Cortex-A8 平台的 μC/OS-Ⅱ及 LwIP 协议栈的移植与实现

马 涛 白瑞林 石 坚

(江南大学轻工过程先进控制教育部重点实验室 江苏 无锡 214122)

摘 要 提出一种嵌入式实时操作系统设备实现接入以太网功能的方案。利用 TI 公司推出的 TMS320DM8168 DaVinci™数字多媒体处理器的 Cortex-A8 平台进行 μC/OS-Ⅱ和 LwIP 协议栈的移植并实现嵌入式 TCP/IP 网络协议。通过对 Cortex-A8 内核体系结构研究,详细阐述了 μC/OS-Ⅱ和 LwIP 针对 Cortex-A8 的移植要点与 LwIP 的高效邮箱机制的实现并给出部分关键代码。通过系统多任务调度及 Web server 功能应用的测试验证 移植操作系统运行正常,网络通信成功。在嵌入式设备上实现了嵌入式 TCP/IP 协议,使嵌入式实时操作系统设备具备了接入以太网的功能。

关键词 μC/OS-Ⅱ Cortex-A8 移植 LwIP

中图分类号 TP316.2 **文献标识码** A DOI 10.3969/j.issn.1000-386x.2014.01.065

TRANSPLANTATION AND REALISATION OF μC/OS-Ⅱ AND LwIP PROTOCOL STACK ON CORTEX-A8 PLATFORM

Ma Tao Bai Ruilin Shi Jian

(*Key Laboratory of Advanced Process Control for Light Industry (Ministry of Education), Jiangnan University, Wuxi 214122, Jiangsu, China*)

Abstract We propose a scheme which realises the function of accessing to the Ethernet using embedded device with real-time operating system. Cortex-A8 platform of TMS320DM8168 DaVinci™ digital media processor launched by TI is used for the transplantation of μC/OS-Ⅱ and LwIP protocol stack and to implement the embedded TCP/IP network protocol. Through studying the core architecture structure of Cortex-A8, we elaborate on the key points of transplantation of μC/OS-Ⅱ and LwIP for Cortex-A8 and the implementation of LwIP efficient mailbox mechanism, and give some of the key codes. It is verified through the test of system multi-task scheduling and Web server function application that the transplanted operating system runs in order, and the network traffic is success. On embedded devices the embedded TCP/IP protocol has been realised, this makes the embedded real-time operating system device has the function of Ethernet access.

Keywords μC/OS-Ⅱ Cortex-A8 Transplant LwIP

0 引 言

嵌入式实时操作系统与网络的结合日趋紧密,越来越多的嵌人式设备需要实现网络化功能,支持嵌入式设备接入网络已成为嵌入式领域重要的研究方向[1]。μC/OS-Ⅱ是一个完整的、可移植、可固化、可裁剪的占先式实时多任务内核[2]。μC/OS-Ⅱ是开放源码的实时操作系统,但仅仅是实时的任务调度及通信内核,缺少对外围设备和接口的充分支持。为了能够使其支持网络就必须进行 TCP/IP 协议移植。LwIP 是 TCP/IP 协议栈的一种实现,是一套用于嵌入式系统的开放源代码的 TCP/IP 协议栈[3]。LwIP 可以通过进程机制来处理通信问题,因此将它移植到 μC/OS-Ⅱ多任务实时操作系统中,可为编写应用程序提供方便,缩短开发周期。

ARM Cortex-A8 处理器是第一款基于下一代 ARMv7 架构的应用处理器,由于使用了 Thumb-2 技术。首次采用强大的 NEON 信号处理扩展集,为 H.264 和 MP3 等媒体编解码提供加速。使用了先进的分支预测技术,具有专用的 NEON 整型和浮点型流水线进行媒体和信号的处理。本文选用 TI 公司的 TMS320DM8168 DaVinci™数字多媒体处理器[4]作为硬件平台,针对 Cortex-A8 的存储器组织和体系结构,对 μC/OS-Ⅱ源码作出相应修改,实现 μC/OS-Ⅱ操作系统在 DM8168 上的移植。并在 μC/OS-Ⅱ操作系统下移植 LwIP 协议,使得在 DM8168 平台上实现嵌入式实时操作系统的网络化需求。

1 μC/OS-Ⅱ移植要点

1.1 μC/OS-Ⅱ移植任务

μC/OS-Ⅱ移植的关键部分是与处理器相关的代码。μC/OS-Ⅱ的软硬件体系结构如图 1 所示,从图中可以看出,与处理器相关的移植代码仅存在于 os_cpu.h、os_cpu_a.asm 以及 os_cpu_c.c 三个文件当中。

收稿日期 2012-09-25。江苏高校优势学科建设工程项目 (PAPD);江苏省科技成果转化项目 (BA2011032)。
马涛,硕士生,主研领域 嵌入式系统与机器人。白瑞林,教授。石坚,硕士生。

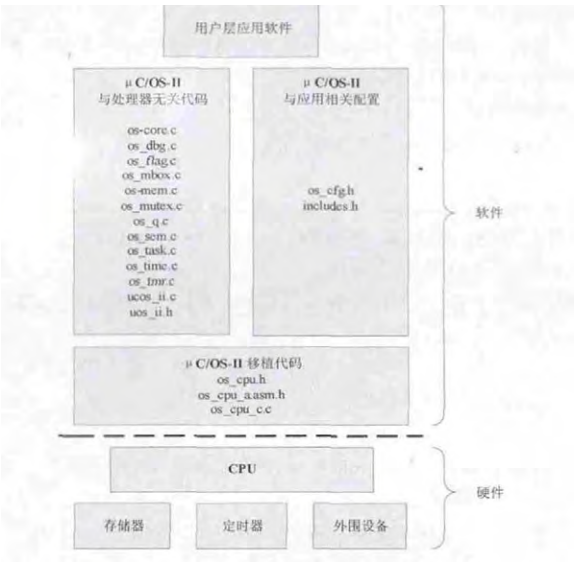

图1 μC/OS-Ⅱ的软硬件体系结构

1.2 μC/OS-Ⅱ移植过程

(1) os_cpu.h 文件编写

首先根据 TMS320DM8168 的 Cortex-A8 内核重新定义数据类型。因为不同的微处理器有不同的字长,所以 μC/OS-Ⅱ的移植包括了一系列的数据类型定义,以确保其可移植性[5]。其次声明开关中断和任务切换的宏,设置常量标识堆栈增长方向。

(2) os_cpu_c.c 文件编写

μC/OS-Ⅱ移植中编写 C 文件时需先确定任务的堆栈结构,任务堆栈结构与内核的体系结构、编译器相关[5]。堆栈需要声明为 OS_STK 类型,并且由连续的内存空间组成。

TMS320DM8168 平台上 Cortex-A8 内核的任务堆栈结构如图 2 所示。

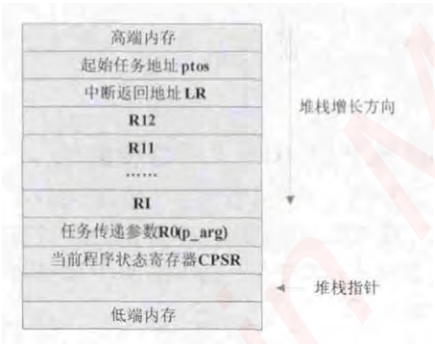

图2 任务堆栈结构

移植时需要实现的 C 函数 堆栈的初始化和钩子函数的实现。最重要的是用来对用户任务堆栈初始化的 OSTaskStkInit ()函数。其他几个 Hook 函数只需进行声明,并不一定要包含任何代码。依据 Cortex-A8 内核的任务堆栈结构,OSTaskStkInit (函数的代码编写如下 :

```
OS_STK * OSTaskStkInit (void *(task)(void * pd), void * p_arg,
OS_STK * ptos, INT16U opt)
{
    OS_STK * stk ;
    opt = opt ;                          /* opt is not used, prevent warning */
    stk = ptos ;                         /* Load stack pointer */
    *(stk) = (OS_STK)task ;              /* Entry Point */
    *(--stk) = (INT32U)0) ;              /* LR */
    *(--stk) = (INT32U)0) ;              /* R12 */
    *(--stk) = (INT32U)0) ;              /* R11 */
    *(--stk) = (INT32U)0) ;              /* R10 */
    *(--stk) = (INT32U)0) ;              /* R9 */
    *(--stk) = (INT32U)0) ;              /* R8 */
    *(--stk) = (INT32U)0) ;              /* R7 */
    *(--stk) = (INT32U)0) ;              /* R6 */
    *(--stk) = (INT32U)0) ;              /* R5 */
    *(--stk) = (INT32U)0) ;              /* R4 */
    *(--stk) = (INT32U)0) ;              /* R3 */
    *(--stk) = (INT32U)0) ;              /* R2 */
    *(--stk) = (INT32U)0) ;              /* R1 */
    *(--stk) = (INT32U)p_arg ;           /* R0 argument */
    *(--stk) = (INT32U)0x00000013L ;
                                         /* CPSR (SVC mode) */
return (stk) ;
}
```

(3) os_cpu_a.asm 文件编写

μC/OS-Ⅱ的移植中在编写 os_cpu_a.asm 文件时需编写 4 个简单的汇编语言函数 OSStartHighRdy ()、OSCtxSw (函数任务级任务切换函数中调用[6] OSIntCtxSw ()中断级任务的切换函数,大多数代码与 OSCtxSw (一样,区别于在 OSIntCtxSw (函数中不需要再保存 CPU 的寄存器 OSTickISR ()提供定时 Tick,调用 OSTimeTick (实现系统切换。

本文使用 CCS 作为编译器,故应注意 CCS 不支持 ldr 伪指令,在声明变量时必须将对应变量的地址保存下来。

(4) 与应用相关的配置代码

μC/OS-Ⅱ的移植中共有两个文件与应用配置相关 os_cfg.h 用来配置内核,设置系统的基本情况 includes.h 系统头文件,包含内核和用户的头文件。用户可根据应用系统来定制合适的内核服务功能。

2 LwIP 基于 μC/OS-Ⅱ的移植

2.1 LwIP 移植任务

LwIP 协议栈在设计时充分考虑了移植问题,体系结构如图 3 所示。可见 LwIP 由相对独立的模块组成 TCP/IP 协议的实现模块 IP、ICMP、UDP、TCP 和相关支持模块[7]。移植的主要工作集中于支持模块 操作系统模拟层的实现。

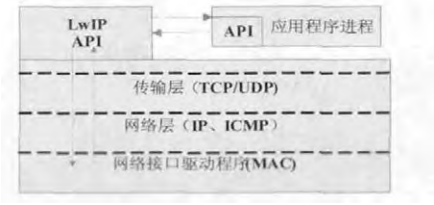

图3 LwIP 的体系结构

2.2 LwIP 移植过程

1) 与 CPU 或编译器相关的 h 文件编写

主要任务是 cc.h、cpu.h、perf.h 中一些与 CPU 或编译器相关的定义。cc.h 定义常用数据类型、同步机制、与编译器相关的 LwIP 结构体封装宏[8]、与平台相关的调试输出。perf.h 定义性能测量使用的宏。需要注意的是 cc.h 定义常用数据类型时将 mem_ptr_t 指定为 INT32U 而不是定义为指针,即 :

```
typedef INT32U mem_ptr_t ;      /* Unsigned 32 bit quantity */
```

2) 与操作系统相关部分的移植

由于 μC/OS-Ⅱ提供了创建任务函数、临界保护函数以及丰

留的信号量和邮箱操作函数，进行一定的修改，便可实现 LwIP 操作系统模拟层的函数。与操作系统相关的结构和函数实现包含在 sys_arch. c 和 sys_arch. h 中：

1）定义操作系统平台需要的数据类型 sys_sem_t，sys_mbox_t，sys_thread_t，sys_prot_t[8]。

2）信号量操作函数的实现 sys_sem_new（）建立并返回一个新的信号量 sys_ sem _free（）释放信号量 sys_ sem _signal（）发送信号量 sys_ arch_sem _wait（）等待由参数 sem 指定的信号并阻塞线程。

3）邮箱操作函数的实现 邮箱一次接收多条消息更高效，邮箱中的消息是指针。本文移植中邮箱的实现机制为 系统同时建立多个邮箱，通过一个单向链表链接在一起。每个邮箱一次可以接收多条消息，接收消息的最大数量由消息数组的大小决定，如图 4 所示。

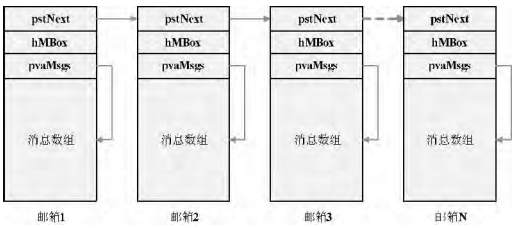

图 4　邮箱实现机制

需要实现的函数 sys_mbox_new（）sys；mbox_free（sys）_；mbox_post（sys）_arch；h_mbox_fetch（。）

4）创建新线程函数的实现：

```
void sys_thread_new (void (* thread )(void* arg ),void * arg );
```

5）系统超时函数的实现：

```
struct sys_timeouts * sys_arch_timeouts (void )
```

//返回目前正处于运行状态的线程所对应的 TIMEOUT 队列指针

3）网络设备驱动程序的实现

在 LwIP 中可以有多个网络接口，每个网络接口都对应了一个 struct netif，包含了相应网络接口的属性、收发函数。在驱动中主要实现的就是网络接口的收、发、初始化以及中断处理函数：

```
void ethernetif_init (struct netif * netif );          //网卡初始化函数
void ethernetif_input (struct netif * netif )//网卡接收函数，从网络接
                //口接收以太网数据报，并把其中的 IP 报文向 IP 层发送
err_t ethernetif_output (struct netif * netif,struct pbuf * p, struct ip_
addr * ipaddr );        //网卡发送函数，给 IP 层传来的 IP 报文加包头
void ethernetif_isr (void );                           //网卡中断处理函数
```

3　移植测试

3.1　μC/OS-Ⅱ移植测试

完成 μC/OS-Ⅱ的移植后，需要验证移植的操作系统是否能够正常运行。为避免复杂化，首先应不添加应用代码，测试 μC/OS-Ⅱ内核自身运行状况。当上述测试步骤成功后。可以运行具体的任务，来进一步验证内核的稳定性和系统性能。

本文通过添加一个多任务调度的应用程序在 CCS 中进行编译、仿真来完成测试。并调用 printf（函数使得在程序运行时，可在 CCS 的 Console 窗口实时观测到系统运行状态。多任务程序主要代码如下：

```
void main (){
EVM816X_init ();
OSInit ();
OSTaskCreate (InitTask, (void * )0,&InitTaskStk [TASK_STACK_
LENGTH - 1],INIT_TASK_PRIO );
OSStart ();
return ;
}
void InitTask (void * pdata ) OSTaskCreate (Task1, (void * )0,
&Task1Stk [TASK_STACK_LENGTH - 1],TASK1_PRIO );
TaskSem = OSSemCreate (0);
while (1) printf ("Task Init..... \n" ); OSTaskDel (OS_PRIO_
SELF );
}
}
void Task1 (void * pdata ) INT8U err ;
OSTaskCreate (Task2, (void* )0,&Task2Stk [TASK_STACK_LENGTH
- 1],TASK2_PRIO );
while (1) OSSemPend (TaskSem, 100, &err ) printf ("Task1
Run..... \n" );
}
}
void Task2 (void * pdata ) {
while (1) printf ("Task2 Run..... \n" )OSSemPost (TaskSem );
}
}
```

程序运行时 CCS 的 Console 窗口显示信息如图 5 所示。由此可验证 μC/OS-Ⅱ工作正常。

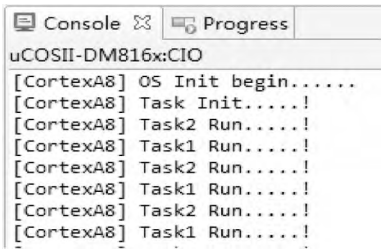

图 5　Console 窗口显示信息

3.2　LwIP 基于 μC/OS-Ⅱ的移植测试

完成 LwIP 在 μC/OS-Ⅱ上的移植后，就可以在 μC/OS-Ⅱ中初始化 LwIP，将 LwIP 作为 μC/OS-Ⅱ的一个任务进行移植测试。以建立工程任务实现一个基本的 Web 服务器功能为例，其任务流程如图 6 所示。

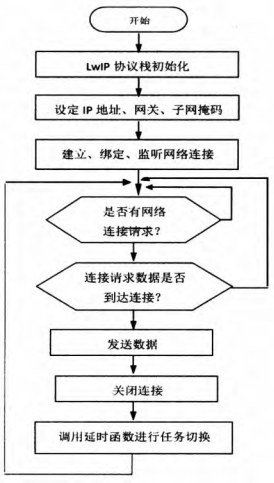

图 6　LwIP 任务流程图

主函数代码如下：

```
void main ( ) {
OSInit ( ) ;
OSTaskCreate (lwip_init_task, &LineNo11, &lwip_init_task [TASK_STACK_LENGTH - 1], INIT_TASK_PRIO ) ;
OSTaskCreate (user_task, &LineNo12, &user_stk [TASK_STACK_LENGTH - 1], INIT_TASK_PRIO + 1 ) ;
OSStart ( ) ;
return ;
}
```

(1) 网络连通功能测试

为了验证 LwIP 能否正常工作把 DM8168 EVM 板的 IP 设置为 172.18.154.1，PC 端 IP 为 172.18.154.176。将工程进行编译并运行后，在 PC 机上的 cmd.exe 中使用 ping 命令测试网络是否连通[10]。结果表明 PC 机网卡发给开发板的 32bit 的数据均成功返回，平均耗时 1ms，测试结果如图 7 所示。证明从 PC 机到嵌入式开发平台的网络是通畅的。

图 7 网络连通验证

(2) 网络服务器功能测试

在程序设计时将 LwIP 协议栈作为一个任务运行在 μC/OS-Ⅱ操作系统上，在这个任务中实现了一个基本的 Web 服务器功能。在 PC 端的 IE 浏览器输入 EVM 板的 IP 地址可浏览开发板提供的网页，如图 8 所示，能够进行正常的网页响应。测试表明 ARP、IP、ICMP、UDP、TCP 协议都已正确运行，由此可进一步验证 ucos-Ⅱ和 LwIP 移植的正确性。

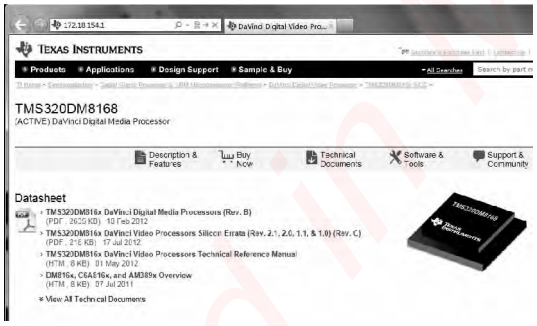

图 8 Web 服务器测试结果

4 结 语

本文详细介绍了利用 TI 公司的 TMS320DM8168 数字多媒体处理器，在 Cortex-A8 平台移植 μC/OS-Ⅱ操作系统的基础上，进一步移植 LwIP 协议栈，使 DM8168 在嵌入式实时操作系统下实现网络化的功能。从而使 TMS320DM8168 得以将其在图像及数据采集和处理方面的优秀能力通过网络功能化，利用其丰富的外围接口，能够充分发挥在音视频采集、网络视频监控等现场，具有一定的实际意义。

参 考 文 献

[1] 苏勇辉. 基于 ARM 微处理器 TCP/IP 协议栈 LwIP 实现[J]. 国外电子测量技术，2009，28(1)：76-79.

[2] Jean J Lbbrosse. 嵌入式实时操作系统 μC/OS-Ⅱ[M]. 2 版. 邵贝贝，等译. 北京：北京航空航天大学出版社，2010.

[3] Dunkel，Adam. Design and Implememation of the LwIP TCP/IP Stack[M]. Swedish Institute of Compumr Science，2001.

[4] TI 公司. TMS320DM816x DaVinci Digital Media Processors Technical Reference Manual[EB/OL]. [2011-04]. www.ti.com.

[5] 王秀霞，孙红艳. μC/OS-Ⅱ在 OMAP5910 上的移植[J]. 电子技术，2010，37(10)：39-42.

[6] 林丽群，刘大茂. μC/OS-Ⅱ在 ARM7 上移植方法的探讨与实现[J]. 现代电子技术，2006，29(18)：47-49.

[7] 杨俊，吕建平，徐峰柳. 基于 μC/OS-Ⅱ和 LwIP 的嵌入式 Web 服务器实现[J]. 电气自动化，2011，33(3)：62-64.

[8] Shang Junyan，Ding Huafeng. Application of lightweight protocol stack LwIP on embedded Ethernet[C]//International Conference on Electrical and Control Engineering(ICECE 2011)，Yichang，China，2011，1.

[9] 向远明，胡健生. 基于 ARM 和 LwIP 的嵌入式以太网接口设计[J]. 计算机光盘软件与应用，2011(15)：171-172.

[10] 邱书波，陈伟. 基于 ARM 的轻量级 TCP/IP 协议栈的研究及移植[J]. 计算机应用与软件，2009，26(8)：90-92.

(上接第 224 页)

参 考 文 献

[1] Wang Y D，Ma X L. 3D facial expression recognition based on encoded templates[C]. 2012 International Symposium on Photonics and Optoelectronics，SOPO 2012，Shanghai，May 21，2012-May 23，2012. IEEE Computer Society，2012.

[2] Zheng H J，Wang T Y，et al. Characteristic parameter extraction of natural quadric surface in STL format for on-machine verification[J]. Journal of Tianjin University Science and Technology，2010，43(4)：293-297.

[3] Sun D Z，Zhu C Z，et al. Research onself-adaptive slicing algorithm for scattered points[J]. Journal of Sichuan University(Engineering Science Edition)，2010，42(1)：216:-219.

[4] 刘云峰，柯映林. 反求工程中切片数据处理及断面特征曲线全局优化技术[J]. 机械工程学报，2006，3(1)：124-129，135.

[5] 柯映林，王青. 反求工程中的点云切片算法研究[J]. 计算机辅助设计与图形学学报，2005，8(1)：1798-1802.

[6] 马淑梅，张涛，李爱平. 基于 SolidWork 的截面草图曲线约束优化技术研究[J]. 现代制造工程，2011，2(5)：55-60，129.

[7] 任朴林，周来水，安鲁陵，等. 基于散乱数据截面线的曲面重构算法研究[J]. 中国制造业信息化，2003，3(8)：82-85.

[8] 刘云峰，柯映林，王秋成，等. 基于特征的反求工程技术研究[J]. 计算机集成制造系统，2006，1(3)：32-37.

[9] 常伟杰，蔡勇，蒋刚，等. 基于支持向量机的点云切片分割技术的研究[J]. 机械，2009，1(1)：16-18.

[10] 朱根松，周天瑞. 反求工程中任意平面切片算法及曲线快速重构[J]. 锻压技术，2008，3(1)：137-140.

[11] 倪小军，姜晓峰，葛亮. 特征保留的点云数据自适应精简算法[J]. 计算机应用与软件，2011，8(3)：38-39，75.

[12] 王慧敏. 反向工程中 NURBS 曲面 CAD 重构技术研究[J]. 计算机应用与软件，2009，1(0)：81-83，238.

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

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

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

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

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

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

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