WIN95 下 DMA 方式编程

徐晓东 杨振坤 （西安交通大学电气学院 710049）

摘要：本文从 DMA 控制器的硬件结构入手，介绍了如何在 Windows95 环境下利用虚拟设备驱动程序实现 DMA 传输。

关键词：Windows95 直接内存存储 虚拟设备驱动程序 VToolsD

1. 引言

Windows95 操作系统的应用越来越广泛，这得益于它的可视化操作功能、强大的内存管理、即插即用等优点，同时它也具有相当的向下兼容性。其对 DOS 应用程序的兼容是通过由保护模式向实模式转换来实现的。但当我们将以前含有硬件处理的 DOS 应用程序移植到 Windows95 环境下时，常常出现程序执行异常或根本无法执行的情况。其原因是 WINDOWS95 采用了一整套新的硬件管理机制（PnP 功能就是基于此而实现的）。

在实时控制过程中，在跟踪监视系统里，在需要大量数据快捷传输的时候，可能会用到 DMA 方式。事实上微机的软盘控制器和普遍使用的声卡都是采用 DMA 操作。本文力图通过对较常用的 DMA 方式编程的介绍来获取对 Windows95 下硬件编程的整体认识。

2. DMA 控制器

在 80386 以上的高性能 PC 机硬件系统中，其核心是由 80x86CPU、80387 数值协处理器、超大规模集成电路的微外部设备 82380，高速缓存存储器 Cache 及其控制器 82385。其中 82380 相当于以前多个附属器件的组合，因此该元件又称集成系统附属器件（integrated system support peripherals）。DMA 控制器就是该器件的功能单元之一，其余还有中断控制器、可编程计时器、DRAM 刷新控制器、等待状态产生器。

82380 内部含有一高速八通道的 32 位 DMA 控制器，可以执行任何由存储器—存储器、存储器—I/O、I/O—存储器、I/O—I/O 的 DMA 传输。其中 03 是低端通道，47 是高端通道。各个通道的操作状态是相互独立的。在启用 DMA 传输时需要设定 DMA 通道号，这时应注意不能和操作系统的缺省硬件设置发生冲突，如一般微机的软盘控制器使用 2 号通道，声卡缺省的是 1 号通

道，若误占了它们，则势必使程序执行异常。

DMA 控制器的结构可分为控制电路和寄存器两部分。控制电路包括 DMA 请求仲裁电路与程序控制，寄存器则主要分控制/形态寄存器和通道寄存器。每一通道均有属于自己的通道寄存器，控制/形态寄存器有两组，通道 03 和 47 各共享一组。其中包括：

·第一控制寄存器 用以将整组的 DMA 通道致能或禁能，设置优先顺序模式。

·第二控制寄存器 设置 DREQn 和 EOP＃输入的检查。

·第一形态寄存器 设置指定通道的传输形态：读取，写入，或检验周期。自动初始设置：致能或禁能。目标地址计数值：递增或递减。数据传输形态：需求式，单一式，数据块式或串接式。

·第二形态寄存器 设置指定通道的要求元件和目标元件类型等。

通过控制/形态寄存器我们可对任一通道进行可能的设置。DMA 还给每个通道分配了一个字节寄存器、一个请求寄存器和一个目标寄存器。字节寄存器含要传输的数据的字节数，该寄存器有 24 位，请求寄存器储存了要求 DMA 服务的存储器或 I/O 的地址，它有 32 位，目标寄存器寄存的是接受 DMA 服务的存储器或 I/O 的地址，也是 32 位。此外，82380 还有一些接口地址，它们在被存取后能使 82380 执行特殊的功能，而实际存取的数据并不重要，关键是写入的动作引起了相应指令的执行。如主清除(master clear)、清除屏蔽暂存器等。

DMA 传输的数据类型可以是字节、字、双字的任意组合。其缓冲区的传输形态有三种：

·单缓冲区(single buffer)

此时 DMA 被设计成传输某一特定的数据块，但下一次传输必须重新设计 DMA 通道。

·缓冲区自动设置(buffer autoinitialize)

这是最为常用的方式，此时同一数据区可在连续的 DMA 传输中重复使用，而不必每次重新设计通道。它要求数据区的物理地址空间必须是连续的。

·缓冲区连锁(buffer chaining)

当使用不连续的数据存储区时，宜采用这种方式。它要求程序先指明一系列要执行的缓冲区传输，通过中断处理程序可以每次重新设计 DMA 控制器，在目前的 DMA 传输完成之后，通道会被自动设计成另一新的缓冲区。

所有的 DMA 传输都涉及三种元件：DMA 控制器、请求者和目标。最常用的 DMA 操作是由 I/O 地址到微机物理内存的数据传输。此时 I/O 为请求者元件，物理存储器是目标元件。可依如下步骤进行：

1. 将 DMA 的通用属性写入两个控制寄存器内。这些通用属性包括优先级、通道组致能、优先顺序模式、以及 DREQn/EOP＃输入的检查。

2. 选定某一通道，并通过两个形态寄存器设置其操作形态：数据和缓冲区的传输形态等。

3. 将符合第二步所选定的操作形态所需的有关内容写入选定通道的字节寄存器、请求寄存器和目标寄存器内。实际写入寄存器的信息以及写入的顺序，依操作形态的不同而异。

4. 设置屏蔽寄存器使新设计好的通道致能。通道即可开始传输数据。

三、虚拟设备驱动程序

Windows95 操作系统接管了微机的所有硬件设备，因此对硬件的编程必须遵循它的规范，在 Windows95 下要实现对中断、DMA、物理地址的访问都要通过虚拟设备驱动程序，即 Virtual Device Driver。它是用来管理系统资源(硬件或软件)的可执行二进制代码。虚拟设备驱动程序一般是以“vxd”为后缀名，VxD 可以动态地加载或卸载，这样就节约了宝贵的系统内存资源。

80386 以上的 CPU 定义了 0、1、2、3 四个特权级，即特权保护。0 是最高特权级，3 是最低特权级。在 Windows95 下，操作系统内核运行于 Ring0 级，一般的应用程序如 Windows 的资源管理器、Microsoft Word 等都是运行在 Ring3 级上。VxD 运行于 Ring0 级上，它在内存中的位置也是处在操作系统保护的空间之内。它在功能上与 DLL(动态链接库)相同，因此可当作运行于 Ring0 级的 DLL。但与一般的 DLL 相比，它的可靠性更高，速度也更快，至少要比 DLL 程序快 23 倍。VxD 的运行都处于称为虚拟机管理程序(VMM)的操作系统部件的监控之下，VMM 位于操作系统的最底层，它的作用是提供虚拟机(硬件)环境、负责多线程占先时间片的调度以及管理虚拟内存等。常规条件下，它对应用程序开发者而言是不可见的。正是 VMM 和 VxD 构成了 Windows95 的 Ring0 级的核心。

虚拟设备驱动程序的生成需要借助 DDK、VToolsD、WinDriver 等开发工具。本程序使用美国 Vireo Software 公司(http://www.vireo.com)的 VToolsD。与传统的设备驱动程序工具 DDK 相比，它既具有较强的开发能力，

又具有较高的开发效率。而 WinDriver 虽然开发效率高，但它的功能有限，不能完全满足特定的需要。VToolsD 包括一个可视化编程的 VxD 代码生成器 QuickVxD、VMM/VxD 服务库、C 运行库、VxD 的 C++ 类库及大量实例。

四、DMA 方式编程

Windows95 下 DMA 方式编程的难点在于：DMA 方式要求直接的物理地址，而我们在应用程序中只能存取 WINDOWS 的虚拟地址。这个问题的解决必须采用设备驱动程序来实现。即在 VxD 中分配一段线性内存，得到其对应物理地址，这就是 I/O 硬件设备所能存取的物理内存，应用程序则通过和 VxD 通信来获取物理地址上的数据。

Windows95 下实现 DMA 传输，首先要向操作系统声明。通过控制面板中的系统选项，进入设备管理子项中的系统设备找到 DMA 控制器，双击后在设置选项选中保留内存，此时可以指定保留的 DMA 缓冲区的大小，同时也可指定是否限定 DMA 向 16 兆以下的内存传送。设置完后重新启动系统，则 WINDOWS 在启动时会预留一段内存空间。还有一种方法是手工改写 system.ini 文件，在其中加入：DMABUFFER = 16（单位为 KB，表示预留 16KB 的内存空间），保存后重新启动即可。

系统保留了 DMA 缓冲区内存后，我们就能在 VxD 中为 DMA 分配一段缓存区了，需要特别指出的是 DMA 缓冲区在分配上存在以下几个限制：

- 物理地址连续。
- 固定和页锁定。
- 64K 边界对齐。
- 必须在 16MB 内存之下，也就是说 DMA 缓存区的地址一定位于 16M 下面。

为满足以上要求，在 VToolsD 中提供了 VDMAD 服务，如 VDMAD-Virtualize-Channel()用于虚拟化 DMA 通道，VDMAD-Request-Buffer()负责申请 DMA 缓冲区等。但在某些情况下，以上调用尤其是分配缓冲区和锁定缓冲区不一定成功。正是基于此种原因，VToolsD 另外提供了两个 DMA 类：VDMAChannel 和 VDMABuffer，利用它们在 VDMAD 调用失败时仍能完成传输。

VDMAChannel 和 VDMABuffer 的类型定义在 vdam.h 头文件中。下面以程序为例介绍它们的用法。该程序实现了利用 DMA3 号通道从 I/O 端口到应用程序的数据传输。

(1)VXD 部分

MyDMA.h 源程序：(略)

(2)应用程序部分

MyDMAAPP.cpp 关键部分代码：

```
DWORD nBytesReturned;
DWORD RetInfo[];
HANDLE hDevice;
long i;
hDevice = CreateFile("\\.\MyDMA.VXD", 0,
0,0,
        CREATE-NEW, FILE-FLAG-DELETE-ON-
CLOSE, 0);
//打开相应的 VXD 文件，VXD 中即执行 OnSysDy-
namicDeviceInit()过程，此处应注意事先将产生的 VXD
文件拷至 WINDOWS\SYSTEM 目录
if (hDevice == INVALID-HANDLE-VALUE)
{
   fprintf("Cannot load VxD, error = %08lx\n",
GetLastError());
   exit(1);
}
i = DeviceIoControl(hDevice, 111, NULL, 0, RetInfo,
sizeof(DWORD), &nBytesReturned, NULL);
//111 为功能号，此调用激发 VxD 中的
OnW32DeviceIoControl()过程，RetInfo[]存放回传数据 if
(i==0) //调用不成功
{
   fprintf("Failed to Communicate\n");
   exit(1);
}
else for(i=0;i<11;i++) //调用成功
   printf("data is %d\n", RetInfo[i]); //显示部分
I/O 传来的数据
}
close(hDevice); //卸载 VxD
```

参考文献

[1] 刘琳、陈红著《80386 硬件与接口技术》北京：学苑出版社 1993

[2] 米东译《Windows95 编程奥秘》北京：电子工业出版社 1996

（来稿时间：1999 年 1 月）