Windows 驱动程序设计

南京东南大学 0499 硕信箱(210096)　　严仪健
南京邮电学院计算机科学与技术系(210096)　王　钧
南京东南大学无线电系(210096)　邹　乐　孟　桥

摘　要：Windows 驱动程序模型的概念及驱动程序的工作原理，讨论了根据 WDM 模型编制的驱动程序的基本框架和程序各部分的功能。

关键词：Windows 驱动程序模型　I/O 请求包　即插即用

几乎所有的计算机外设在完成了与计算机的电气连接之后，都必须安装合适的驱动程序，这样设备才能正常工作。可以说驱动程序是连接应用程序、硬件以及操作系统的桥梁。目前 PC 机中大量使用的是微软公司的 Windows 系列操作系统，它们不同于以前的 DOS 系统。Windows 采用了各种保护措施来维护系统的稳定与安全，因此编写 Windows 下的驱动程序也变得更为复杂。为方便广大制造商编写其设备在 Windows 下的驱动程序，微软公司提出了 Windows 驱动程序模型 WDM(Windows Driver Model)的概念。

1　驱动程序模型简介

WDM 的概念是随着 Windows for Workgroups 3.10 操作系统的不断完善和改进而形成。在 Windows 98 和刚刚推出的 Windows 2000 中，设备驱动程序必须根据 Windows 驱动程序模型(WDM)设计。

WDM 来源于 Windows NT 式驱动模型，并在其中加入了即插即用(PNP)、总线和类驱动等一些新内容。Windows 驱动程序模型有二个重要方面。首先是核心模型描述设备驱动程序的标准结构，这一结构描述驱动程序的加载以及如何响应用户应用程序的请求和与硬件打交道；其次，Microsoft 为常见类型的设备提供了一系列的总线驱动程序和类驱动程序。这些驱动程序是为符合同样标准或某些通过同种总线接入计算机的一类设备服务的，它使该类设备驱动程序的编写变得简单和方便。用户的驱动程序可以不必直接跟硬件打交道而只需将各种请求经过整理之后传递给下层的总线或类驱动程序，由它们统一与硬件打交道。这样，使设计人员只需根据自己的硬件设备的特点编写有针对性的代码而不必考虑那些烦琐的硬件接口编程，因为这些都由总线或类驱动程序完成了。WDM 的另一个特点是支持即插即用(PNP)，这是对 NT 式驱动模型的一大改进。

2　驱动程序的工作原理及驱动程序的分层设计

用户对计算机的各种设备的使用和操作往往是通过运行相应的应用程序进行，而用户的操作最后却是由驱动程序来处理。其实现过程是：首先应用程序对设备 I/O 进行 Win32 调用，这个调用由 I/O 系统接受；接着由 I/O 管理器根据这个请求构造一个 I/O 请求包 IRP(I/O Request Packet)。例如：应用程序对 CreateFile 的调用最终将产生发送给驱动程序的“创建”IRP(IRP_MJ_CREAT)，而 ReadFile 和 WriteFile 将最终产生“读”IRP(IRP_MJ_READ)和“写”IRP(IRP_MJ_WRITE)。一般情况下，I/O 管理器会直接把 IRP 传递给设备的驱动程序，由驱动程序中的相应例程处理该 IRP。由于 WDM 使用层次式的驱动程序概念，所以必要时驱动程序还要把这个 IRP 传递给驱动程序设备栈中的下一级驱动程序。当然如果当前的驱动程序不适合处理该 IRP，也可以直接把 IRP 向下层传递。在完成了 IRP 的处理后，I/O 管理器把数据和结果返回给 Win32 和用户应用程序。图 1 描述了驱动程序的调用过程。除此之外，驱动程序还要处理系统根据设备状态要求 I/O 管理器发送给设备的一些主功能代码 IRP_MJ_PNP(即插即用 IRP)，实现设备的即插即用，如设备的启动、挂起、停止等。

Windows Driver Model 规范了驱动程序的层次划分，以适应即插即用系统。如图 2 所示，图的左边是 1 个设备对象(device objects)栈，设备对象是系统为便于软件对硬件进行管理、控制而创建的数据结构。最底层的设备对象称为物理设备对象 PDO(physical device object)，栈中间的设备对象称为功能设备对象 FDO(functional device object)，在 FDO 的上层和下层还可能存在过滤设备对象(filter device object)。

系统的即插即用(PnP)管理器根据分层设备驱动程序

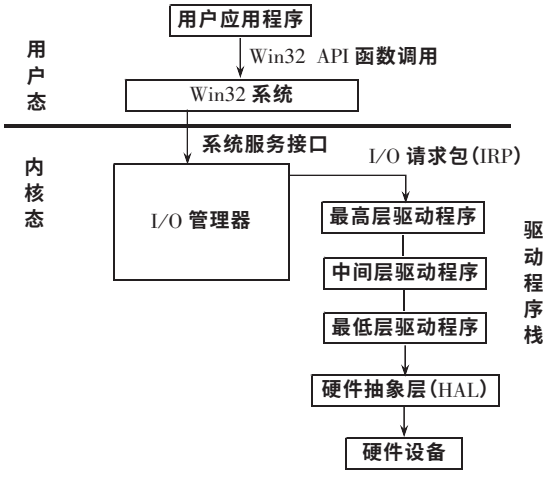

图 1　设备驱动程序的调用

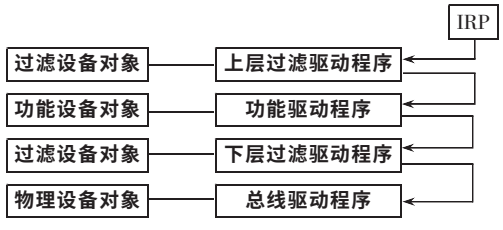

图 2　设备对象和驱动程序的分层

的需要创建了设备对象栈。设备栈表示了处理请求的驱动程序层次，为同一硬件设备服务的不同层次的驱动程序占据设备栈的相应位置以实现各自不同的功能。总线驱动程序负责发现总线上的全部设备，它管理设备与计算机之间的连接，检测设备何时添加和删除。总线驱动会为它发现的设备创建一个 PDO。功能驱动程序知道如何控制设备的主要功能，它在总线驱动程序的上面，并在设备栈中创建 FDO。一般情况下，一旦总线驱动程序接管控制权，它即可以直接访问硬件。但是，也有些设备必须通过它的类驱动程序进行访问，如 USB 设备。

3　WDM 驱动程序的基本结构

WDM 驱动程序根据硬件设备的不同和所要实现功能的区别，其个体间的差异很大。但 WDM 驱动程序的基本框架是每个程序所必须遵守的，有一些例程是每个驱动程序必不可少的。

(1)DriverEntry 例程

每个驱动程序必须有一个“初始化”模块，即必须命名为 DriverEntry 的例程，它有标准的函数原型。正如例程的名字一样，它是驱动程序的初始化入口点，有时一个驱动程序可能要服务于多个同样的硬件设备。而作为全局初始化的 DriverEntry，只需在驱动程序第一次加载时执行一次。

系统内核将各种对硬件的控制和操作转化为 I/O 请求包(IRP)的形式发送到驱动程序，驱动程序根据接受到的 IRP 执行相应的代码完成操作。DriverEntry 的主要工作就是负责设置一系列的回调(CallBack)例程，其中大部分例程的任务是用来处理某个特定的 IRP。常用的 IRP 包括“创建”、“读”、“写”、“关闭”以及 IOCTL IRP 等，处理这些 IRP 的例程往往又称为分发例程。还有一些回调例程用来完成特殊的工作，如 AddDevice 和 StartIo 例程，它们的功能将在后面介绍。

DriverEntry 例程返回 NTSTATUS 类型的变量，表示例程完成的状态，如 STATUS_SUCCESS。以下是一个基本的 DriverEntry 范例，其中还可以根据不同需要添加相应的初始化语句。

```
NTSTATUS DriverEntry( IN PDRIVER_OBJECT
    DriverObject,IN PUNICODE_STRING RegistryPath)
{
    // Export other driver entry points……
    DriverObject->DriverExtension->AddDevice=AddDevice;
    DriverObject->DriverStartIo=DriverStartIo;
    DriverObject->DriverUnload=DriverUnload;

    DriverObject->MajorFunction[IRP_MJ_CREATE]=Dis-
                                        patchCreate;
    DriverObject->MajorFunction[IRP_MJ_CLOSE]=Dis-
                                        patchClose;

    DriverObject->MajorFunction[IRP_MJ_PNP]=DispatchPnp;
    DriverObject->MajorFunction[IRP_MJ_POWER]=Dis-
                                        patchPower;

    DriverObject->MajorFunction[IRP_MJ_READ]=Dis-
                                        patchRead;
    DriverObject->MajorFunction[IRP_MJ_WRITE]=Dis-
                                        patchWrite;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CON-
                        TROL]=DispatchDeviceControl;
    DriverObject->MajorFunction[IRP_MJ_SYSTEM_CON-
                        TROL]=DispatchSystemControl;
    //……Add your initialize codes here;

    NTSTATUS status=STATUS_SUCCESS;
    return status;
}
```

(2)即插即用的实现

即插即用是 WDM 驱动程序所要实现的重要功能。支持即插即用主要是指实现 1 个 AddDevice 例程和 1 个 IRP_MJ_PNP 处理程序。

AddDevice 例程在 DriverEntry 例程中设置，它负责完成设备对象的创建工作，是 WDM 驱动程序中十分重要的环节。在进行了全局初始化并执行完 DriverEntry 例程之后，系统的 PNP 管理器便会调用驱动中的 AddDevice 例程。如果驱动程序驱动的是多个同样的设备，管理器将为每一个设备调用一次 AddDevice。该例程的主要功能是

创建一个设备对象并把它加入设备栈中。具体包括下面几个步骤：

①调用 IoCreateDevice 例程创建设备对象和设备扩展对象的实例。设备扩展结构是驱动程序编写者自己定义的，用来存储一些程序所要用到的额外信息。

②使设备对 Win32 程序可见，以便于应用程序找到设备。具体实现有二种可选的方式：ⓐ为设备命名，即为设备创建符号链接名，应用程序通过该链接名找到设备；ⓑ使用设备接口并用全局唯一描述符(GUID)来标识这个接口，以此方便应用程序找到设备接口。

③对设备扩展进行初始化并设置设备对象的“标记”(Flags)成员变量。

④最后调用 IoAttachDeviceToDeviceStack 函数将新的设备对象加入堆栈中。

IRP_MJ_PNP 处理程序处理的是主功能代码为 IRP_MJ_PNP 的 IRP。每个 IRP_MJ_PNP 还有几个次功能代码，包括 IRP_MN_START_DEVICE(启动设备)、IRP_MN_QUERY_REMOVE_DEVICE(查询删除)、IRP_MN_REMOVE_DEVICE (删除设备)、IRP_MN_STOP_DEVICE(停止设备)等多项关于设备即插即用状态的代码。主 IRP_MJ_PNP 例程可以根据 IRP_MJ_PNP IRP 的次功能代码有选择地执行相应的处理程序，进行适当的操作。它们在编程时往往用 case 语句实现，以使设备在各种状态(启动、挂起、停止、意外删除、删除等)间转换。这其中较基本的次功能有：与 AddDevice 相对应的删除设备、启动和停止设备等例程。

删除设备例程(相应 PNP 次功能代码为 IRP_MN_REMOVE_DEVICE)所做的工作与 AddDevice 正好相反，它调用 IoDeleteDevice 删除该设备对象，而在此之前必须先使用 IoDetachDevice 将设备对象从设备栈脱开。

与 IRP_MN_START_DEVICE 次功能代码对应的是启动设备例程。系统在创建设备后或需要将设备从停止状态启动时会给驱动程序发送“启动设备”的 IRP，由启动设备例程处理。系统会告诉驱动程序它的设备的资源分配，然后驱动程序开始与设备进行相应的对话进行资源配置，并将该 IRP 沿设备栈向下传递。而次功能代码为 IRP_MN_STOP_DEVICE 的停止设备例程将停止设备的工作并释放所占用的资源，以便 PNP 管理器重新分配这些资源。

(3)分发例程的处理

如前所述，Win32 应用程序对设备的访问和各种操作是通过 IRP 传递到驱动程序的。下面就来介绍一下处理应用程序请求所产生的 IRP 的几种主要的分发例程。它们处理打开、关闭、读、写和 IOCTL 请求。

应用程序通常用来访问设备的 Win32 函数及其所产生的 IRP 的主功能代码如表 1 所示。

表 1　Win32 函数与产生的 IRP 对照表

Win32 函数	IRP 主功能代码
CreateFile	IRP_MJ_CREATE
CloseHandle	IRP_MJ_CLOSE
ReadFile	IRP_MJ_READ
WriteFile	IRP_MJ_WRITE
DeviceloControl	IRP_MJ_DEVICE_CONTROL

所有的分发例程有相同的函数原型，均须传递一个指向设备对象的指针和 IRP，同时该函数要返回一个合适的 NTSTATUS 类型的值(如 STATUS_SUCCESS)。

处理器执行任务是有优先级的，优先级通常根据中断来划分，优先级低的任务不能对优先级高的任务请求中断。WDM 驱动程序一般使用 3 个中断级。由低到高分别是 PASSIVE_LEVEL(无中断级)、DISPATCH_LEVEL(软件中断级)以及优先级最高的 DIRQL(硬件中断级)。不同的中断级对驱动程序的编写也做了不同的限定，例如运行于硬件中断级的例程只能访问非分页内存。驱动程序运行的中断优先级非常重要，为了避免在某个例程运行时被其它例程中断，必须明确驱动程序各个例程运行的中断级，有时可以采取必要的措施将某个例程提升到 DIRQL 来运行，以避免被硬件中断。

分发例程通常运行在 PASSIVE_LEVEL，所以它们很容易被内核的其它部分中断，但在这一优先级运行可以对大多数的内核调用。

“创建”、“关闭”、“读”和“写”的 IRP 分别由相应的分发例程来处理。系统调用 DeviceIoControl( )函数可以产生主功能代码为 IRP_MJ_DEVICE_CONTROL 的 IRP，使驱动程序可以通过缓冲区访问技术获取应用程序发出的一组预先定义好的(驱动程序作者定义)IOCTL 命令，实现各种操作。

“创建”和“关闭”分发例程根据各个驱动程序的不同功能分别做一些初始化和设备关闭后的善后工作。对于相应的 IRP 一般采用立即处理的方式。而对于“读”、“写”以及 IOCTL 等 IRP 的处理，虽然也可以立即处理，但为了程序的安全和稳定，目前采用较多的是串行化处理的方法，即将此类 IRP 放在一个队列中，依次或按一定的优先级顺序进行排队处理。

初学驱动程序编写的读者也许会迫切地想了解如何直接对硬件进行访问，其实串行化对硬件的访问更重要。访问硬件有现成的内核函数可以调用，如 READ_PORT_UCHAR 等，且对于硬件的访问只占 WDM 驱动程序的一小部分；而处理不好对硬件的顺序访问将会严重影响系统的稳定工作。驱动程序要串行化对硬件的访问是因为：同一个设备可能同时被多个应用程序打开，特别是在多处用户、多任务的时候这种情况更易出现，而且由于分发例程在较低的中断优先级下调用，所以也容易被其它的例程中断而造成错误。解决这些问题的方法就是将这些与访问硬件有关的 IRP 放入一个队列中，顺序执行。实际

(接上页)

上在 WDM 驱动程序中，排队后的 IRP 由在 DriverEntry 例程中设定的 StartIo 例程处理，该例程一次一个地处理排队后的 IRP。由于排队的 IRP 往往具有不同的主功能代码，所以，StartIo 例程内部通常都有一个“ case ”语句用于处理不同的 IRP。此时原来处理这些 IRP 的分发例程所要做的仅仅是对 IRP 的初始处理：调用 IoMarkIrpPending( )将当前 IRP 标记为挂起；然后调用 IoStartPacket( )把该 IRP 插入队列中；最后，分发例程返回 STATUS_PENDING，确认该 IRP 被挂起，并将由后续的处理例程来完成该 IRP。当一个 IRP 被插入队列后，如果队列中没有排队的 IRP，该 IRP 被马上发送，在 StartIo 中处理，否则将按先后顺序处理(实际上 IoStartPacket( )可以适当地设定一些关键字优先级以便进行特殊顺序处理)。StartIo( )例程一次只处理一个排队的 IRP，要连续不断地处理 IRP 队列，就必须使 StartIo 循环运行。所以往往在 StartIo 例程中进行 IoStartNextPacket( )内核调用来启动下一个 StartIo。如果队列中还有 IRP 则 IoStartNextPacket( )启动 StartIo 例程进行处理，否则调用很快返回。还需要说明一点，StartIo 例程在 DISPATCH_LEVEL 被调用，所以它不易被中断而且它访问的代码和变量必须放置在非分页内存中。

4 结束语

WDM 是编写 Windows98、Windows2000 驱动程序的基础，它定义了驱动程序的明确框架和驱动程序的层次结构。WDM 的内容包括很多方面，其中许多内容涉及 Windows 的系统内核。总之，学习和掌握 WDM 对于开发硬件设备的 Windows 驱动程序和了解 Windows 内核的工作原理都很有益处。

参考文献

1 Cant C.Windows WDM 设备驱动程序开发指南.北京：机械工业出版社，2000

2 Oney W.Programming the Microsoft Windows Driver Model.微软公司，1999

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
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计
64. 一种新型 MVB 通信板的探究
65. 具有 MVB 接口的输入输出设备的分析
66. 基于 STM32 的 GSM 模块综合应用
67. 基于 ARM7 的 MVB CAN 网关设计
68. 机车车辆的 MVB CAN 总线网关设计
69. 智能变电站冗余网络中 IEEE1588 协议的应用
70. CAN 总线的浅析 CANopen 协议
71. 基于 CANopen 协议实现多电机系统实时控制
72. 以太网时钟同步协议的研究
73. 基于 CANopen 的列车通信网络实现研究
74. 基于 SJA1000 的 CAN 总线智能控制系统设计
75. 基于 CANopen 的运动控制单元的设计
76. 基于 STM32F107VC 的 IEEE 1588 精密时钟同步分析与实现

77. 分布式控制系统精确时钟同步技术
78. 基于 IEEE 1588 的时钟同步技术在分布式系统中应用
79. 基于 SJA1000 的 CAN 总线通讯模块的实现
80. 嵌入式设备的精确时钟同步技术的研究与实现
81. 基于 SJA1000 的 CAN 网桥设计
82. 基于 CAN 总线分布式温室监控系统的设计与实现
83. 基于 DSP 的 CANopen 通讯协议的实现

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
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
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计
42. 基于 VXWORKS 的 RS485MVB 网关的设计与实现
43. 实时操作系统 VxWorks 在微机保护中的应用
44. 基于 VxWorks 的多任务程序设计及通信管理
45. 基于 Tilcon 的 VxWorks 图形界面开发技术
46. 嵌入式图形系统 Tilcon 及应用研究
47. 基于 VxWorks 的数据采集与重演软件的图形界面的设计与实现
48. 基于嵌入式的 Tilcon 用户图形界面设计与开发
49. 基于 Tilcon 的交互式多页面的设计
50. 基于 Tilcon 的嵌入式系统人机界面开发技术
51. 基于 Tilcon 的指控系统多任务人机交互软件设计
52. 基于 Tilcon 航海标绘台界面设计
53. 基于 Tornado 和 Tilcon 的嵌入式 GIS 图形编辑软件的开发
54. VxWorks 环境下内存文件系统的应用
55. VxWorks 下的多重定时器设计
56. Freescale 的 MPC8641D 的 VxWorks BSP
57. VxWorks 实验五[时间片轮转调度]
58. 解决 VmWare 下下载大型工程.out 出现 WTX Error 0x100de 的问题
59.

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
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发

48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究
51. 基于缓存竞争优化的 Linux 进程调度策略

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
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计
27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现
28. 基于 XPE 的数字残币兑换工具开发
29. Windows CENET 下 ADC 驱动开发设计

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
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计
25. 基于 PowerPC603e 通用处理模块的设计与实现
26. 嵌入式微机 MPC555 驻留片内监控器的开发与实现
27. 基于 PowerPC 和 DSP 的电能质量在线监测装置的研制
28. 基于 PowerPC 架构多核处理器嵌入式系统硬件设计
29. 基于 PowerPC 的多屏系统设计
30. 基于 PowerPC 的嵌入式 SMP 系统设计

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
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析
34. 基于 ARM S3C6410 的 uboot 分析与移植
35. 基于 S5PC100 移动视频监控终端的设计与实现

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
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化
33. 基于 GPU 的 FDTD 算法
34. 基于 GPU 的瑕疵检测
35. 基于 GPU 通用计算的分析与研究
36. 面向 OpenCL 架构的 GPGPU 量化性能模型
37. 基于 OpenCL 的图像积分图算法优化研究
38. 基于 OpenCL 的均值平移算法在多个众核平台的性能优化研究
39. 基于 OpenCL 的异构系统并行编程
40. 嵌入式系统中热备份双机切换技术研究

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
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案
10. 基于小波变换与偏微分方程的图像分解及边缘检测
11. 基于图像能量的布匹瑕疵检测方法
12. 基于 OpenCL 的拉普拉斯图像增强算法优化研究
13. 异构平台上基于 OpenCL 的 FFT 实现与优化
14. 数据结构考题 - 第 4 章 串
15. 数据结构考题 - 第 4 章 串答案
16.

FPGA / CPLD:

1. 一种基于并行处理器的快速车道线检测系统及 FPGA 实现
2. 基于 FPGA 和 DSP 的 DBF 实现
3. 高速浮点运算单元的 FPGA 实现
4. DLMS 算法的脉动阵结构设计及 FPGA 实现
5. 一种基于 FPGA 的 3DES 加密算法实现
6. 可编程 FIR 滤波器的 FPGA 实现
7. 基于 FPGA 的 AES 加密算法的高速实现
8. 基于 FPGA 的精确时钟同步方法
9. 应用分布式算法在 FPGA 平台实现 FIR 低通滤波器
10. 流水线技术在用 FPGA 实现高速 DSP 运算中的应用
11. 基于 FPGA 的 CAN 总线通信节点设计
12. 基于 FPGA 的高速时钟数据恢复电路的实现
13. 基于 FPGA 的高阶高速 FIR 滤波器设计与实现
14. 基于 FPGA 高效实现 FIR 滤波器的研究
15.