·电子机械工程·

Windows2000下数控系统运动控制器的应用研究

黄大贵 ， 李再银

(电子科技大学机械电子工程学院 成都 610054)

【摘要】通过对自行研制的GD102运动控制器特性的研究，将其驱动程序划分为内核支持层、寄存器读写层、基本命令层和应用层；分析了各层次的功能和实现原理及各层次间的通信机制。在分析Windows2000下内核模式驱动程序的构成和I/O请求处理流程的基础上，以DriverWorks为工具开发了运动控制器的驱动程序，并对驱动程序开发中的一些关键问题进行了论述。该驱动程序已被成功地应用到实际开放式数控系统项目，并取得了良好的效果。

关 键 词 驱动程序； I/O请求包； 运动控制器； 开放式数控

中图分类号 TP311 文献标识码 A

Research on Application of the Motion Controller Based on Windows 2000

HUANG Da-gui and LI Zai-yin

(School of Mechatronics Engineering, University of Electronic Science and Technology of China Chengdu 610054)

Abstract Based on modularization, this paper proposes the 4-layer model of GD102 motion controller's driver through research on its characteristics. The four layers are kernel layer, register layer, basic command layer, and application layer. This paper analyses the function and implementation method of each layer. The communications between layers are also referred to. The driver of the motion controller is developed by the tool of DriverWorks through analyzing the composition of the kernel mode driver and the processing of the IRP in Windows2000. Some key problems during processing of program development are discussed in this paper. The driver has been applied in an open numerical control system successfully.

Key words driver; IRP; motion controller; open NC

采用工业PC机和运动控制器构成开放式数控系统是数控技术发展的趋势。运动控制器是开放式数控系统的核心。目前，国内的数控系统大多采用国外的运动控制器或少量的国内产品，成本极高[1]。开发运行可靠、价格低廉的运动控制器对国产数控系统的发展具有重要意义。Windows2000操作系统是开发开放式数控系统的理想平台[2]，因此有必要为GD102运动控制器开发Windows2000下的驱动程序，使它能被应用到基于Windows2000的数控系统中。本文研制的GD102运动控制器能完成两轴控制功能。

1 驱动程序的模块化设计

GD102运动控制器的控制核心为MCX312芯片，其基本指令系统分为数据写命令、数据读命令、驱动命令和插补命令。它提供一套独有的命令操作规则，开发者必须严格按照规则进行各种操作。整个驱动程序的开发就是使用这些命令和规则不断往上扩展的过程。根据芯片的这一特点，本文对驱动程序进行模块划分，如图1所示。

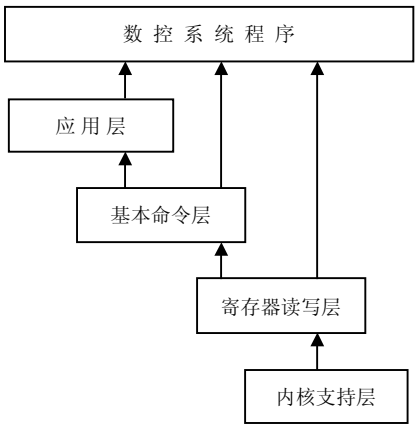

图1 驱动程序层次结构图

收稿日期：2006 – 03 – 15；修回日期：2006 – 09 – 22
基金项目：国家自然科学基金(60302001)
作者简介：黄大贵(1952 –)，男，教授，博士生导师，主要从事智能机电系统方面的研究.

驱动程序分为内核支持层、寄存器读写层、基本命令层和应用层。最底层是一个*.sys文件，上三层被封装到一个动态链接库中。各功能层之间相互独立，下层模块通过接口为上层模块提供服务，上层模块调用下层模块的服务，完成更复杂的功能，最终为用户提供不同级别的调用接口。

1.1 内核支持层

1.1.1 引入内核支持层的必要性

在Windows2000下，软件有内核模式和用户模式两种运行模式。用户程序运行在用户模式，硬件I/O指令不能被直接执行，硬件操作必须通过陷阱门来请求操作系统内核。因此，要完成对运动控制器上寄存器的读写，须编写一个特定的读写模块——内核支持层，并使它运行在内核模式下。

1.1.2 内核支持层的构成

内核支持层在本质上属于一个内核模式的驱动程序，它与一般的应用程序不同。内核模式的驱动程序可以作为被操作系统软件调用的各种例程集合，它本身并不能直接执行，只有当操作系统的I/O管理器调用时，这些例程才会执行。驱动程序的主要例程包括：(1) 当打开和关闭设备时，调用CreateDispatch和CloseDispatch例程。(2) 当驱动程序加载时，调用DriverEntry例程。(3) 当用户程序发出I/O请求时，调用I/O系统服务派遣例程。(4) 当设备开始数据传输时，调用Start I/O例程。(5) 当设备产生中断时，调用中断服务例程ISR；如果需要，调用延迟过程调用例程DPC[3]。

1.1.3 内核支持层的工作原理

内核支持层运行在内核模式下，而运动控制器驱动程序的上三层(dll部分)运行在用户模式，它们依靠Windows2000自有的一套I/O机制来完成通信，如图2所示。驱动程序的寄存器读写层通过Win32 API 函数调用操作系统的服务，这些函数主要为[4]：CreateFile、ReadFile、WriteFile、DeviceIoControl、CloseHandle。在Windows2000下，几乎所有的I/O都是包驱动的，每个工作命令都由一个输入/输出请求包(I/O request packets, IRP)来表示。当操作系统接收到API调用操作后，它的I/O管理器从未分页的系统存储器里分配一个IRP，并将它向下传递给驱动程序的内核支持层。内核支持层根据IRP中提供的信息操纵运动控制器，并将操作的具体结果写入IRP中，传回给I/O管理器。

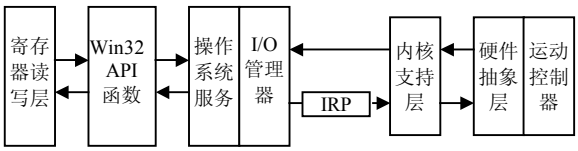

图2 内核支持层的工作原理图

1.1.4 内核支持层的实现[5-7]

用来开发内核模式驱动程序的工具很多，如DDK2000、DriverWorks、Windriver。DriverWorks对DDK的函数进行了封装，开发者只需从其类库中直接选用合适的类或进行简单的派生，并使用其成员函数进行相应的操作，便可快速生成一个内核模式的驱动程序。Windows2000操作系统的设计充分体现了面向对象的特点。与驱动程序相关的核心对象有驱动程序对象、设备对象、中断对象等。DriverWorks对上述对象进行了封装和实现，在开发过程中使用的几个主要类别如下：

(1) KDriver，用于实现驱动程序对象。一个驱动程序对象可以管理由它创建的一个或多个设备对象。当用户程序发出I/O请求时，I/O管理器将该请求发送给管理对应设备对象的驱动程序对象。

(2) KDevice，实现设备对象。设备对象是整个驱动程序的核心，所有的硬件操作都在设备对象中进行。设备对象接收由管理它的驱动程序对象发来的各种操作命令，并对这些命令进行实际的处理。

(3) KResourceRequest和KResourceAssignment，用于系统资源的申请。KResourceRequest用于描述驱动程序中需要使用的各种资源，包括I/O地址、中断号、DMA控制器等。KResourceAssignment用于确认系统实际分配的资源。

(4) KioRange，用于对寄存器进行读写操作。

(5) KIrp，实现I/O请求对象。KIrp类封装了IRP的所有信息，以及关于IRP的各种操作。

(6) KMemory，实现内存对象。内存对象又称为内存描述符表，描述一片虚拟内存，包括起始地址、字节长度等信息。当进行设备直接I/O操作时，I/O管理器将用户缓冲区锁定，并创建一个内存对象描述它。使用KMemory类的MapToSystemSpace函数可以获得虚拟内存在系统空间的地址指针，运行在内核模式的驱动程序，就可以使用该指针直接对用户缓冲区进行访问。

DriverWorks安装成功后，便在Visual C++集成开发环境中生成一个DriverWizard向导。利用该向导可以方便地生成驱动程序的框架代码(类似于MFC的向导)；再根据自身的需要填写相应的代码，便可

完成运动控制器端口读写。内核支持层的主要例程如下：

(1) 分配资源并初始化。

```
NTSTATUS GD_102Device::InitResources(void)
{…KResourceRequest theRequest(Isa, (ULONG) 0, (ULONG) 0); //创建资源申请对象
Add_IoPortRange(&theRequest); //申请I/O端口资源
//确认实际获取的资源并初始化
KResourceAssignment
AssignedIoPortRange(&theRequest,
CmResourceTypePort, 0);
status = m_IoPortRange.Initialize(Isa, 0,
AssignedIoPortRange.Start().QuadPart,
AssignedIoPortRange.Length(),TRUE); …}
```

运动控制器申请ISA总线上的一系列地址，分配给各个寄存器。

(2) StartIo例程。

```
VOID GD_102Device::StartIo(KIrp I)
{…switch (I.MajorFunction())
{ case IRP_MJ_DEVICE_CONTROL: //判断功能代码
switch (I.IoctlCode())
{case GD_102_IOCTL_SETWR0: //判断操作指令
Serial_GD_102_IOCTL_SETWR0_Handler(I); break;
case GD_102_IOCTL_GETRR1:
Serial_GD_102_IOCTL_GETRR1_Handler(I);
break; …}
```

StartIo例程从队列中取出IRP，检查它是否已经被取消；如果没有，则调用它的处理函数。GD_102_IOCTL_SETWR0等宏表明硬件操作的类型，是API函数DeviceIoControl的一个参数，从应用程序传来，然后随该函数的其他参数被I/O管理器一起包装在IRP中。在内核支持层解包时，可根据该宏来确定操作类型，并且连同其他的IRP信息对运动控制器进行读写。

(3) 端口读写例程[8-10]。

在利用DriverWizards生成驱动程序时，可选择用户缓冲区直接访问方式。由于用户程序运行在Windows2000的低2 GB程序地址空间，而内核代码运行在高2 GB空间，必须进行内存映射。当执行设备写操作时，用户输入缓冲区被复制到系统缓冲区，其指针可以通过KIrp ::IoctlBuffer()访问。当执行设备读操作时，用户输出缓冲区被I/O管理器锁定在物理内存，由KMemory类将其映射到系统缓冲区。

```
VOID
GD_102Device::Serial_GD_102_IOCTL_SETWR1_Handler(KIrp I)
{…UCHAR* pBuffer = (UCHAR*)I.IoctlBuffer();
//获取数据缓冲区地址
m_IoPortRange.outb(ByteOffset, pBuffer, Count);
//调用KIoRange的成员函数进行写操作
m_IoPortRange.outb (ByteOffset+1, pBuffer+1,
Count);…}
VOID
GD_102Device::Serial_GD_102_IOCTL_GETRR3_Handler(KIrp I)
{…KMemory Mem(I.Mdl());
//根据内存描述符表(MDL)创建内存对象
PUCHAR pBuffer = (PUCHAR)
Mem.MapToSystemSpace();
//将用户缓冲区映射到系统缓冲区
m_IoPortRange.inb(ByteOffset, pBuffer, Count);
//调用KIoRange的成员函数进行读操作
m_IoPortRange.inb(ByteOffset+1, pBuffer+1,
Count); …}
```

1.2　寄存器读写层

MCX312的寄存器分为写寄存器和读寄存器两类。写寄存器主要用于写入各种命令，设定芯片工作模式和插补模式及需要写入的数据等。读寄存器用于读取芯片的工作状态、外部硬件输入信号、运行过程中的数据信息和出错信息。寄存器读写层通过Win32 API函数DeviceIoControl调用内核支持层的功能，将寄存器的读写独立划分为一层，向用户屏蔽了API函数的调用以及底层硬件操作的细节。因此，用户不必直接调用操作系统的功能，也无须关心程序到底运行在内核模式还是用户模式，只需调用驱动程序dll中的引出函数，便可实现底层的端口读写。

1.3　基本命令层

基本命令层接收应用层传来的基本命令序列，并将它们解释成一系列的寄存器读/写操作，然后传递给寄存器读写层，其结构如图3所示。

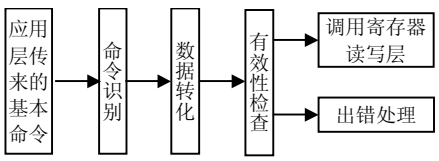

图3　基本命令层的工作过程

应用层传来的基本命令主要包括四类：(1) 数据

写命令，如速度参数设定、插补终点设定、减速点设定、限位寄存器设定等。(2) 数据读命令，如速度/加速度读出、实位计数器读出等。(3) 驱动命令，主要用于单轴的驱动。(4) 插补命令，驱动命令和插补命令只是单个的指令，而非最终提供给用户的插补驱动函数。基本命令层识别应用层传来的各种命令，将其中的数据参数转化成芯片指定的格式，并进行有效性检查。一旦检查成功，便按照相应的规则将命令解释为一系列的寄存器读写操作，交给寄存器读写层去完成。

1.4 应用层

应用层向用户提供最终的功能，主要包括运动控制器初始化与关闭、定量/连续驱动、单段直线/圆弧插补、连续插补、位模式插补等。用户只需在应用层提供的函数中，设定驱动或插补的起点与终点、速度与加速度、加减速方式等参数，便可以实现机床的运动控制。应用层接收来自用户程序的调用，按照相应规则转化为基本命令序列，再交给基本命令层去完成。

2 驱动程序的创建结果

内核支持层被成功地编译和连接后，会生成一个*.sys文件，一般将它放到\WINNT\System32\Driver目录下，然后创建相应的注册表键，使其运行在内核模式下等待调用。驱动程序的上三层被封装到动态连接库中，供应用程序直接调用。驱动程序向用户提供三个层次不同级别的引出函数，用户既可以直接调用最终的应用层函数，直接服务于数控系统，又可以调用低层的基本功能，对运动控制器进行再次开发，以满足特殊的需求。

3 结 束 语

本文阐述了Windows2000下运动控制器驱动程序的开发过程；提出了驱动程序分层模块化的设计思想，并对内核模式下的端口读写进行了分析；以Visual C++6.0和DriverWorks为工具，开发出了GD102运动控制器驱动程序。

参 考 文 献

[1] 何 航. Windows2000下开放式数控系统软件设计与研究[D]. 成都: 电子科技大学, 2004.
[2] 张 敏, 王晓明. 基于Windows2000的开放式数控系统中WDM驱动程序的开发[J]. 组合机床与自动化加工技术, 2002, 11: 52-55.
[3] BAKER A, LOZANO J. Windows2000设备驱动程序设计指南[M]. 施 诺, 译. 北京: 机械工业出版社, 2001.
[4] 王峰博, 崔慧娟. WDM设备驱动程序的研究及实现[J]. 计算机应用, 2003, 23(6): 98-100.
[5] 武安河, 邰 铭, 于洪涛. Windows2000/XP WDM设备驱动程序开发[M]. 北京: 电子工业出版社, 2003.
[6] 郭 艳, 苗克坚. Windows2000下WDM驱动程序的研究与开发[J]. 计算机工程, 2006, 22: 266-268.
[7] 刘 伟. 基于WDM模型的USB设备驱动程序开发[J]. 情报杂志, 2006, 5: 78-82.
[8] 李维民, 瞿安连. WDM设备驱动程序开发中的若干问题[J]. 微计算机信息, 2005, 15: 123-125.
[9] 彭寿全, 宋 杰, 严海锦. UNIX设备驱动程序的剖析与实例[J]. 电子科技大学学报, 1998, 27(1): 78-82.
[10] 李 盛, 张 扬. 嵌入式通信设备驱动程序设计标准化. 电子科技大学学报, 2005, 34(3): 355-358.

编 辑 黄 莘

(上接第250页)

[5] WILLEMS D A. Multifunction small-signal chip set for transmit/Receiver modules[J]. IEEE MTT, 1990, 38(12): 2007-2015.
[6] 李 超. Ka频段集成前端[D]. 成都: 电子科技大学, 2001.
[7] LAVEDAN L J. Design of waveguide-to-microstrip transitions specially suited to millimeter application[J]. Elctron Lett, 1977, 13(20): 604-605.
[8] SHIH Y C, TON T N, BUI L Q. Waveguide-to-microstrip transitions for millimeter-wave appilications[C]//IEEE MTT-S, 1988. [S.l.]: IEEE, 1988.

编 辑 税 红