中断调用方式的 ARM 二次开发接口设计

李　硅

（江西理工大学 信息工程学院，赣州 341000）

摘要：为了满足每个用户对各自独立的嵌入式定制程序的需求，缩短二次开发周期，本文提出了一种基于中断调用方式的二次开发接口的设计思路。这种二次开发接口具有简易、安全、更新方便的特点。早期 DOS 功能调用会普遍采用中断调用方式实现 API。用户可以独立、简单、快捷地开发出适合自己需求的程序。

关键词：ARM；二次开发；中断；GCC

中图分类号：TP39　　**文献标识码**：A

ARM Secondary Development Interface Based on Interrupt Call

Li Gui

(School of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China)

Abstract: In order to meet the demands of independent embedded custom programs for each user, and to shorten the second development cycle, this paper presents a design concept of secondary development interface based on interrupt call. The secondary development interface has the features of simple, safe, convenient updating. Early DOS function call commonly uses interrupt calls to actualize API. Users can easily develop programs suited for them quickly.

Key words: ARM; secondary development; interrupt; GCC

引　言

随着信息技术的飞速发展，基于 ARM 芯片的可编程智能嵌入式设备在我们的日常生活中扮演着越来越重要的角色。由于程序的设计者在设计软件程序和功能的时候，所针对的对象是社会上的通常情况，并没有根据最终不同的使用者来加以设计，因此各个软件的实际使用者在使用软件时会根据自己的使用要求和市场需求预测，对软件再加以开发，即俗称的二次开发[1]。如今，Android 系统和苹果产品等都给用户提供开发平台，便于用户开发自己的程序。为了适应用户简易嵌入式应用程序的开发，缩短二次开发周期，开发一种低成本的专用嵌入式二次开发工具就显得十分必要。其中，二次开发工具最重要的部分就是二次开发接口的设计。

通过对现有嵌入式二次开发技术的研究发现 3 个问题。问题一，现有嵌入式二次开发技术主要是直接给用户提供系统函数的 OBJ 中间文件，以及函数声明的头文件。用户通过调用系统函数编写自己的程序，进行编译链接后直接烧录进芯片进行使用，但是这样会造成芯片存储空间的浪费。问题二是系统程序函数的安全问题，它直接给用户提供系统 OBJ 中间文件，通过反编译能够很容易地破解出原有系统函数，系统程序的保密性将大大地降低。问题三是用户编写程序的复杂性问题，一般专业的嵌入式工程师编写的系统函数考虑到重用性，会把一些过程细分为多个子函数，普通用户为实现一个功能也需要调用多个函数，对于二次开发十分不利。函数一般采用嵌入式专业术语进行命名，普通用户很难通过函数名直接了解此函数具体的用途。

1　存在的问题与解决方法

二次开发接口的构架如图 1 所示，通过二次开发接口可以轻松地实现系统程序和用户应用程序的分离。

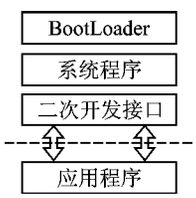

图 1　基于嵌入式系统的二次开发程序结构示意图

通过这种应用程序和系统程序的分离方式，可以轻松地实现系统程序的升级和应用程序的通用。

1.1　Flash 存储问题

系统程序中自带函数 Funtion1 实体、Funtion3 实体，如果用户自己定制的应用程序中还需使用到 Funtion1 和 Funtion3 这 2 个函数，则在编译应用程序文件的时候，编

译器会从系统提供的 OBJ 文件中把 Funtion1 和 Funtion3 这 2 个函数的实体再次编译链接进入 bin 文件，从而造成 bin 文件的冗余。当把应用程序的 bin 文件烧录至 Flash 中时会浪费大量的空间资源。其结果如图 2 所示。

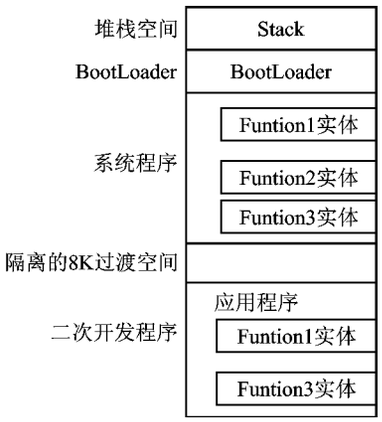

图 2　Flash ROM 存储结构示意图

针对 Flash 的冗余问题，笔者提出了一个方案，就是在应用程序中保存的不是系统函数本身，而是其地址，当运行到那个函数的时候则实现跳转到指定地址，函数运行完成后跳转回来，应用程序继续执行下去。

1.2　系统函数的安全问题

因为使用基于 1.1 节中的地址直接跳转方式，所以不需再向用户提供函数体本身，而是提供函数体的地址，相对于原始方法来说进行反编译更难。

1.3　简化代码编写难度

可以针对用户群进行函数的二次封装，把一些实现整体动作的函数，重新打包成一个新的函数，函数名直接采用直译方式命名，更易明白。比如显示屏特定位置显示字符的过程，可以直接把清屏、输出指针移动、输出字符这 3 个函数封闭在一起。

2　改进方案

在实现接口的设计过程中发现 2 个问题。第 1 个问题，在给用户提供可用函数库的时候需要提供所有函数的地址，对于二次接口的编写者来说工作量比较大。第 2 个问题，在系统程序更新后，所有函数在 Flash 中存放的地址会发生变动，在更新以前编写的应用程序都需要重新定义函数地址才可以再次使用。这一点不利于用户应用程序的开发，没有考虑到应用程序的通用性。

通过研究中断过程发现，可以借鉴 DOS 中断响应时的跳转原理来解决以上 2 个问题。比如 21 号 DOS 的中断调用：

```
MOV AH 4CH
INT 21H
```

是返回 DOS 系统的通过给 AH 寄存器赋值 4CH，然后调用 INT 21H 指令，计算机就会根据 AH 寄存器中的值执行相应的操作。其中，4CH 返回 DOS 系统，还可以给 AH 寄存器赋其他值[2]。

同理这里提出了一种基于上面函数地址跳转的改进方法，即通过设置一个中间跳转函数（这里函数名设置为 SysCallLib）来解决接口的设计过程中发现的 2 个问题。改进的基于地址跳转方法的实现效果如图 3 所示。通过这种方法，把 SysCallLib 函数体固定在特定的地址，用户只需要知道一个地址即可实现跳转。SysCallLib 函数体内部通过 Case 语句来实现内部函数的跳转。

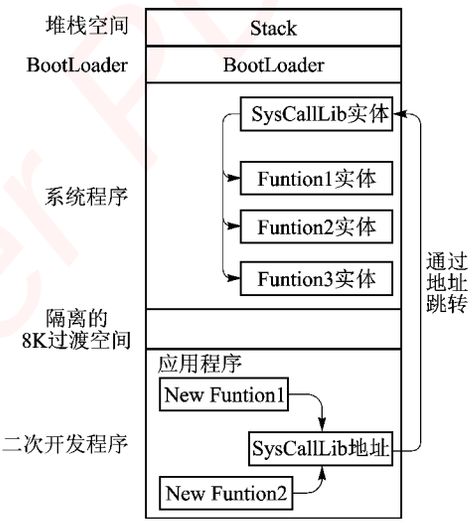

图 3　改进的基于地址跳转方法的 Flash ROM 存储结构示意图

3　二次开发接口的建立

3.1　GCC 开发环境的建立

GNU 工具链在 ARM 产品开发中使用的很广泛，有些为 ARM 打造的开发工具也是基于 GNU 工具链的。在目前，支持 CM3 的 GNU 工具链可以由 CodeSourcery 网站免费下载。

而 GNU 的 C 编译器则在以后支持 Cortex-M3。和其他 ARM 开发工具相似，GNU 工具链也包含了编译器、汇编器和链接器，使得源代码既可以使用 C 语言，也可以使用汇编完成，基于 GNU 工具链的开发模式图如图 4 所示[3]。

这里选用 EABI 版本，其基本操作命令如表 1 所列。Cortex-M3 内核在地址 0x00000000 处存放一个向量表，向量表的第 0 个单元即地址 0x00000000 处存放的是堆栈顶的地址。Cortex-M3 复位后即从该处取出数据，用以初始化 MSP 寄存器。向量表中的内容是 32 位的地址，这些

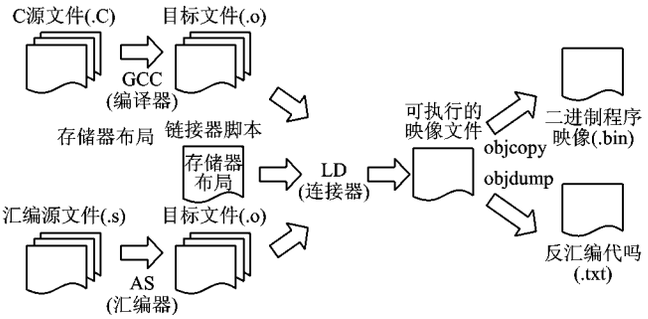

图 4　基于 GNU 工具链的开发模式图

地址是中断异常服务程序的入口地址，其中向量表的第一个单元即地址 0x00000004 处存放的是复位向量，也就是说 Cortex-M3 复位后，执行该向量（可理解为函数指针）指向的复位代码[4]，代码如下：

```
__attribute__((section(".stackarea")))
static unsigned long pulStack[STACK_SIZE];
```

表 1　winARM20080331 GNU 工具链的命令名称

功　能	命　令
汇编器	arm-none-eabi-as
编译器	arm-none-eabi-gcc
链接器	arm-none-eabi-ld
二进制映像产生器	arm-none-eabi-objcopy
反汇编器	arm-none-eabi-objdump

这一句定义了一个 pulStack 数组，程序把这个数组作为了堆栈区。这条语句使用了__attribute__((section(".stackarea")))，把数组定位在了.stackarea 这个段中。

```
typedef void (* pfnISR)(void);
__attribute__((section(".isr_vector")))
pfnISR VectorTable[]={
    (pfnISR)((unsigned long)pulStack + sizeof(pulStack)),
    ResetISR, //初始化堆栈
    NMIException, //复位句柄
    HardFaultException
};
```

定义了一个数组 VectorTable，作为向量表，定位于.isr_vector 段中。通过链接脚本的控制，这个表将放在正文区的最开始，正文区又将从 Flash 开始存放，这样这个向量表就会起到存放在 0x00000000 开始的地址空间的效果。向量表的第 0 个单元是((unsigned long)pulStack+sizeof(pulStack))，这是数组的最后一个元素，因为 Cortex-M3 的堆栈是向下增长的。向量表的第 1 个单元是 ResetISR，它指向复位处理的代码也是整个程序的入口。本程序用它来实现启动代码的功能。

```
extern unsigned long _etext;
extern unsigned long _data;
extern unsigned long _edata;
extern unsigned long _bss;
extern unsigned long _ebss;
void ResetISR(void){
    unsigned long *pulSrc, *pulDest;
    //将保存于 Flash 中的初始化数据复制到 SRAM 中
    pulSrc=&_etext;
    for(pulDest=&_data; pulDest < &_edata;){
        *pulDest++=*pulSrc++;
        }
    // 将.bss 段清零
    for(pulDest=&_bss; pulDest<&_ebss;){
        *pulDest++=0;
    }
    main(); //最后调用 main 进入主程序
}
```

这段代码用到了通过连接器赋值的几个变量值。_etext的值为正文段结尾处的地址，这之后的 Flash 空间是初始化的数据值，应该复制到 SRAM 中去。__data、__edata 的值分别为数据段的开始和结尾处的地址，这部分应该是 SRAM 的地址。

这部分代码的主要功能就是将保存于 Flash 中的初始化数据复制到 SRAM 中。然后将.bss 段清零。最后调用 main 进入到我们的主程序。

3.2　Gcc 二次开发接口设计中的关键技术点

(1) 连接器中 SysCallLib 地址的赋值

连接器所做的工作简单地讲就是把所有目标文件相应的段连接到一起，并把目标文件中的“变量地址”、“函数地址”重新定位至正确的地址空间。通过在 SECTIONS 段中添加函数 PROVIDE 来声明函数地址，代码如下：

```
SECTIONS{
    PROVIDE(SysCallLib= 0x08005935);
    /*对于 Cortex 器件来说，启动代码的开始段 VectorTable 一般存储在.isr_vector 段中*/
    .isr_vector :{
        . =ALIGN(4);
        KEEP(*(.isr_vector)) /*启动代码*/
        . = ALIGN(4);
    } >FLASH
}
```

(2) 用户程序 Flash 空间和 SRAM 空间的设定

对于 STM32 系列芯片来说向量表、.text 和.rodata 就应该放到从 0x08000000 开始的 Flash、.data、.bss 和堆栈，就应该定位至从 0x20000000 开始的 SRAM 中。但是对于用户程序，这些地址空间都需要重新设定。这些定位都可以通过链接脚本进行控制。

```
MEMORY{
    FLASH(rx) :
    ORIGIN=0x08000000, LENGTH=0x20000
    SRAM (rwx):
    ORIGIN=0x20000000, LENGTH=0x5000
}
```

这些语句说明了 Flash 和 SRAM 开始的地址以及大小。只需要修改这里，ORIGIN 的值即可改变用户程序存储的物理地址，通过修改 LENGTH 来修改空间大小。

(3) SysCallLib 函数的编写

SysCallLib 函数在系统程序中进行编写，主要通过 Case 语句来实现跳转。形式如下：

```
void SysCallLib(unsigned int Index, int * Ret, unsigned char Argc, unsigned int Arg[]);
```

通过函数的参数来判断需要调用的系统函数。通过把 SysCallLib 函数作为过渡函数来为用户编写全新的 API 函数，用户使用起来更加方便，以 LCD 显示图片的函数为例，代码如下：

```
void mLCD_DisplayImage(unsigned char x, unsigned char y, const unsigned char * ImagePt, unsigned char ImageXSize, unsigned char ImageYSize ){
    int RetVal;
    unsigned int Arg[5];
    unsigned char Argc=5;
    Arg[0]=x;
    Arg[1]=y;
    Arg[2]=(unsigned int)ImagePt;
    Arg[3]=ImageXSize;
    Arg[4]=ImageYSize;
    SysCallLib(__SYSCALL__LCD__DIS__IMAGE, &RetVal, Argc, Arg);
}
```

用户只需调用简单的 API 函数 mLCD_DisplayImage 即可在 LCD 上显示图片。

结　语

通过实验证明，具有 C 语言基础的用户只需参看工具手册即可轻松编写出自己的定制程序。这种二次接口的设计优点可以总结如下：解决了二次开发过程中用户程序的冗余问题；提高了系统函数的安全性；系统程序的更新不会影响到以往版本的使用，地址固定后，只需添加新增的函数地址即可实现更新；用户定制程序编写简单，用户只需调用提供的 API 函数，即可编写出所需定制程序。

参考文献

[1] 彭强. 论软件二次开发知识产权保护[D]. 武汉：华中师范大学，2009.

[2] 杨延双，魏坚华，张晓冬. 微机原理及汇编语言教程[M]. 北京：北京航空航天大学出版社，2002.

[3] 姚文详. ARM Cortex-M3 权威指南[M]. 宋岩，译. 北京：北京航空航天大学出版社，2009.

[4] 在 Linux 下用 Gcc 4.3.1 进行 STM32 开发入门教程[EB/OL]. [2012-11]. http://www.amobbs.com/forum.php?mod=viewthread&tid=1444935.

李硅(硕士)，主要研究方向为嵌入式可视化编程。

(责任编辑：杨迪娜　收稿日期：2012-11-13)

13 到短消息，不保存短消息，直接由 MODEM 转发至单片机。这时单片机和 MODEM 之间的通信方式必须为全双工，类似于中断方式，只是不再需要读短消息和删除短消息了。这时转发的短消息均是以“+CMT:”开头的，可以根据是否收到“+CMT:”来判断收到的数据是否是短消息，再对短消息进行处理，在这里就不详细介绍了。

结　语

本文介绍的两种收发方式虽说是对 PDU 方式而言的，但也可用于文本方式的短消息的收发。在应用中，可以根据单片机的硬件资源情况和对短消息实时性的要求，来选择相应的收发方式，在资源较少的单片机应用中，可选择查询方式，在实时性要求较高且资源丰富的单片机应用中，可选择中断方式。查询方式和中断方式均在实际的应用中都取得了良好的效果。

参考文献

[1] European Telecommunications Standards Institute. GSM 03.40 version 7.3.0 Release 1998. [EB/OL]. (1999-11)[2012-11-19]. http://www.etsi.com.

[2] European Telecommunications Standards Institute. GSM 03.38 version 7.0.0 Release 1998. [EB/OL]. (1998-07)[2012-11-19]. http://www.etsi.com.

[3] WAVECOM. An introduction to the SMS in PDU modeGSM Recommendation phase 2[EB/OL]. (2000-01-01)[2012-11-19]. http://www.wavecom.com.

[4] WAVECOM. AT commands interface[EB/OL]. (2000-12)[2012-11-19]. http://www.wavecom.com

刘海涛(工程师)，从事嵌入式系统研发工作。

(责任编辑：杨迪娜　收稿日期：2012-11-22)

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
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化

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
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析

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
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发

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
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统

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
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计

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
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现

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
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析