例如： MSR CPSR\_F, #&F00000000

R3, CPSR;CPSRR3

x：扩展域，即PSR[15:8] s：状态域，即PSR[23:16] f：标志域，即

;设置所有标志位

(9)异常/中断指令（续）a)SWI SWI {} 产生SWI异常中断，以实现在用户模式下调用操作系统的监控功能程序。它将处理器置于管理(SVC)模式。例如：

SWI 0x01

;调用编号为01的系统例程

b)BKPT BKPT 产生软件断点中断，可用于程序调试。例如 BKPT 0xF010(10)协处理器指令 a)CDP CDP {} , , CRd, CRn, CRm {,} 编号为CP#的协处理器接受指令并执行。具体操作由Cop1和Cop2定义，CRn和CRm为源操作数，结果CRd。例如：

CDP P6, 2, C5, C10, C3, 16

;激活协处理器P6的操作，操作码1和操作码2值分别为2和16，目标寄存器为C5，源操作数寄存器为C10和C3 b)MRC和MRC MRC | MCR {} , , CRd, CRn, CRm {,} 在ARM与协处理器寄存器间传送数据。MRC：ARM寄存器协处理器寄存器 MCR：协处理器寄存器ARM寄存器 例如： MCR P3, 3, R0, C4, C5, 6;ARM处理器R0协处理器P3的寄存器C4和C5 c)LDC和STC 在协处理器寄存器与存储器间传送数据 LDC：协处理器寄存器存储器

STC：存储器协处理器寄存器 前变址格式

LDC | STC {} {L} , CRd, [Rn, ]{!} 后变址格式

LDC | STC {} {L} , CRd, Rn, 3.3 Thumb指令集

(1)Thumb指令集的特点

 是ARM指令集压缩形式的子集，所有Thumb指令均有对应的ARM指令。 采用16位二进制编码，代码密度小。

 执行Thumb指令时，先动态解压缩，然后作为标准的ARM指令执行。 如何区分指令流取决于CPSR的位T。 大多Thumb数据处理指令采用2地址格式。 移位操作变成单独指令。

 没有协处理器指令、单寄存器交换指令、乘加指令、64位乘法指令及程序寄存器处理指令。 仅分支指令B有条件执行功能。

(2)Thumb状态切换

a)ARM状态进入Thumb状态 执行带状态切换的转移指令BX。例如： BX R0;若R0最低位为1，则转Thumb状态

异常返回。通常用于返回到进入异常前所执行的指令流，而不是特地用于切换到Thumb模式。适用于异常前执行的是Thumb指令。

b)Thumb 状态进入ARM状态 执行BX指令。

利用异常进入ARM指令流。(3)Thumb指令寄存器的使用  对R0~R7具有全部访问权限。

 R8~R12的访问受到限制，只能通过MOV、ADD和CMP访问。

 在每一种特权模式下都有一组SP、LR和SPSR，分别对应ARM状态的R13、R14和R15。(4)Thumb指令与ARM指令的相似点  Load/Store结构。

 支持8位字节、16位半字和32位字数据类型。 半字以2字节边界对准，字以4字节边界对准。(5)Thumb指令与ARM指令差异点

 跳转指令：条件跳转在范围上有更多的限制，转向子程序只具有无条件转移。

 数据处理指令：对通用寄存器进行操作，操作结果需放入其中一个操作数寄存器。因此，许多Thumb指令采用2地址格式，ARM数据处理指令大多采用3地址格式。 单寄存器加载和存储指令：只能访问R0~R7。

 PUSH和POP指令使用堆栈指针R13作为基址实现满递减堆栈，除R0~R7外，PUSH指令还可存储R14，POP指令可加载程序指令PC。

 大多数Thumb指令是无条件执行的，所有ARM指令都是条件执行的。 由于采用高密度编码，Thumb指令格式没有ARM指令格式规则。(6)Thumb分支指令

a)B 指令——目标为Thumb代码 B 偏移量左移1位

B指令是Thumb指令集中唯一可条件执行的指令。PC = PC +(SignExtend(signed\_immed\_8)

3.4 3.5 35

3.6 嵌入式系统设计与开发过程

(1)嵌入式软件开发的特点 a)需要交叉开发环境

 交叉开发环境：实现编译、链接和调试应用程序代码的环境，它分散在有通信连接的宿主机与目标环境之中。

 宿主机(Host)是一台通用计算机，一般是PC机。

 目标机(Target)可以是嵌入式应用软件的实际运行环境，也可以是能替代实际环境的仿真系统。

 交叉软件开发工具包括：交叉编译器、交叉调试器和模拟软件等。b)引入任务设计方法

 嵌入式应用系统以任务为基本执行单元  用多个并发的任务代替通用软件的多个模块 c)需要固化程序 d)软件开发难度大

 实时性。

 稳定性、可靠性、抗干扰性。(2)软硬件平台的选择 选择处理器需考虑的因素  处理性能  技术指标、功耗

 软件支持工具、是否内置调试工具  供应商是否提供评估板 软件平台的选择

 操作系统。开发工具，与硬件接口难度，内存，是否提供硬件驱动程序，可裁剪性、实时性等

 编程语言。通用性、可移植性、执行效率、可维护性等  集成开发环境

(3)嵌入式系统设计与开发流程  需求分析  体系结构设计

 硬件的设计、制作及测试  软件的设计、实现及测试  系统集成

 系统性能测试及可靠性测试

STM32微控制器

教学目的：使学生对STM32的性能指标、硬件结构有一定的了解。教学重点：STM32的硬件结构。教学难点：STM32的硬件结构。

教学方法与教学手段：课堂讲授，多媒体教学。教学时间：4课时。教学内容

4.1 STM32微控制器的性能指标

(1)STM32F103的主要功能

 ARM Cortex-M3核  最高主频72MHz  512kB Flash、64kB RAM  1个USB、1个CAN、5个USART、3个SPI、2个I2C、2个I2S、1个SDIO、112个GPIO

 3个ADC、2个DAC、8个定时器

 FSMC总线（支持NOR，NAND，SRAM） 12通道DMA控制器

 Serial wire debug(SWD)、JTAG 接口  启动：用户Flash、系统存储器、SRAM(2)JC-STM32CB核心板

 核心芯片：ST STM32F103ZET6  8M NOR Flash：SST39VF6401，16bit  128M NAND Flash：K9F1G08U0C，8bit  8MHz CPU晶振，32768Hz RTC晶振  1个10M/100M以太网接口  1路音频输出和1路音频输入  1个10-pin JTAG调试接口  1个4-pin SWD串行调试接口  Reset按钮、自定义LED，电源LED  1个精密可调电阻连接到片内ADC输入

 使用CPLD扩展IO口及地址译码，型号EPM240T100C5N  板载10/100M网卡控制器，型号DM9000AEP，16bit总线接口

STM32

本文档由站牛网zhann.net收集整理，更多优质范文文档请移步zhann.net站内查找