

嵌入式软件开发导论

徐钦桂 老师

xqg@dgut.edu.cn

东莞理工学院计算机学院

2008

简介

- 嵌入式领域的一门技术基础课
- 3学分课程，每周3课时，共48学时
- 已成功开设过3年

目标

- 使学生理解嵌入式系统和嵌入式软件的基本概念和特点
- 理解 **Linux**的功能和体系结构,特别是它与其它嵌入式操作系统的差异
- 了解**Linux**内核定制、设备驱动程序编写、嵌入式应用程序开发方法
- 通过实验和综合实验让学生有更多的实践

成绩评定

- **20%作业+30%实验+50%考试**

课程安排

嵌入式系统和嵌入式操作系统概念：2 学时	Qt图形界面移植与编程：4学时
嵌入式Linux使用 and 开发基础：2学时	嵌入式Web服务器：2学时
Linux shell编程：2学时	AD/DA设备驱动程序：4学时
构建嵌入式Linux系统：6学时	电机驱动设备驱动程序：4学时
Bootloader分析与移植：4学时	CAN总线设备驱动程序：4学时
Linux内核分析：4学时	应用案例1：4学时
Linux根文件系统创建：2学时	应用案例2：4学时
嵌入式Linux多线程编程：2学时	

学生如何学习

- 1. 教师：必要的解说、实验演示
- 2. 学生：
 - 掌握必要的原理；
 - 仿照教师完成每个实验；
 - 阅读、思考和理解各个实验的源程序；
 - 将各实验联合起来运行形成独立功能的嵌入式软件程序。
- 3. 学生之间：交流

教材及参考书

- 许信顺，贾智平. “嵌入式Linux应用编程”，机械工业出版社
- 李俊. 嵌入式Linux设备驱动开发详解. 人民邮电出版社

嵌入式操作系统

第一讲. 嵌入式系统和 嵌入式操作系统

东莞理工学院软件学院

2008

目 录

- 什么是嵌入式系统？
- 嵌入式系统特点
- 嵌入式系统结构
 - 嵌入式硬件
 - 嵌入式软件
- 什么是嵌入式操作系统？
- 常用嵌入式操作系统简介
 - **Windows Embedded**
 - **VxWorks**
 - **Embedded Linux**

1.什么是嵌入式系统？

定义：嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。

简而言之，它是完成特定任务的计算机系统。

什么是嵌入式系统？

- 程序特点：嵌入式系统里的程序是被**写死**的。系统上电后程序开始执行直至系统关闭，程序是**不能被改变**的，除非开发人员采用特定的方法才能对程序进行改进并重新写入系统；
- 存在形式：嵌入式系统往往做为一个大型系统的组成部分被嵌入到该系统中(这也是它名称的由来)，嵌套关系可能相当复杂，也可能非常简单，它的表现形式多种多样。

• 嵌入式系统应用



PDA



消费电子



信息家电



移动通信



GPS



智能识别系统



导航系统



汽车电子



cellular phones

•嵌入式系统应用

- ✓ 工业控制
- ✓ 火控系统
- ✓ 数字电视
- ✓ 飞行控制系统
- ✓ 测试仪器
- ✓ 医疗设备
- ✓ 游戏机
- ✓ 等等



Mars, December 3, 1999
Crashed due to uninitialized variable



2. 嵌入式系统特点

- 实时性
- 小尺寸（资源，代码，规格.....）
- 低功耗
- 高效率
-

什么是实时性？

定义：对于实时系统，它的正确性不仅与系统的逻辑正确性相关，而且与系统的响应时间相关。如果系统的响应不能满足时限要求，即使它能得到正确的输出，我们也只能说它是一个失败的响应。

特点：

- 在时限范围内的正确响应
 - ✓ 通常是嵌入式系统
 - ✓ 通常是分布式系统
 - ✓ 对时限要求的不同
- 硬实时系统
- 软实时系统

硬实时和软实时

- 硬实时

系统对时限的要求特别严格，如果不满足时限要求会给系统带来灾难性后果。如飞行控制系统。

- 软实时

系统对时限的要求不是很迫切，如果不能满足时限要求，系统仍然可以正常工作，只是性能有所影响而已。如数据采集系统。

实时系统实例

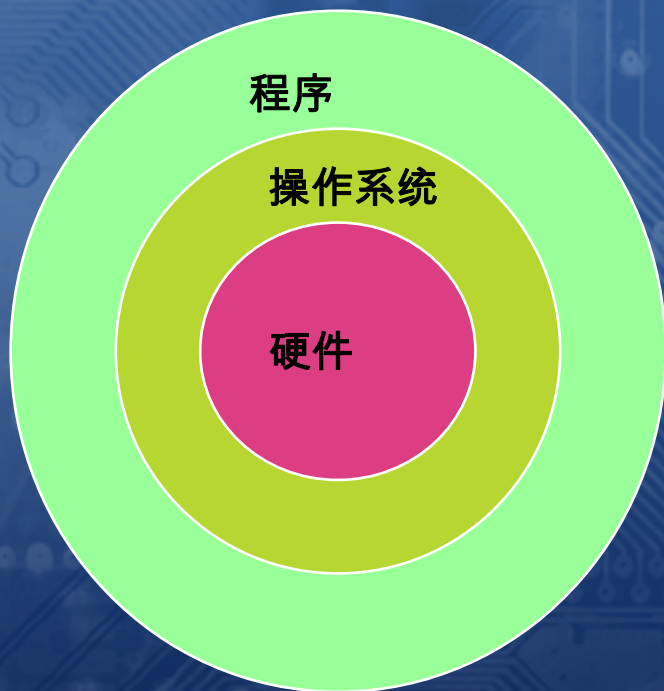
- 硬实时系统应用

- 汽车 (沃尔沃S80有19台计算机)
- 飞机 (JAS)
- 医疗设备
- 空间设备 (火星探测器)
- 军方系统
- 工业自动化

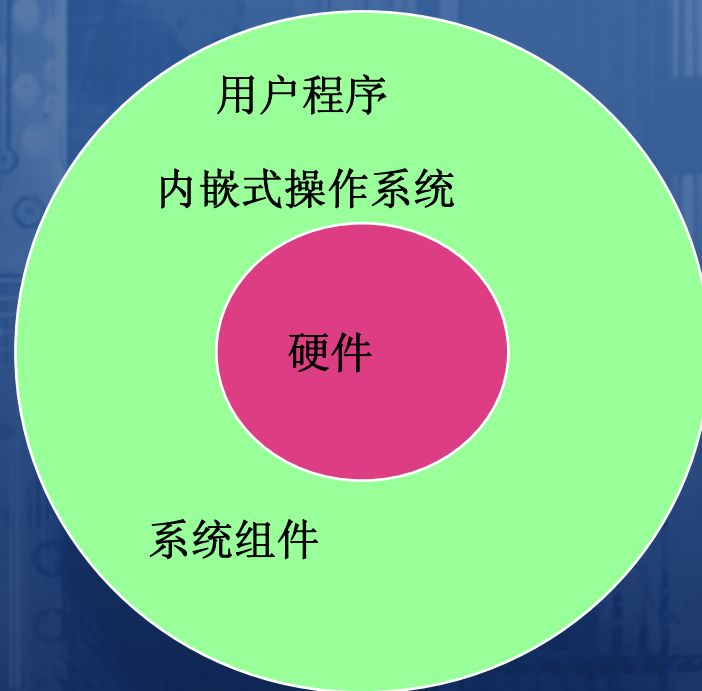
- 软实时系统应用

- 游戏
- DVD (MPEG 编码)
- 英特网视频和广播
- 通讯

3. 嵌入式系统结构



计算机系统结构



嵌入式系统结构

(1) 嵌入式系统硬件

- CPU
- 内存
- 外部设备
 - 传感器(Sensors)
 - A/D转换
 - 显示设备

● CPU

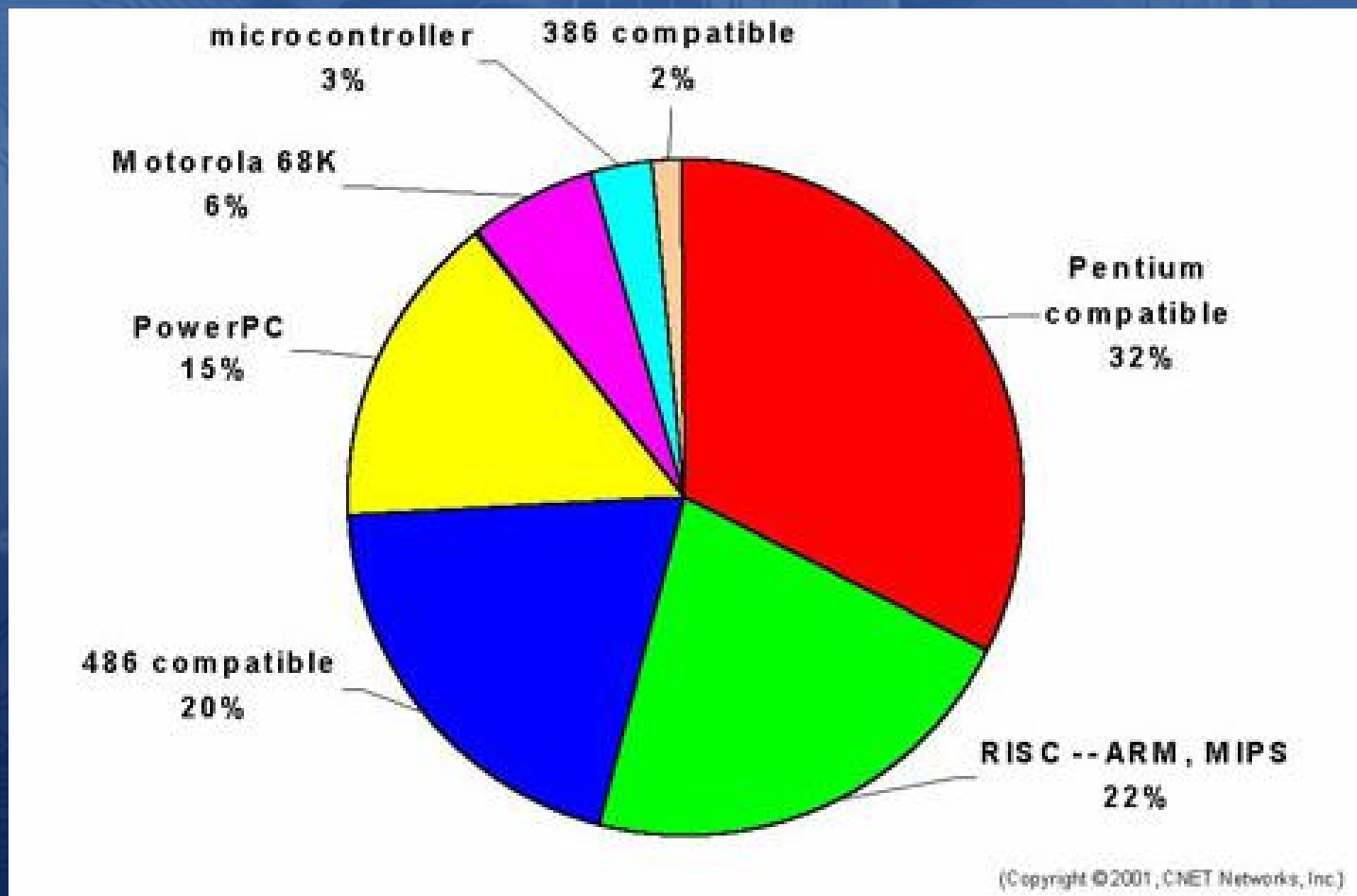
- Intel x86
- PowerPC(Mac) G3,G4,G5
- SPARC, Alpha
- ARM
- MIPS
-

● 位宽

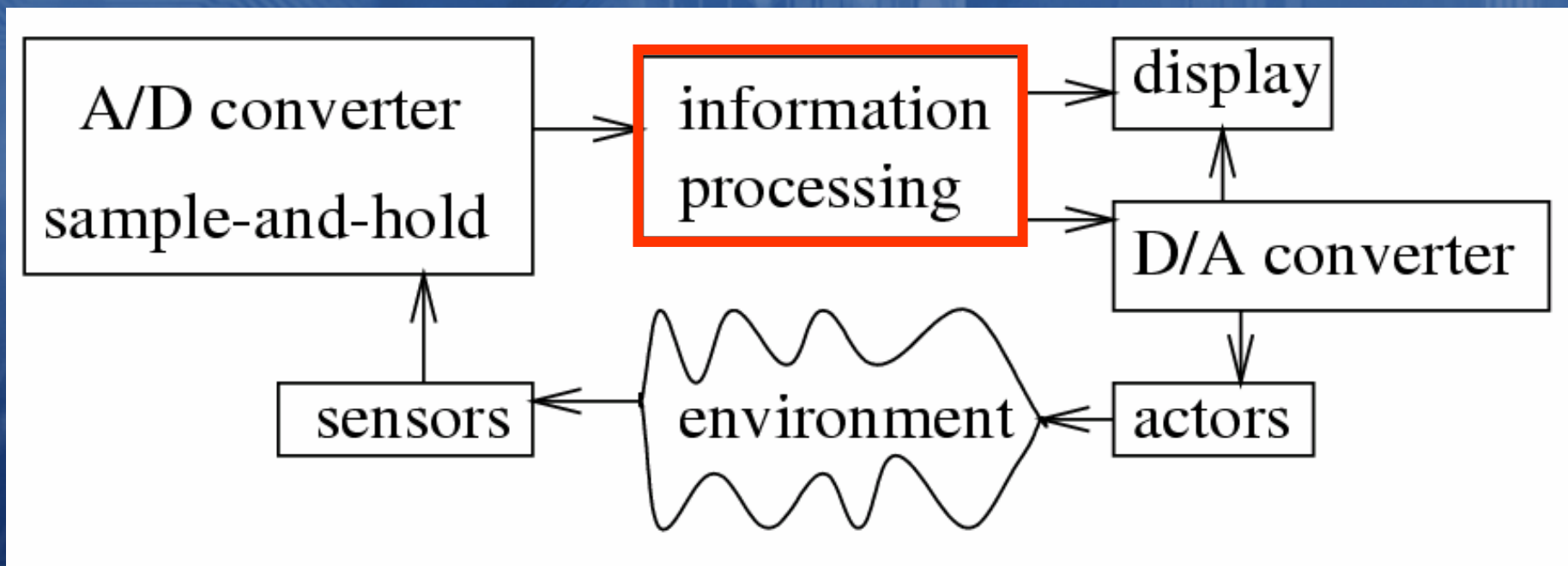
- 8位控制器(仍然存在!)
- 16位控制器(主流)
- 32位控制器(开始流行)
- 64位控制器(高性能)

嵌入式系统硬件

● CPU市场份额



嵌入式硬件实例



问题:功耗和能量

- “电能是嵌入式系统的最大约束”

[in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

- 目前的**UMTS**电话系统几乎不可能在数据一直被传输的情况下被操作多于一小时的时间。

[from a report of the Financial Times, Germany, on an analysis by Credit Suisse First Boston; <http://www.ftd.de/tm/tk/9580232.html?nv=se>]

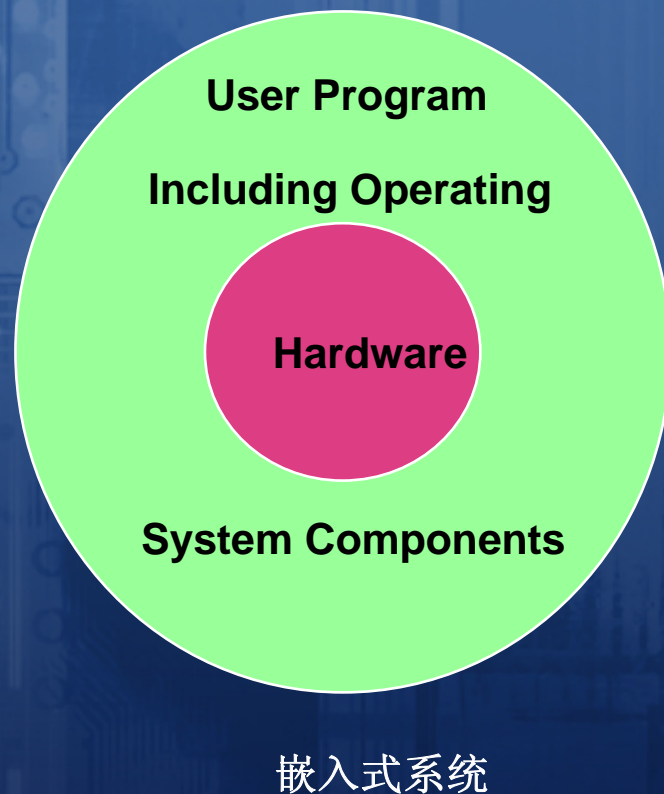
(2)嵌入式软件 ——目标

- 设计可靠、稳定、高效的嵌入式软件，
需要考虑

- 并行性
- 兼容性
- 实时性
- 层次
- 继承性
- 有限的资源
- 多样性
- 可读性

嵌入式系统软件的特点

- 有限的资源
- 实时性
- 操作系统与用户软件没有明显的界线
- 开发模式



嵌入式系统软件的今天



嵌入式系统软件的明天



第一章 作业

- 你是如何理解嵌入式系统的？
- 在日常生活中，你接触过哪些嵌入式产品？他们都有些什么功能？
- 嵌入式系统中的软件有哪些特征？
- 比较嵌入式系统与普通的计算机系统存有哪些相同和不同之处。

4.什么是嵌入式操作系统

各式各样的OS:

- 桌面机
 - Windows (9X, XP Home, XP/2000 Pro)
 - Mac
- 服务器
 - Windows (XP/2000 Server &Advanced Server)
 - Unix Varieties
- 嵌入式
 - Many

嵌入式操作系统定义：

- 嵌入式系统是使用特定嵌入式软件完成特定功能的计算机系统，嵌入式操作系统作为**软件的组成部分，为嵌入式软件的开发和运行提供良好的环境。**
- 嵌入式系统可以是**基于ROM或者是磁盘的系统**，类似**PC**，但它并不能替代通用计算机系统。

嵌入式操作系统特点：

- 模块化
- 可升级
- 可配置
- 等等...
- 小内存损耗
- **CPU**支持
- 设备驱动

实时操作系统RTOS的构成:

- 多线程和抢占式调度
- 必须支持可预测线程同步机制
- 优先级继承系统

5. 常用嵌入式操作系统:

- **Microsoft**

- **Embedded NT/XP**

- “实时” 控制

- **Windows CE (CE.NET)**

- Internet 设备

- **Pocket PC 2002**

- Handheld PC's and PDA's

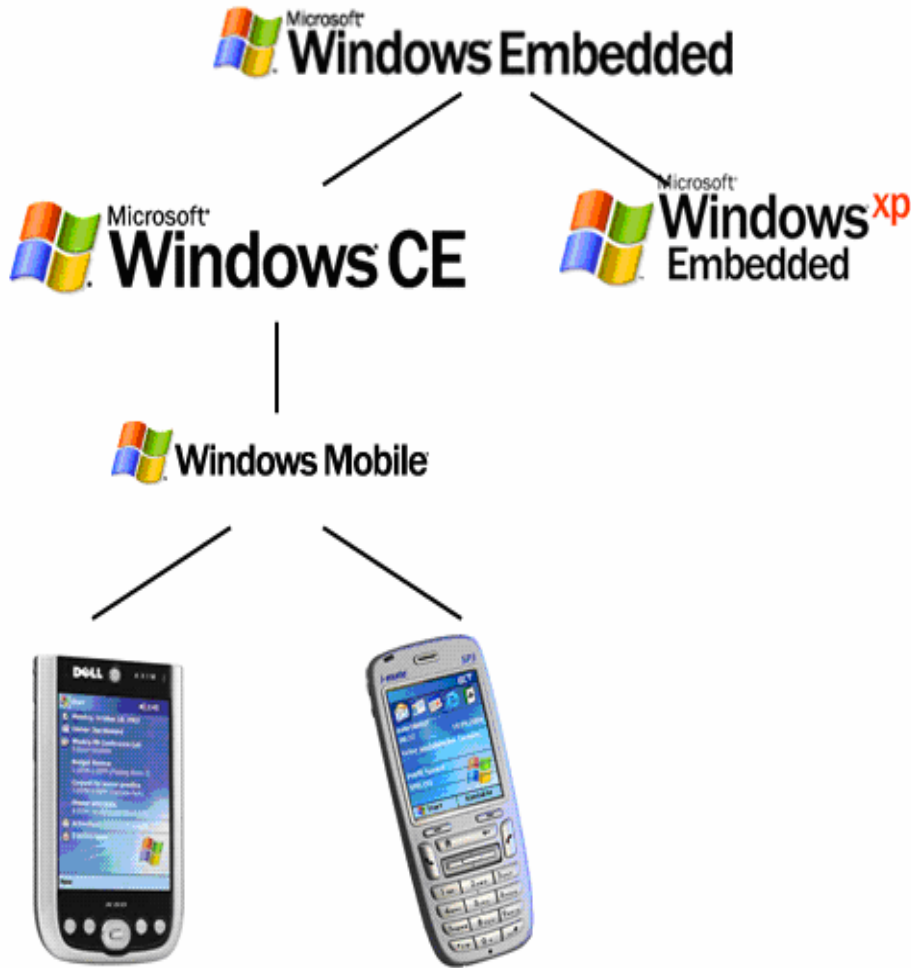
- **Wind River Systems**

- **VxWorks**

- **pSOS**

- **嵌入式Linux**

(1) Windows Embedded 家族



1.Windows XP embedded:

Windows XP的可裁剪组件化版本,在嵌入式应用领域取代PC;

2.Windows CE:重新设计的小型化嵌入式Windows版本,与桌面Windows在开发模式和运行方式上兼容;

3.Windows Mobile: 针对移动通信应用升级的将Windows CE;

微软的移动平台

Pocket PC

- 信息消费
- 浏览和输入数据
- 把电话融入PDA
- 可以与Office, Exchange和SQL Server交互
- .NET Compact Framework
- ASP.NET 移动控件

Smartphone

- 信息消费
- 基本数据浏览
- 把PDA融入电话
- 可以与Exchange交互
- .NET Compact Framework
- ASP.NET 移动控件

小型个人产品

- 单向网络
- 信息消费

Windows CE

Windows Mobile

Windows XP/XPE

更强的功能



笔记本PC

- 复杂的文档编辑和读写
- 桌面键盘输入
- 键盘和鼠标输入法
- 完整的.NET framework支持



平板电脑

- 复杂的文档编辑和读写
- 桌面键盘输入
- 支持数字墨水
- 可以支持键盘, 也可以把键盘拿走
- 键盘、鼠标、数字墨水和语音输入
- 完整的 .NET framework支持
- 提供笔, 数字墨水, 手写和语音识别API

(2) VxWorks

特点:

- **VxWorks** 是风河公司开发的一款**商用硬实时操作系统**
- 主要思想: 在嵌入式系统中**最大限度地实现内核的时间可预测性, 根据用户定义的任务优先级对任务实现调度。**
- **给用户最大的控制权**

VxWoks

设计目标:

- 为追求系统的实时性而设计的，并不是以通用**OS**为设计目标。
- 去掉了一些**OS**模块，因为这些模块在某种程度上会影响系统的实时性 (如在内存管理中沒有采用页面管理模式，采用的是平板式内存。

VxWoks

任务调度:

- 任务调度采用的是**基于优先级的抢占式任务调度模式**，**优先级分256级(0-255)**
- 用户可以动态的改变优先级，但是这种做法不提倡
- 用户可以**锁定一个任务**使它不被更高的任务或中断抢占
- 允许使用固定优先级响应时间来检查任务调度的性能

VxWoks

实时性能实现技术:

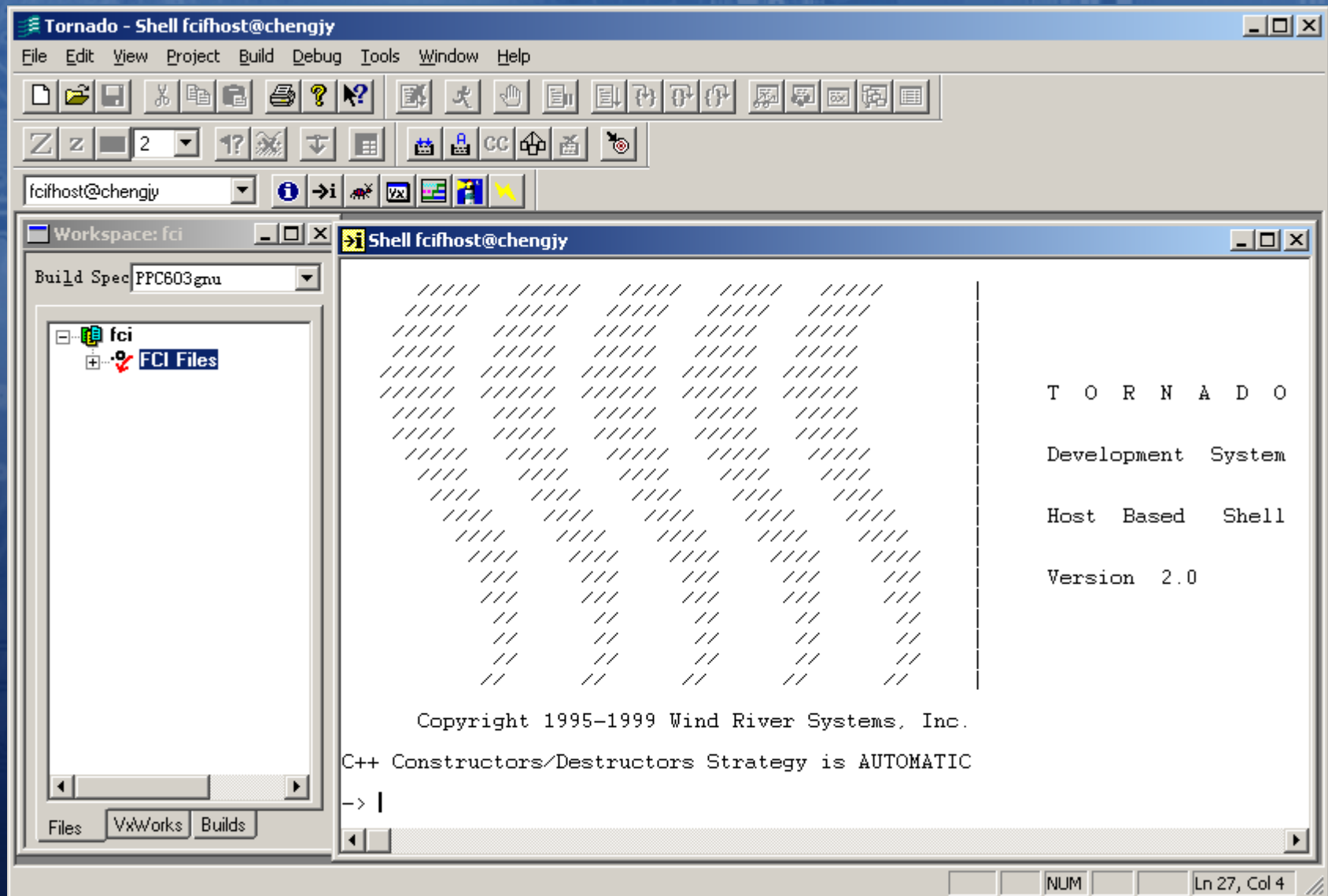
- 资源共享和优先级继承机制
- 采用最优化的上下文切换和中断返回机制.
- 内核从不禁止非屏蔽中断 **NMI (non-maskable interrupts)**

VxWoks

缺点:

- 缺少某些OS特性
- 保证时限要求是设计者自己的任务(系统的灵活性带来的弊端)
- 不支持很多应用和APIs(只支持部分POSIX标准的函数集)
- 尽管采用了平板式内存管理，但是由于内存的动态分配，仍然存在内存段，这样仍然存在时间上的不可预测性
- 应用领域主要局限在对实时性要求较严格的硬实时系统中

Tornado —— 集成开发环境



GDB —— 调试工具

The screenshot shows the Tornado IDE interface for debugging a C++ program. The main window displays the source code of `player.cpp` at line 118, column 30. A context menu is open over the code, with the `Run to Cursor` option selected. The right sidebar contains two panels: **Locals** and **Watch**.

Locals Panel:

Name	Value
<code>this</code>	<code>0x3fe4fc</code>
<code>Party_T</code>	
<code>Runnable</code>	
<code>pMyLink_</code>	<code><incomplete typ</code>
<code>static count_s</code>	<code>6</code>
<code>theDemo_</code>	<code>0x3fe944</code>
<code>pThisTask</code>	<code>0x3fe4fc</code>

Watch Panel:

Name	Value
<code>pMsg</code>	<code>0x384c4c</code>
<code>sync</code>	<code><incomplete type></code>
<code>out</code>	<code>@0x9858a: <incomplete t</code>
<code>pNextPlayer</code>	<code>0x3fe454</code>

The bottom status bar shows the program is stopped at line 118, column 30. The command window at the bottom left shows the command `(gdb) display /W this` and the output `display /W this`.

WindView —— 多任务跟踪和观察工具



(3)嵌入式Linux

- 嵌入式Linux概览
- 使用嵌入式Linux的开发过程
- 嵌入式Linux与Windows CE

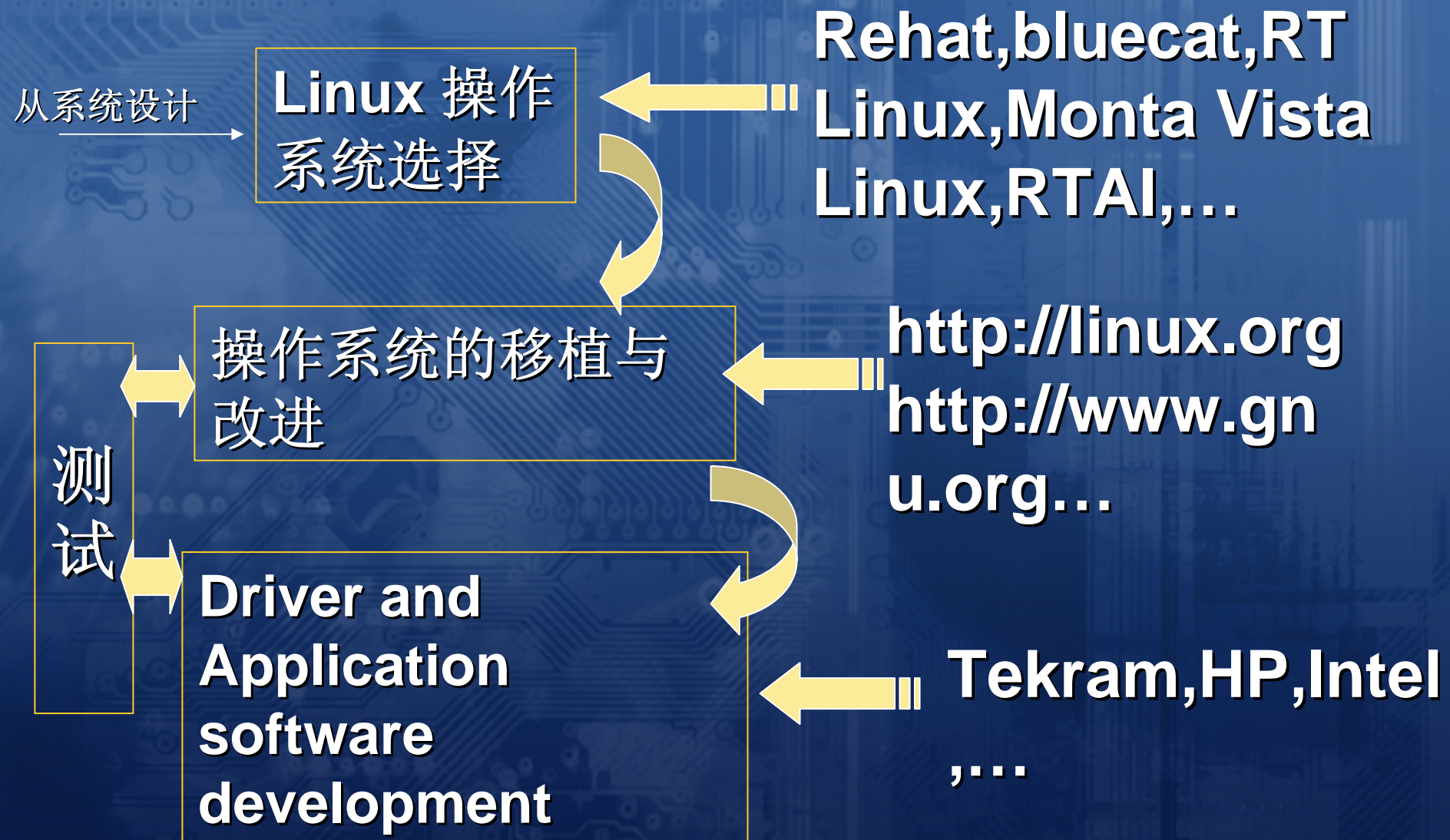
什么是嵌入式Linux

嵌入式Linux系统就是利用Linux其自身的许多特点，把它应用到嵌入式系统里。

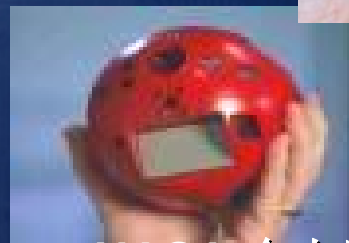
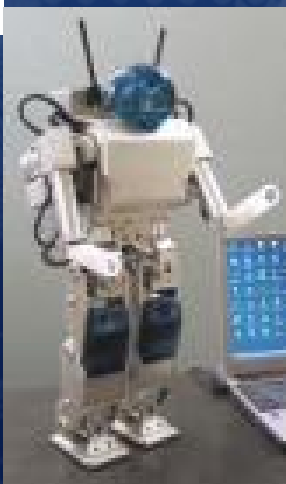
优势：

- Linux是开放源代码的，不存在黑箱技术，遍布全球的众多Linux爱好者又是Linux开发者的强大技术支持；
- Linux的内核小、效率高，内核的更新速度很快；
- Linux是免费的OS，在价格上极具竞争力。
- Linux适应于多种CPU和多种硬件平台，是一个跨平台的系统。到目前为止，它可以支持二三十种CPU。而且性能稳定，裁剪性很好，开发和使用都很容易。
- 近年来，出现很多嵌入式Linux：Embedix，ETLinux，LEM，Linux Router Project，LOAF，uCLinux，muLinux，ThinLinux，FirePlug，Linux和PizzaBox Linux

开发过程



将Linux嵌入到 各种设备中:

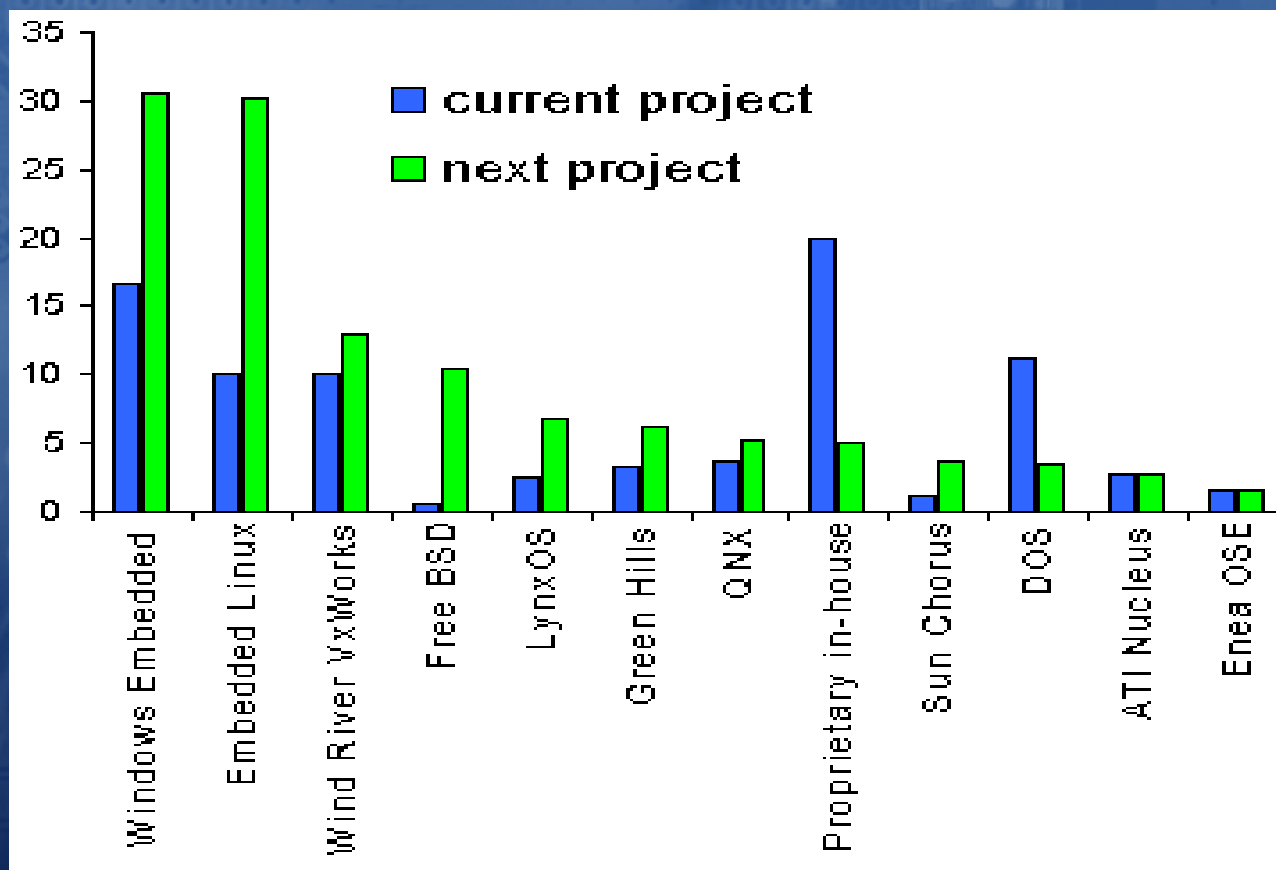


NASA 个人助理

嵌入式Windows 与 嵌入式Linux

版权费	非常便宜	并非所有的嵌入式 windows产品都便宜
开发环境	更好	不同意
功能	更多	不同意
上市时间	更短	不确定
开发人员数	更少	不确定
硬件支持	差不多	Linux更好

嵌入式Windows 与嵌入式Linux



两者都
呈强健
的上升
趋势!

目前嵌入式系统项目中使用的目标操作系统以及
二年后使用的操作系统，**2002**，数据来自**EDC**

作业

1. **Vxworks**操作系统有哪些优缺点？
2. 什么是嵌入式**embedded Linux**？给出几个例子。
3. 与嵌入式**Linux**相比嵌入式**Windows**有何优点？
4. 应用**Windows CE**与嵌入式**Linux**进行嵌入式系统开发，二者的开发流程有什么不一样？

第一讲 嵌入式Linux使用 和编程基础

教学目标

- 熟练掌握嵌入式Linux应用开发必要的操作命令、编辑软件、项目管理工具、开发环境；
- 掌握Linux应用编程方法
- 熟悉shell编程

目录

1. 认识Linux
2. Linux文件与目录系统
3. Linux基本操作命令
4. Linux编程环境
5. Shell编程
6. 编译运行嵌入式应用程序的过程

1.认识Linux

Linux发展历史及与Unix的关系:

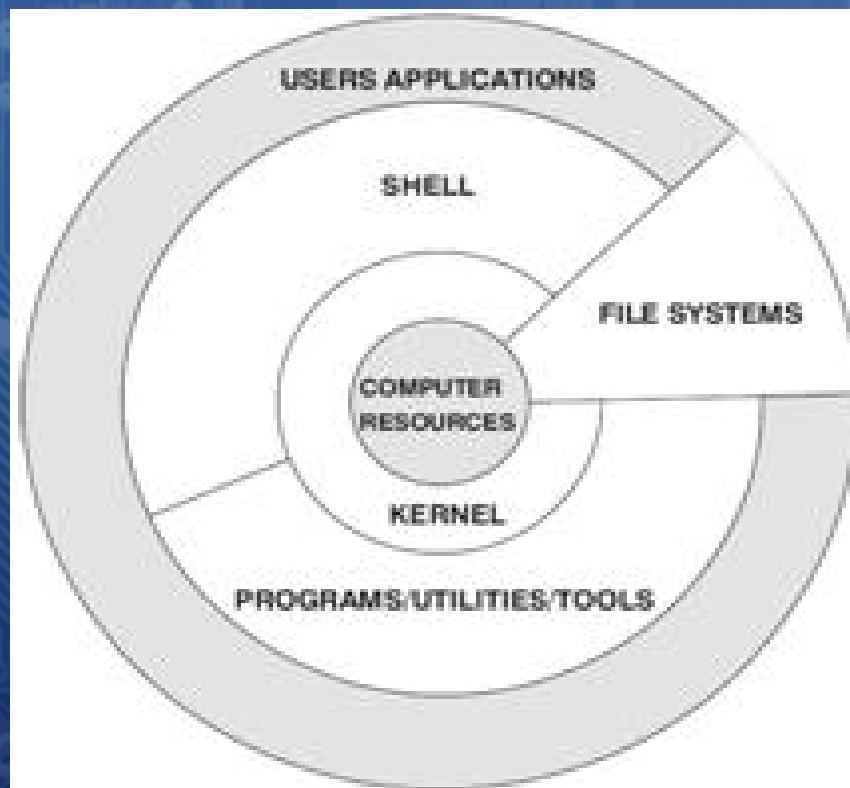
- **Linux**是为了让广大计算机爱好者获得学习和使用**UNIX**机会而开发的一种开源**UNIX**系统;
- **Linux**与**GNU**结合促进了二者的双赢发展: **Linux**采用**GNU**的**GPL**实现版权保护, 集成了**GNU**成功的软件开发模式, 并且避免了**UNIX**发展过程中存在的不兼容问题, 借用**GNU gcc**作为编译工具; **GNU**软件以**Linux**作为试验平台;
讨论: 如果没有Linux和GNU gcc, 计算机的应用和发展会有何影响?
- **Linux**和**GNU**的发展使广大计算机用户直接受益, 推动了计算机技术和应用的发展

Linux的特点:

- 出色的速度性能
- 多用户多任务
- 丰富的网络功能: ?
- 开放性: ?
- 可靠的系统安全: ?
- 良好的用户界面
- 良好的可移植性
- 良好的兼容性

Linux系统组成:

- Linux内核
- Linux shell
- Linux文件系统
- Linux应用程序



Linux版本

- 内核版本: **Linux Torvalds**领导下的开发小组开发的系统内核版本: **r.x.y**

- **r**:内核主版本
- **x**:偶数: 稳定版本, 奇数: 开发版本
- **y**:错误修补的次数

- **Linux发行版本** 一些组织和厂商将Linux打包

表1-1 常见的Linux发行版本

	发行版本	公司网址
国际发布与国内发布	Red Hat Linux	http://www.redhat.com/
	Mandrake Linux	http://www.linux-mandrake.com/en/
	SUSE Linux	http://www.suse.com/
	Debian Linux	http://www.debian.org/
	Caldera Linux	http://www.caldera.com/
	Redflag Linux	http://www.redflag.com.cn/
安全发布与小型发布	Astaro Security Linux	http://www.astaro.org/
	Engarde Secru Linux	http://www.engardelinux.org/
	ClarkConnect	http://www.clarkconnect.org/
	Linux Router Project	http://www.linuxrouter.org/

2.Linux文件系统

(1) Linux文件管理特点

- 要求：支持多种不同物理文件系统：
ext,ext2,ext3,msdos,vfat,ntfs,iso9660,devfs,.....;
- 问题：不同文件系统间差别很大：组织结构、操作函数、驻留载体、...;
- 解决思路：Linux把各种不同文件系统的操作和管理纳入到一个统一框架中，让系统中的文件系统界面成为一条文件系统总线，对不同文件系统进行操作；该框架对用户程序隐去各种不同文件系统的实现细节，为用户提供统一的、抽象的、虚拟的文件系统界面---虚拟文件系统（VFS）；
- VFS的应用程序接口：VFS对用户程序的接口由一组标准、抽象的文件操作构成，以系统调用形式提供给用户程序：
read,write,lseek,...；用户程序把所有文件看出一致的、抽象的VFS文件；

(2) VFS原理描述

文件操作实例：

```
fd=open("/home/user1/abc",RD_ONLY);
```

```
read(fd,buf,100);
```

文件系统号=?; 文件操作函数号=?

已安装文件系统表：

0	文件系统 类型: vfat	read函数 入口地址	write函数 入口地址	lseek函数 入口地址
1	文件系统 类型: ext2	read函数 入口地址	write函数 入口地址	lseek函数 入口地址
2	文件系统类 型:is09660	read函数 入口地址	write函数 入口地址	lseek函数 入口地址
3	文件系统类 型:proc	read函数 入口地址	write函数 入口地址	lseek函数 入口地址

(3) Linux文件系统类型

- **ext:** 第一个专门为Linux的文件系统类型，叫做扩展文件系统。它在1992年4月完成的。它为Linux的发展取得了重要作用。但是在性能和兼容性上存在许多缺陷。现在已经很少使用了；
- **ext2是:** 为解决ext文件系统的缺陷而设计的可扩展的高性能的文件系统，特点为存取文件的性能极好，对于中小型的文件更显示出优势，这主要得利于其簇快取层的优良设计，缺点是系统突然断电可能造成在文件系统就会处于不一致的状态；
- **ext3:** 由开放资源社区开发的日志文件系统,可靠性好,具有较好的恢复功能；
- **jfs**提供了基于日志的字节级文件系统，该文件系统是为面向事务的高性能系统而研发的，具有快速重启能力：**Jfs** 能够在几秒或几分钟内就把文件系统恢复到一致状态。
- **ReiserFS:** 基于平衡树结构，根据需要动态地分配索引节，而不必在

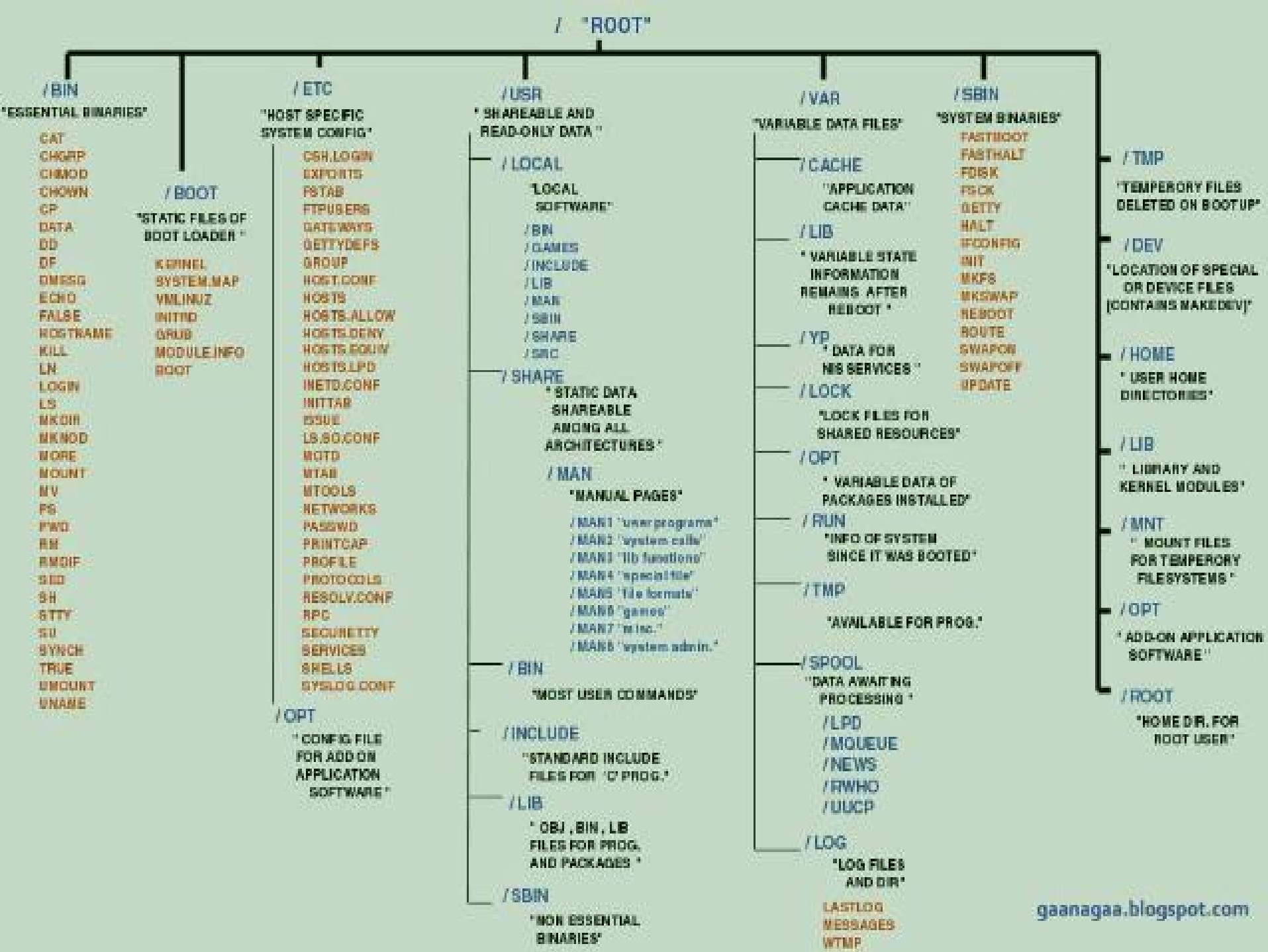
(3) Linux文件系统类型

- **NTFS**: 微软视窗系统 NT内核的系列操作系统支持的、一个特别为网络和磁盘配额、文件加密等管理安全特性设计的磁盘格式;
- **Cramfs**, 是2.4系列Linux内核提供的一种新的文件系统。它是一种压缩的、只读的文件系统。它主要的优势是所有存储的文件都是压缩的,而且这些文件只是在被访问到的时候才解压到**RAM**中,而不在访问之列的文件并没有被解压到**RAM**中。这样, **Cramfs**能有效减少**Flash**和**RAM**的占用量,但不足之处是需要的指令比较多,不支持**XIP**特性。;
- **JFFS** (**J**ournaling **H**ash **F**ile **S**ystem), 是专门针对嵌入式系统中**Hash**存储器的特性而设计的一种日志文件系

3.Linux 目录结构

UNIX和Linux共有的目录结构规范:

```
/
/bin
/boot
/dev
/etc
/home
/lib
/mnt
/proc
/root
/sbin
/tmp
/usr
/var
```



4.Linux常用操作命令

- 关机重启
- 文件及目录操作
- 信息显示
- 备份压缩
- 系统管理

(1) 重启重启

- 图形界面操作:
- 命令操作:
 - 关机: `#init 0`
 - 重启: `# init 6`
 - 警告重启: `#shutdown -h +5`

(1) 文件目录操作

更改当前目录	cd	cd /etc cd work	当前目录设为/etc目录 当前目录设为./work
文件复制	cp	cp /etc/passwd ~/. cp passwd /tmp	将/etc目录下的文件passwd复制到当前目录 将当前目录下passwd文件复制到/tmp目录
文件改名 文件移动	mv	mv f1 f2 mkdir dir1 mv f2 dir1	文件f1更名为f2 创建目录dir1 将f2移动到目录dir1
显示文本 文件内容	cat less more	more passwd	显示passwd的内容
删除文件 或目录	rm	rm /tmp/passwd rm -rf dir1	删除目录/tmp下文件passwd 删除目录dir1

(2) 信息显示

显示用户信息	who whoami		
显示网络接口性能系	ifconfig	#ifconfig	显示网络接口ip地址、 MAC地址、路由表等
操作系统信息	uname	uname	
测试网络连通性	ping	ping 219.222.171.9 more /proc/cpuinfo	
显示CPU、内容、设 备、内核模块信息	Proc文 件系统	more /proc/meminfo more /proc/devices more /proc/modules	

(3) 系统管理命令

改变文权限	chmod	chmod +x test	给 test 文件添加执行权限
安装文件系统	mount	mount -t vfat /dev/hda1 /mnt/c mount /dev/sda1 /mnt/udisk	将 windows 的盘挂载到 /mnt/c 将 u 盘挂载到 /mnt/udisk
切换用户身份	su	su -	切换到根用户身份
显示进程信息	ps	ps -ef	显示系统所有进程信息
杀死进程	kill	kill -9 12345	杀死 id 号为 12345 的进程
软件包管理	rpm	rpm -ivh ddd.rpm	安装软件包 ddd

(3) 作业与练习

1. 用5分钟时间练习上面讲过的3类命令
2. 课后完成实验一的各项任务

5.Linux编程环境

(1) 基于文本模式的开发平台

- 编辑工具: **vi,emacs,gedit**
- 编译工具: **gcc**
 - 编译过程中调用的工具还有: **bintuils**工具链,**glibc**库等
- 调试工具
 - **gdb**
 - **ddd:gdb**的图形界面
- 项目管理工具: **make**

好处: 可看到程序编辑、编译、运行、调试的全过程
是嵌入式Linux应用开发的主要形式

(2) 集成开发平台eclipse+CDT

- eclipse是一个开源开发环境，CDT是eclipse支持C/C++开发的一组插件；
- CDT依赖于基于文本命令的开发工具：
gcc、gdb、make, 是文本开发工具的图形界面；
- CDT可对应用程序的源代码和可执行代码打包，就像Microsoft VC++;

6.常用开发工具的使用

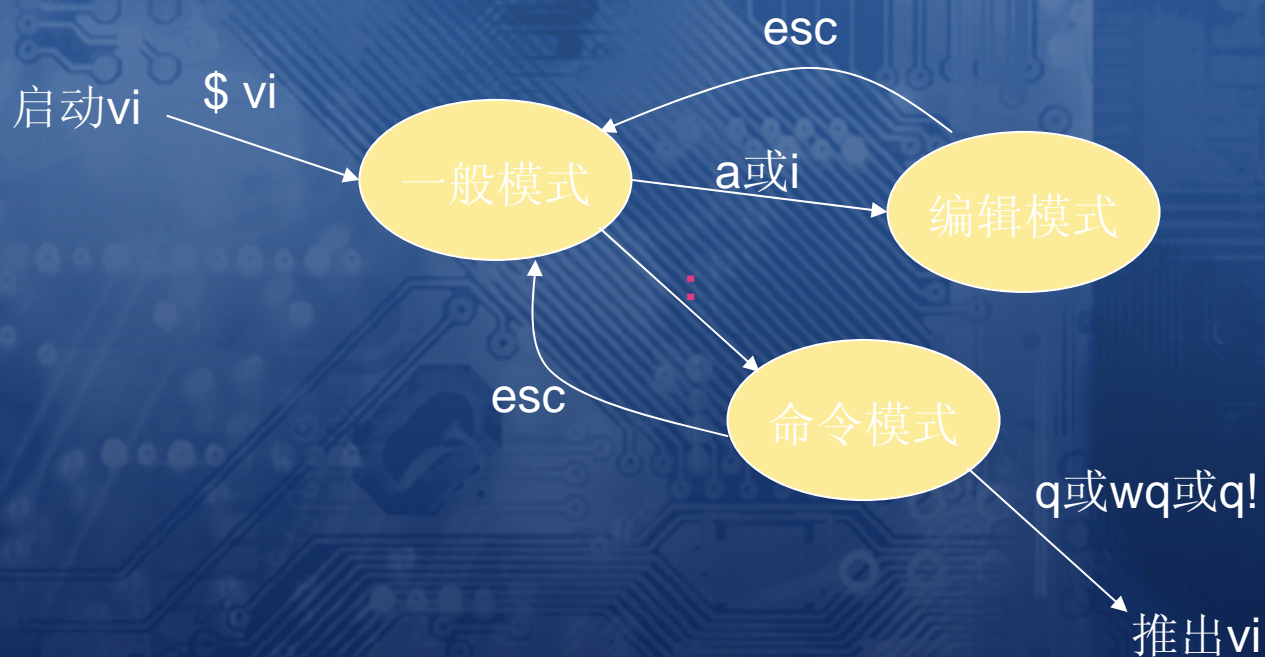
(1) 编辑工具gedit

- 图形化全屏幕源程序或文本编辑工具，只能在Linux图形界面下使用，使用方法像Windows下的记事本和写字板
- 使用方法：
 - 从命令行启动：\$ gedit f1.c
 - 从图形界面启动：开始菜单→编程工具→gedit

(2) 编辑工具Vi和vim

- 可在Linux命令界面下使用，也可在远程登录终端中使用，用来编辑源程序
- 启动命令实例：\$ vi 或\$ vi p1.c
- 工作模式模式：
 - 一般模式：进入vim所处的模式，发接受光标移动、串搜索和单字符命令（删除一个字符等）；
 - 编辑模式：在一般模式下按字母i或a将进入编辑模式，终端左下角显示编辑模式字样，该模式可输入文本信息；

● 工作模式转换



● 常用编辑命令

一般模式	i,a	进入编辑模式
	光标键	上、下、左、右
	:	进入命令模式
	x	删除当前字符
	/ 文本	搜索和定位文本
编辑模式	可视字符	输入字符
	esc	回到一般模式
	光标键	光标上、下、左、右移动
命令模式	esc	回到一般模式
	q	退出 vi
	wq	保存文件并退出 vi
	q!	存盘强行退出 vi
	s/被替换串/替换	文本替换
	123	光标移到 123 行

- 练习3分钟:用**vim**创建文件**hello.c**, 内容为:

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int a,b,c;
```

```
    a=3;b=4;c=a+b;
```

```
    printf("a+b=%d\n",c);
```

```
    exit(0);
```

```
}
```


(3) gcc编译工具集

- 构成:
 - **cpp,gcc,g++,libgcc,libstdc++,as,ld**
- 程序编译实例
 - **gcc -g -v -o test test.c**
 - **gcc -c -O -D__KERNEL__ -DMODULE -I/usr/share/src/linux/include -o testdrv.o testdrv.c**
- 编译选项:
 - **-c: compile**
 - **-O: optimization**
 - **-o: output**
 - **-D: define**
 - **-g: global**
 - **-v: verbal**, 显示编译过程中每一步用到的命令
- 练习: 编译运行程序**hello.c**

(4) 基于源程序的调试工具gdb

- 要求:

- 编译程序时使用-g选项
- 启动gdb: `$ gdb hello` 进入调试界面, `hello`等待运行

- 调试命令:

- `list`: 从上次调用`list`或源程序开始处列出10行源程序
- `break 10`: 在第10行处设置断点
- `run`: 运行程序直到断点或运行结束
- `cont`: 断点后继续运行程序
- `next`: 单步执行程序,函数调用一步执行完
- `step`: 单步执行程序,进入函数单步执行
- `watch i`: 在程序运行过程中监视变量*i*的变化

(5) 项目管理工具Make

- 考虑2个源程序:

f1.c:

```
#include <stdio.h>
int main(){
    int a,b,c;
    a=3;b=4;
    c=add(a.b);

    printf("a+b=%d\n"
);
```

f2.c:

```
int add(int x,int
y){
    int z;
    z=x+y;
    return z;
```

```
}
```


- 编译运行方法1:

编译f1.c: `$ gcc -c f1.c`

编译f2.c: `$ gcc -c f2.c`

链接f1.o和f2.o: `$ gcc -o test f1.o f2.o`

运行: `$/test`

- 编译运行方法2:

编译链接: `$ gcc -o test f1.c f2.c`

运行: `$/test`

- 问题:

1. 如果f1.c和f2.c自上次编译以来未曾修改, 则上述编译步骤可以不做, 当源程序文件数量很多时, 可以节省大量的编译时间, 如编译内核;
2. 如果一个可执行程序要通过编译很多源程序文件 (如1000个) 链

- 解决途径：采用**Make**管理源程序及编译链接过程。方法是在当前目录下创建一个**Makefile**或**makefile**文件；其中用规则列出各种可执行文件、目标代码文件、源程序文件间的依赖关系，**每条规则下面退格写出从被依赖文件产生依赖文件的命令**；

➤ **Makefile/makefile**文件内容：

```
test: f1.o f2.o
```

```
    gcc -o test f1.o f2.o
```

```
f1.o: f1.c
```

```
    gcc -c -o f1.o f1.c
```

```
f2.o: f2.c
```

```
    gcc -c f2.c
```

- 执行命令 **\$ make test**，**make**扫描和分析**Makefile/makefile**文件呢绒，**当依赖文件的最后修改时间早于被依赖文件最后修改时间时**，**将调用其命令重新生成依赖文件**，**保证依赖文件是用最新的被依赖文件生成的**，**否则不执行相应的命令**。

- 练习：编写**makefile**文件管理**f1.c**和**f2.c**的编译过程

- 在**Makefile**使用变量增强**Makefile**文件的通用性

- 如将编译程序命令名变成变量**CC**:

CC=gcc

all: f1.o f2.o

\$(CC) -o test f1.o f2.o

f1.o: f1.c

\$(CC) -c -o f1.o f1.c

f2.o: f2.c

\$(CC) -c f2.c

- 采用不同编译命令不需修改**Makefile**

- **make CC=arm-linux-gcc**

将用**arm-linux-gcc**编译该程序，使其可在**arm linux**平台运行。

- 采用内置变量

test: f1.o f2.o

\$(CC) -o \$@ \$^

f1.o: f1.c

\$(CC) -c -o f1.o f1.c

f2.o: f2.c

\$(CC) -c \$<

6.shell编程

(1) shell编程原理和用途

- 原理：将多条Linux命令放在一个文本文件（**shell**程序）中，成批自动执行；可以在**shell**程序中使用变量，增强**shell**命令功能的通用性；还可以加上控制结构，使其中的命令有条件执行或反复执行；
- 用途：不用高级语言编程，创建出效率更高、功能更强更灵活、满足应用实践需要的Linux命令；
- 实例：**init**命令
 - **init 5**

(2) Shell编程要点

- **shell程序规范**

- 文本文件，以`#!/bin/bash`作为第1行，表示将调用`/bin/bash`解释执行shell程序；
- 通常每行一条Linux命令或结构语句

- **Shell程序的执行**

- 添加了可执行权限：像执行普通Linux命令一样执行，如：`$ init 0`
- 无执行权限：`$ /bin/bash init 3`

(1) Shell编程要点

● shell 变量

- 定义及赋值：直接赋值，不需定义，自动识别数据类型，如 **count=5**
- 引用：变量名前加\$，如： **echo \$count**
- 系统变量（环境变量）：如**HOME**、**PATH**、**PS1**等
- 特殊变量：
 - **\$?**: 上一条命令执行的返回值
 - **\$#**: 命令行参数个数
 - **\$0**: 本**shell**程序的文件名
 - **\$\$**: **shell**进程**PID**

- 数值运算命令**expr**
 - **expr 2*\3+4\)**
- 关系运算符、逻辑运算符、引号作用
 - 看实例
- 运算符、条件表达式、**case**表达式、循环语句、函数
 - 看实例

(3) Shell程序分析阅读

```
#!/bin/sh
SAVEDIFS=$IFS
IFS=:
INPUT_FILE=names2.txt
NAME_HOLD="Peter James"
LINE_NO=0

if [ -s $INPUT_FILE ] then
    while read NAME DEPT ID
    do
        LINE_NO='expr $LINE_NO + 1'
        if [ "$LINE_NO" -le 2 ] then
            # 如果LINE_NO的值大于2, 则跳过
            continue
        fi
        if [ "$NAME" = "$NAME_HOLD" ] then
            # 如果NAME_HOLD的值为Peter James, 则跳过
            continue
        else
            echo "Now processing ...$NAME $DEPT $ID"
            # 所有的处理都到达此外
        fi
    done < $INPUT_FILE
    IFS=$SAVEDIFS
else
    echo "'basename $0':Sorry file not found or there is no data in the file">&2
    exit 1
fi
```

(4) For循环实例

```
for file
do
    tr a-z A-Z < $file >$file.caps
done
```


(5) 练习

- 执行实验2的shell编程

6. 编译运行嵌入式应用程序的过程

- 源程序: `hello.c`

```
#include <stdio.h>
int main(void)
{
    printf ("Hello world, Linux programming!\n");
    return 0;
}
```

- 目标: 假定有了开发环境和运行Linux内核的目标嵌入式系统设备, 如何在目标嵌入式系统设备已经运行的条件下, 如何使hello程序加载到其上运行
- 讲述内容
 - 开发环境构成

(1)嵌入式应用开发环境构成



- **开发主机：**安装了交叉开发工具链（面向目标平台的gcc、glibc、as、ld等）；
- **目标嵌入式系统：**拥有CPU和操作系统（EOS），运行嵌入式应用程序；
- **网络线：**嵌入式操作系统运行起来后可以用于联网；
- **串口线：**将嵌入式系统从启动开始的输出送往开发主机，通过超级终端显示,并以超级终端控制嵌入式系统的控制终端窗口；

(2)建立开发机与目标机连接

1. 用串口线将开发机的串口1与目标机的串口1相连，用交叉双绞线将目标机和开发机的以太网口相连；
2. 打开开发机的超级终端（开始→程序→附件→通讯→超级终端→**arm.ht**（速率设置为**115200,N-8-1**）；
3. 打开目标机电源，如果在超级终端看到信息显示，则连接成功；
4. 在超级终端等到其**Linux**系统启动完毕，出现命令提示符，此时超级终端就是嵌入式系统的命令输入和信息显示窗口。

(3) 源程序编译链接

- 在开发主机上采用面向arm体系结构的交叉编译器进行编译：

```
$ arm-linux-gcc hello.c -o hello
```


(4) 将可执行程序hello下载到目标嵌入式设备

方法1: 采用U盘复制

开发主机:

1. 将hello复制到Windows与虚拟机的共享文件夹/mnt/hgfs/,即d:\share

```
$ cp hello /mnt/hgfs/.
```

2. 将hello复制到U盘,将U盘卸下插到嵌

• 目标嵌入式设备:
嵌入式系统设备

3. 安装U盘文件系统

```
# mount /dev/sda1 /mnt/udisk
```

4. 将hello程序从U盘复制到/root目录下

```
# cpp hello /root
```


方法2：通过另一串口传送到嵌入式系统设备

开发主机：

1.将hello复制到Windows与虚拟机的共享文件夹/mnt/hgfs/,即d:\share

```
$ cp hello /mnt/hgfs/.
```

2. 将hello复制到U盘,将U盘卸下插到嵌入式系统设备

目标设备：

3. 在/root目录下执行串口接收文件命令

```
# rz
```

开发主机：

4.通过超级终端的菜单采用zmodem方式发送文件d:\share\hello

- **方法3：通过以太网传输**
 - 设置好目标机和开发主机**IP**地址，使其在同一**IP网段**内
 - 开发主机运行**tftp server** 或**ftp server**
 - 目标机执行**ftp**或**tftp**将**hello**下载到本地

(5)在目标机上运行hello

- 运行hello
./hello

(6)嵌入式应用程序调试

- 设计思路：采用远程调试的方法，将gdb的功能分成2部分：用户交互部分gdb_client和调试控制部分gdb_server，分别在开发机和目标机运行，二者通过TCP/IP网或串口通信；

1) 在开发板上启动

- 在远程调试的第一步就是要在开发板上启动gdbserver工具，可以采用两种不同的连接方式

主机ip地址及端口号

```
[/mnt/yaffs].gdbserver 192.168.0.47:1234 hello
```

```
Process hello created; pid = 105
```

- 网口

开发板串口号

```
[/mnt/yaffs].gdbserver /dev/tts/0 hello
```

```
Process hello created; pid = 112
```

```
Remote debugging using /dev/tts/0
```

- 串口

2) 在主机上启动arm-linux-gdb

- 在开发板上启动gdbserver以后，第二步就是在主机上运行arm-linux-gdb来启动gdb调试应用程序，同样针对于启动的连接方式不同选用不同的参数

```
[root@localhost gce-gdb]# armv4l-unknown-linux-gdb hello
GNU gdb 5.2.1
```

Copyright 2002 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "--host=i686-pc-linux-gnu --target=armv4l-unknown-linux"...

(gdb)

3) 启动远程调试功能

开发板ip地址及端口号

(gdb) **target remote 192.168.0.115:1234**

Remote debugging using 192.168.0.115:1234

0x40000d00 in ?? ()

(gdb) list

1 #include <stdio.h>

2

3 int main () {

4

5 int i;

6 for (i =0; i < 10; i++)

7 {

8 printf ("helloworld.....NOM %d\n", i);

9 }

10 return 0;

(gdb)

启用串口方式调试:

(gdb) target remote /dev/ttyS0

4) 设置断点

(gdb) **break main**

Breakpoint 1 at 0x20003a0: file hello.c, line 6.

(gdb) **break 8**

Breakpoint 2 at 0x20003bc: file hello.c, line 8.

(gdb) **c**

Continuing.

Breakpoint 1, main () at hello.c:6

6 for (i =0; i < 10; i++)

(gdb) **c**

Continuing.

Breakpoint 2, main () at hello.c:8

8 printf ("helloworld.....NOM %d\n", i);

(gdb)

第三讲 嵌入式Linux内核烧 写与启动

主要内容

1. 嵌入式Linux系统软件构成
2. 建立数据传输通道
3. Linux内核的编译
4. 嵌入式Linux内核的加载
5. 嵌入式Linux的启动过程

1.嵌入式Linux系统软件构成

Flash/RAM:Root Filesystem

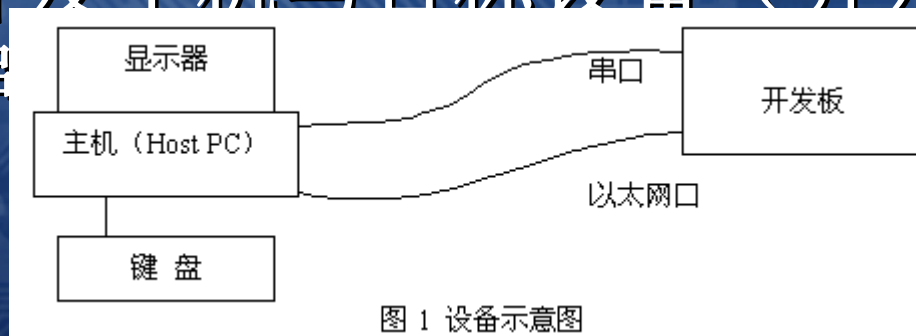
Flash/RAM:Linux Kernel

Flash: Bootloader vivi

2.烧写准备

(1) 内核烧写原理:

- 烧写的目的是把内核映像文件传到目标嵌入式系统设备的存储器中（**flash/RAM**);
- 假定开发主机与目标设备（开发板）已经通过串口传输文件;



(2) 建立数据传输通路

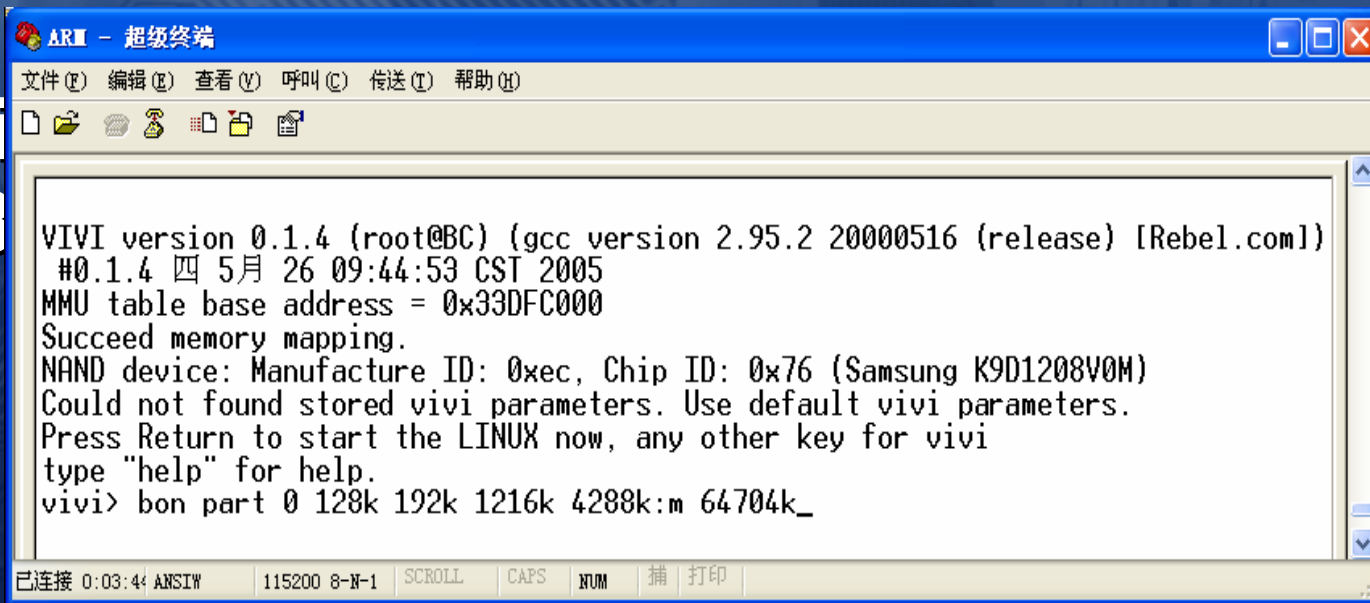
- 用串口线将开发机的**COM1**与**2410-S**开发板的**COM1**连接;
- 开发主机: 打开超级终端, 选择**COM1**, 波特率设置为: **115200bps**(即打开**arm.ht**)

3.使用串口下载烧写程序

(1) 格式化flash

- 打开超级终端，先按住PC 机键盘的Back Space 键，然后启动2410-S，进入vivi，按照以下命令格式化flash，重新分区，如图：

● vivi
428



```
ARM - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
VIVI version 0.1.4 (root@BC) (gcc version 2.95.2 20000516 (release) [Rebel.com])
#0.1.4 四 5月 26 09:44:53 CST 2005
MMU table base address = 0x33DFC000
Succeed memory mapping.
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
Could not found stored vivi parameters. Use default vivi parameters.
Press Return to start the LINUX now, any other key for vivi
type "help" for help.
vivi> bon part 0 128k 192k 1216k 4288k:m 64704k_
```

已连接 0:03:44 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

(2) 烧写vivi

这时已格式化flash，运行的是SDRAM 中的vivi.注意如果这时重启或断电会丢失所有数据，否则必须用Jtag 重新烧写vivi。

- vivi>load flash vivi x 回车

此时超级终端提示：

Ready for downloading using xmodem...

Waiting...

- 点击超级终端任务栏上“传送”下拉菜单中的“发送文件”，选择协议为Xmodem，选择镜像文件：vivi.bin，点击“发

(3) 烧写内核映像zImage

- **vivi>load flash kernel x** 回车

出现提示:

Ready for downloading using xmodem...

Waiting...

- 点击超级终端任务栏上“传送”下拉菜单中的“发送文件”，选择镜像文件**zImage**，协议为**Xmodem**，点击“发送”，如图1.5.5，4分钟左右**zImage**传输完毕，**zImage** (**d:\image\zImage**)先传输到**SDRAM**中，再把数据从**SDRAM**复制到**flash**里。请等

(4) 烧写根文件系统 (root)

- **vivi>load flash root x** 回车

Ready for downloading using xmodem...

Waiting...

- 点击超级终端任务栏上“传送”下拉菜单中的“发送文件”，选择镜像文件 **root.cramfs**，协议为**Xmodem**，点击“发送”，如图1.5.6,8 分钟左右**root.cramfs** 烧写完毕

4. 使用Jtag 烧写VIVI

1、连接线路

并口线连接到pc 机并口端，同时，并口线另一端与 Jtag 简易仿真器相连，Jtag 简易仿真器接开发板的14 针JTAG 口，启动2410-S。

2、软件准备

把整个GIVEIO 目录(在\img\flashvivi 目录下)拷贝到C:\WINDOWS 下，并把该目录下的giveio.sys 文件拷贝到C:\WINDOWS\system32\drivers 下。

3、添加驱动

在控制面板里，选添加硬件，下一步>→选择“是，我已经连接了此硬件”，下一步>→选择“添加新的硬件设备”（见图1.5.10），下一步

4、烧写vivi

- **vivi** 是由韩国**MIZI** 公司提供的一款针对**S3c2410** 芯片的**Bootloader**。烧写之前把要烧写的**vivi** 拷贝到**jf2410-s** 所在的**flashvivi (d:\image\flashvivi)**目录下，在开始→运行中输入**cmd**，进入命令行格式，进入**flashvivi** 目录，运行**jf2410-s** 命令，格式如下：**sjf2410-s /f:vivi**，如图**1.5.12**所示。
- 一切正常，**sjf2410-s** 会自动找到**CPU** 的**ID**。在此后出现的三次要求输入参数，第一次是选择**flash** 类型，输入**0**；第二次是

5、完成vivi 烧写

烧录vivi 后关闭2410-S，拔掉JTAG 简易仿真器，连接好串口线，准备烧写
vivi,kernel,root

第四讲 嵌入式Linux构建

主要内容

1. **Bootloader(vivi)**的源码分析与编译
2. **Linux**内核源代码结构与编译
3. 根文件系统的构建

1. Bootloader(vivi)的源码分析与编译

- 概述
- **bootloader** 的功能
- **vivi bootloader**的框架与编译
- **vivi**的启动流程
- **vivi**的命令使用
- 添加**vivi**命令
- **vivi bootloader**的下载

(1) bootloader 概述

- **bootloader / bootstrap**
 - 操作系统的引导加载程序
 - **Program that loads the “first program” (the kernel)**

(2)bootloader 的功能

- 启动设备
 - 系统储存、IO外设的工作模式以及中断的配置
- 装载、执行主要的软件组件：
 - 装载内核启动参数：设置内核的方式
 - 装载操作系统映像：Linux内核zImage映像
- 通过命令行，提供控制、检测功能：
 - 设备测试功能：
 - LCD、Flash等
 - 内存读写测试
 - 文件下载功能：
 - 串口下载（xmodem、zmodem 协议的支持）
 - 网络下载（tftp协议，基本的网卡驱动）

(3) vivi bootloader的框架与编译

➤ arch/

- s3c2410/
 - head.S smdk.c ...
- vivi.lfs

➤ Documentation/

➤ drivers/

- mtd/
- net/
- serial/

➤ include/

➤ init/

- main.c

➤ lib/

- boot_kernel.c boot_ucos.c string.c exec.c

(4) vivi的启动流程 (1)

- 硬件设备初始化
- 为加载 **bootloader**的 **stage2** 准备 **RAM** 空间
- 拷贝 **bootloader**的 **stage2** 到 **RAM** 空间中
- 设置好堆栈
- 跳转到 **stage2** 的 **C** 入口点

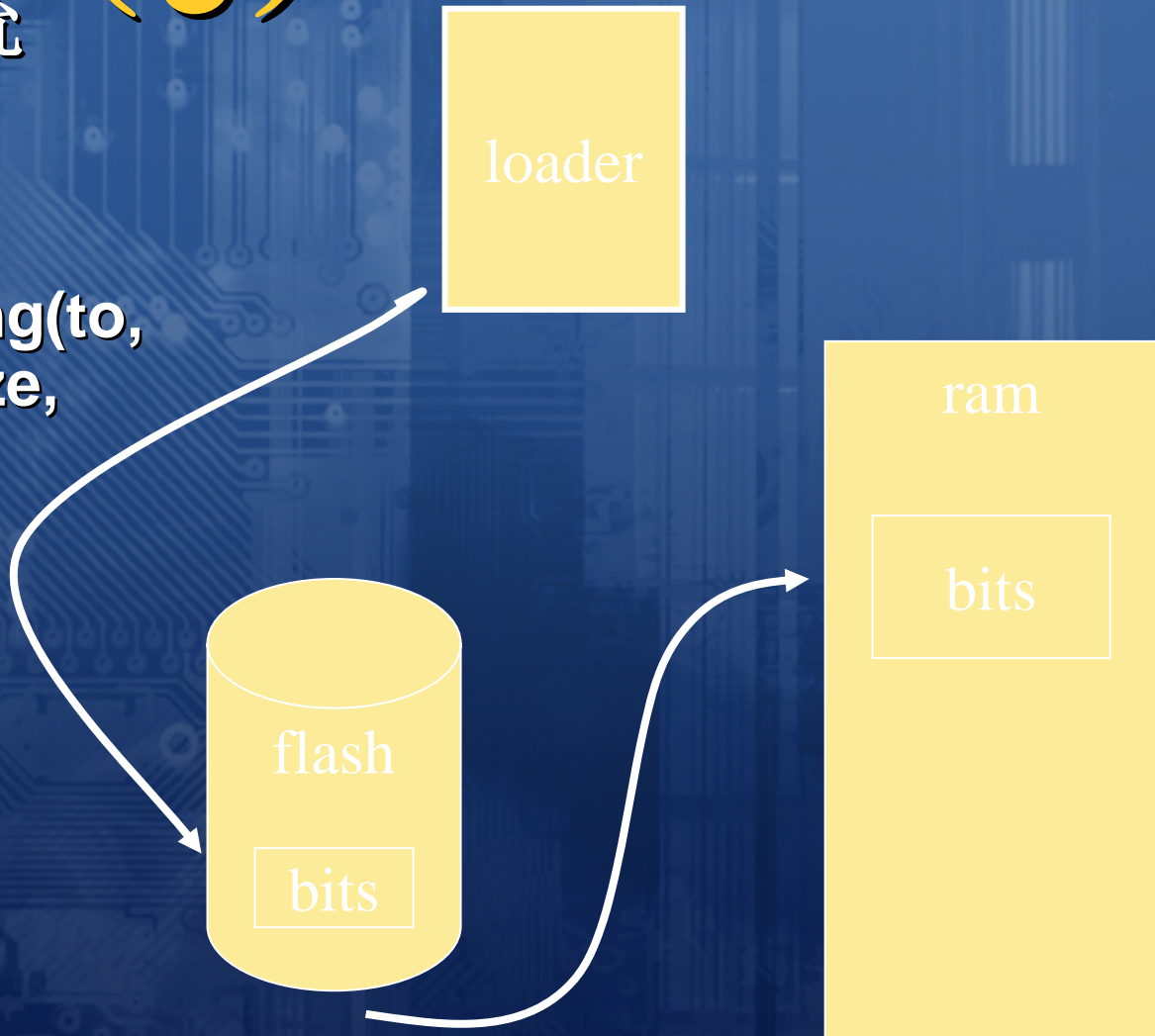
(4)vivi的启动流程 (2)

- **bootloader的 stage2 通常包括以下步骤 (以执行的先后顺序):**
 - 初始化本阶段要使用到的硬件设备
 - 检测系统内存映射(**memory map**)
 - 将 **kernel** 映像和根文件系统映像从 **flash** 上读到 **RAM** 空间中
 - 为内核设置启动参数
 - 调用内核或进入命令行

(4) vivi的启动流程—装载

主要组件 (3)

- 初始化文件系统
- 装载组件:
 - `boot_kernel.c:`
 - `copy_kernel_img(to, (char *)from, size, media_type);`
- 运行装载后文件:
 - `boot_kernel.c:`
 - `call_linux(0, mach_type, to);`



(5) vivi的命令使用

- **vivi> help**
 - **cpu [{cmds}]** -- Manage cpu clocks
 - **bon [{cmds}]** -- Manage the bon file system
 - **reset** -- Reset the system
 - **param [set|show|save|reset]** -- set/get parameter
 - **part [add|del|show|reset]** -- Manage MTD partitions
 - **mem [{cmds}]** -- Manage Memory
 - **load {...}** -- Load a file to RAM/Flash
 - **go <addr> <a0> <a1> <a2> <a3>** -- jump to <addr>
 - **dump <addr> <length>** -- Display (hex dump) a range of memory.
 - **call <addr> <a0> <a1> <a2> <a3>** -- jump_with_return to <addr>
 - **boot [{cmds}]** -- Booting linux kernel
 - **bootucos [{cmds}]** -- Booting ucos system

(6)添加vivi的命令

- 命令初始化流程:
- **main.c**
 - **init_builtin_cmds**
 - **add_command**
- 添加命令的流程示例:
 - **cp -af ./arch/s3c2410/smdk2410_test.c ./lib/my_test.c**
 - 编辑 **./lib/Config_cmd.in**
 - **bool 'built-in my test command' CONFIG_MY_TEST**
 - 编辑 **./lib/Makefile**
 - **obj-\$(CONFIG_TEST) += my_test.o**
 - 编辑**my_test.c**文件
 - **init_builtin_cmds**中添加命令:

(7) vivi bootloader的下载

1、load flash ucos x

(注意: 不是load flash ucos)

2、使用x-modem协议下载

3、bootucos

4、help命令:

mytest [{cmds}]
functions

-- Test

5、mytest命令:

vivi> mytest

Usage:

test sleep

-- Test sleep mode.

(7) vivi的编译

- # make menuconfig
- # make

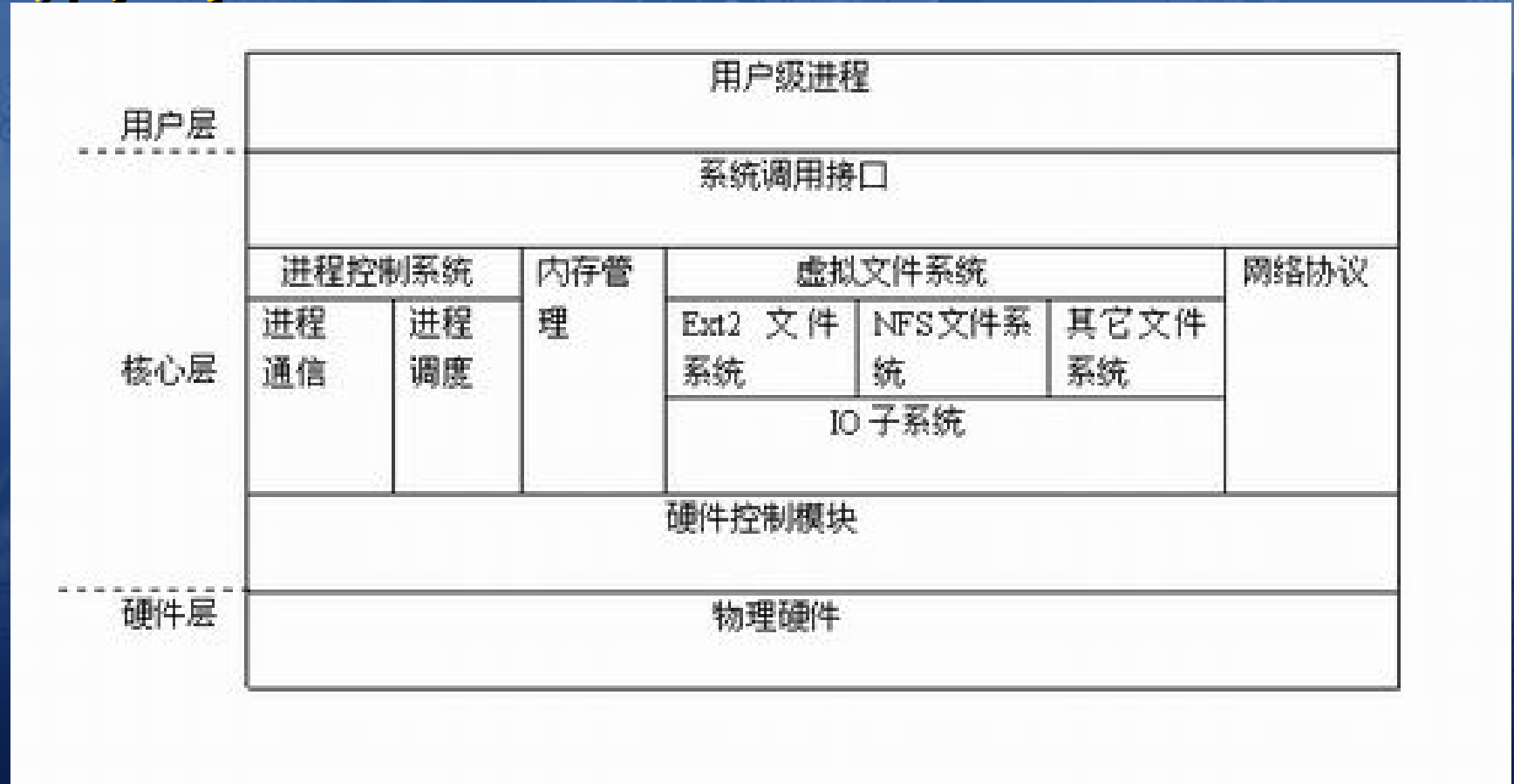
•,

(8)vivi功能源代码分析实例

- 启动过程：
 - 初始化、将信息写到串口、网络功能启动、加载OS
- 格式化flash，各种映像的存放位置
- 参考 《 Bootloader(Vivi)源代码分析 》

2.Linux内核源代码结构与

(1) Linux操作系统层次结构

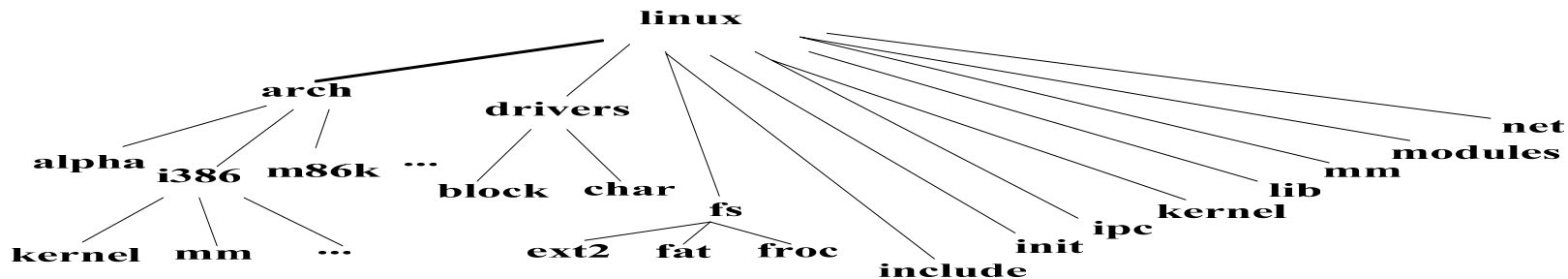


(2) Linux内核的功能

- **Linux**内核实现进程管理、内存管理、文件系统、设备驱动和网络系统等功能；
- **Linux Kernel**由五个主要的子系统组成：进程调度（**SCHED**）、内部进程通讯（**IPC**）、内存管理（**MM**）、虚拟文件系统（**VFS**）、网络接口（**NET**）

(3) Linux内核源代码结构

Linux内核源程序安装在/usr/src/linux下:



Linux内核目录结构

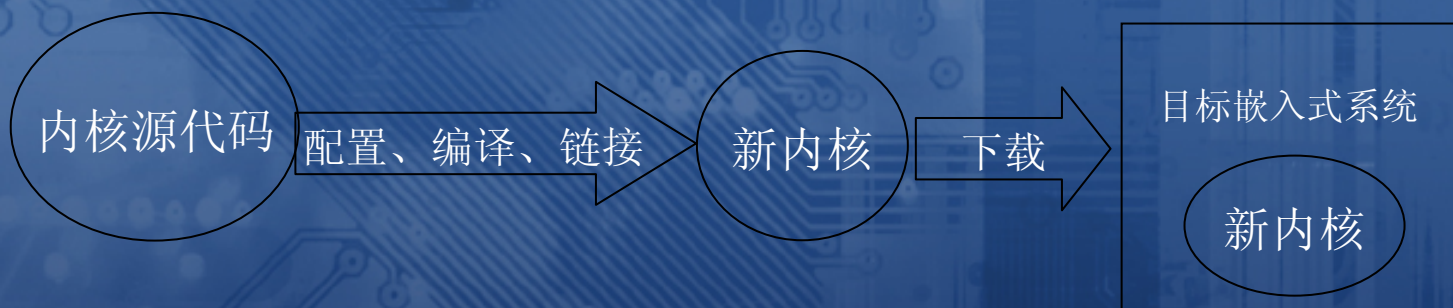
- **arch**: 该子目录包括了所有和体系结构相关的内核代码。它的每一个子目录都代表一种支持的体系结构，例如**i386**就是关于**intel cpu**及与之相兼容体系结构的子目录。PC机一般都基于此目录。
- **Include**: 该子包括编译内核所需要的大部分头文件。与平台无关的头文件在**include/linux**子目录下，与**intel cpu**相关的头文件在**include/asm-i386**子目录下，而**include/scsi**目录则是有关**scsi**设备的头文件目录。
- **init**: 该子目录包含内核的初始化代码，

Linux内核目录结构

- **kernel**: 主要的核心代码, 此目录下的文件实现了大多数linux系统的内核函数, 其中最重要的文件当属**sched.c**; 同样, 和体系结构相关的代码在**arch/*/kernel**中。
- **drivers**: 放置系统所有的设备驱动程序; 每种驱动程序又各占用一个子目录: 如, **/block**下为块设备驱动程序, 比如**ide (ide.c)**。设备初始化程序在**drivers/block/genhd.c**中的**device_setup()**。
- **lib**: 放置核心的库代码。
- **net**: 核心与网络相关的代码。
- **ipc**: 这个目录包含核心的进程间通讯的代码。
- **fs**: 所有的文件系统代码和各种类型的文件操作代码, 它的每一个子目录支持一个文件系

(4) Linux内核的重新编译

内核编译过程



1) 为PC Linux重新编译内核

- 下载内核源代码

```
# mv linux-2.4.18.tar.gz /root/src/. //把内核代码文件移到相应的目录
```

```
# cd /root/src
```

```
# tar zxvf linux-2.4.18.tar.gz //解压
```

```
(bzip2 -dc linux-2.6.14.tar.bz2 | tar xvf -)
```

- 配置内核

```
# cd linux
```

```
# make mrproper //清除目录下配置文件和中间文件
```

```
# make xconfig //配置内核
```


重新编译Linux内核

- 配置启动文件

```
# cp /usr/src/linux/arch/i386/boot/zImage  
/boot/zImage-2.4.18
```

```
# cp /usr/src/linux/System.map  
/boot/System.map-2.4.18
```

```
# ln -sf /boot/System.map-2.4.18  
/boot/System.map /* 内核符号表 */
```

- 使用**lilo**启动Linux:

- 编辑/etc/lilo.conf文件
- 执行lilo

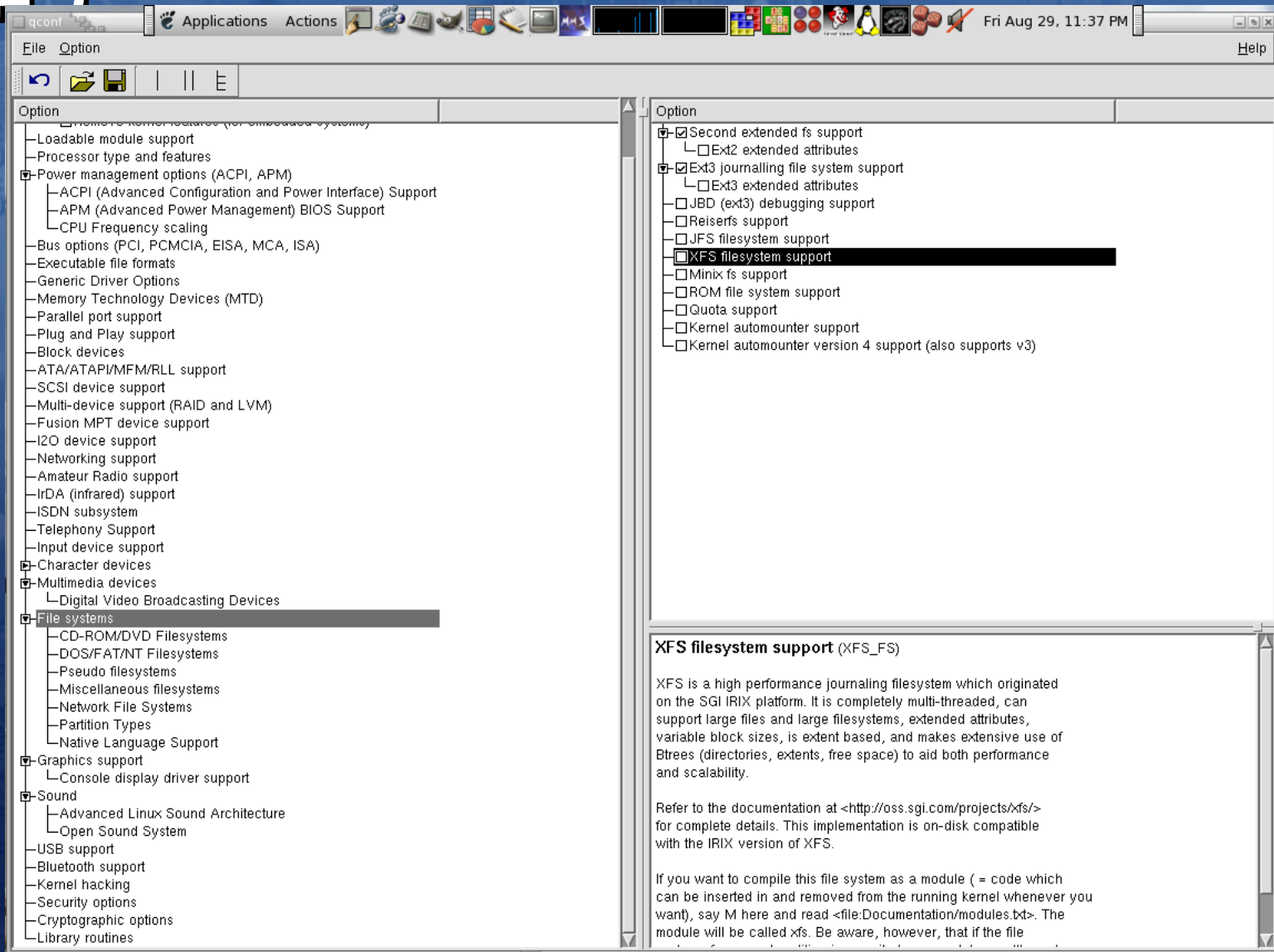
2) 重新编译ARM Linux内

核

Makefile:

- 顶层 Makefile 读取 `.config` 中的配置选项 `CROSS_COMPILE` `?= /usr/local/arm-linux/bin/arm-linux-` 指定编译内核所使用的编译器
- `include arch/(ARCH)/Makefile`, 包含了特定 CPU 体系结构下的 Makefile, 指定了平台相关的信息。
- 各个子目录下的 Makefile: 比如 `drivers/Makefile`, 负责所在子目录下源代码的编译管理。
- `.config`: 内核配置文件, 包含由用户选择的

a. 基于 Xwindows 图形界面的 #make xconfig



b.Linux内核的编译

```
# tar jxvf uptech-root.tar.bz2 -----解压缩内核源程序
# make xconfig      ----配置编译选项
# make dep          ----生成变量依赖关系信息.
# make clean        ----删除生成的模块和目标文件.
# make zImage       ----编译内核生成映象.
# make modules      ----编译模块.
# make modules_install ----安装编译完成的模块.
```

压缩内核影像所在路径:

arch/arm/boot/zImage

c. Linux内核文件

- vmlinuz与vmlinux
- vmlinuz是可引导的、压缩的、可执行的内核。
- 老的zImage解压缩内核到低端内存（第一个640K），bzImage解压缩内核到高端内存（1M以上）。如果内核比较小，那么可以采用zImage或bzImage之一，两种方式引导的系统运行时是相同的。大的内核采用bzImage，不能采用zImage。
- vmlinux是未压缩的内核，vmlinuz是vmlinux的压缩文件。

d.内核映像的烧写

- 两种烧写方式:
 - 1、使用vivi中提供的xmodem协议下载
 - 2、在开发板的linux系统启动后，使用imagewrite工具：
`imagewrite /dev/mtd/0 zImage:192k`

3)如何在内核配置中添加一个编译模块 (1)

- 1、在内核的驱动目录下编写驱动模块代码
本例中kernel-2410s/drivers/char/demo.o
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量
如: **CONFIG_S3C2410_DEMO**
该变量可以设置为三种状态:
Y---将该功能模块编译进内核
N ---不将该功能模块编译进内核
M ---将该功能编译成模块的方式, 可以在需要时动态插入到内核中的模块

如何在内核配置中添加一个编译模块 (2)

- 3、在**Makefile**中将编译选项与具体要编译的代码相关联，根据编译选项变量的状态决定编译
Object file lists.
obj-y :=
obj-m :=
obj-n :=
obj-\$(CONFIG_S3C2410_DEMO) += demo.o
- 4、使用**make menuconfig**对该功能模块进行配置，设置完成后，会生成更新的**.config**文件
- 拷贝**demo.o**，运行测试程序**test_demo**

3.建立根文件系统

- 根文件系统内容

uptech-root.tar.bz2

- 根文件系统root.cramfs的建立
tar jxvf uptech-root.tar.bz2

```
#!/bin/sh
```

```
if [ -z "$1" ]; then \
```

```
    mkcramfs root ../img/root.cramfs
```

```
else \
```

```
    mkcramfs $1 ../img/root.cramfs
```


第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

第五讲 开发环境若干难点问题

主要内容

1. 虚拟机与Windows共享方法
2. **zImage** 位置
3. 可装载模块放在哪里？ 如何在内核增加一个模块
4. 如何调通虚拟机网络
5. 如何使用**gdbserver**
6. 检测**Lib.a**的内容

1.虚拟机与Windows共享方法

- 1. `cd /root/`
- 2. 运行`rpm -ihv VMwareTools-5.5.2-29772.i386.rpm`
- 3. 运行`/usr/bin/vmware-config-tools.pl`
按提示，选择默认操作。
- 4. `ls /mnt/hgfs/` 存在此目录，表明安装成功。

2. zImage 目标位置

- 在子目录Makefile文件中
- 可用`grep zImage * -r` 查找

3. 如何在内核增加一个模块

- 1、将设备驱动程序的源代码demo.c从exp目录复制到kernel-2410s/drivers/char
- 2、在该级目录下的Config.in中添加对该模块的编译条件变量

```
dep_tristate 'S3C2410 DEMO' CONFIG_S3C2410_DEMO  
$CONFIG_ARCH_S3C2410
```

- 3、在该级目录的Makefile文件中增加一行：

```
$(CONFIG_S3C2410_ADC) += s3c2410-  
adc.o
```

- 4、重新配置内核：S3C2410_ADC的方式选择M（内核模块）

- 5 重新编译加载内核

- 可装载模块应该在**zImage**中

4.如何调通虚拟机网络

方法一、Linux虚拟机作为主操作系统
Windows XP的内部网节点上网

1、windows网卡（本地连接）属性→高级→Internet连接共享→运行其他网络用户通过此计算机的Internet连接来连接→选择家庭网络连接：VMWare Network Adapter VMNet1

配置结果：VMNet1的网络接口的IP地址被设置为192.168.0.1，将作为虚拟Linux的网关

3、VMWare虚拟机：设置→以太网→自定义：指定虚拟网络→VMNet1（仅主

方法二、**Linux**虚拟机作为与**Windows XP**连在同一交换机上的节点上网，其**IP**地址在同一网段

- 1、**Windows**网络配置：本地连接（网卡）属性→高级→取消**Internet**共享
- 2、**VMWare**虚拟机设置：以太网→网络连接→桥接：直接连接到物理网络
- 3、**Linux**虚拟机网络设置：**IP**地址与主机**IP**地址在同一网段，网关和**DNS**与主机配置相同
- 4、**Windows**主机：关掉防火墙或关掉防火墙中的某些过滤规则。

配置结果：将**Windows XP**当做交换机，**Linux**虚拟

5.gdbserver的使用

嵌入式操作系统

CAN总线通信及 Linux设备驱动程序

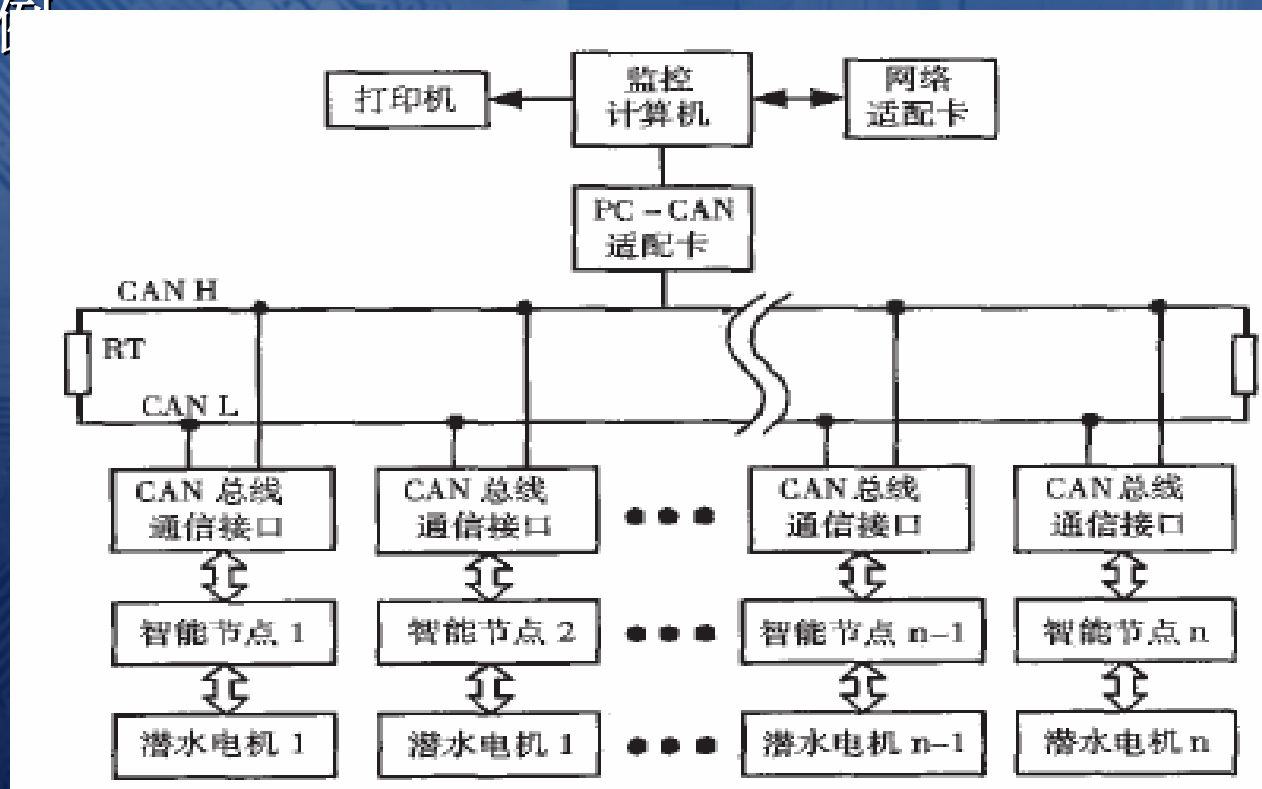
提纲

- CAN总线工作原理
- CAN总线通信协议
- 接口寄存器
- 控制功能
- 设备驱动程序
- 测试过程

一.CAN总线工作原理

(1) CAN总线概念

- 控制器局域网**CAN(Controller Area Net)**是一种**现场总线**，主要用于各种过程检测及控制。由于其卓越性能现已广泛应用于工业自动化、多种控制设备、交通工具、医疗仪器以及建筑、环境控制等众多部门
- 总线结构实例



(2)实验内容

- 学习CAN总线基本工作原理
- 分析CAN总线设备驱动程序，掌握总线控制的方法，理解其驱动程序的编写方法
- 运行测试程序，调用CAN总线设备驱动程序，研制驱动程序的正确性

UP-NETARM2410-S

教学科研系统

The diagram illustrates the UP-NETARM2410-S teaching and research system, centered around a main board with various components and optional modules.

Main Board Components:

- 电源插口 (Power Jack)
- 电源开关 (Power Switch)
- 核心模块 (Core Module)
- 复位按钮 (Reset Button)
- 从USB (Slave USB)
- 四主USB口 (Four Master USB Ports)
- JTAG
- 8" TFT LCD
- 音频插口 音量调节 (Audio Jack Volume Control)
- MIC
- PS/2
- 数码管 (7-segment Display)
- 小键盘 (Numpad)
- AD电位器 (AD Potentiometer)
- 直流电机 步进电机 (DC Motor Stepper Motor)
- IC卡 (IC Card)
- CF卡 (CompactFlash Card)
- IDE硬盘 (IDE Hard Drive)
- PCMCIA接口 (PCMCIA Interface)
- SD卡 (SD Card)
- IrDA红外 (IrDA Infrared)
- CAN Bus
- 双DA (Dual DA Converter)
- 双RS232口 (Dual RS232 Ports)
- 168Pin 扩展插槽 (168Pin Expansion Slot)
- RS485
- 双以太网口 (Dual Ethernet Ports)

Optional Modules:

- (可选) GPS+GPRS模块 (Optional GPS+GPRS Module)
- (可选) FPGA模块 (Optional FPGA Module)
- (可选) CAN结点模块 (Optional CAN Node Module)
- (可选) 蓝牙模块 (Optional Bluetooth Module)
- (可选) 摄像头模块 (Optional Camera Module)
- (可选) 无线上网卡 (Optional Wireless Internet Card)
- (可选) 无线网卡 (Optional Wireless Network Card)

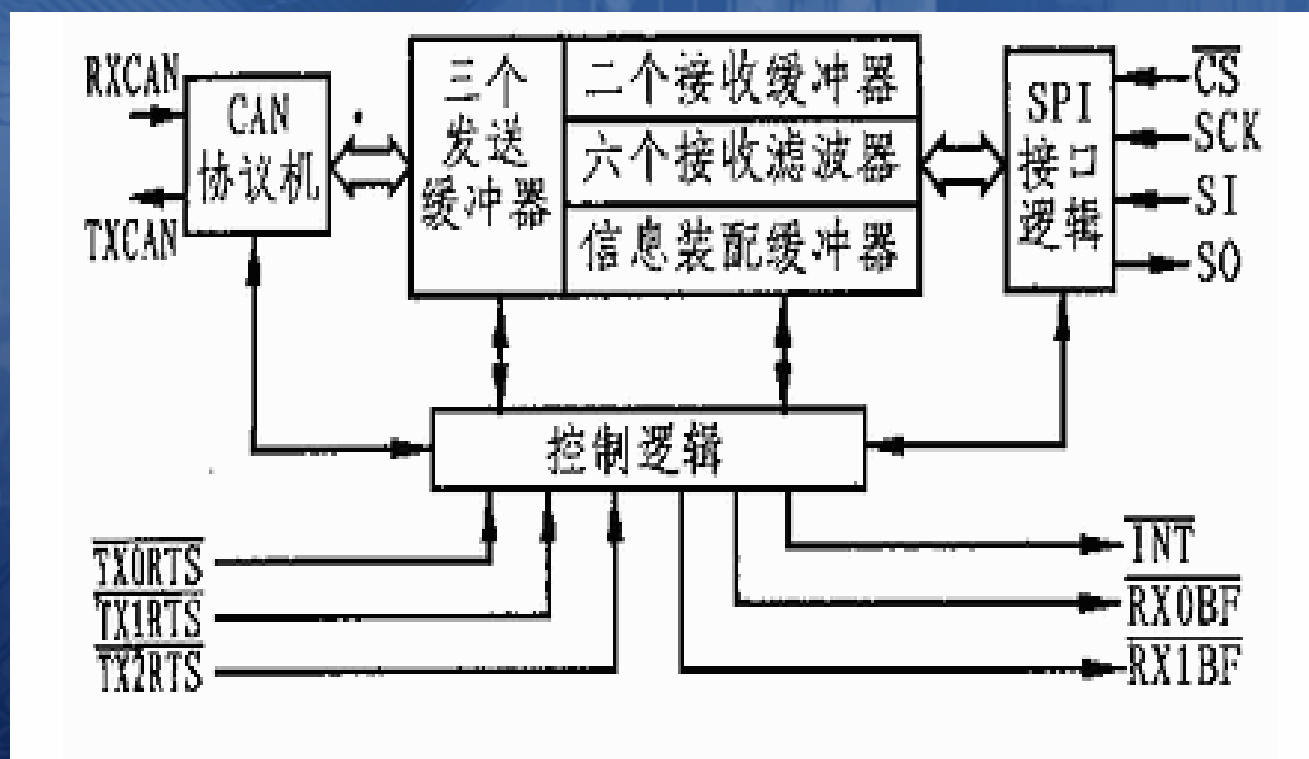
Peripherals:

- 鼠标 键盘 (Mouse Keyboard)

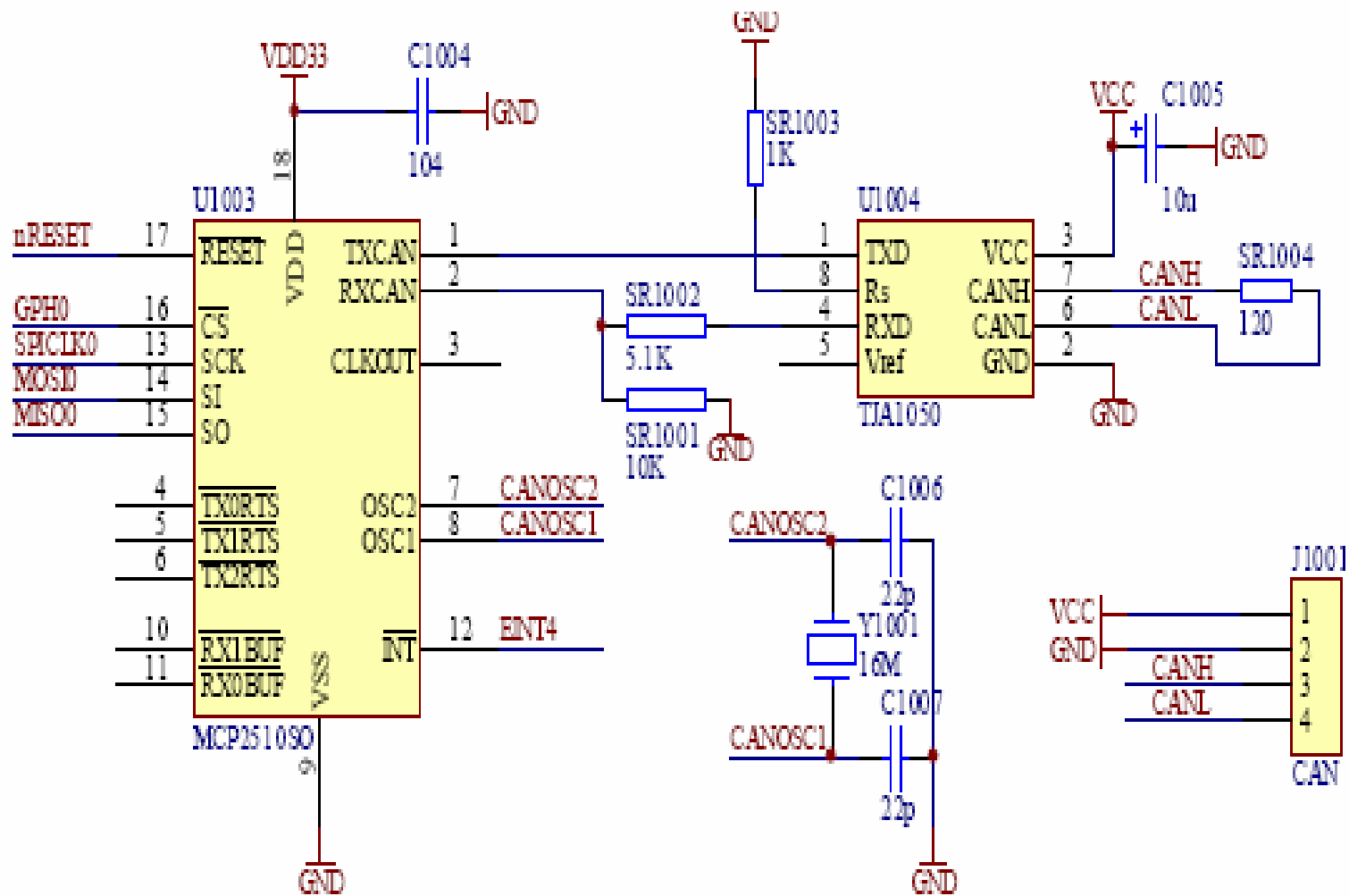
Logo and Contact Information:

博创科技
www.UP-TECH.COM

(4) CAN总线接口逻辑结构



(5) CAN总线接口硬件电路图



(6)控制软件设计要解决的问题

- 硬件上如何连接**CAN**总线硬件逻辑
- 如何对**CAN**总线进行编程控制
- **CAN**总线协议协议
- 如何编写设备驱动程序
- 如何测试确认电机控制驱动程序的正确性

二.CAN总线通信协议

1. 信号表示

采用差分电压信号

- 隐性: $CAN_H = CAN_L = 2.5V$, 表示信号1,
- 显性: CAN_H 比 CAN_L 高, $CAN=3.5V$,
 $CAN_L=1.5V$, 表示信号0.

2. CAN总线位时间

CAN 总线的**一个位时间**可以分成四个部分：同步段，传播段，相位段 1 和相位段 2，每段的**时间份额**的数目都是可以通过 CAN 总线控制器（比如 MCP2510）编程控制的，而**时间份额**的大小 t_q 由系统时钟 t_{sys} 和波特率预分频值 BRP 决定： $t_q = BRP / t_{sys}$ 。如图 2.6.1 所示：

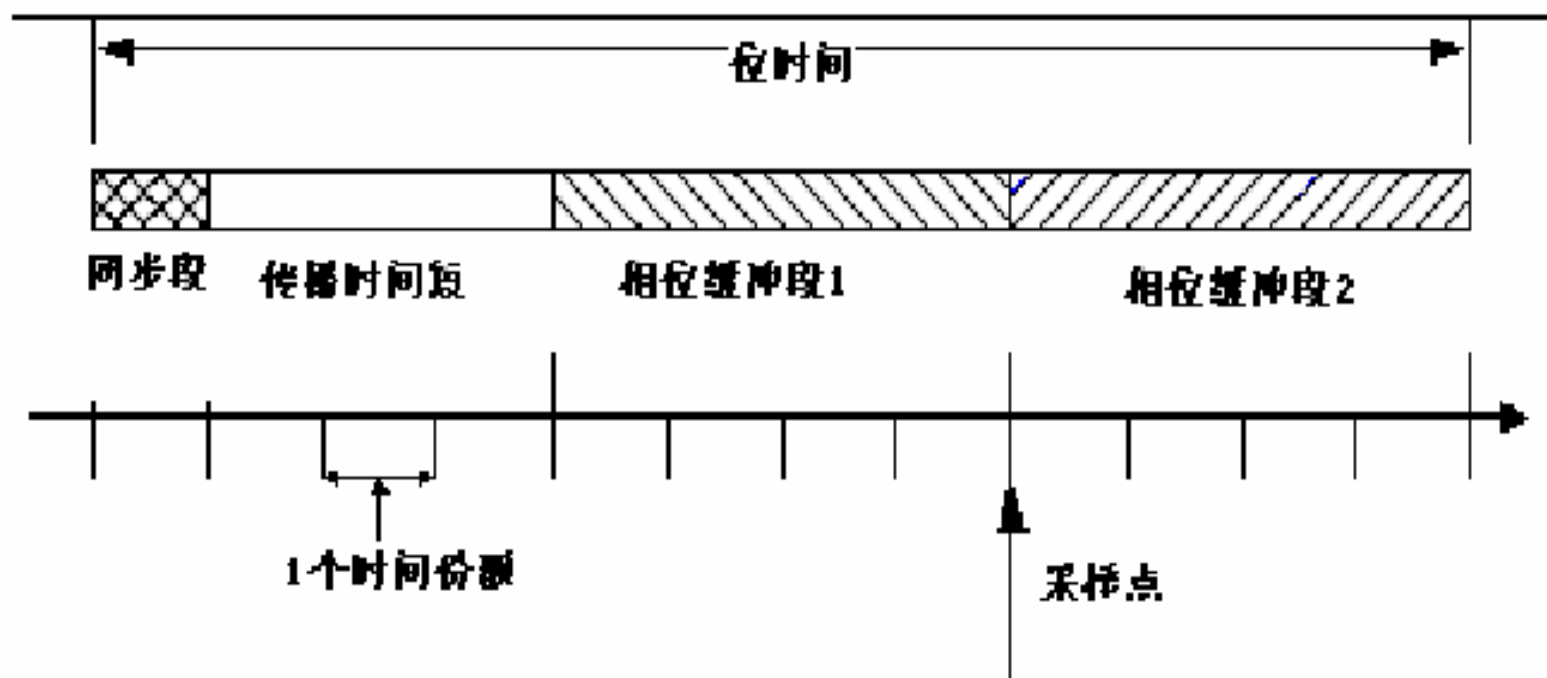
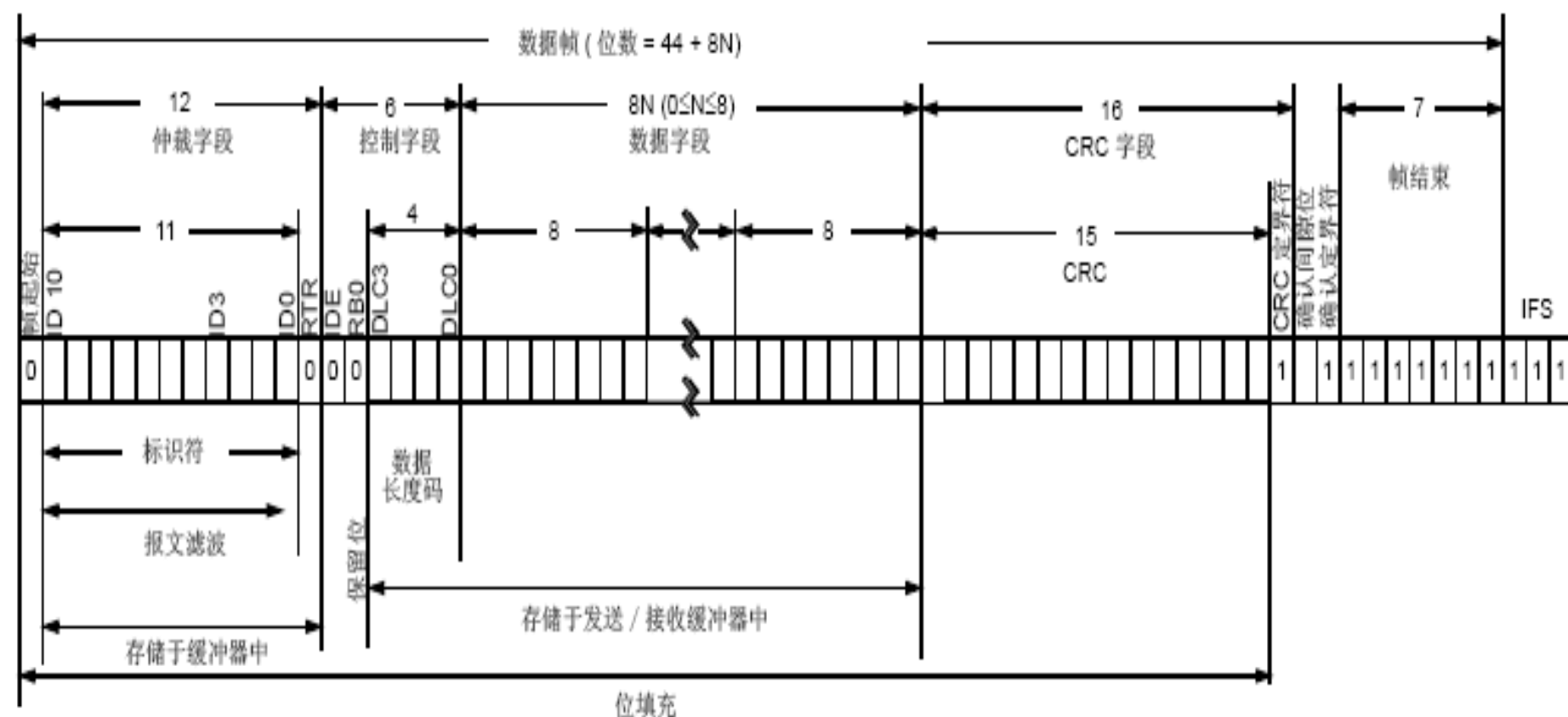


图 2.6.1 CAN 总线的**一个位时间**

3.CAN总线MAC帧



图 2.6.2 CAN 总线的帧数据



● 报文帧

- 标准数据帧
- 扩展数据帧
- 远程帧
- 错误帧
- 过载帧

三.MCP2510控制字

表 2.6.1 MCP2510 中的命令

命令	格式	定义
复位	1100 0000	设置内部寄存器为默认值，并设置 MCP2510 到配置状态
读取	0000 0011	从选定的寄存器的地址开始读取数据
写入	0000 0010	向选定的寄存器的地址开始写入数据
发送请求	1000 0nnn	设置一个或者多个发送请求位，发送缓冲区中的数据
读取状态	1010 0000	轮流检测发送或者接收的状态
修改位	0000 0101	按位修改寄存器

4.总线控制方式

- 主从关系

1 读取命令

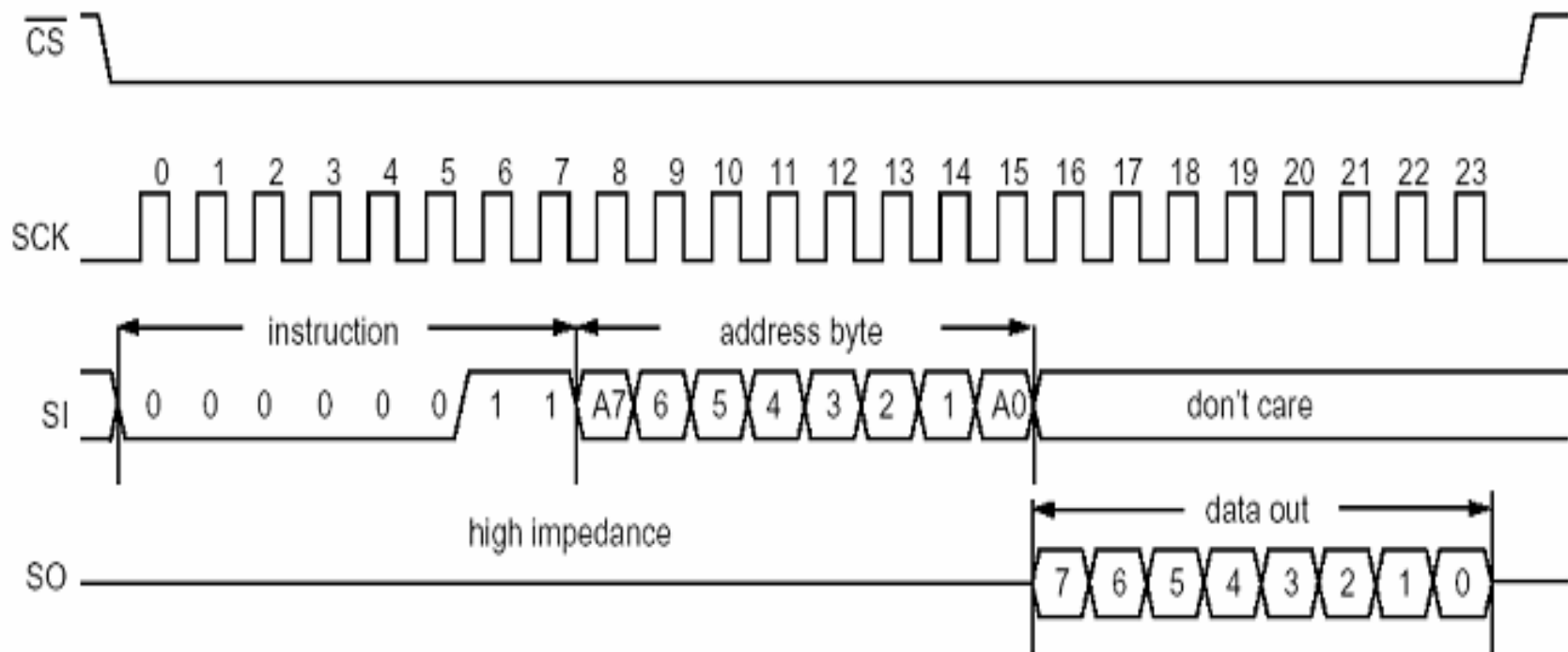


图 2.6.4 读取命令

2 字节写入命令

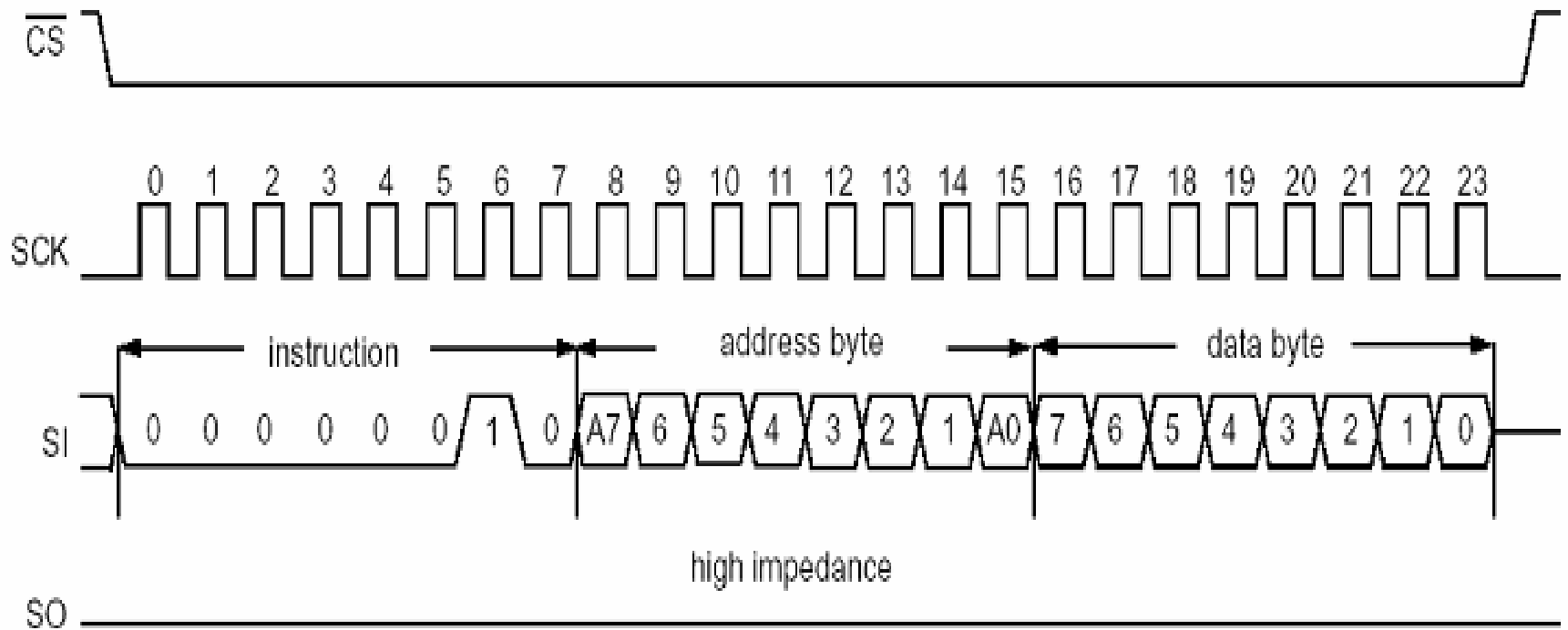


图 2.6.5 单字节写入命令

3 发送请求命令

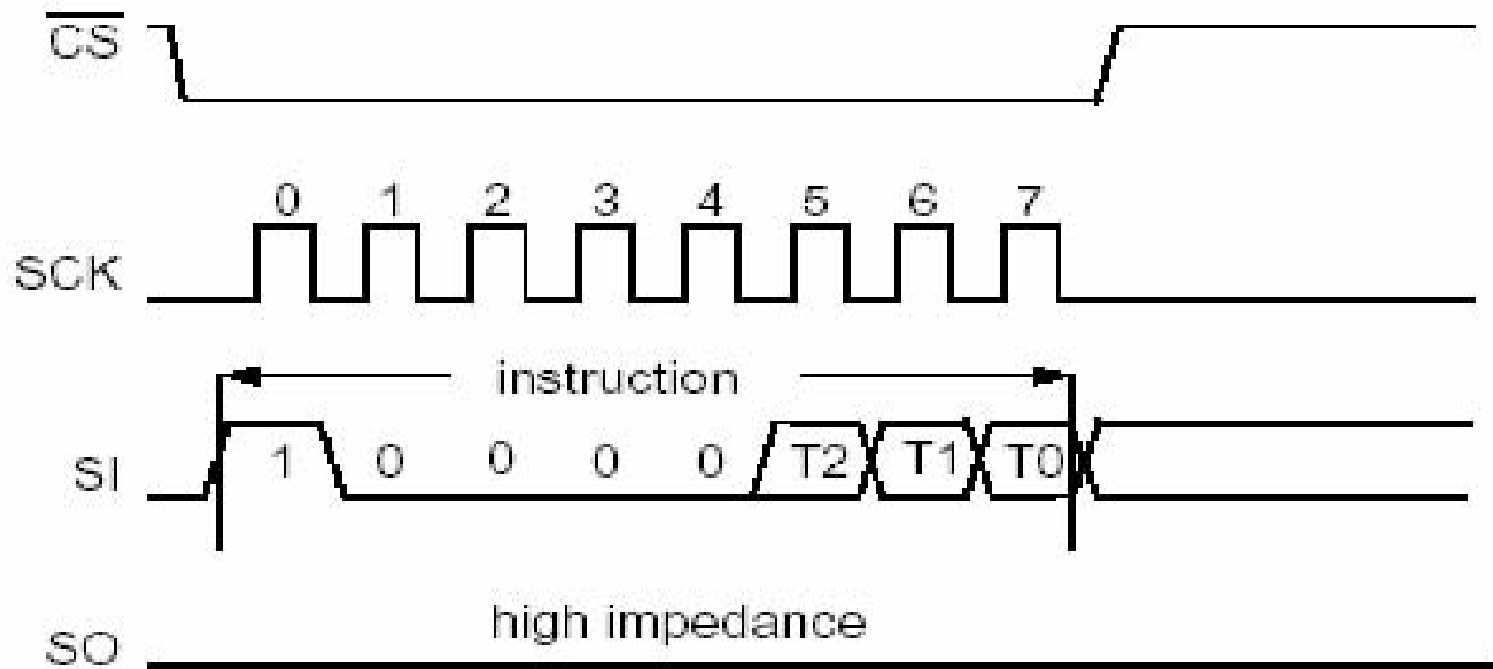


图 2.6.6 发送请求命令

4 修改位命令

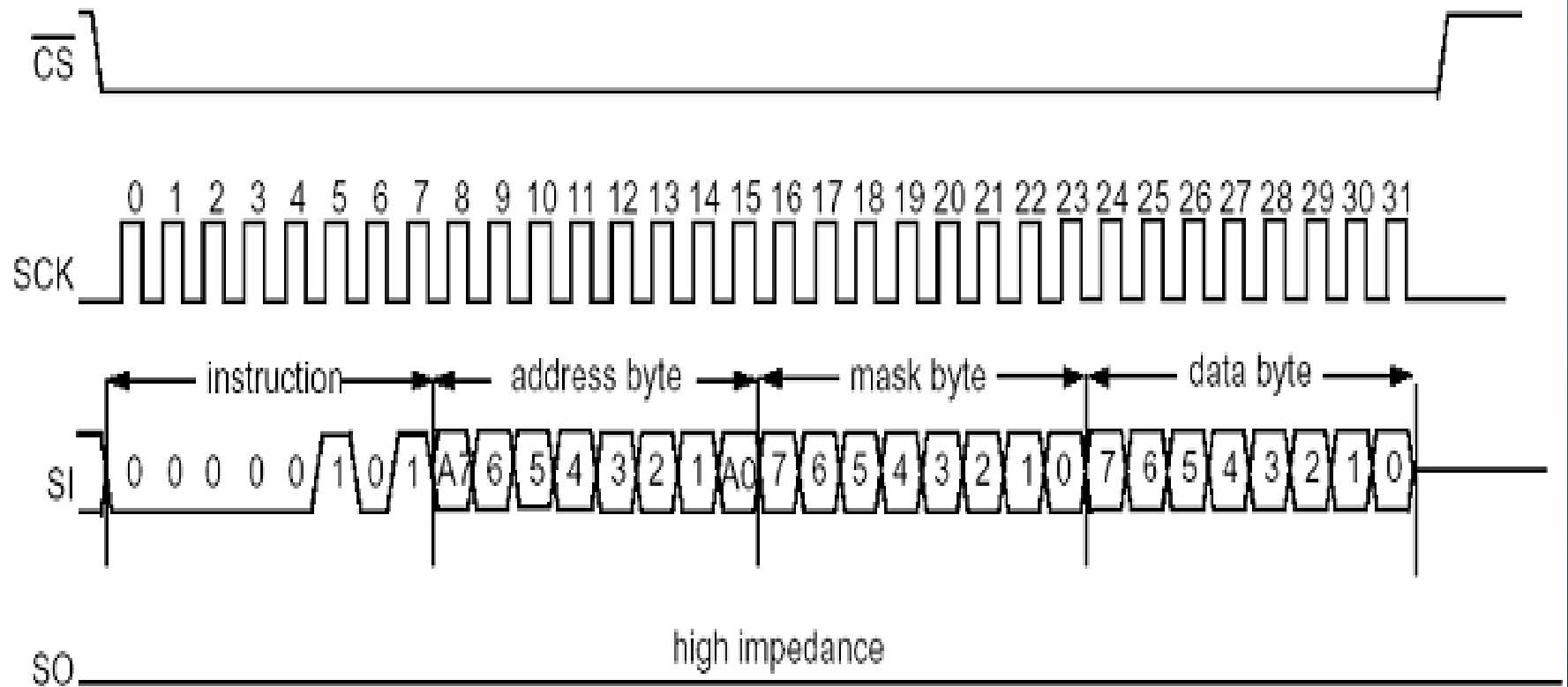


图 2.6.7 修改位命令

5 状态读取命令

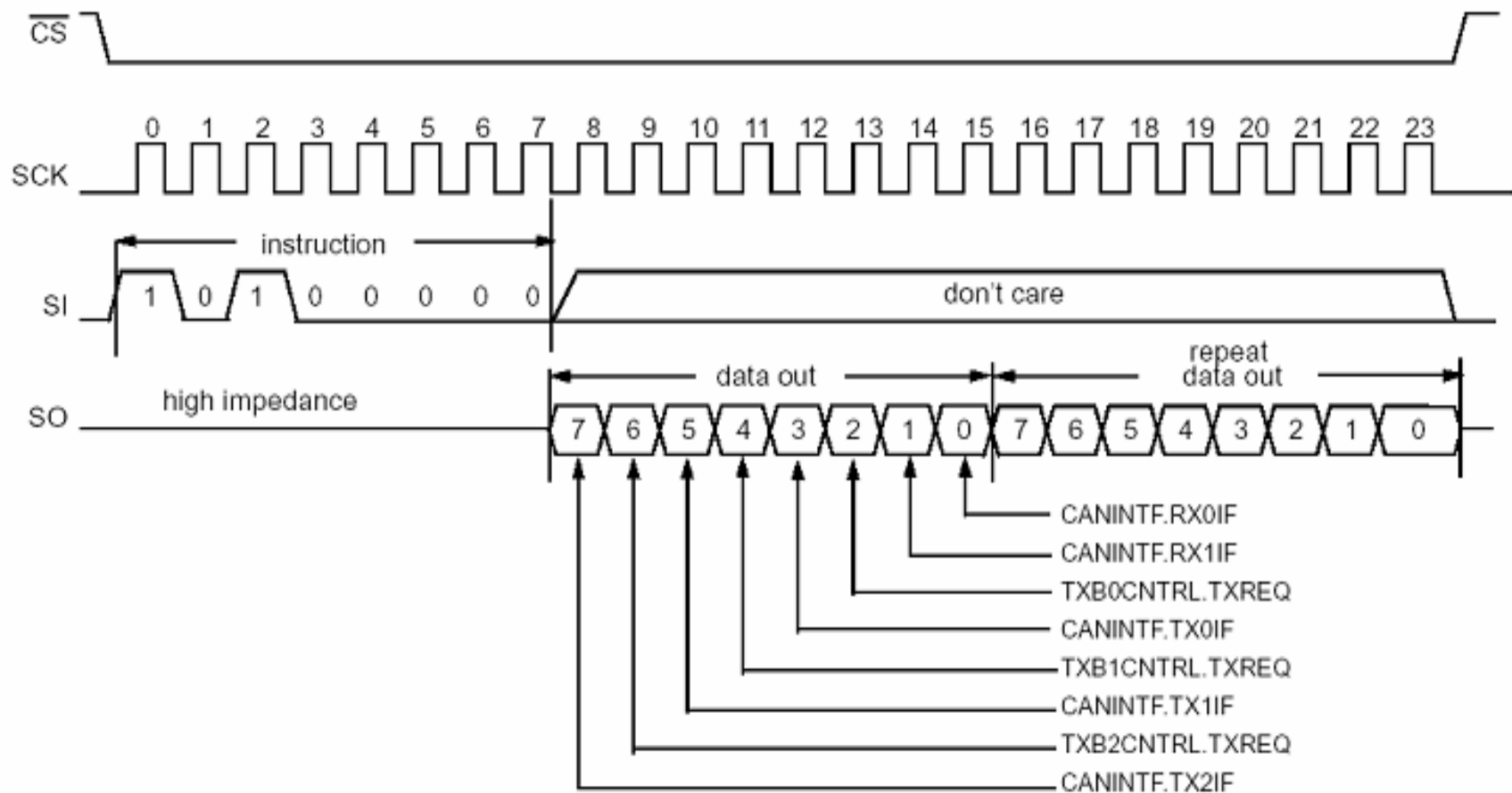


图 2.6.8 状态读取命令

5.波特率设置

假定MCP2510的输入时钟为16 MHz

表 2.6.2 MCP2510 的波特率设置

CAN 波特率	同步段	传输段	相位 1	相位 2	CNF1	CNF2	CNF3
125Kpbs	1	7	4	4	0x03	0x9E	0x03
250Kpbs	1	7	4	4	0x01	0x9E	0x03
500Kpbs	1	7	4	4	0x00	0x9E	0x03
1Mbps	1	3	2	2	0x00	0x9E	0x03

6.接收过滤器设置

表 2.6.3 Mask 和 Filter 的控制规律

Mask	Filter	发送方的 ID	是否接收数据
0	x	x	是
1	0	0	是
1	0	1	否
1	1	0	否
1	1	1	是

7.MCP2510初始化

MCP2510 的初始化如下步骤：

1. 软件复位，进入配置模式
2. 设置 CAN 总线波特率
3. 关闭中断
4. 设置 ID 过滤器
5. 切换 MCP2510 到正常状态（Normal）
6. 清空接受和发送缓冲区
7. 开启接收缓冲区，开启中断（可选）

8.编程问题

- 如何使能**CAN**总线控制器**MCP2510**?
- **CAN**总线接口的地址? 如何通过**SPI**控制**CAN**总线逻辑? 如何编程**SPI**? **SPI**的地址?
- 如何向**CAN**总线控制器发送命令、数据、**ID**标志符等?

三.硬件连接及端口地址

1. GPH端口

Register	Address	R/W	Description	Reset Value
GPHCON	0x56000070	R/W	Configure the pins of port H	0x0
GPHDAT	0x56000074	R/W	The data register for port H	Undefined
GPHUP	0x56000078	R/W	Pull-up disable register for port H	0x0
Reserved	0x5600007C	—	Reserved	Undefined

GPH0为output: GPHCON[1:0]=01

使MCP2510的CS为0则: GPHDAT=xx...x0

参阅: 2410S硬件手册P.14的18和芯片手册P.9-19的GPH寄存器介绍

2. SPI寄存器

(1)状态寄存器

SPI STATUS REGISTER

Register	Address	R/W	Description	Reset Value
SPSTA0	0x59000004	R	SPI channel 0 status register	0x01
SPSTA1	0x59000024	R	SPI channel 1 status register	0x01

SPSTAn	Bit	Description	Initial State
Reserved	[7:3]		
Data Collision Error Flag (DCOL)	[2]	This flag is set if the SPTDATn is written or the SPRDATn is read while a transfer is in progress and cleared by reading the SPSTAn. 0 = not detect, 1 = collision error detect	0
Multi Master Error Flag (MULF)	[1]	This flag is set if the nSS signal goes to active low while the SPI is configured as a master, and SPPINn's ENMUL bit is multi master errors detect mode. MULF is cleared by reading SPSTAn. 0 = not detect, 1 = multi master error detect	0
Transfer Ready Flag (REDY)	[0]	This bit indicates that SPTDATn or SPRDATn is ready to transmit or receive. This flag is automatically cleared by writing data to SPTDATn. 0 = not ready, 1 = data Tx/Rx ready	1

(2)SPI控制寄存器

SPI Channel control register

Register	Address	R/W	Description	Reset Value
SPPIN0	0x59000008	R/W	SPI channel 0 pin control register	0x02
SPPIN1	0x59000028	R/W	SPI channel 1 pin control register	0x02

SPPINn	Bit	Description	Initial State
Reserved	[7:3]		
Multi Master error detect Enable (ENMUL)	[2]	The /SS pin is used as an input to detect multi master error when the SPI system is a master. 0 = disable (general purpose) 1 = multi master error detect enable	0
Reserved	[1]	This bit should be '1'.	1
Master Out Keep (KEEP)	[0]	Determine MOSI drive or release when 1byte transmit is completed (only master). 0 = release, 1 = drive the previous level	0

(3)SPI波特率缩放因子寄存器

SPI Baud Rate Prescaler Register

Register	Address	R/W	Description	Reset Value
SPPRE0	0x5900000C	R/W	SPI cannel 0 baud rate prescaler register	0x00
SPPRE1	0x5900002C	R/W	SPI cannel 1 baud rate prescaler register	0x00

SPPREn	Bit	Description	Initial State
Prescaler Value	[7:0]	Determine SPI clock rate as above equation. Baud rate = PCLK / 2 / (Prescaler value + 1)	0x00

(4)数据发送和接收寄存器

SPI Tx Data Register

Register	Address	R/W	Description	Reset Value
SPTDAT0	0x59000010	R/W	SPI channel 0 Tx data register	0x00
SPTDAT1	0x59000030	R/W	SPI channel 1 Tx data register	0x00

SPTDATn	Bit	Description	Initial State
Tx Data Register	[7:0]	This field contains the data to be transmitted over the SPI channel.	0x00

SPI Rx Data Register

Register	Address	R/W	Description	Reset Value
SPRDAT0	0x59000014	R	SPI channel 0 Rx data register	0x00
SPRDAT1	0x59000034	R	SPI channel 1 Rx data register	0x00

SPRDATn	Bit	Description	Initial State
Rx Data Register	[7:0]	This field contains the data to be received over the SPI channel.	0x00

3.SPI总线控制方式

- 主从关系

4. CAN 寄存器

(1) CAN控制寄存器

寄存器 9-1: CANCTRL - CAN 控制寄存器 (地址: XFh)

R/W-1	R/W-1	R/W-1	R/W-0	U-0	R/W-1	R/W-1	R/W-1
REQOP2	REQOP1	REQOP0	ABAT	—	CLKEN	CLKPRE1	CLKPRE0
bit 7							bit 0

bit 7-5 **REQOP<2:0>**: 工作模式设定

- 000 = 设定为正常工作模式
- 001 = 设定为休眠模式
- 010 = 设定为回环模式
- 011 = 设定为监听模式
- 100 = 设定为配置模式

REQOP 不应设定为其它任何值, 任何其它设定值皆为无效。

注: 上电复位时, REQOP = b'111'

bit 4 **ABAT**: 报文发送中止设定

- 1 = 请求中止所有等待发送的发送缓冲器
- 0 = 终止报文发送中止请求

bit 3 **未用**: 读作 '0'

bit 2 **CLKEN**: CLKOUT 引脚使能设定

- 1 = CLKOUT 引脚使能
- 0 = CLKOUT 引脚禁止 (引脚处于高阻状态)

bit 1-0 **CLKPRE <1:0>**: CLKOUT 引脚预分频器设定

- 00 = FCLKOUT = 系统时钟频率 / 1
- 01 = FCLKOUT = 系统时钟频率 / 2
- 10 = FCLKOUT = 系统时钟频率 / 4
- 11 = FCLKOUT = 系统时钟频率 / 8

(3)寄存器映射表

表 10-1: CAN 控制寄存器映射表

低地址位	高地址位							
	x000 xxxxx	x001 xxxxx	x010 xxxxx	x0011 xxxxx	x100 xxxxx	x101 xxxxx	x110 xxxxx	x111 xxxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID0	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

注：可以采用位修改指令对表中阴影部分的寄存器进行单独位的位修改

(4)寄存器总汇

表 10-2: 控制寄存器汇总

寄存器名称	地址 (Hex)	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	上电复位值
BFPCTRL	0C	—	—	B1BFS	B0BFS	B1BFE	B0BFE	B1BFM	B0BFM	--00 0000
TXRTSCTRL	0D	—	—	B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM	--xxx x000
CANSTAT	xE	OPMOD2	OPMOD1	OPMOD0	—	ICOD2	ICOD1	ICOD0	—	100- 000-
CANCTRL	xF	REQOP2	REQOP1	REQOP0	ABAT	—	CLKEN	CLKPRE1	CLKPRE0	1110 -111
TEC	1C	发送错误计数器								0000 0000
REC	1D	接收错误计数器								0000 0000
CNF3	28	—	WAKFIL	—	—	—	PHSEG22	PHSEG21	PHSEG20	-0-- -000
CNF2	29	BTLMODE	SAM	PHSEG12	PHSEG11	PHSEG10	PRSEG2	PRSEG1	PRSEG0	0000 0000
CNF1	2A	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	0000 0000
CANINTE	2B	MERRE	WAKIE	ERRIE	TX2IE	TX1IE	TX0IE	RX1IE	RX0IE	0000 0000
CANINTF	2C	MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF	0000 0000
EFLG	2D	RX1OVR	RX0OVR	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000
TXB0CTRL	30	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
TXB1CTRL	40	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
TXB2CTRL	50	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
RXB0CTRL	60	—	RXM1	RXM0	—	RXRTR	BUKT	BUKT	FILHIT0	-00- 0000
RXB1CTRL	70	—	RSM1	RXM0	—	RXRTR	FILHIT2	FILHIT1	FILHIT0	-00- 0000

(2)CAN状态寄存器

寄存器 9-2: CANSTAT - CAN 状态寄存器 (地址: XEh)

R-1	R-0	R-0	U-0	R-0	R-0	R-0	U-0
OPMOD2	OPMOD1	OPMOD0	—	ICOD2	ICOD1	ICOD0	—
bit 7				bit 0			

bit 7-5 **OPMOD<2:0>**: 工作模式

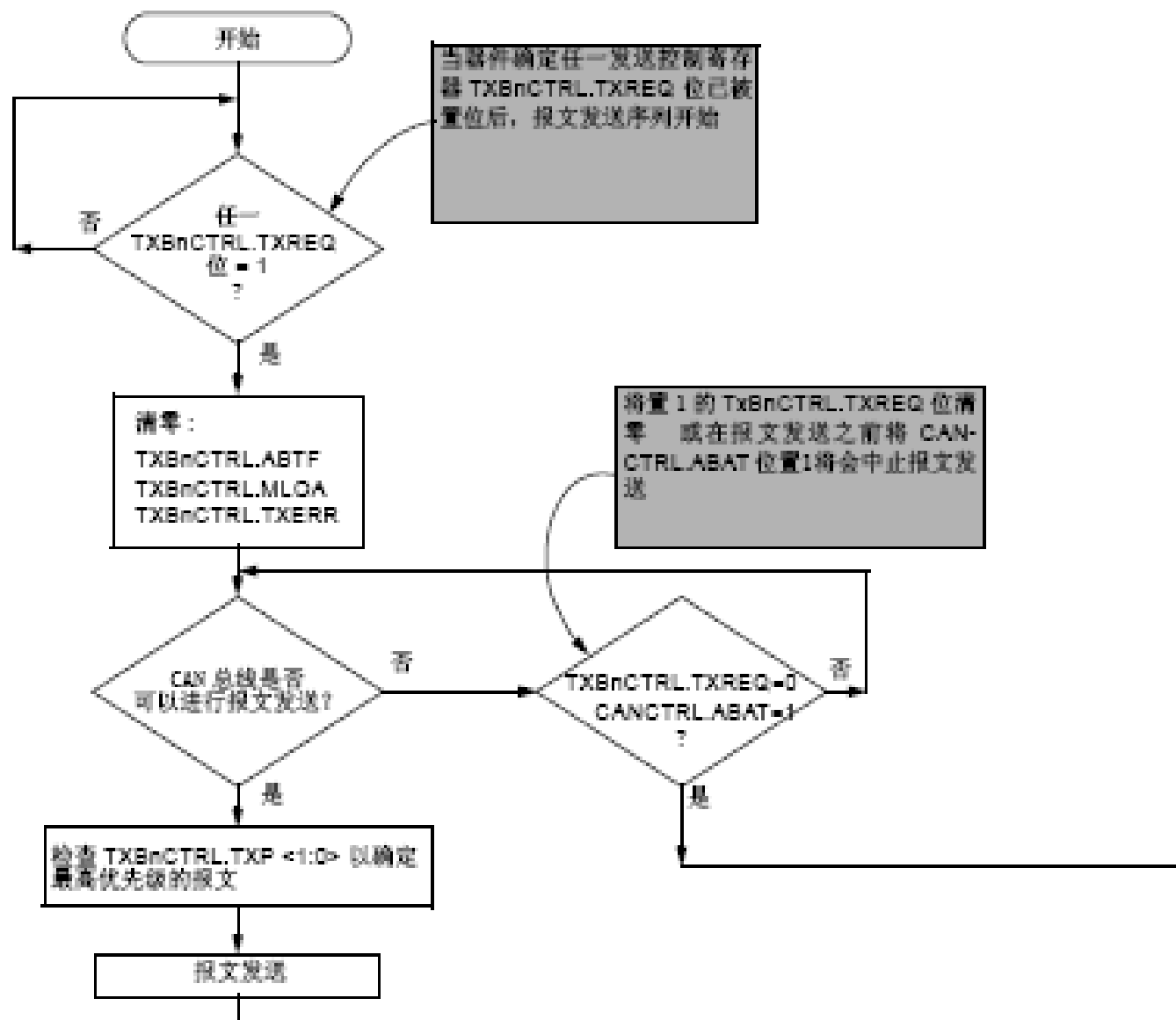
- 000 = 器件处于正常工作模式
- 001 = 器件处于休眠模式
- 010 = 器件处于回环模式
- 011 = 器件处于监听模式
- 100 = 器件处于配置模式

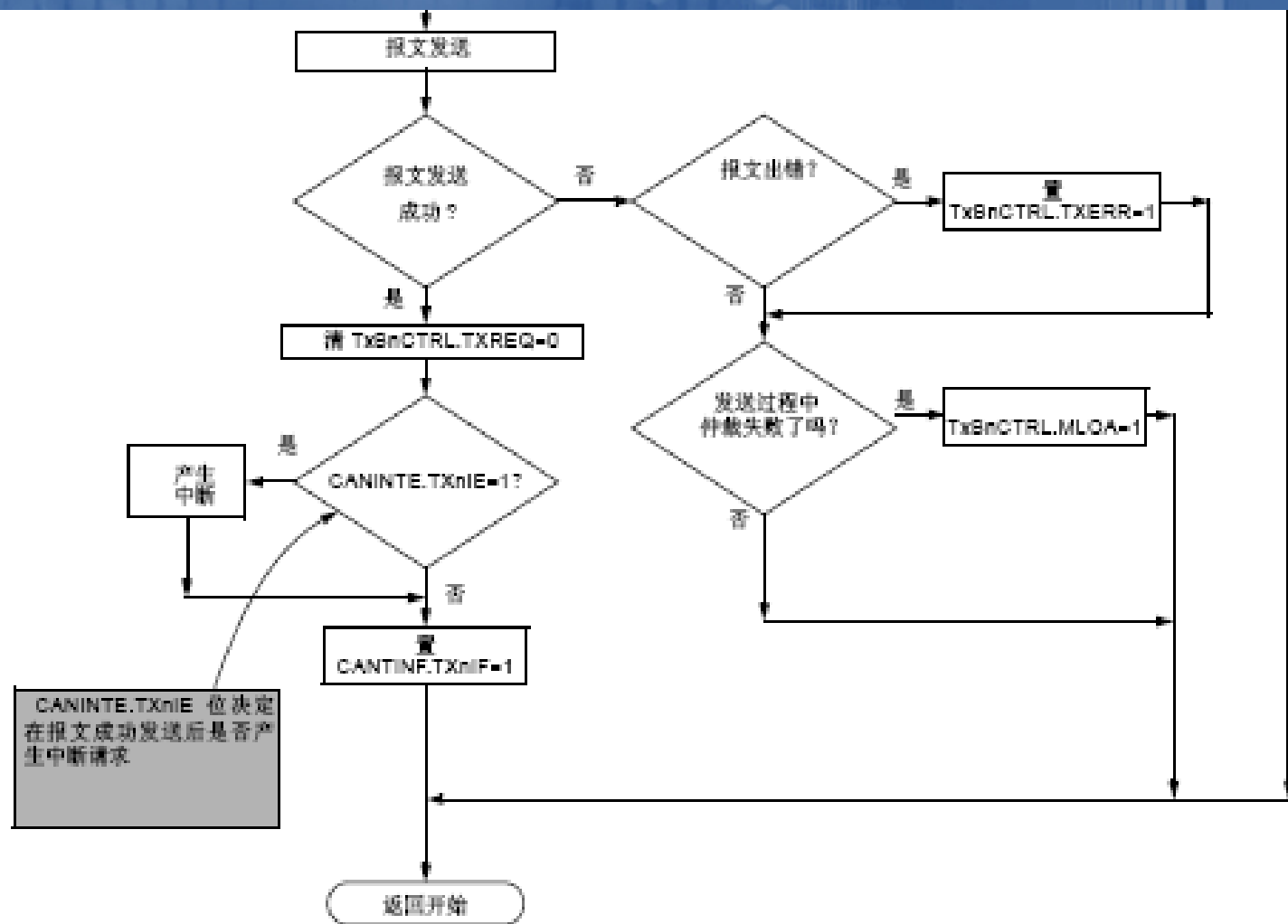
bit 4 **未用**: 读作 '0'

bit 3-1 **ICOD<2:0>**: 中断标志码

- 000 = 无中断
- 001 = 出错中断
- 010 = 唤醒中断
- 011 = TXB0 中断
- 100 = TXB1 中断
- 101 = TXB2 中断
- 110 = RXB0 中断
- 111 = RXB1 中断

bit 0 **未用**: 读作 '0'





四 关键编程实现

1. MCP2510的Enable和Disable

- `#define MCP2510_Enable() do {GPHDAT &= ~MCP2510_PIN_CS;udelay(1000);}while(0);`
- `#define MCP2510_Disable() do {GPHDAT |= MCP2510_PIN_CS;}while(0);`
- `#define MCP2510_OPEN_INT()
enable_irq(MCP2510_IRQ) //added by wb`

2. static int __init s3c2410_mcp2510_init(void)

```
set_gpio_ctrl(GPIO_MCP2510_CS); //使能port GPH0  
init_MCP2510(BandRate_250kbps); //初始化2510及波特率
```


3. **init_MCP2510(CanBandRate bandrate)**

```
MCP2510_Reset();
```

```
MCP2510_SetBandRate(bandrate,FALSE);
```

```
// Disable interrupts.
```

```
MCP2510_Write(CANINTE, NO_IE);
```

```
// Mark all filter bits as don't care:
```

```
MCP2510_Write_Can_ID(RXM0SIDH, 0, TRUE);
```

```
MCP2510_Write_Can_ID(RXM1SIDH, 0, TRUE);
```

```
// Anyway, set all filters to 0:
```

```
MCP2510_Write_Can_ID(RXF0SIDH, 0, 0);
```

```
MCP2510_Write_Can_ID(RXF1SIDH, 0, 0);
```

```
MCP2510_Write_Can_ID(RXF2SIDH, 0, 0);
```

```
MCP2510_Write_Can_ID(RXF3SIDH, 0, 0);
```

```
MCP2510_Write_Can_ID(RXF4SIDH, 0, 0);
```

```
MCP2510_Write_Can_ID(RXF5SIDH, 0, 0);
```

```
//Enable clock output
```

```
MCP2510_Write(CLKCTRL, MODE_LOOPBACK | CLKEN | CLK1);
```

```
// Clear, deactivate the three transmit buffers
```

```
// and the two receive buffers.
```

```
MCP2510_Write(RXB0CTRL, 0);
```

```
MCP2510_Write(RXB1CTRL, 0);
```

(4)

**open/write/read/interrupt/io
ctl**

嵌入式操作系统

步进电机驱动及 Linux设备驱动程序

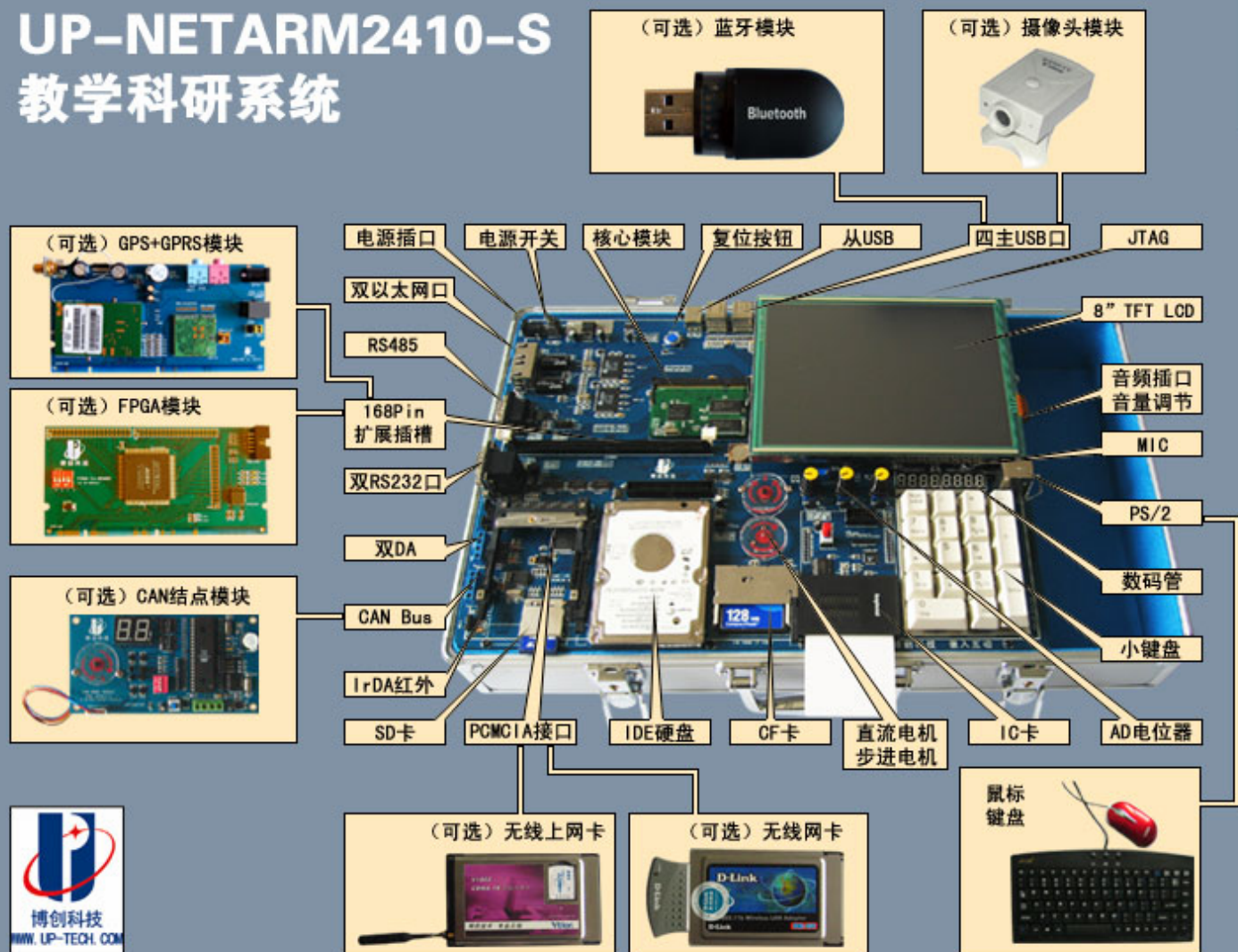
提纲

- 步进电机驱动工作原理
- 步进电机控制方法
- 控制功能
- 设备驱动程序
- 测试过程

1. 步进电机工作原理

(1) 步进电机在实验板的位置

UP-NETARM2410-S 教学科研系统



(2)实验内容

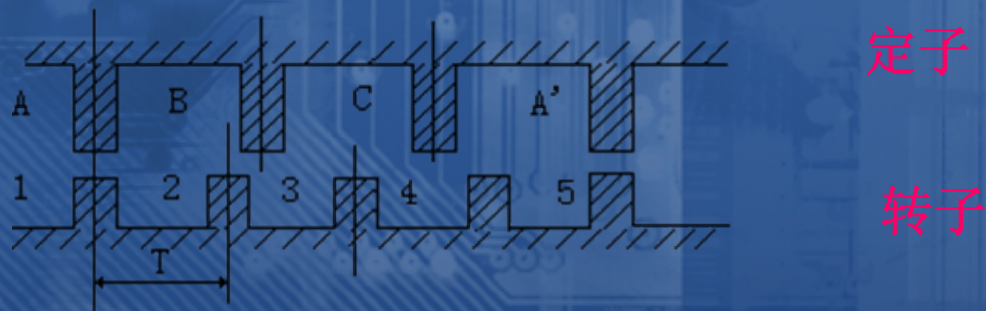
- 了解步进电机控制工作原理
- 分析步进电机设备驱动程序，掌握设备驱动程序编写方法
- 运行测试程序，调用步进电机设备驱动程序，验证设备驱动程序的正确性

(3) 步进电机工作机制

- 步进电机是一种能够将电脉冲信号转换成角位移或线位移的机电元件，它实际上是一种单相或多相同步电动机。
- 每输入一个脉冲到脉冲分配器，电动机各相的通电状态就发生变化，转子会转过一定的角度（称为步距角），步进电机转过的总角度和输入的脉冲数成正比
- 连续输入一定频率的脉冲时，电动机的转速与输入脉冲的频率保持严格的对应关系，不受电压波动和负载变化的影响。
- 由于步进电动机能直接接收数字量的输入，所以特别适合于微机控制
- 应用：影碟机、光驱、软驱等

(4) 步进电机转动原理

以三相反应式步进电机为例. 以下为定转子的展开图)



- 电机转子均匀分布着很多小齿，定子齿有三个励磁绕组，其几何轴线依次分别与转子齿轴线错开
- 如A相通电，B，C相不通电时，由于磁场作用，齿1与A对齐，（转子不受任何力以下均同）。
- 如B相通电，A，C相不通电时，齿2应与B对齐，此时转子向右移过 $1/3\tau$ ，
- 如C相通电，A，B相不通电，齿3应与C对齐，此时转子又向右移过 $1/3\tau$ ，此时齿4与A偏移为 $1/3\tau$ 对齐。
- 如A相通电，B，C相不通电，齿4与A对齐，转子又向右移过 $1/3\tau$ 这样经过A、B、C、A分别通电状态，齿4（即齿1前一齿）移到A相，电机转子向右转过一个齿距，如果不断地按A，B，C，A.....通电，电机就每步（每脉冲） $1/3\tau$ 向右旋转。

(5) 电机速度和步距角:

- 电机的位置和速度由**导电次数（脉冲数）**和**频率**成一一对应关系。而方向由**导电顺序**决定。
- **步距角 α** ：步进电机定子绕组的通电状态每改变一次，它的转子便转过一定的角度
- 假定相邻两转子齿轴线间的距离为齿距以 τ 表示，采用**A--B-A** 导电状态时步距角为 $1/3\tau$ ；采用**A-AB-B-BC-C-CA-A**这种导电状态，**步距角**为 $1/6\tau$ 。
- 步距角 $=360^\circ / mCZ_k$

其中**m**:相数， **Z_k** 为转子小齿数，**C**为1或2

- 系统中采用的是四相单、双八拍控制方法，所以步距角为 $360^\circ / 512$ 。但步进电机经过一个**1/8** 的减速器引出，实际的步距角应为 $360^\circ / 512/8$

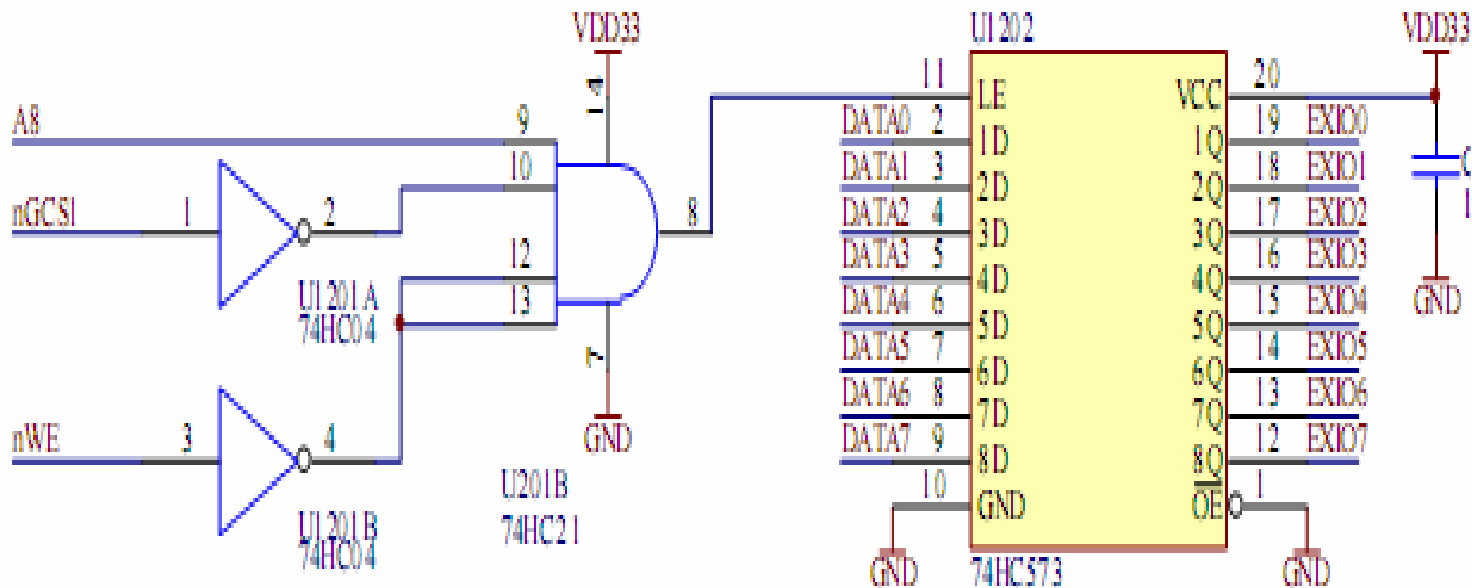
(6) 控制软件设计要解决的问题

- 利用那条信号线控制步进电机工作
- 如何产生电脉冲信号
- 如何实现电机控制软件(**电机控制驱动程序 s3c2410-exio.c**)
- 如何测试确认电机控制驱动程序的正确性 (**stepmotor.c**)

2.步进电机控制方法

(1) 硬件连接及端口地址

- 开发平台中使用EXI/O 的高四位控制四相步进电机的四个相。
- 端口地址：**0x08000100**



(2) 电机转动的控制方法

- 按照四相单、双八拍控制方法，电机正转时的控制顺序为**A→AB→B→BC→C→CD→D→DA**

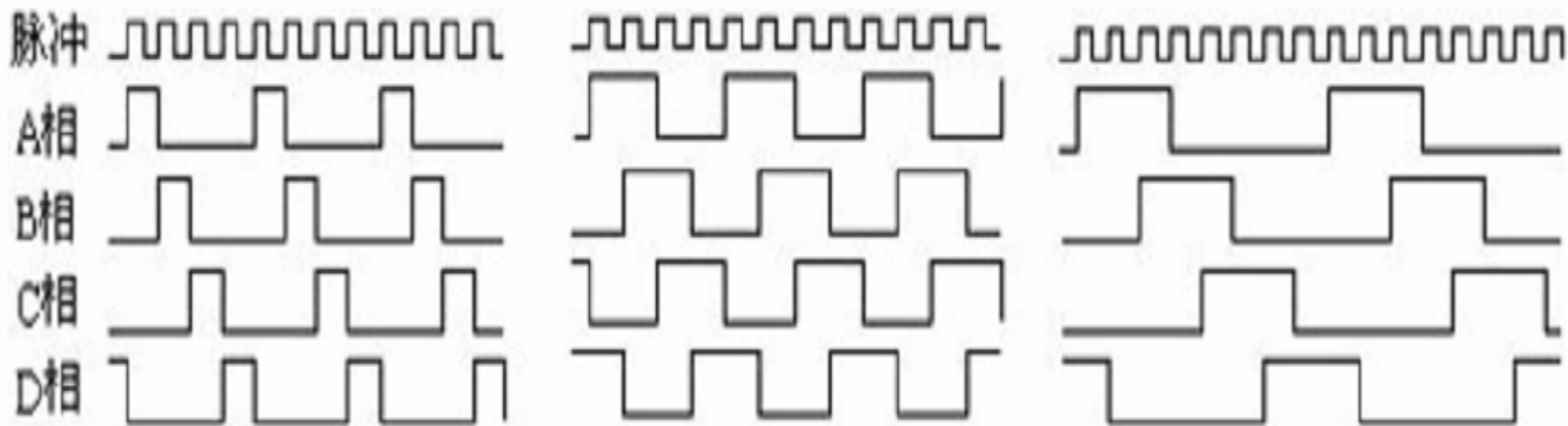
讨论：如何反转？

- EXIO** 的高四位值：

十六进制	二进制	通电状态
1H	0001	A
3H	0011	AB
2H	0010	B
6H	0110	BC
4H	0100	C
CH	1100	CD
8H	1000	D
9H	1001	DA

- 编程方法：依次向**EXIO**端口高4位写入**1H、3H、2H、6H、4H、CH、8H、9H**，电机即循环正转

各相波形变化情况：（哪一个？）



3.控制功能

i.改变步进电机相位

```
#define EXIODEV_PHY_START 0x08000100
```

```
s3c2410_exio_base = (unsigned int)  
ioremap(EXIODEV_PHY_START, SZ_1K); //端口虚拟地址
```

```
bak = readw(s3c2410_exio_base); //读接口寄存器
```

```
bitops_mask_bit(phase, 0xf0, &bak);//清除4-8位然后再  
设置phase传进来的位（也是4-8位）
```

```
writew(bak, s3c2410_exio_base); //调用内核函数将bak的  
值写入相应的寄存器中
```

ii. 设置写入值

```
static void bitops_mask_bit(int nr, int mask,  
    volatile void *addr)
```

```
{  
*((unsigned int *) addr) &= ~mask;  
*((unsigned int *) addr) |= nr;  
}
```


4.设备驱动程序

(1) 设备驱动程序文件: **s3c2410-exio.c**

(2) 文件操作接口

```
static struct file_operations s3c2410_exio_fops = {  
    owner:    THIS_MODULE,  
    open:     s3c2410_exio_open,  
    ioctl:    s3c2410_exio_ioctl,  
    release:  s3c2410_exio_release,  
};
```

(3) **s3c2410_exio_ioctl**内核函数

do stepmotor run((char)arg);

(4)

```
static int do_stepmotor_run(char phase)
```

```
{
```

```
    unsigned int bak;
```

```
    bak = readw(s3c2410_exio_base);
```

```
    bitops_mask_bit(phase, 0xf0, &bak);
```

```
    writew(bak, s3c2410_exio_base); //enable jtag  
    output
```

```
    bak = readw(s3c2410_exio_base);
```

```
    return 0;
```

```
}
```

5.测试程序

(1) stepmotor.c

```
char stepdata[]={0x10,0x30,0x20,0x60,0x40,0xc0,0x80,0x90};  
//各个相位对应的值  
for (;;) {  
    for (i=0; i<sizeof(stepdata)/sizeof(stepdata[0]); i++) {  
        ioctl(step_fd, STEPMOTOR_IOCTL_PHASE,  
               stepdata[i]);  
    }  
}
```


6.总结

编写步进电机控制软件的关键：

- 查阅硬件连接，获得设备接口寄存器地址
- **掌握接口编程控制方法**
- 实现设备驱动程序框架

7.改进设备驱动程序

改进方法:

- 修改应用程序，让步进电机反方向运转
- 在2个*ioctl*系统调用间添加时间延迟
- 在*do_stepmotor_run()*核心函数中添加时间延迟

测试过程:

- 重新编译和设备驱动程序
- 运行应用程序，观察、分析运行结果

嵌入式操作系统

直流电机驱动及 Linux设备驱动程序

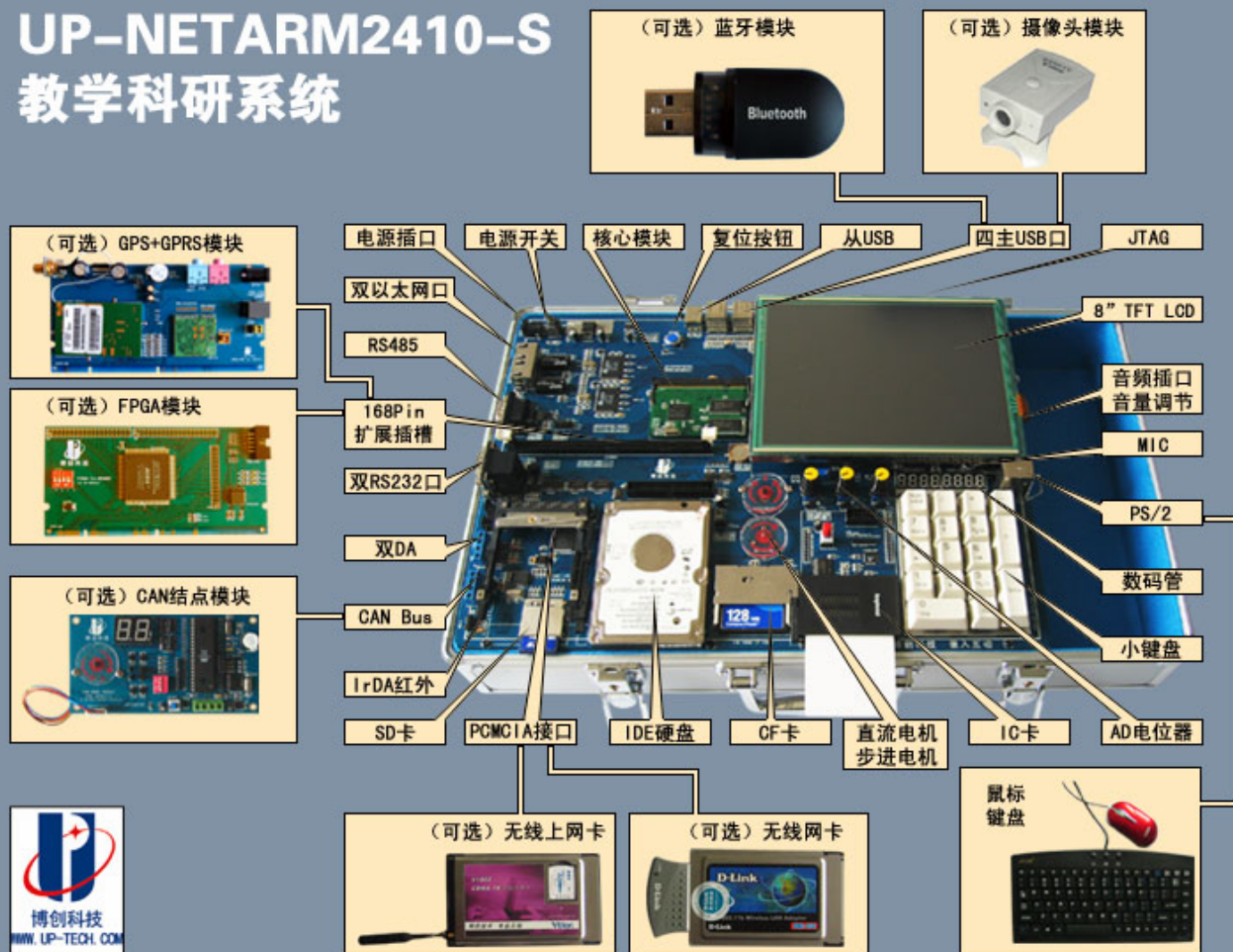
提纲

- 直流电机驱动工作原理
- 直流电机控制方式
- 脉冲控制信号的产生
- 接口寄存器
- 控制功能
- 设备驱动程序
- 测试过程

1.直流电机驱动工作原理

(1) 直流电机在实验板的位置

UP-NETARM2410-S 教学科研系统



(2)实验内容

- 学习和掌握步进电机控制基本原理
- 分析步进电机设备驱动程序，掌握直流电机设备驱动程序的结构和编写方法
- 运行测试程序，调用电机控制设备驱动程序，验证驱动程序的正确性

(3) 直流电动机的PWM 等效电路

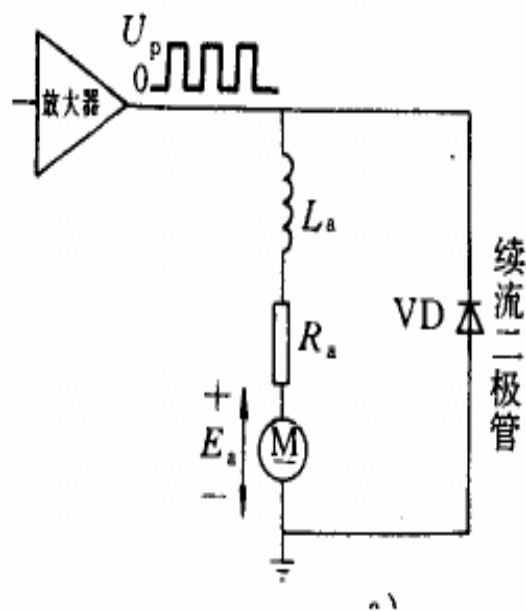


图 2.9.2 (a) 等效电路

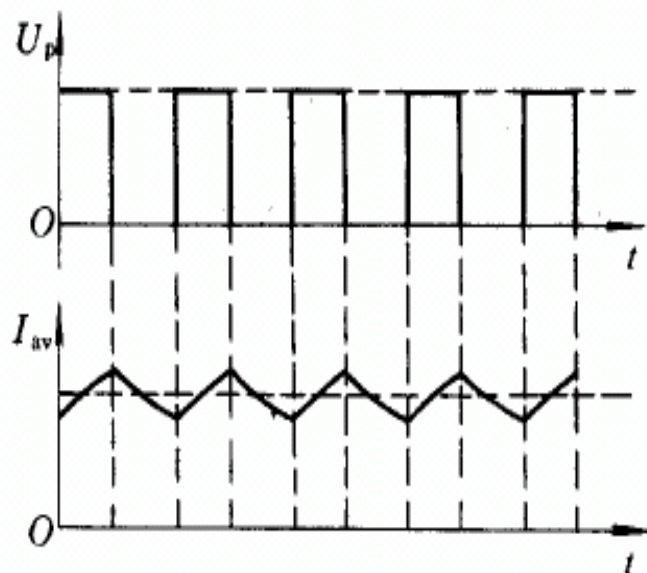


图 2.9.2 (b) PWM 电路中电流和电压波讨论

(4) 直流电动机的PWM 电路原理

- 通过改变导通角 α 的大小（晶体管的导通时间），来改变加在负载上的平均电压的大小，以实现 对电动机的变速控制，

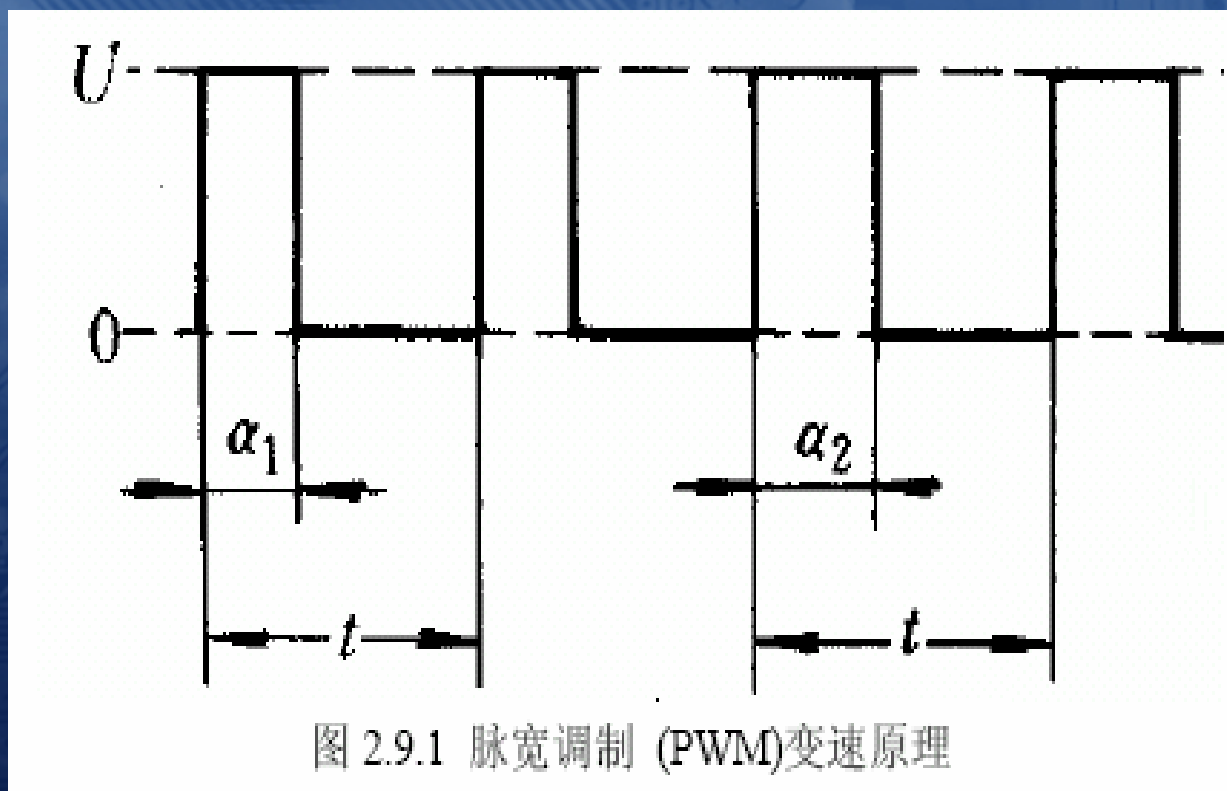


图 2.9.1 脉宽调制 (PWM)变速原理

(5)控制软件设计要解决的问题

- 用什么**信号控制线**连接电机，并在其上加载控制信号(**GPB0,TOUT0**)
- 如何产生控制信号(**PWM**)
- 如何实现电机控制软件(**电机控制驱动程序 s3c2410-dc-motor.c**)
- 如何测试确认电机控制驱动程序的正确性(**dcm_main.c**)

2.直流电机控制方法

(1)控制信号

- 直流电机: PWM TOUT0

参考: 2410硬件说明书P.15,P.6

2410芯片手册P.10-2 (PWM)

(2) 芯片引脚

Table 9-1. S3C2410X Port Configuration (Sheet 2 of 5)

Port B	Selectable Pin Functions			
GPB10	Input/output	<u>nXDREQ0</u>	–	–
GPB9	Input/output	<u>nXDACK0</u>	–	–
GPB8	Input/output	<u>nXDREQ1</u>	–	–
GPB7	Input/output	<u>nXDACK1</u>	–	–
GPB6	Input/output	<u>nXBREQ</u>	–	–
GPB5	Input/output	<u>nXBACK</u>	–	–
GPB4	Input/output	<u>TCLK0</u>	–	–
GPB3	Input/output	<u>TOUT3</u>	–	–
GPB2	Input/output	<u>TOUT2</u>	–	–
GPB1	Input/output	<u>TOUT1</u>	–	–
GPB0	Input/output	<u>TOUT0</u>	–	–

参考：2410芯片手册P.9-3 (I/O ports)

(3) 控制方法

- 在**GPB0**加载一定大小、频率和宽度的脉冲电压，改变加在负载上的平均电压的大小，从而实现对电动机的变速控制
- 设置**GPB0**的连接属性及输入输出属性
- 对**PWM**计时器进行编程在**GPB0**上产生一定宽度的脉冲

2.脉冲控制信号的产生

(1) PWM时钟频率的产生

An 8-bit prescaler and a 4-bit divider make the following output frequencies:

4-bit divider settings	Minimum resolution (prescaler = 0)	Maximum resolution (prescaler = 255)	Maximum interval (TCNTBn = 65535)
1/2 (PCLK = 50 MHz)	0.0400 us (25.0000 MHz)	10.2400 us (97.6562 KHz)	0.6710 sec
1/4 (PCLK = 50 MHz)	0.0800 us (12.5000 MHz)	20.4800 us (48.8281 KHz)	1.3421 sec
1/8 (PCLK = 50 MHz)	0.1600 us (6.2500 MHz)	40.9601 us (24.4140 KHz)	2.6843 sec
1/16 (PCLK = 50 MHz)	0.3200 us (3.1250 MHz)	81.9188 us (12.2070 KHz)	5.3686 sec

输入时钟频率 = $\text{PCLK} / (\text{prescaler value} + 1) / (\text{divider value})$

参考: 2410芯片手册P.10-3 (PWM)

(2) PWM脉冲(TOUT)频率和脉冲宽度的确定

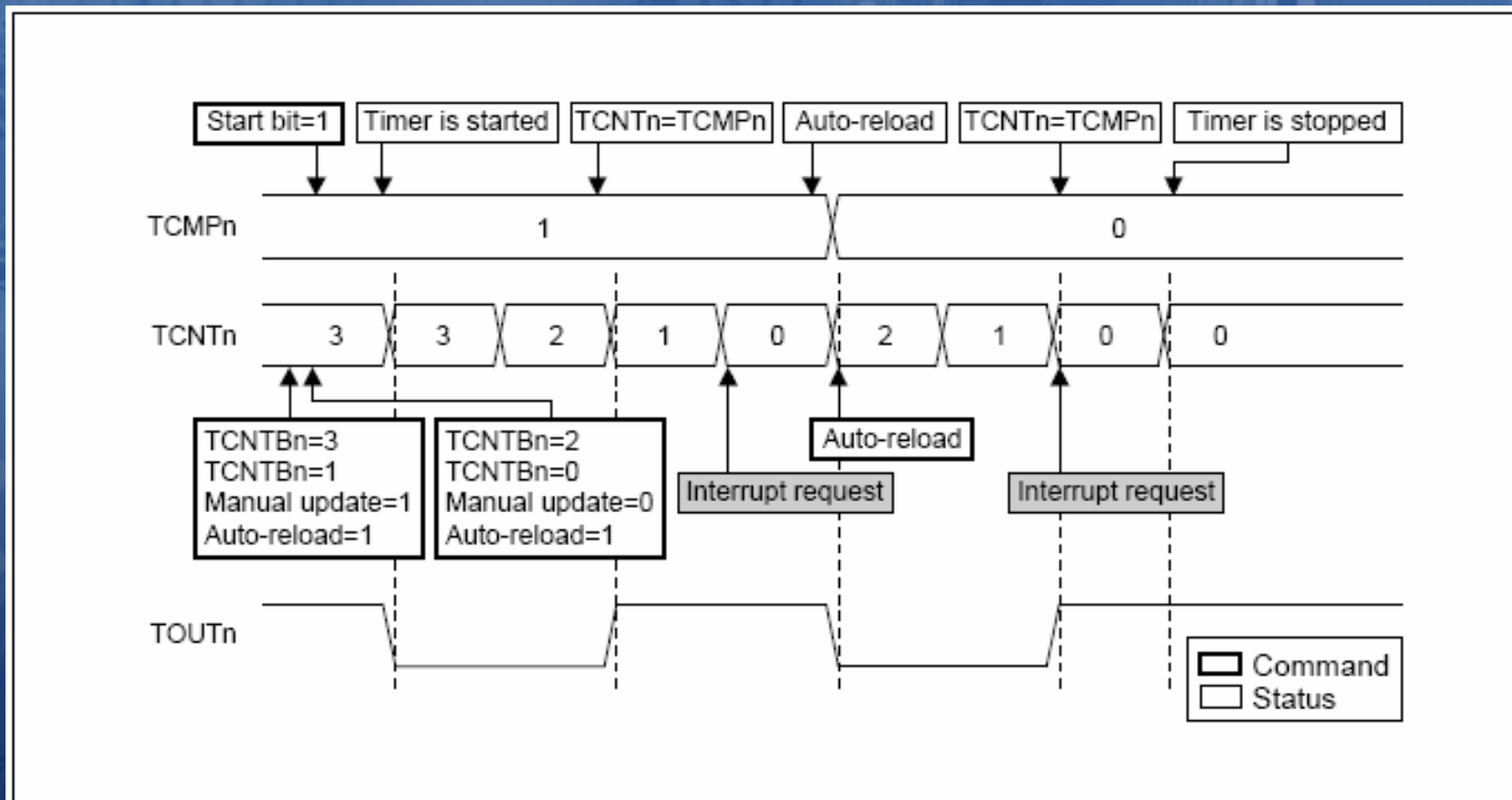


Figure 10-2. Timer Operations

TCNTB0 决定了脉冲的频率, *TCMPB0* 决定了*TOUT*正脉冲的宽度:
当 $TCMPB0 = TCNTB0 / 2$ 时, 正负脉冲宽度相同;
当*TCMPB0* 由0 变到*TCNTB0* 时, 负脉冲宽度不断增加

(3) 小结：编程工作

- 将**GPB0**设置为**TOUT0**
- 设置输入时钟频率的参数：**prescaler value**和**divider value**
- 设置**TCNTB0**和**TCMPB0**寄存器的值
- 控制计时器的工作：**TCON**

4.接口寄存器

(1) GPB端口属性设置寄存器

PORT B CONTROL REGISTERS (GPBCON, GPBDAT, and GPBUP)

Register	Address	R/W	Description	Reset Value
GPBCON	0x56000010	R/W	Configure the pins of port B	0x0
GPBDAT	0x56000014	R/W	The data register for port B	Undefined
GPBUP	0x56000018	R/W	Pull-up disable register for port B	0x0
Reserved	0x5600001C	-	Reserved	Undefined

GPB3	[7:6]	00 = Input 10 = TOUT3	01 = Output 11 = reserved
GPB2	[5:4]	00 = Input 10 = TOUT2	01 = Output 11 = reserved]
GPB1	[3:2]	00 = Input 10 = TOUT1	01 = Output 11 = reserved
GPB0	[1:0]	00 = Input 10 = TOUT0	01 = Output 11 = reserved

GPBUP	Bit	Description
GPB[10:0]	[10:0]	0: The pull-up function attached to to the corresponding port pin is enabled. 1: The pull-up function is disabled.

编程方法：将GPBCON寄存器(存储器地址为0x56000010)后4为的值设置为1010

参考：2410芯片手册P.9-9 (I/O ports)

(2) PWM产生有关寄存器

i. 定时器配置寄存器TCFG0

表 2.9.1 TCFG0 寄存器

寄存器	地址	读/写	描述	复位值
TCFG0	0x51000000	R/W	Configures the two 8-bit prescalers	0x00000000

TCFG0	位	描述	初始状态
Reserved	[31:24]		0x00
Dead zone length	[23:16]	These 8 bits determine the dead zone length. The 1 unit time of the dead zone length is equal to that of timer 0.	0x00
Prescaler 1	[15:8]	These 8 bits determine prescaler value for Timer 2, 3 and 4.	0x00
Prescaler 0	[7:0]	These 8 bits determine prescaler value for Timer 0 and 1.	0x00

参考：Dead zone length=0；prescaler value=2。

编程方法实例：将TCFG0寄存器(存储器地址为0x51000000)后2位设置为10，预置分频值为：2

ii. 定时器配置寄存器TCFG1

输入时钟频率 = $50\text{MHz}/3/2$

表 2.9.2 TCFG1 寄存器

寄存器	地址	读/写	描述	复位值
TCFG1	0x51000004	R/W	5-MUX & DMA mode selecton register	0x00000000

TCFG1	位	描述	初始状态
Reserved	[31:24]		00000000
DMA mode	[23:20]	Select DMA request channel 0000 = No select (all interrupt) 0010 = Timer1 0100 = Timer3 0110 = Reserved 0001 = Timer0 0011 = Timer2 0101 = Timer4	0000
MUX 4	[19:16]	Select MUX input for PWM Timer4. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK1	0000
MUX 3	[15:12]	Select MUX input for PWM Timer3. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK1	0000
MUX 2	[11:8]	Select MUX input for PWM Timer2	0000
MUX 1	[7:4]	Select MUX input for PWM Timer1. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK0	0000
MUX 0	[3:0]	Select MUX input for PWM Timer0. 0000 = 1/2 0001 = 1/4 0010 = 1/8 0011 = 1/16 01xx = External TCLK0	0000

编程方法示例：将TCFG1寄存器(存储器地址为0x51000004)后4位设置为0000，分频系数为1/2

iii. 定时器计数缓冲区寄存器和比较缓冲区寄存器 TCNTB0& TCMPB0

TIMER 0 COUNT BUFFER REGISTER & COMPARE BUFFER REGISTER (TCNTB0/TCMPB0)

Register	Address	R/W	Description	Reset Value
TCNTB0	0x5100000C	R/W	Timer 0 count buffer register	0x00000000
TCMPB0	0x51000010	R/W	Timer 0 compare buffer register	0x00000000

TCMPB0	Bit	Description	Initial State
Timer 0 compare buffer register	[15:0]	Set compare buffer value for Timer 0	0x00000000

TCNTB0	Bit	Description	Initial State
Timer 0 count buffer register	[15:0]	Set count buffer value for Timer 0	0x00000000

iv. TCON 定时器控制寄存器

TIMER CONTROL (TCON) REGISTER

Register	Address	R/W	Description	Reset Value
TCON	0x51000008	R/W	Timer control register	0x00000000

TCON	Bit	Description	Initial state
Reserved	[7:5]	Reserved	
Dead zone enable	[4]	Determine the dead zone operation. 0 = Disable 1 = Enable	0
Timer 0 auto reload on/off	[3]	Determine auto reload on/off for Timer 0. 0 = One-shot 1 = Interval mode(auto reload)	0
Timer 0 output inverter on/off	[2]	Determine the output inverter on/off for Timer 0. 0 = Inverter off 1 = Inverter on for TOUT0	0
Timer 0 manual update (note)	[1]	Determine the manual update for Timer 0. 0 = No operation 1 = Update TCNTB0 & TCMPOB0	0
Timer 0 start/stop	[0]	Determine start/stop for Timer 0. 0 = Stop 1 = Start for Timer 0	0

参考：dead zone operation enable; Inverter off

编程方法实例：启动和停止计时器： ?

人工装载递减计数器和比较计数器： 0x1x

自动装载递减计数器和比较计数器： 1x0x

5. 控制功能

(1) 设置直流电机控制信号

```
#define tout01_enable() \  
    ({ GPBCON &=~ 0xf; \  
        GPBCON |= 0xa;    }) //GPB0和GPB1为TOUT0和TOUT1
```

```
#define tout01_disable() \  
    ({ GPBCON &=~ 0xf; \  
        GPBCON |= 0x5;    \  
        GPBUP &=~ 0x3;    }) //设置GPB0和GPB1为OUTPUT  
                                // 设置GPB0和GPB1为pull-up
```


(2) 设置TCMPB0 和TCNTB0 寄存器的计数值

```
#define DCM_TCNTB0    (16384)
```

```
TCNTB0 = DCM_TCNTB0;    /* less than 10ms */
```

```
TCMPB0 = DCM_TCNTB0/2;
```

(3) 改变电机速率

```
TCMPB0 = DCM_TCNTB0/2 + v
```

(4) 停止定时器

```
TCON &= ~0x1;
```

(5) 启动定时器

```
#define DCM_TCFG0 (2)
```

```
TCFG0 &= ~(0x00ff0000);
```

```
TCFG0 |= (DCM_TCFG0); //prescaler=2
```

```
TCFG1 &= ~(0xf); //divider=1/2
```

```
TCNTB0 = DCM_TCNTB0; //DCM_TCNTB0=16384
```

```
TCMPB0 = DCM_TCNTB0/2; //  
TCON &= ~(0xf);
```

```
TCON |= (0x2); //更新 TCNT0和TCMB0
```

```
TCNT0 = (0xf);
```

6.设备驱动程序分析

```
static struct file_operations
s3c2410_dcm_fops = {
    owner:      THIS_MODULE,
    open:  s3c2410_dcm_open,
    ioctl:  s3c2410_dcm_ioctl,
    release:   s3c2410_dcm_release,
};
```


7. 编写步进电机控制软件的关键:

- 查阅硬件连接, 获得设备接口寄存器地址
- **掌握接口编程控制方法**
- 实现设备驱动程序框架

8.测试程序分析

- 改变速率:

```
ioctl(dcm_fd, DCM_IOCTL_SETPWM, (setpwm *  
factor)); //change dc motor speed
```

- 启动PWM

```
dcm_fd=open(DCM_DEV, O_WRONLY);
```

- 停止PWM

```
close(dcm_fd);
```

9.改进设备驱动程序功能

修改方法:

- 修改应用程序延迟时间
- 修改设备驱动程序参数,如:

prescale=255

devider=1/8

测试过程:

- 运行、观察和分析运行结果