基于 ARM 系统下映像文件的执行与中断运行机制的实现

冯　玮,喻晓峰
(国防科技大学 信息系统与管理学院,湖南 长沙 410073)
(fwfwfwfw_82@126.com)

摘　要:阐述了 ARM 系统下的映像文件执行和调用过程,并以 ARM9200 下对 FIQ 相应的处理过程为例阐述了中断运行的相关机制。

关键词:中断;映像文件;ARM;重映射

1　ARM 映像文件的概念

ARM 中的各种源文件(包括汇编程序、C 语言程序以及 C++语言程序)经过 ARM 编译后生成 ELF 格式的目标文件,该类文件和相应的运行时库经过 ARM 连接器处理后,生成 ELF 格式的映像文件(image),这种格式的映像文件可以被写入到 ROM 中。

通常 ARM 映像文件有 axf 和 bin 两种格式,对于裸机而言(Flash 里面没有任何的信息)在 ARM9200 上电或复位后从 0x0 处开始执行,bin 文件是真正的可执行文件,而 axf 文件是 ARM 特有的调试文件,里面除了 bin 文件的内容之外,还有很多的附加调试信息,这些调试信息可以在 ARM 的开发环境 ADW 或者 AXF 的 loadimage 中调试。

2　ARM 映像文件的构成

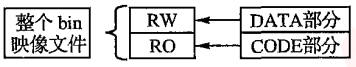

图 1　ARM 映像文件

ARM 映像文件是一个层次性结构的文件,其中包含了域(region),输出段(output section)和输入段(input section)。输入与输出段有其相应的属性,可以为只读(RO),可读写(RW)以及初始化成 0 的(ZI),一个域中包含 1～3 个输出段。以下面的程序为例:

```
AREA  INIT, CODE, READONLY (代码部分 RO)
CODE32
…
ENTRY
Start
…
AREA  LINEOUT, DATA, READWRITE (数据部分 RW)
…
END
```

从上面的代码可以发现:

域　整个 bin 映像文件,也就是说整个程序的映像文件中只有一个域(加载域)。

输出段　有两个输出段 RO 和 RW。

输入段　两个输入段,即程序的 CODE 部分和 DATA 部分。CODE 部分 READONLY,属于 RO 输出段,DATA 部分是 READWRITE,属于 RW 输出段。

3　ARM 系统下映射文件的执行

ARM 中映像文件各组成部分在存储系统中的地址有两种:一种是该映像文件开始运行之前的地址,即加载地址;另一种是映像文件运行时的地址,即运行地址。在 ARM 公司提供的 ADS 调试环境中可以设置加载 RO 和 RW 的地址 ro_base 和 rw_base 的值,假定 RW 段加载时的地址为 0x4000,该地址位于 ROM 中;RW 段运行时的地址为 0x8000,该地址位于 RAM 中,则加载时地址映射关系和运行时地址映射关系如图 2 所示。

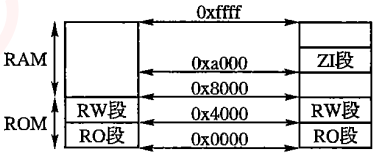

图 2　加载时地址映射关系和运行时地址映射关系 1

如果系统采用地址重映射的方式,假设 ro_base = 0x00000000, ro_base = 0xa00000 则运行和加载地址的映射关系如图 3 所示。

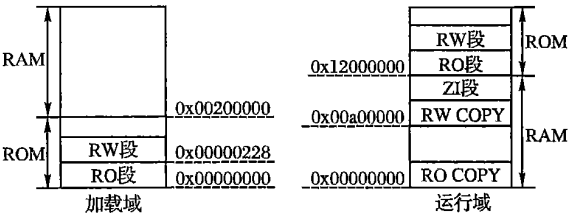

图 3　加载时地址映射关系和运行时地址映射关系 2

所谓地址重映射就是通过系统的存储管理部件改变系统中各个存储器的映射地址,在 ARM9200 中,通过改变系统控制寄存器组来改变存储器的映射地址。从图 3 可以看出在重映射以前,先将 RO 的备份拷贝到 RAM 中,然后将 RAM 重映射为 0x00000000,重映射后 FLASH 的地址为 0x12000000,此时系统执行的是 RAM 中 ROcopy 的指令;然后由 BOOTLOAD 将 ROM 中的 RW 数据段移到 RAM 中相应的地方。

采用地址重映射的原因首先是由于 ARM9200 系统上电后的实际地址在 0x0 处,其中在某种类型的 ROM 中保存了中断向量表,当用户要修改或者使用相应的中断向量指针时,由于不能直接在 ROM 中进行修改或赋值,给程序的编写带来了困难,所以采用了重映射的机制,即通过软件的方法,为别的存储器(主要是快速的 RAM)分配以 0 起始的地址,将

收稿日期:2005-09-15

作者简介:冯玮(1982-),男,湖南岳阳人,硕士研究生,主要研究方向:嵌入式系统;　喻晓峰(1958-),男,湖南长沙人,副教授,主要研究方向:嵌入式系统,硬件系统开发.

ROM中的内容进行数据搬移,拷贝到速度相对较快的 RAM中,这样不单单可以加快程序的执行速度,而且可以对中断向量表中的中断向量指针进行修改。

4 ARM系统中的中断相应流程

ARM中共有 7种类型的异常中断,按中断优先级从高到低可以分为复位 (Reset)、数据访问中断 (Data Abort)、快速中断请求 (FIQ)、外部中断 (IRQ)、指令预取中断 (Prefetch Abort)、软件中断 (SWI)、未定义指令中断 (Undefined instruction),在 ARM9200中系统上电后从 0x0处开始执行指令,中断向量表具体如表 1所示[1]。

表 1 中断向量表

中断向量地址	异常中断类型	异常中断模式	优先级 (6为最低)
0x0	复位	特权模式	1
0x4	未定义指令	未定义指令终止模式	6
0x8	软件中断	特权模式	6
0x0c	指令预取中断	终止模式	5
0x10	数据访问中断	终止模式	2
0x14	保留	未使用	未使用
0x18	外部请求中断	外部中断模式	4
0x1c	快速中断请求	快速中断模式	3

向量表里面的每一条指令直接跳向对应的异常处理函数。其中 FIQ_Handler()可以直接从地址 0x1C处开始。ARM的跳转指令 (B)是有范围限制的 (±32MB),但很多情况下不能保证所有的异常处理函数都定位在向量表的 32MB范围内,需要大于 32MB的长跳转,并且由于向量表空间的限制只能由一条指令完成,所以通过 MOV PC, #imme_data指令给 PC直接赋值或者用 LDR PC, [PC + offset]指令把目标地址先存储在某一个合适的地址空间,然后把这个存储器单元上的 32位数据传送给 PC来实现跳转,如图 4所示。这种方法对目标地址值没有要求,可以是任意有效地址,但是存储目标地址的存储器单元必须在当前指令的 ±4KB空间范围内。

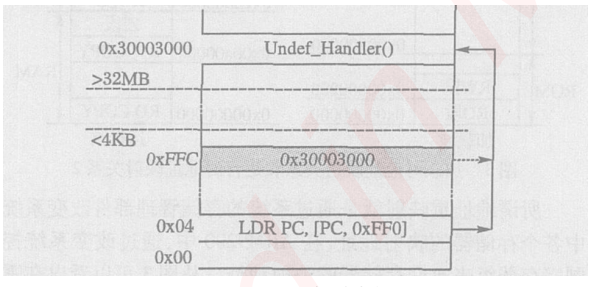

图 4 ARM中断流程

ARM内核只有两个外部中断输入信号 FIQ和 IRQ,但对于一个系统来说,中断源可能多达几十个。为此,在系统集成的时候,一般都会有一个异常控制器来处理异常信号。ARM9200中采用的是优先中断控制器即 (AIC)[2]来区分中断源,如图 5所示。

这时候,用户程序可能存在多个中断处理函数,当从向量表起始跳转时,为了最终能找到正确的处理函数入口,需要设计一套处理机制和方法。针对这种情况有软件和硬件处理两种方式,在硬件处理上有的系统在 ARM的异常向量表之外,又增加了一张由中断控制器控制的特殊向量表。当由外设触发一个中断以后,PC能够自动跳到这张特殊向量表中去,特殊向量表中的每个向量空间对应一个具体的中断源;软件处理方式是我们开发 ARM9200时常用的一种方式,当外部请求中断 IRQ发生的时候,在中断控制器中找到其所对应的中断源,通过中断源所对应的中断向量指针,完成一次中断响应的跳转。这种方法的好处是用户程序在运行过程中,能够很方便地动态改变异常服务内容,在 ARM9200中通过设定对应中断状态下的 AIC_SVR寄存器来设定中断服务例程的地址。

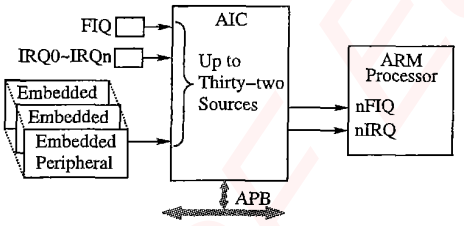

图 5 ARM9200中断控制器的应用

5 中断处理函数的设计

当任何一个异常发生并得到响应时,ARM内核自动完成以下动作[3]:

1)拷贝 CPSR到 SPSR_<mode>;

2)设置适当的 CPSR位:改变处理器状态进入 ARM状态,改变处理器模式进入相应的异常模式,设置中断禁止位禁止相应中断;

3)更新 LR_<mode>;

4)设置 PC到相应的中断向量。

注意当响应中断后,不管中断发生在 ARM还是 Thumb状态下,处理器都将自动进入 ARM状态。还应注意的是中断使能会被自动关闭,也就是说缺省情况下中断是不可重入 (嵌套)的。

从中断返回时常用的操作是:

1)恢复 CPSR状态寄存器,即将 SPSR_<mode>拷贝到 CPSR;

2)返回到发生中断的指令的下一条指令处执行,即将 lr_mode寄存器的内容恢复到程序计数器中。

在 ARM架构里,PC值指向当前执行指令的地址加 8处。也就是说,当执行第一条指令 (地址 0x4000)时,PC等于第三条指令的地址 (0x4008)。假如第一条指令是“BL”指令,则当执行时,会把 PC(=0x4008)保存到 LR寄存器里面,但是接下去处理器会马上对 LR进行一个自动的调整动作:LR=LR-0x4。这样,最终保存在 LR里面的是 B指令的地址,所以当从 BL返回时,LR里面正好是正确的返回地址。

同样的调整机制在所有 LR自动保存操作中都存在,比如进入中断响应时处理器所做的 LR保存中,也进行了一次自动调整,并且调整动作都是 LR=LR-0x4。由此我们可以对不同异常类型的返回地址进行依次比较:

1)如果发生的是软件中断,即第二条指令是“SWI”指令

从 SWI中断返回后下一条执行指令即第三条指令,正好是 LR寄存器保存的地址,所以只要直接把 LR恢复给 PC。

2)如果发生的是“IRQ”或“FIQ”等指令

因为外部中断请求中断了第二条指令的执行,当中断返回后,需要重新回到第二条指令的执行,也就是返回地址应该是 B(0x4004),需要把 LR-4。

3)如果发生的是“Data Abort”

在第二条指令上进入数据异常的响应,但导致数据异常的原因却应该是上一条指令即第一条指令。当中断处理程序修复数据异常以后,要回到第一条上重新执行导致数据异常的指令,因此返回地址应该是 LR-8。

以 IRQ和 FIQ中断为例,这两个异常中断的引脚都是低

电平有效。当程序状态寄存器的 CPSR中的控制位 I为 1时，FIQ和 IRQ异常中断被屏蔽；当控制位为 0时，FIQ和 IRQ的异常中断被打开。异常中断的操作如下：

```
IRQ_Handler;                    //中断响应，从向量表直接跳来
STMFD SP!, {R0 - R12, LR};
                                //保护现场，一般只需保护{r0 - r3, lr}即可
BL IrqHandler;                  //进入普通处理函数，C或汇编均可
LDMFD SP!, {R0 - R12, LR};      //恢复现场
SUBS PC, LR, #4;                //中断返回，注意返回地址
```

通用寄存器的恢复采用一般的堆栈操作指令，而 PC和 CPSR的恢复可以通过一条指令来实现，下面是 3个例子：MOVS pc, lr或 SUBS pc, lr, #4或 LDMFD sp!, {pc}这几条指令都是普通的数据处理指令，特殊之处就是把 PC寄存器作为了目标寄存器，并且带了特殊的后缀“S”或“^”，在特权模式下，“S”或“^”的作用就是使指令在执行时，同时完成从 SPSR到 CPSR的拷贝，达到恢复状态寄存器的目的。

对于不可重入的 IRQ/FIQ异常中断处理程序，为了方便使用高级语言直接编写异常处理函数，ARM编译器对此作了特定的扩展，可以使用函数声明关键字 __irq，这样编译出来的函数就满足异常响应对现场保护和恢复的需要，并且自动加入对 LR进行减 4的处理，符合 IRQ和 FIQ中断处理的要求。在没有 __irq标识的高级语言程序中，就不会有对状态寄存器的恢复。

```
__irq void IRQ_Handler(void)
{…}
```

如果在可重入的 IRQ/FIQ异常中断处理过程中调用了子程序，子程序的返回地址将被保存到 LR_irq中，这样上次中断过程中保存的值将会被破坏，那么子程序就不能正常返回。因此在处理第一次中断的时候就要对寄存器进行相应的处理和保护。

对于一次可重入中断的处理过程如下：1)进入普通不可重入的中断处理；2)保护寄存器：LR, SPSR等；3)与中断控制器进行通话；4)切换到 system状态开中断使能；5)中断处理（现在中数据可开）；6)关闭中断使能，切换控制器状态到 IRQ；7)恢复寄存器：PC, CPRS等；8)结束一次可重入中断处理。

6 FIQ中断响应在 ARM9200下的实现

如图 4所示，FIQ快速响应中断由 AIC中断控制器来控制，当触发 FIQ中断时，通过 AIC处理后的中断信号由 ARM进行处理。根据开发板原理图我们可以知道 FIQ由 PB28信号线与 AIC相连，所以首先要将 PIO（Parallel Input/Output Controller)控制器所控制的第 28根信号线分配给外围设备 B来进行控制，具体实现通过函数 AT91F_PIO_CfgPeriph (AT91C_BASE_PIOB, ((unsigned int) AT91C_PB28_FIQ), 0)来实现。

分配好相应的信号线后就要对 AIC控制器进行配置，包括中断触发模式和优先级的选择，并且设置中断服务例程 AT91F_FIQ_ASM_HANDLER的入口地址给 AIC_SVR寄存器。我们可以通过函数：

```
AT91F_AIC_ConfigureIt(
    AT91C_BASE_AIC,                     //AIC baseaddress
    AT91C_ID_FIQ,                       //System peripheral ID
    AT91C_AIC_PRIOR_HIGHEST,            //Max priority
    AT91C_AIC_SRCTYPE_INT_EDGE_TRIGGERED,
    AT91F_FIQ_ASM_HANDLER);
```

来具体实现，接下来就可以将中断源所对应的中断使能位打开。值得注意是我们在中断信号的选择上选取的是边缘触发模式，即 Edge-triggered模式，所以当中断相应后系统会自动清除中断源。

在中断使能之后，当中断产生时调用相应的中断服务例程。

```
EXPORT AT91F_IRQ0_ASM_HANDLER
IMPORT AT91F_IRQ0_HANDLER
  AT91F_IRQ0_ASM_HANDLER
  IRQ_ENTRY                         //中断前产生时环境的保护
    ldr r1, =AT91F_IRQ0_HANDLER
                                    //调用 C中定义的中断服务例程
    mov r14, pc
    bx r1
  IRQ_EXIT                          //中断后环境的恢复
```

在定义中断服务例程中也可以通过定义 __irq void AT91F_IRQ0_HANDLER(void)直接在原程序中定义中断服务例程，从而减少了通过再次编写汇编程序来完成中断前后环境变量的保护和恢复。

参考文献：

[1] 杜春雷. ARM体系结构[M]. 北京：清华大学出版社，2003.

[2] AT91ARM9200 Introduction[Z]. ATMEL, 2002.

[3] ARM Software Development Toolkit Version 2.0 Programming Techniques ARM[Z]. ARM, 1995.

[4] DDI0151C_920T_TRM ARM[Z]. ARM, 1995.

(上接第 249页)

显然在模型驱动框架里面模型是非常重要的角色。模型不再是软件开发用以参考的图片。因为模型被直接实现，所以必须跟以前写程序一样的精确和高质量。

5 结语

本文介绍并分析了 MDA架构和解决方案。目前 MDA已经有 AndroMDA、OpenMDX等的开源项目。其中 OpenMDX开源项目已经实现了大量的工作，并且已近存在一个基于 OpenMDX开发的功能强大的 CRM开源产品 OpenCRX；但是 OpenMDX还存在很多不完善的地方，如不支持模型的系统行为图、平台的工作流组件不完善、没有集成开发环境等。我们可以在已有的开源项目的基础上完善这些不足和添加一些其他的客户需求。

参考文献：

[1] Copyright 2003 Object Management Group. MDA Guide Version 1.0.1[Z].

[2] Object Management Group. Unified Modeling Language: Superstructure, version 2.0[Z].

[3] Object Management Group. MOF 2.0/XMI Mapping Specification, v2.1[Z].

[4] Object Management Group. XML Metadata Interchange (XMI), version 1.2[Z].

[5] Object Management Group. Common Warehouse Metamodel (CWM) Specification, Version 1.1[Z].

[6] Sun. Java Metadata Interface (JMI) Specification; Frequently Asked Questions[Z].

[7] Sun. Java Data Objects, version 1.0[Z].

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计

Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究

16. 基于 UEFI 的 Application 和 Driver 的分析与开发

Programming:

1. 计算机软件基础数据结构 - 算法