VxWorks 环境下 socket 的实现

周　进，　王秀丽，　郁立虎
(上海船舶运输科学研究所，上海 200135)

摘　要:介绍了 TCP/ IP 模型下 socket 的实现过程,详细阐述了 socket 接口所提供的常用函数及其使用方法,解决了嵌入式系统同 PC 或工作站之间的网络通信问题,并给出一种基于 VxWorks 操作系统的 TCP 服务器端的实现方法。

关键词: 网络通信; TCP/ IP; socket; TCP
中图分类号: TP316.89　　**文献标志码**: A

Realization of Socket under VxWorks

ZHOU Jin,　*WANG Xiuli*,　*YU Lihu*
(Shanghai Ship and Shipping Research Institute, Shanghai 200135, China)

Abstract: The development of socket under TCP/ IP is described, and the normal functions provided by socket interface and the methods of using them are introduced in details. The network communication problems between the embedded system and the PC or workstation are settled and a realization method of TCP server based on VxWorks operating system is given.

Key words: network communication; transmission control protocol/ internet protocol; socket; transmission control protocol

1　TCP/ IP 模型简介

TCP/ IP 是一组协议的代名词,传输层的 TCP 协议、网际互联层的 IP 协议和许多别的协议共同构成了 TCP/ IP 协议簇。其中最重要的两个核心协议是 TCP 协议和 IP 协议。基于 TCP/ IP 的体系结构只有 4 层,从下到上分别为网络接口层、网际互联层、传输层和应用层。与 ISO 的 7 层 OSI 参考模型相比,结构更为简单。TCP/ IP 协议簇的部分协议含义如下:

IP(Internet Protocol) ——网际协议,提供无连接的数据传送和路由服务,位于体系结构的网际互联层。

ARP(Address Resolution Protocol) ——地址解析协议,用于查找与给定 IP 地址相对应的物理地址,位于体系结构的网际互联层。

RARP(Reverse Address Resolution Protocol) ——反向地址解析协议,用于解决物理地址到 IP 地址的转换问题,位于体系结构的网际互联层。

ICMP(Internet Control Message Protocol) ——因特网控制报文协议,用于对 IP 数据报的传送进行差错控制,对未能完成传送的数据报给出出错原因,位于体系结构的网际互联层。

TCP(Transmission Control Protocol) ——传输控制协议,用于提供一种可靠的面向连接的数据传输服务,位于体系结构的传输层。进行通信的双方在传输数据前,必须建立连接。此外,TCP 还具有确认和重传机制、差错控制和流量控制等功能,以确保报文段传送的顺序和传输无错。

UDP(User Datagram Protocol) ——用户数据报协议,用于提供一种无连接的不可靠的数据传输服务,位于体系结构的传输层。尽管 UDP 提供的是不可靠的服务,但是它开销小,效率高。

TCP/ IP 体系结构的分层工作原理以及主机通过 2 个网络互联的结构示意如图 1 所示。

图 1 中,在数据逐个通过每一层直到被当做一串比特流送入网络的过程中,每一层对收到的数据都要增加一些首部信息(有时还要增加尾部信息),这一过程称为封装;接收过程则相反,即逐层拆装。

TCP 协议传送给 IP 的协议数据单元称为 TCP 报文段或简称 TCP 段,UDP 协议传送给 IP 的协议数据单元称为 UDP 数据报;IP 协议传送给网络接口层的协议数据单元称为 IP 数据报;通过以太网传输的比特流称为数据帧。

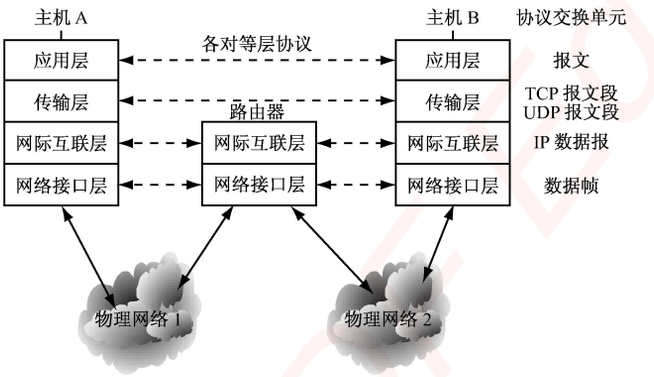

图 1 TCP/ IP 分层工作示意图

2 socket 接口及函数

socket 接口是 TCP/ IP 网络的 API(Application Programming Interface,应用程序编程接口),socket 接口定义了许多函数或例程,程序员可以用它们来开发 TCP/ IP 网络上的应用程序。常用的 socket 类型有 2 种:流式 socket (SOCK_STREAM)和数据报式 socket(SOCK_DGRAM)。流式是一种面向连接的 socket,针对面向连接的 TCP 服务应用;数据报式 socket 是一种无连接的 socket,对应于无连接的 UDP 服务应用。

2.1 socket 创建

为了创建 socket,程序可以调用 socket 函数,该函数返回一个整型 socket 描述符,该描述符作为 socket 的标识传递给其它 socket 接口函数。socket 函数原型为:

```
int socket(int domain, int type, int protocol);
```

domain 指明所使用的协议族,一般为 AF_INET;

type 参数指定 socket 的类型: SOCK_STREAM 、SOCK_DGRAM 或 SOCK_RAW ;

protocol 通常赋值“0”。

2.2 socket 配置

通过调用 socket()返回一个 socket 描述符后,在使用 socket 进行网络传输以前,必须配置该 socket。无连接 socket 的客户端和服务器端以及面向连接 socket 的服务器端通过调用 bind 函数来配置本地信息。

bind 函数将 socket 与本机的网络地址相关联,该网络地址包含本机 IP 地址及端口号等信息。bind 函数原型为:

```
STATUS bind(int s,struct sockaddr *name, int namelen);
```

s 是传递给函数的 socket 描述符;

name 是一个指向包含有本机 IP 地址及端口号等信息的 sockaddr 类型的指针;

namelen 常被设置为 sizeof(struct sockaddr)。

struct sockaddr 结构类型是用来保存 socket 信息的:

```
struct sockaddr {
unsigned short sa_family; /* 地址族, AF_xxx */
char sa_data[14]; /* 14 字节的协议地址 */
};
```

sa_family 一般为 AF_INET,代表 Internet(TCP/ IP)地址族;

sa_data 则包含该 socket 的 IP 地址和端口号。

另外还有一种结构类型:

```
struct sockaddr_in {
```

short int sin_family; /* 地址族 */
unsigned short int sin_port; /* 端口号 */
struct in_addr sin_addr; /* IP 地址 */
unsigned char sin_zero[8]; /* 填充 0 以保持与 struct sockaddr 同样大小 */
};

这个结构更方便使用。sin_zero 用来将 sockaddr_in 结构填充到与 struct sockaddr 同样的长度,可以用 bzero()或 memset()函数将其置为零。指向 sockaddr_in 的指针和指向 sockaddr 的指针可以相互转换,这意味着如果一个函数所需参数类型是 sockaddr 时,就可以在函数调用的时候将一个指向 sockaddr_in 的指针转换为指向 sockaddr 的指针;或者相反。

2.3 连接建立

对于面向连接的客户程序,需要在本机 socket 与远程 socket 之间建立一个联接的虚电路,在 socket 中,这种联接通过调用 connect 函数来配置,其函数原型为:

STATUS connect(int s, struct sockaddr *name,int namelen);

s 是传递给函数的 socket 描述符;

name 是要连接主机的网络地址指针,包含远端主机 IP 地址和端口号;

namelen 是远端地址结构的长度。

面向连接的服务器也从不启动一个连接,它只是在协议端口监听客户的请求。

listen 函数使 socket 处于被动的监听模式,并为该 socket 建立一个输入数据队列,将到达的服务请求保存在此队列中,直到程序处理它们。

STATUS listen(int s, int backlog);

s 是传递给函数的 socket 描述符。

backlog 指定在请求队列中允许的最大请求数,进入的连接请求将在队列中等待 accept()它们。如果一个服务请求到来时,输入队列已满,该 socket 将拒绝连接请求,客户将收到一个出错信息。

accept()函数让服务器接收客户的连接请求。在建立好输入队列后,服务器就调用 accept()函数,然后睡眠并等待客户的连接请求。

int accept(int s, void *addr, int *addrlen);

s 是传递给函数的 socket 描述符;

addr 通常是一个指向 sockaddr_in 变量的指针,该变量用来存放提出连接请求服务的主机的信息,包括该主机的 IP 地址及发出请求的端口号;

addrlen 通常为一个指向值为 sizeof(struct sockaddr_in)的整型指针变量。

当 accept()函数监视的 socket 收到连接请求时,socket 执行体将建立一个新的 socket,执行体将这个新 socket 和请求连接进程的地址联系起来,收到服务请求的初始 socket 仍可继续在以前的 socket 上监听,同时可以在新的 socket 描述符上进行数据传输操作。

2.4 数据传输

write()和 read()这 2 个函数可用于面向连接的 socket 上进行数据传输。

write()函数原型为:

int write (int fd, char *buffer, size_t nbytes);

fd 用来传输数据的 socket 描述符;

buffer 是一个指向要发送数据的指针;

nbytes 是要传送的数据的长度,以字节为单位。

read()函数原型为:

int read (int fd, char * buffer, size_t maxbytes);

fd 是接受数据的 socket 描述符;

buffer 是存放接收数据的缓冲区;

maxbytes 是缓冲区的长度,以字节为单位;

read ()返回实际上接收的字节数,当出现错误时,返回 ERROR。

sendto()和 recvfrom()常用于在无连接的数据报 socket 方式下进行数据传输。由于本地 socket 并没有与远端机器建立连接,所以在发送数据时应指明目的地址。

sendto()函数原型为:

```
int sendto(int s, caddr_t buf, int buflen, int flags, struct sockaddr * to, int tolen);
```

s 是传递给函数的 socket 描述符;

buf 是要传送的数据区;

buflen 是要送的数据长度,以字节为单位;

flags 置 0;

to 接收方的网络地址,包括 IP 地址和端口号;

tolen 接收方地址结构的长度,常置为 sizeof (struct sockaddr)。

recvfrom()函数原型为:

```
int recvfrom(int s, char *buf, int bufLen, int flags, struct sockaddr * from, int * pFromLen)
```

s 是传递给函数的 socket 描述符;

buf 是接收数据缓冲区的指针;

buflen 缓冲区长度,以字节为单位;

flags 置 0;

from 用于存放数据来源方的地址信息,包括 IP 地址和端口号;

pFromLen 接收方地址结构的长度指针。

不管是否建立了联接 recvfrom()函数都能通过 socket 接收数据。recvfrom()返回后,from 中存放了数据来源方的地址信息,包括 IP 地址和端口号。recvfrom ()函数返回接收到的字节数或当出现错误时返回 ERROR。

2.5 结束传输

当所有的数据操作结束以后,就可以调用 close()函数来释放该 socket,从而停止在该 socket 上的任何数据操作:

```
close(sockfd);
```

3 VxWorks 网络通信机制

VxWorks 作为应用最为广泛的实时操作系统,率先集成了 TCP/IP 网络协议栈,利用 VxWorks 对 socket 的良好支持,可以方便地实现网络通信。这不仅使嵌入式系统与 PC 或工作站之间的网络通信成为可能,同时也丰富了系统的实时配置和调试方法,从而进一步扩展了 VxWorks 的功能和灵活性。

网络是 VxWorks 系统之间以及其与其它系统联系的主要途径,在 VxWorks 网络结构的最底层通常使用以太网作为传输媒介,而在传输媒介的上一层则使用 TCP 和 UDP 协议。在网络应用中,通信的两个任务间的常见模式是客户机/服务器模式,即客户先向服务器提出服务请求,服务器接收到请求后提供相应的服务。

VxWorks 为用户提供了 2 种 socket,即流式 socket (SOCK_STREAM)和数据报式 socket(SOCK_DGRAM)。前者定义了一种可靠的面向连接的服务,实现了双向的有序的无重复的数据传输。后者则定义了一种无连接的服务,数据通过相互独立的报文进行传输,是无序的,而且不保证数据的可靠和无差错。

面向连接的服务器首先调用 socket 函数建立流式 socket,然后用 bind 函数将此 socket 和本地地址和端口绑定,调用 listen 函数准备接收客户端的连接,然后调用 accept 函数接受。当接收到客户端的请求后,则连接建立,accept 函数返回新的 socket 描述符,就可以在这个新的 socket 上进行数据的传输,原来的 socket 则可以继续通过 accept 函数调用等待另一个连接。

4 实现代码

根据以上论述,笔者实现了在 VxWorks 下的 TCP 服务器端的编程,部分程序代码如下:

```
STATUS tcpServer(void)
{
    struct sockaddr_in serverAddr; /* 服务器端 socket 地址 */
    struct sockaddr_in clientAddr; /* 客户端 socket 地址 */
    int sockAddrSize;
    int sFd; /* socket 描述符 */
    int newFd; /*接收后的 socket 描述符 */
    ...
    sockAddrSize = sizeof (struct sockaddr_in);
    bzero ((char *) &serverAddr, sockAddrSize);
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_len = (u_char) sockAddrSize;
    serverAddr.sin_port = htons (SERVER_PORT_NUM); /* 指定端口号 */
    serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);/* 填入本机 IP 地址 */
/* 创建 TCP socket */
    if ((sFd = socket (AF_INET, SOCK_STREAM, 0)) == ERROR)
    {
    perror ("socket");
    }
/*绑定 socket */
    if (bind (sFd, (struct sockaddr *) &serverAddr, sockAddrSize) == ERROR)
   {
    perror ("bind");
   }
/* 创建客户端联接队列 */
if (listen (sFd, SERVER_MAX_CONNECTIONS) == ERROR)
    {
    perror ("listen");
    }
/* 接受一个新的联接请求,并发起一个新的任务来处理相关联接 */
FOREVER
    {
    if ((newFd = accept (sFd, (struct sockaddr *) &clientAddr, &sockAddrSize)) == ERROR)
    {
    perror ("accept");
    continue;
    }
while(1 == flag){
/* 接收数据 */
if ((nRead = recvfrom (newFd,(char *) buf,
       sizeof (buf),
       0,
       (struct sockaddr *) &clientAddr,
        &sockAddrSize)) == ERROR)
    {
    perror ("recvfrom");
    }
if(nRead > 0)
    {
```

```
    …
/ *收到关闭联接命令 将循环标志 flag 置 0 */
    if((nRead = =8) &&((strncmp(buf,"shutdown",8) = =0)))
    {
    sprintf(replyMsg,"The connect has been shut down");
    flag = 0;
    }
    …
/ *数据发送 */
    if (write (newFd, replyMsg, strlen (replyMsg)) = = ERROR)
    {
    perror ("write");
    flag = 0;
    }
    }//if(nRead > 0)
    }//while(flag)
/ *结束数据传输,关闭 socket */
    close (newFd);
    …
    }
}
```

5 结 语

本服务器程序应用在某型船的嵌入式网关上,用于网关的远程在线配置和管理的通信。经证明,通过 VxWorks 提供的 socket 机制,实现了嵌入式系统同 PC 和工作站等不同平台之间的网络通信,效果良好。

参考文献:

[1] Wind River system Inc. VxWorks Programmer's Guide[M]. USA:Atlantic Avenue Alameda, 1999.
[2] Wind River system Inc. VxWorks Network Programmer's Guide[M]. USA:Atlantic Avenue Alameda,1999.
[3] 陈智育. VxWorks 程序开发实践[M]. 北京:人民邮电出版社,2004.
[4] Kenneth D. Reed. TCP/IP 基础[M]. 张文,译. 第 7 版. 北京:电子工业出版社,2003.

(上接第 46 页)

3 结 语

对三相电压型 PWM 整流器在 *dq* 坐标系下的数学模型及其前馈解耦控制策略进行了研究。通过 PSCAD/EMTDC 软件环境下的仿真建模,对 PWM 整流器在单位功率因数整流和有源逆变进行了比较,结果表明,直流侧电流动态性能良好,无功功率接近于 0,能量可以实现双向流动,对 PWM 整流器在电力推进船舶的进一步应用具有现实参考价值。

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

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

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
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