摘 要

数据采集是测试系统的关键环节，提高数据采集系统的性能对于提高整个测试系统的性能有着重要意义。

该数据采集系统利用新型微机总线 PCI 总线解决了数据传输中的瓶颈问题，并且 PCI 总线支持即插即用，便于系统自动分配资源，论文对 PCI 总线的接口问题进行了详细的阐述。系统采用双缓冲技术实现连续数据采集，采用硬件中断保证了数据采集的速度与精度。

作者完成了数据采集系统的硬件设计和软件编程部分以及可编程逻辑器件下载电缆线的制作。系统硬件部分主要包括 A/D 转换、先进先出数据缓存器、控制读写逻辑的可编程逻辑器件和 PCI 总线接口芯片 S5933。软件部分即数据采集卡的驱动程序和应用程序。

本数据采集卡的驱动程序利用 Numega 公司的 VtoolsD2.5 编写，并由 SoftICE4.0 调试。系统采用 DMA 方式传送数据。利用 VC++6.0 编写了一个简单的应用程序来加载该驱动程序并取得所采集的数据。可编程逻辑器件由 Altera 公司的 MAX+PLUS II 编程并仿真，由所制作的并口传输电缆线下载到硬件。

经过努力，该数据采集系统的硬件和软件的性能均达到预期的结果。

由于作者的经验不足和时间的限制，该数据采集系统还存在一些有待于优化的方面，敬请批评指正。

关键词：数据采集；PCI 总线；S5933；A/D 转换；DMA；VxD

Abstract

Data acquisition is the key part of an inspecting system. Improving the performance of the data acquisition system is important for improving the performance of the whole inspecting system.

PCI bus, the new computer bus is used by the author to solve the bottle-neck problem existing in data transfers, also the PCI bus is plugging and playing which is convenient for the operation system to assign resource. The author discusses the PCI bus interface and data transfers mechanism in details. Double-buffer is adopted to ensure gaining the speed an precision of the data acquisition.

The author accomplishes the data acquisition system including hardware design, software programming and making the byteblaster parallel port download cable of the programmable logic device. The hardware consists of A/D converter, first in first out buffers, programmable logic device and PCI bus controller s5933. The software consists of hardware driver and application program.

The driver of the data acquisition is compiled with VtoolsdD 2.5 of Numega Corporation, and debugged with SoftICE 4.0. Data transfer is through DMA mode. A simple program is compiled with VC++ 6.0 to load the driver, and receive the data sampled by A/D converter. The programmable logic device is programmed and simulated with MAX+PLUS II of Altera Corporation, and the program is down loaded to the device through byteblaster parallel port download cable .

The performance of the hardware and software of the data acquisition gains the expected result which due to the hard work of the author and guidance of her teacher.

Because of the insufficiency of the author's experience and the limit of the time, there may be some aspects of the data acquisition system that need optimized, it is my luck to receive the readers' advise that can help me to do the work better.

Key Words: Data Acquisition; PCI Bus; S5933; A/D;DMA; VxD

第 1 章 绪论

1.1 论文选题背景

现代微电子技术、计算机技术、软件技术和网络技术的高度发展及其在电子测量技术与仪器上的应用，新的测试理论、测试方法、测试领域以及新的仪器结构不断出现，为测试仪器的发展提供了强大的推动力[1]。从模拟仪器到数字化仪器，再由智能仪器发展到今天的虚拟仪器，已经完全摆脱了传统仪器独立使用、功能单一、精度低、用户无法改变的模式[2]。虚拟仪器是由软件结合硬件实现的。自从“软件就是仪器”这一口号提出后，虚拟仪器开发厂商与研究人员就尽量减少硬件在仪器中的使用，而采用软件替代这些硬件，例如，数据处理与显示部分完全可以用软件来实现。这样不仅降低了仪器成本，而且减少了硬件难于维护的困难。

虚拟仪器主要包括数据采集与处理及最终结果显示三个模块。数据采集作为虚拟仪器的核心部分，则必须用软件结合硬件实现。其性能的提高不仅依赖于微机总线的性能，而且依赖于一些硬件，如 A/D、采样保持和其它一些逻辑器件的性能，但主要是取决于 A/D 转换器件的性能。

在基于 PC 机的测试系统中，首先要进行数据采集[3]，然后才能对获得的数据进行测试。很多厂商推出了各种通用的数据采集卡，只需要在其前端接上传感器，后端加上后继数字信号处理部分便可使用。这些数据采集卡虽然使用方便，但价格也很高，并且也不一定满足具体的测试过程的需要。随着新的微机总线技术的出现和发展以及 A/D 转换技术的发展，为我们自行研发高性能的数据采集卡提供了的条件。提高高速数据采集卡的性能，关键有两点，一是微机总线的选择，二是在于 A/D 转换芯片[4]。

1.2 关于微机总线的发展

传统的微机总线（如 ISA、EISA 和 MCA 等）由于带宽的限制，已成为制约微机性能的瓶颈，也不能满足高速数据采集与处理的要求。ISA 只有 8

位和 16 位两档，最高传输速率只有 8MB/s；EISA 兼容 ISA，虽然能支持 32 位数据、32 位地址，速率可达 32MHZ，但其成本比较高，它应用与服务器内较多。IBM 公司的微通道（MC）可以认为是标准总线，由于其专利的封闭性而难以广泛流行。

局部总线 PCI（peripheral component interconnect）即外围设备互连的出现解决了这一问题（此外，还有另一种流行比较广泛的局部总线 VESA，但其特点更适用于视频显示信号，而 PCI 总线信号适用性更强）。PCI 总线是 32 位并可升级到 64 位的独立于 CPU 的总线结构，总线速度高达 33/66MHZ，同步控制、猝发（burst）传送使得数据传送速率高达 132MB/sec（32 位总线）、264MB/sec（64 位总线）[5]。线性猝发成组数据传输是 PCI 总线的基本传输机制：一次猝发传输通常由一个地址周期和一个或多个数据周期组成。它解决了总线的速度问题，为 PCI 外设提供了一个高带宽的数据通道，将外设从 I/O 总线上移下来，不需处理器的介入便可进行数据传输。PCI 总线可进行隐式仲裁。

PCI 有三个相互独立的物理空间：存储器地址空间、I/O 地址空间和配置地址空间。配置地址空间是 PCI 所特有的一个物理空间，所有的 PCI 设备必须提供配置地址空间。在桥和配置地址空间的支持下，PCI 提供了功能强大而又方便灵活的配置能力。

由于 PCI 总线协议非常复杂，通常采用两种接口方案来执行 PCI 协议，专用芯片（AMCC 公司的 S593x，PLX 公司的 PCI9052、PCI9054、PCI9050 等，放置于系统或插卡与 PCI 总线之间，提供数据和控制信号的接口电路）和 PCD（ALTERA 公司的 FLEX8000（CPLD），Xilinx 的 XC3100A(FPGA)等，不受插卡功能限制）。利用专用芯片简单方便，使设计者不需在处理系统与 PCI 总线接口的问题上花很多时间[6]。

1.3 A/D 转换技术的发展

在信号的采集与处理中，必须将模拟信号转换为数字信号，这就用到了模/数（A/D）转换技术。现在 A/D 芯片的最高采样速率可达 1GSPS 以上，属于超高速 A/D 转换芯片。常用的超高速 A/D 芯片有 AD9038，采样速率

300MSPS，分辨率 8bit；MAX100，采样速率 250MSPS，分辨率 8bit 等。转换速率低于 60MSPS 的常用高速 A/D 转换芯片有 AD9058，采样速率 50MSPS，分辨率 8bit 等。

A/D 芯片采样速率提高的同时，转换精度也大大提高了。并且近年来兴起的$\sum-\Delta A/D$转换技术能以较低的成本获取高分辨率，使分辨率高达 16、24 位。$\sum-\Delta A/D$转换器以很低的采样分辨率（1）位和很高的采样速率将模拟信号数字化，利用过采样技术、噪声整形和数字滤波技术增加有效分辨率。其内部含有自采样和跟踪电路，不需外加采样保持或跟踪保持电路，从而提高了采样速率，降低了孔径误差。ADI 公司的 A/D7705/6、AD7723/2/1/0 等为典型的 16 位$\sum-\Delta A/D$转换器；典型的 24 位芯片有 AD7714/5/6 等[7]。

1.4 数据采集系统中的数据传输与存储技术

一般低速数据采集系统是通过软件查询和中断方式与主机交换数据，速度较高的系统用 DMA（直接存储）方式传送数据。但是对于过高的采样频率，例如兆级以上，用 CPU 进行控制数据采集是办不到的（系统 I/O 读速度只有几百 K），因此高速数据采集系统在采集数据时对于系统 CPU 必须独立，有些数据采集系统在采集卡上放置处理速度高的单片机（一般用于数据分析），但最终必须满足系统 CPU 对采集数据进行处理和读取的同时，采集系统能够单独采集数据。解决的方案是先将数据存储到系统外部存储器，需要的时候再通过计算机接口电路的控制送往内存或存储器如硬盘，用于数据分析、波形显示及信号处理等[8]。

存储器可以选用高速 FIFO 芯片或 SRAM。SRAM 为静态存储器，需要地址译码，才能将数据写入。FIFO 为先进先出缓存器，不需要与主计算机地址相关的地址就可以存取数据[9, 10]。

1.5 驱动程序开发

由于本次设计的数据采集系统是运行与 Windows98 平台上，所以必须开发 Windows 下的设备驱动程序。在 Windows 平台上，用户如果想控制硬件中断、I/O、DMA 或者访问物理地址都必须通过虚拟设备驱动程序 VxD。虚拟

设备驱动程序是用来管理系统资源（硬件或软件）的可执行的而进制代码，一般以“VxD”为后缀名。VxD 运行于 Ring0 级上，在内存中的地址也是处在操作系统保护空间之内。通常用三种方法开发 VxD：DDK、VtoolsD 和 WDM。

Microsoft 为开发设备驱动程序提供了设备驱动程序工具箱（DDK）。但它提供的许多 VMM 服务都使用寄存器的调用方式，使得要用 32 位汇编代码编写 VxD 极不方便，因为需要对 Wind9x 体系内核结构相当了解[11]。

VtoolsD 是美国 Vireo Software 公司出品的用于 Windows 虚拟设备驱动程序开发的工具包，它包括一个可视化编程的 VxD 代码生成器 QuickVxD、ANSIC 运行库、VMM/VxD 服务库、VxD 的 C++类库、实用工具及大量的实例。所写的代码可以用 Visual C++或 Borland C++的 32 位 C/C++编译器编译。VtoolsD 的类库提供了 VxD 程序的类框架，绝大多数的 VMM 和 VxDs 的服务都可以通过类成员函数来实现。它还通过提供一组附加的类来简化一般的 VxD 编程任务。除了 QuickVxD 之外，VtoolsD 提供的工具还包括 VxDLoad、VxDview 等。VxDLoad 通过命令行方式加载或卸载 VxD，而 VxDview 则可以给出当前系统中所有已加载的 VxD 的状态信息。这两个工具在调试 VxD 的过程中起了重要作用[12]。

1996 年的 WinHEC 会议上，Microsoft 宣布了一种新的 Windows 设备驱动程序模型—Win32 Driver Model（WDM），这种设备驱动程序的模型是 NT4 Kernel Mode Driver + Power Management + Pnp，它是 Windows2000 的核心，驱动程序的后缀名由“.VxD”改为“.SYS”。VxD 最终将被 WDM 取代，因为 Windows 系列与 Windows NT 最终统一起来，VxD 是 Windows 下的设备驱动程序，KMD（Kernel Model Driver）是 Windows NT 下的设备驱动程序，WDM 将这两者统一起来了。

第2章 PCI总线概述

2.1 PCI总线的特点

PCI是先进的高性能的局部总线，可同时支持多组外围设备。PCI局部总线不受制于处理器，为中央处理器及高速外围设备提供一座桥梁，更可控制总线之间的数据传输，提高数据吞吐量。PCI采用高度综合化的局部总线结构，其优化的设计可充分利用今日最先进的微处理器及个人电脑科技，它可确保电脑部件、附加卡及系统间的运作可靠，并能完全兼容现有的ISA/EISA/Micro Channel扩充总线[13]。总之，PCI局部总线具有如下特点：

1 高性能

PCI局部总线以33MHz的时钟频率操作，采用32位数据总线，可支持多组外围部件及附加卡。数据传输速率可高达132MB/s，远远超过标准ISA总线5MB/s的速率。即使在32位的情况下也能支持奔腾（Pentium）级电脑的图形数据传送速率。

2 线性突发传输

PCI支持线性突发的数据传输模式，可确保总线不断满载数据。外围设备一般会由内存某个地址顺序接收数据，这种线性或顺序的寻址方式，意味着可以由某一个地址起读写大量数据，然后每次只需将地址自动加1，便可接收数据流内下一个字节的数据。线性突发传输能够更有效地运用总线的带宽去传送数据，以减少无谓的地址操作。

PCI总线的全部读写传送都可以用突发传送。突发传送的长度由总线主设备决定，在数据传送开始时，目标得到起始地址和交易类型，但没有传送长度。当主设备准备传送每一个数据项时，主设备通知目标是否为最后一个数据项。当最后一个数据项传送后本次数据传送即告结束。

3 极小的存取延误

支持PCI的设备，存取延误很小，能够大幅度减少外围设备取得总线控制权所需的时间。

4 采用总线主控和同步操作

PCI的总线主控和同步操作功能有利于PCI性能的改善。总线主控是大多

数总线都具有的功能，目的是让任何一个具有处理能力的外围设备暂时接管总线，以加速执行高吞吐量、高优先级的任务。PCI 独特的同步操作功能可保证微处理器能够与这些总线主控同时操作，不必等待后者的完成。

5 不受处理器限制

PCI 独立于处理器的结构，形成一种独特的中间缓冲器设计方式，将中央处理器子系统与外围设备分开。用户可随意增添外围设备，以扩充电脑系统而不必担心在不同时钟频率下会导致性能的下降。

6 适合于各种机型

PCI 局部总线不只是为标准的桌面(台式)电脑提供合理的局部总线设计，同时也适用于便携式电脑和服务器。

在服务器环境下，PCI 支持分级式外围设备的特性，可使一个 PCI 界面支持一组级连 PCI 局部总线；也可以使设置为多组 PCI 总线的服务器增添额外的扩展插槽，提供更多 I/O 接口，并将高带宽与低带宽的数据分隔开来。

7 兼容性强

由于 PCI 的设计是要辅助现有的扩展总线标准，因此它与 ISA、EISA 及 MCA 总线完全兼容。虽然现有电脑系统的插槽数目有限，但 PCI 局部总线可提供“共用插槽”，以便接插一个 PCI、ISA、EISA 及 MCA 插头。

8 预留了发展空间

PCI 总线在开发时预留了充足的发展空间，例如，它支持 64 位地址/数据多路复用。PCI 的 64 位延伸设计，可将系统的数据传输速率提高到 264MB/s。

PCI 还提供了自动配置功能，从而保证了用户在安装外围卡时，不需要手工调整跨接线。

9 低成本、高效益

PCI 芯片将大量系统功能高度集中，节省了逻辑电路，耗用较小的电路板，成本降低。PCI 部件采用地址/数据线复用，从而使 PCI 部件用以连接其它部件的引脚数减至 50 以下[14]。

2.2 PCI 总线协议基础

2.2.1 PCI 数据传输机制

PCI 基本总线协议传输机制是猝发成组传输。一个猝发分组由一个地址节拍和一个或多个数据节拍组成。PCI 支持存储器空间和 I/O 空间的猝发传输。

这里的猝发传输是指主桥（位于主处理机和 PCI 总线指间）可以将多个存储器写访问在不产生副作用的前提下合并为一次传输。一个设备通过将基址寄存器的预取位置 1，来表示允许预读数据和合并写数据。一个桥可利用初始化时配置软件所提供的地址范围，来区分哪些地址空间可以合并，哪些不能合并。当遇到要写的后续数据不可预取或者一个对任何范围的读操作时，在缓冲器的数据合并操作必须停止并将以前的合并结果清洗。但其后的写操作，如果在预取范围内，便可与更后面的写操作合并，但无论如何不能与前面合并过的数据合并[15]。

只要处理机发出的一系列写数据（双字）所隐含的地址顺序相同，主桥路总是可以将它们组合成突发数据。

PCI 总线上的所有数据传输基本上都是由以下三条信号线控制的：

FRAME#：由主设备驱动，指明一个数据传输的起始和结束。

IRDY#：由主设备驱动，允许插入等待周期。

TRDY#：由从设备驱动，允许插入等待周期。

当数据有效时，数据资源需要无条件设置 XRDY#信号（写操作为 IRDY#，读操作为 TRDY#）。接受方可以在适当的时间发出它的 XRDY#信号。FRAME#信号有效后的第一个时钟前沿是地址周期的开始，此时传送地址信息和总线命令。下一个时钟前沿开始一个或多个数据周期，每逢 IRDY#和 TRDY#同时有效时，所对应的始终前沿就使数据在主、从设备之间传送，在此期间，可由主设备或从设备分别利用 IRDY#和 TRDY#的无效而插入等待周期。

一旦主设备设置了 IRDY#信号，直到当前数据周期结束为止，主设备不能改变 IRDY#信号和 FRAME#信号。而一个从设备一旦设置了 TRDY#信号或 STOP#信号，就不能改变 DEVSEL#、TRDY#或 STOP#，直到当前的数据周期完成。

当最后一次数据传输时（有时紧接地址周期之后）主设备应撤消 FRAME#信号，而建立 IRDY#信号，表明主设备已做好了最后一次传输数据的准备，待到从设备发出 TRDY#信号后，就说明最后一次数据传输已完成，FRAME#和 IRDY#信号均撤消，接口回到了空闲状态。

2.2.2 PCI 编址

PCI 定义了三个物理空间：存储器地址空间，I/O 地址空间和配置空间。

这些编址是分布式的，每个设备都对自己的地址空间负责。PCI 总线支持正向编码和反向编码两种类型。存储器和 I/O 地址空间为全 32 位地址。在配置地址空间，由 AD[7:2]寻址 64 个双字寄存器，当一条配置指令的地址被译码，IDSEL 有效且 AD[1:0]=00 时，设备判定是否是寻址自己的配置寄存器。如果不是则不理会当前操作。

字节允许用来指出哪一个字节是有效数据，在每个新的数据节拍上，可改变字节允许位，使之对数据的有效性和有效部分进行界定。这一功能称为“字节校正”。

2.3 PCI 总线数据传输

PCI 是地址/数据复用总线，每一个 PCI 总线传送由两个节拍组成：地址节拍和数据节拍。一个地址节拍由 FRAME#信号从非激活状态（高电平）转换到激活状态(低电平)的周期开始。在地址节拍，总线主设备通过 C/BE[3:0]#端发送总线命令，如果总线读命令，紧接着地址节拍的时钟周期叫做总线转换周期，在这一个时钟周期内，AD[3:0]既不被主设备驱动也不被从设备驱动，以避免总线冲突。对于写操作，就没有总线转换周期，总线直接从地址节拍进入数据节拍。

地址节拍的时间是一个 PCI 时钟周期，数据节拍取决于要传送的数据个数，一个数据节拍至少需要一个 PCI 时钟周期，在任何一个数据节拍都可以插入等待周期。FRAME#从有效变为无效表示当前正在进行最后一个数据节拍。

总线操作结束有多种方式，通常由主设备和从设备共同撤消准备信号：TRDY#和 IRDCY#；如果从设备不能够继续传送，可以设置 STOP#信号，表示从设备撤消与总线的连接；所寻址的从设备不存在或者 DEVSEL#信号一直为无效状态都可能导致主设备结束当前总线操作，使 FRAME#和 IRDY#变成无效，回到总线空闲状态。

在存储器指令传送期间，所有从设备都应检查 AD[1:0]，并且提供所要求的猝发顺序，或在第一个数据节拍之后让从设备脱离总线。所有支持猝发的设备都要求线性出发顺序。对于采用高速缓存线触发器例外。在存储器空间，是对于 AD[31:2]译码得到的双字地址进行操作。在线性增加模式下，在每个数据节拍后，地址增加 4 个字节，直到传送结束。

在存储器命令期间，AD[1:0]有如下意义：

AD1	AD0	猝发顺序
0	0	线性增加
0	1	高速缓存线触发器模式
1	X	保留

2.3.1 读操作

当FRAME#有效时，读传送开始，在AD[31:0]上保持有效地址信号，同时C/BE[3:0]#上保持一个总线命令。如果总线命令为0110B（存储器命令），同时AD[31:0]的地址又在目标设备的地址范围内，该设备将置DEVSEL#信号有效（低电平），然后主设备停止驱动AD[31:0]，置IRDY#为低，表明主设备准备好接受数据。第一个数据节拍产生于第三个时钟周期（第二个时钟为转换周期）。在这个数据期内，C/BE[3:0]信号是字节允许，以表示数据总线的哪些字节有效。主设备在接下来的每个时钟周期的上升沿检查TRDY#信号，如果TRDY#为高电平（无效），表示从设备没有准备好，主设备自动插入等待周期，反之，将传送数据，完成一个数据节拍。当主设备使FRAME#从有效变为无效，表示当前是最后一个数据节拍。

2.3.2 写操作

在写传送中，由于地址和数据都是由主设备提供，不存在AD[31:0]切换驱动的问题，所以没有转换周期。除此之外，写传送和读传送类似，数据节拍完成的工作是相同的。

2.3.3 传送终止

总线主设备和从设备都可以终止PCI传送。无论什么原因引起终止，当FRAME#和IRDY#都无效时，所有传送将被终止，进入IDLE周期。

1 总线主设备引起终止

由于某种原因，总线主控设备常常要终止一个PCI传送，最典型的是一次数据传送结束；或者由于总线上具有更高优先级的设备请求占用总线而由总线仲裁器取消当前主设备的总线主控权（移去了GNT#信号）；或者主设备发出了FRAME#信号后在规定的时间内没有检测到目标设备DEVSEL#响应信号，都可能导致终止。

2 从设备引发的终止

从设备可以通过STOP#信号请求总线主控设备终止传送，一旦STOP#信号

有效，必须保持有效直到主设备置 FRAME#无效。IRDY#与 TRDY#之间的关系与 STOP#与 FRAME#之间的关系无关。所以，在从设备请求终止期间（置 STOP#有效），数据仍可以传送，这仅取决于当时 IRDY#和 TRDY#的状态。

从设备可由以下两种原因导致终止：

重入：从设备正处于不能传送数据状态而导致终止；

解除连接：导致从设备解除连接的原因有很多。例如，设备响应速度太慢，从设备可解除连接，让出总线，以允许访问更快的设备；猝发传送中，从设备检测到下一个数据的地址已经超出规定的范围，也可能导致从设备解除连接。

2.4 PCI 总线仲裁机制

PCI 总线主设备要求使用 PCI 总线执行数据传输时，它必须从 PCI 总线仲裁器请求使用总线。每个主设备都有各自的请求先 REQ#和批准线 GNT#。在任意时刻，一个或多个 PCI 总线主设备可能要求使用 PCI 总线，执行数据传送到另一个 PCI 设备。每个发出请求的主设备有效其 REQ#输出，通知总线仲裁器它正在请求总线。总线仲裁器根据系统指定的算法确定哪个请求的设备该获得总线使用权后，有效其 GNT#信号线。

PCI 采用隐式总线仲裁。即在当前主设备正在执行数据传输时，PCI 机理允许总线仲裁发生。如果仲裁器决定将下一次交易的总线所有权授于某个主设备，而不是当前交易的主设备，它从当前主设备取回 GNT#（即先取得）并将之发给总线的下一个所有者，但是，直到当前主设备让总线空闲，下一个所有者才取得总线所有权。隐式总线仲裁不必占用 PCI 总线周期，提高了总线使用效率。但是如果在总线空闲期，就不一定采用隐式仲裁。

PCI 总线仲裁主要利用 REQ#和 GNT#两个信号线实现。仲裁器可以在任何时钟上置某一设备的 GNT#无效。若某一设备要利用 PCI 总线传输数据时，必须确保它的 GNT#信号在此时有效。下面给出 PCI 总线仲裁的协议规则：

(1) 若设置了 GNT#信号无效而 FRAME#有效时，当前的数据传输是合法的且继续进行下去。

(2) 若总线不处于空闲状态，则一个设备 GNT#信号有效和另一个设备的 GNT#信号无效之间必须有一个延迟时钟，否则会在 AD 线上和 PAR 线上出现时序竞争。

(3) 当 FRAME#无效时，为了响应更高优先级主设备的服务，可以在任意时刻设置 GNT#和 REQ#无效。若总线占用者在 GNT#和 REQ#设置后，在处于空闲状态 16 个 PCI 时钟后，仍未开始数据传输，仲裁器允许当前主机“打破”这个状态。仲裁器也可以在任意时刻移去 GNT#，以便服务于一个更高优先级的设备。

2.5 配置空间

为了实现自动配置，每个 PCI 设备必须提供 256 字节的配置空间结构。该空间是具有特定记录结构或模型的地址空间，它又分为头标区和设备有关区两部分。设备在每个区中只实现必要的和与之相配的寄存器。一个设备的配置空间不仅在系统自举时可以访问，在其它时间内也是可以访问的。

配置空间头标区占 64 字节，每个设备都必须支持该区的寄存器分配，该区中的各个字段用来唯一地识别设备，并使设备能以一般方法控制。设备寄存器占 64～255 字节之间，因设备而异。软件可以通过检查 PCI 总线，确定总线上的设备，所以配置软件必须读取每个 PCI 槽位上的设备供应商识别码。如果读取的位置上不存在一个设备，则从宿主总线连到 PCI 桥必须准确无误地报告出来。由于 0FFFH 是一个非法的供应商识别码，所以宿主总线连到 PCI 桥可以返回一个全“1”，作为一个设备的配置空间寄存器的读出值，以表示设备不存在。

所有符合 PCI 要求的从设备都必须支持供应商识别、设备识别、命令和状态字段、其他寄存器的实现是可选的，也就是可以将它们作为保留的寄存器来处理，具体视设备的功能而定。如果一个设备支持的功能是某个寄存器所管辖的，则它必须实现此寄存器，并且要符合规定的位置和功能。

设备状态寄存器用来记录与 PCI 总线有关事件的信息。状态寄存器的读操作与一般寄存器相同，写 1 到某一位可以使该位清零。

配置空间还有等待时间寄存器、内部自检寄存器、指定寄存器、配置空间寄存器操作，它们完善了 PCI 总线的自适应能力，提供了友好的用户界面。

PCI 总线规范为专用设备的初始化和系统启动提供了扩展 ROM 结构。一般情况下，POST 系统对扩充的 PCI 设备和母板上的设备同等对待，只是在处理扩展 ROM 时有例外。

第 3 章 PCI 总线接口芯片 S5933

AMCC 的 S5933 芯片是功能较强 PCI 控制芯片，支持各种层次的接口环境。在最低级别，S5933 可以作为一个简单的 PCI 总线从设备接口。在高层应用，S5933 可以充当系统主设备接口，最大传送速率可达 132MB/S（32 位数据总线）[16-17]。

3.1 S5933 结构概述

3.1.1 S5933 的功能单元

图 3-1 给出了 S5933 内部的主要功能模块，它提供了三个物理总线接口：PCI 总线接口；外加总线接口和可选的 NV 存储器接口。数据传送可以在 PCI 总线与外加总线之间进行，也可以在 PCI 总线与 NV 存储器之间进行。PCI 总线与外加总线的数据传送可以通过信箱寄存器、FIFO 或者 PASS-THRU 通道。使用 FIFO 传送数据，可以由硬件控制也可以由软件控制。在本论文所设计的数据采集系统中，采用了硬件来控制 FIFO 传送数据[18]。

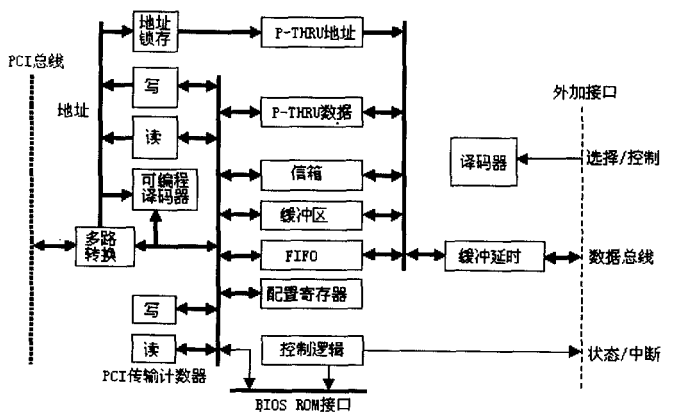


图 3-1 S5933 结构框图

3.1.2 信箱

S5933 的信箱寄存器提供了一个双向数据通道，可以在软件的控制下完成系统平台与外加设备之间的数据传送。这些信箱可以当作用户命令、状态或命令参数寄存器使用，其用途由用户自己定义，当定义的信箱事件发生，可以在 PCI 总线或者外加总线产生中断。

3.1.3 FIFO

在 S5933 的内部有两个独立的 FIFO 数据通道。一个是用于 PCI 到外加数据总线的数据传送（图 3-2），另一个是用于外加总线到 PCI 总线的数据传送（图 3-3）。两个 FIFO 都支持 FIFO 主控。每一个 FIFO 都有一个地址指针和传送计数器以实现 PCI 数据传送。

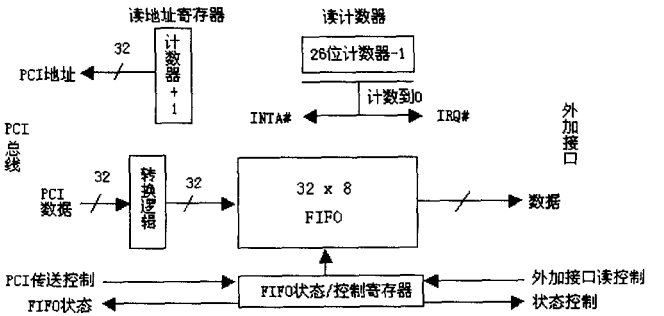


图 3-2 PCI 总线到外加接口的 FIFO 寄存器

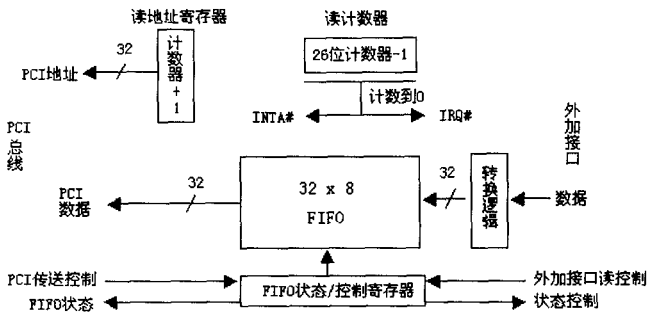


图 3-3 PCI 总线到外加接口 FIFO 寄存器

3.1.4 PASS-THRU

图 3-4 是 PASS-THRU 逻辑结构图。它允许 PCI 传送操作与外加接口操作同时进行。PASS-THRU 方式可以获得计高高的数据传送速率，数据只受限于 PCI 总线和外设的速度。

PASS-THRU 逻辑包含一个地址寄存器和两个数据寄存器（一个数据寄存器对应一个传送方向），并为外加接口提供控相应的制信息[19]。

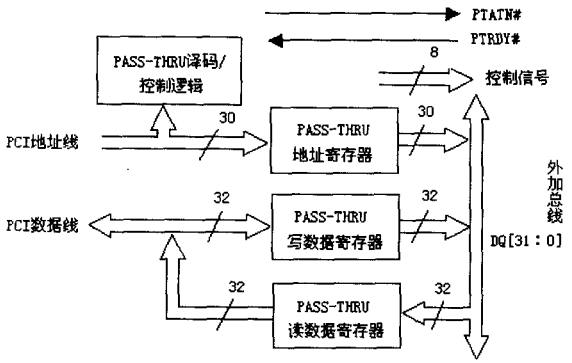


图 3-4 PASS-THRU 逻辑结构

3.2 S5933 引脚信号描述

图 3-5 是 S5933 引脚信号图。S5933 信号符合 PCI 总线 2.1 版，为了方便起见，我们将 S5933 的信息分为三组，即 PCI 接口信号、外加接口信号和 NV 存储器接口信号。PCI 总线接口信号与标准 PCI 接口信号相同，不再赘述。下面介绍外加接口信号和 NV 存储器接口信号。

3.2.1 NV 存储器接口信号

NV 存储器接口信号用于连接外部存储器，它可以连接串行存储器，也可以连接并行存储器，取决于用户的选择。NV 存储器可以在系统初始化时下装 PCI 寄存器的内容，通常将相对固定的参数从 NV 存储器下装，比如 PCI 配置空间的生产厂商和设备 ID 号等。AMCC 公司提供写 NV 存储器的软件，可完全脱离用户程序来写 NV 存储器。

1） 串行存储器接口信号

SCL ：输入，串行时钟输出端。使用并行存储器时，该信号被定义为读选通信号 ERD#。

SDA ：漏极开路，串行地址/数据线。使用并行存储器时，该信号被定义为写选通信号 EWR#。

SNV ：输入，串行 NV 存储器选择信号。为高，表示使用串行存储器；为低，则表示使用并行存储器。

作者设计的 PCI 数据采集系统中采用的是串行 NV 存储器 AT24C02，容量为 256 字节，与 S5933 的接口如图 3-6 所示[20]。

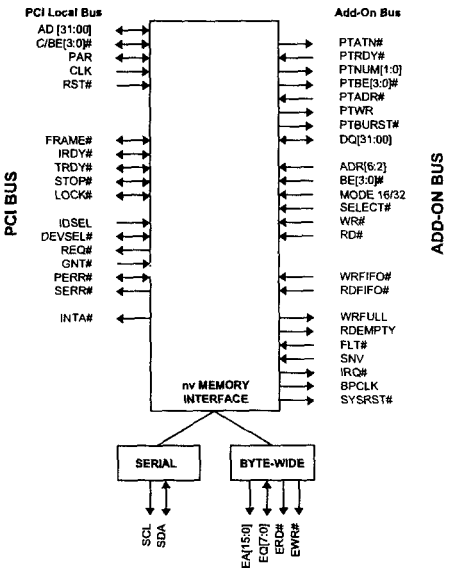


图 3-5 S5933 引脚信号

2）并行存储器接口信号

EA[15:0] ：双向三态信号，外部存储器地址信号端。

ERD# ：输出，外部存储器读选通信号，与 SCL 共用。

EWR# ：双向三态信号，外部存储器写选通信号，与 EWR 共用。

EQ[7:0] ：双向三态信号，外部存储器数据总线。

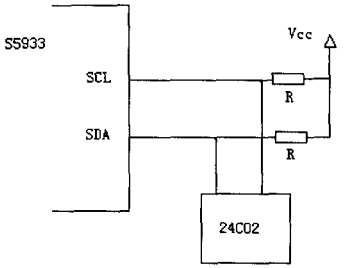


图 3-6 NV 存储器与 S5933 的接口

3.2.2 外加总线接口信号

PTATN# ：输出，PASS-THRU attention 信号，该信号表示一个 PCI 总线周期正在进行，外加接口必须写数据到 PASS-THRU 数据寄存器，或从 PASS-THRU 数据寄存器中读取数据。

PTBURST# ：输出，PASS-THRU 猝发传送，该信号表示当前 PCI 总线操作是猝发方式。

PTRDY# ：输入，PASS-THRU 准备好信号，若有效，表示外加接口已准备好，可进行下一个操作。

PTNUM[1:0]：输出，该信号表示在 PASS-THRU 激活周期，地址落在 4 个基地址的哪一个范围中。

PTBE[3:0]：输出，PASS-THRU 字节允许。

PTADR# ：PASS-THRU 地址请求信号。

PTWR：输出，PASS-THRU 读写控制信号。

3.2.3 寄存器访问控制端

DQ[31:0]：双向三态，与外设的数据总线连接。

ADR[6:2]：输入，外加地址总线，用于选择 16 个内部寄存器。

BE3# or ADR1：字节允许 3（32 位方式）或 ADR1（16 位方式），该信号端与读选通（RD#）或写选通（WR#），SELECT#信号一起使用。

SELECT#：输入，外加接口选择信号。

WR# ：写选通信号，当该信号与 SELECT#共同有效时，写数据到外加接口寄存器。

RD# ：读选通信号，当该信号与 SELECT#共同有效时，从外加接口寄存器读取数据。

MODE：输入，数据总线宽度控制信号，该端为高电平时，外加数据总线为 6 位，如果为低电平，外加数据总线宽度为 32 位。

3.2.4 FIFO 访问控制端

WRFIFO# ：输入，写 FIFO 控制端，它提供了直接访问 FIFO 寄存器的方式，不需要 SELECT#有效和 ADR[6:2]=01000B。

RDFIFO# ：输入，读 FIFO 控制端，直接访问 FIFO 寄存器的方式，不需要 SELECT#有效和 ADR[6:2]=01000B。

WRFULL：输出，写 FIFO 满，表示 FIFO 寄存器满，不能写入新的数据。

RDEMPTY：输出，读 FIFO 空，表示 FIFO 寄存器中没有数据可用。

3.2.5 系统控制信号

SYSRST#：输出，系统复位信号，低电平有效。

BPCLK：输出，缓冲后的 PCI 时钟信号。

IRQ# ：输出，中断请求信号，低电平有效。

FLT# ：输入，浮动，低电平有效。有效时，S5933 的所有输出端将处于高阻状态[21]。

3.3 S5933 内部操作寄存器

S5933 控制器内部包含 16 个双字寄存器，这些寄存器地址可以映像到存储器空间，也可以映像到 I/O 空间。其实地址由 PCI 基地址寄存器 0 的内容决定。

1. 输出信箱寄存器（OMB）

输出信箱寄存器 1~4 提供了一种发送命令和参数到外加系统的方式。PCI 总线对这些寄存器的操作可以是 8 位、16 位或 32 位。当写数据到这些寄存器时，可以在外加接口产生中断请求，中断允许由中断控制/中断状态寄存器的相应控制。

2. 输入信箱寄存器（IMB）

输入信箱寄存器 1~4 提供了一种从外加系统接收数据的方式。PCI 总线对这些寄存器的操作可以是 8 位、16 位或 32 位。当读这些寄存器时，可以在外加接口产生中断请求，中断允许由中断控制/中断状态寄存器相应控制。

3. FIFO 寄存器端口（FIFO）

该寄存器为双向，输入和输出是两个独立的寄存器，它们使用同一端口地址。

4. 主机写地址寄存器（MWAR）

该寄存器用来存放 PCI 写存储器期间的存储器地址。在数据传送过程中，MWAR 寄存器连续更新。

5. 主控写传送计数器（MWTC）

该计数器存放传送数据字节数，计数器值随传送而减少，直到为 0。该计数器的高 6 位总为 0，低 26 位是有效数据，即一次最多传送 64MB 数据。

6. 主控读地址寄存器（MRAR）

该寄存器用来在 PCI 存储器读期间将数据从 PCI 总线传送到外加总线。它的高 30 位是存储器双字地址，低 2 位为 0。

7. 主控读传送计数器（MRTC）

该计数器用来控制读传送字节数，每传送一个数据，计数器内容自动递减，当减到 0 时，将导致中断请求信号，该中断请求信号可以传送到 PCI 总线，也可以传送到外加总线。MRTC 计数器的低 26 位有效。

8. 信箱空/满状态寄存器（MBEF）

该寄存器包含信箱当前状态。高 16 位记录输入信箱状态，低 16 位记录输出信箱状态。如果某一位为 1，表示对应信箱中已经写入数据，还没有被目标接口读取。输入信箱的数据传送方向是从外加总线带 PCI 总线，而输出信箱的数据流向是 PCI 总线到外加总线。

9. 中断控制/中断状态寄存器（INTCSR）

该寄存器控制什么条件下在 PCI 总线上产生中断请求、查看中断产生的条件，以及相应中断的方式。寄存器各位的定义见图。FIFO 管理和控制各位的定义见图 3-7，3-8。

10 总线主控控制/状态寄存器（MCSR）

该寄存器为 S5933 提供全面控制。D7~D0 是状态位，D31~D8 是控制位。寄存器各位定义见图 3-9。

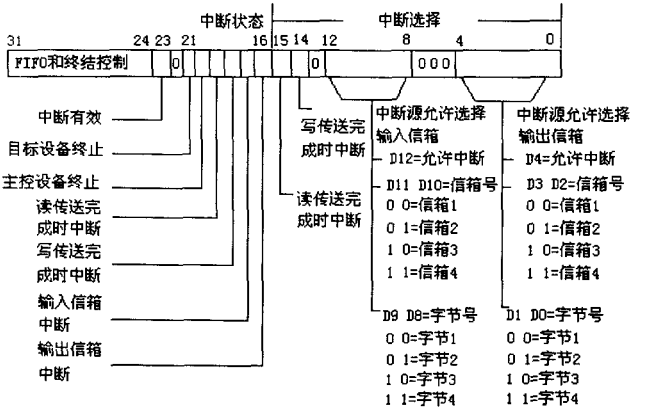


图 3-7 中断控制/状态寄存器

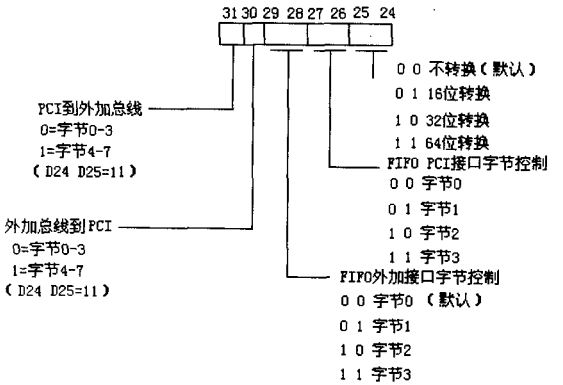


图 3-8 FIFO 管理和控制字节

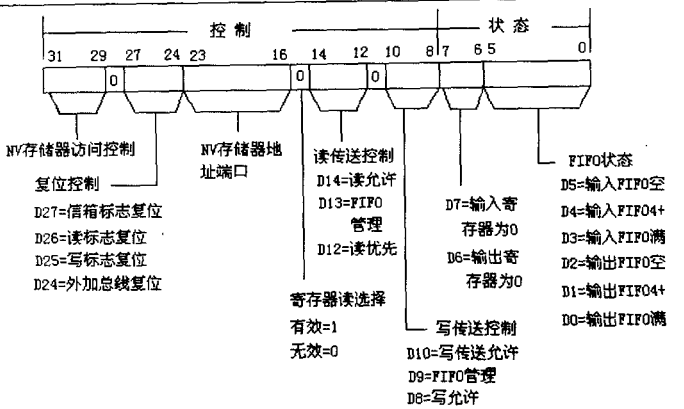


图 3-9 总线主控控制/状态寄存器

3.4 S5933 外加总线操作寄存器

S5933 外加总线接口中有 18 个 32 位寄存器，这些寄存器中包含外加总线接口的数据、控制和状态信息。外加接口使用片选信号 SELECT#和读选通信号 RD#或写选通信号 WR#实现对这些寄存器的访问。使用这些信号可以实现在外加中线的 DMA 操作或者与外部 FIFO 连接。

1. 外加接口输入信箱寄存器（AIMB1~4）

这 4 个寄存器用来从 PCI 总线接收数据、命令和命令参数。外加接口可以使用 8 位、16 位或 32 位宽度读取这些寄存器，当 PCI 中断控制/状态寄存器对应位允许时，读这些寄存器可以导致 PCI 总线中断。

2. 外加接口输出信箱寄存器（AOMB1~4）

这 4 个寄存器用来发送数据、命令和命令参数到 PCI 总线。外加接口可以使用 8 位、16 位或 32 位宽度写这些寄存器，当 PCI 中断控制/状态寄存器对应位允许时，写这些寄存器可以导致 PCI 总线中断。

3. 外加接口 FIFO 寄存器（AFIFO）

对 FIFO 的访问，FIFO 状态可以通过读主控控制/状态寄存器 AGCSTS 查询。

4. 外加接口 PASS-THRU 地址寄存器

该寄存器用于存放 PASS-THRU 方式地址信息。

5. 外加接口 PASS-THRU 数据寄存器（APTD）

在 PASS-THRU 方式，该寄存器存放传送数据，当 PCI 总线周期的地址在 S5933 定义的地址范围内，该器件设置 PTATN#信号有效，直到数据传送结束或被系统主设备终止。

6. 外加接口信箱寄存器（AMBEF）

该寄存器记录信箱中每一个字节单元空/满状态。高 16 位反映输入信箱各字节状态，低 16 位反映输出信箱状态。如果某一位为 1，表示对应字节已经由源接口写入到数据寄存器，还没有被目标接口读取。

7. 外加接口中断控制/中断状态寄存器（AINT）

该寄存器用于设置外加接口中断条件，记录当前外加接口中断状态。

8. 外加接口控制/状态寄存器（AGCSTS）

对外加接口提供如下控制：复位 PCI 到外加接口方向标志；复位外加接口到 PCI 方向标志；复位信箱状态标志；读写外部 NV 存储器。

3.5 S5933 的 FIFO

如上所述，在 S5933 的内部有两个独立的 FIFO 数据通道。一个是用于 PCI 到外加数据总线的数据传送（图 3.2），另一个是用于外加总线到 PCI 总线的数据传送（图 3.3）。使用 FIFO，S5933 可以充当 PCI 总线主控设备接口，也可以充当 PCI 总线的目标设备接口。如果是目标设备接口，允许总线主控设备通过 FIFO 访问外加总线上的数据；S5933 作为总线主控设备接口，允许使用总线地址寄存器和主控字节计数器实现 DMA 传送。

S5933 FIFO 接口的使用非常灵活，其中包括几种不同的 FIFO 管理机制、endian 转换机制，以适应各种外加 CPU、FIFO 指针前移条件使用户可以选择不同的外加总线宽度。

1. 在 PCI 总线主控方式使用 FIFO

如果将 S5933 设置成总线主控方式，则 S5933 可以通过 FIFO 接口启动 PCI 周期。启动一个总线主控传输需要设置相应的地址和传输的字节数，这些参数可以通过 PCI 总线设置，也可以通过外加接口设置。数据传送结束时，S5933 可以在 PCI 总线或外加接口产生中断请求。在数据传输过程中不占用 CPU 资源，提高了系统性能。

1） 从外加接口启动总线主控

如果外部串行存储器的 45H 号单元的 bit7=0，那么只有外加接口可以访问总线主控读地址寄存器（MRAR）、总线主控写地址寄存器（MWAR）、读传送计数器（MRTC）和写传送计数器（MWTC）。如果 S5933 使用并行存储器，外加接口仍不能访问这些寄存器。一旦设定由外加接口启动总线主控传送，S5933 将控制整个数据传送，直到传送计数器为零，也可以不使用传送计数器。从外加接口启动总线主控的前提是外加接口必须有 CPU。

2） 从 PCI 总线启动总线主控

如果外部串行存储器的 45H 号单元的 bit7=1，那么只有 PCI 总线可以访问总线主控读地址寄存器（MRAR）、总线主控写地址寄存器（MWAR）、读传送计数器（MRTC）和写传送计数器（MWTC）。在这种方式下，S5933 传送数据直到计数器变为零，但是不能禁止使用传送计数器。如有外部 NV 存储器，S5933 默认从 PCI 总线启动总线主控传送。

2. 总线接口

S5933 的缓冲器可以从 PCI 总线访问，也可以从外加总线访问，S5933 为外加接口提供了完整的控制信号和状态信号，这样使得外加接口中的 CPU 和可编程逻辑与 FIFO 的接口变得非常简单。

1） FIFO 与 PCI 接口（目标模式）

S5933 的 FIFO 缓冲器可以作为一个标准 PCI 目标。PCI 总线主控可以通过 MCSR 寄存器了解 FIFO 的状态，FIFO 缓冲器在 PCI 操作寄存器占据一个 32 位寄存器位置。在猝发传送的每一个数据节拍，PCI 启动设备将自动增加地址计数器、跟踪当前地址、检测撤消连接事件，这样可以使启动设备在中断位置重新启动传送。

对于 PCI 从外加总线→PCI 的 FIFO 寄存器中读取数据操作，S5933 置 TRDY#信号有效，并完成 PCI 周期，如果 PCI 主机企图从一个空的 FIFO 缓冲器读取数据，S5933 立即宣布撤消连接。对于 PCI 写数据到 PCI→外加总线 FIFO 缓冲器，S5933 设置 TRDY#信号有效结束 PCI 周期，如果 PCI 企图写数据到已满的 FIFO，也将导致 S5933 撤消连接。

2） FIFO PCI 接口（主控模式）

S5933 在 PCI 总线上可以作为总线主控者，允许该器件控制 PCI 总线通过 FIFO 发送和接收数据。S5933 内部的地址寄存器和字节计数器控制数据传

送地址和传送数据块的字节数。

对于 FIFO 读传送（传送数据从 PCI 到外加接口），读操作执行直到下列情况发生：MRTC 寄存器为 0、PCI→外加接口 FIFO 满、PCI 总线仲裁器撤消 S5933 的 GNT#信号或 AMREN=0 无效。如果传送寄存器不为 0、GNT#保持有效、AMREN=1，FIFO 将继续从 PCI 总线读取数据直到 PCI→外加接口 FIFO 满。如果外加总线从 FIFO 提取数据速度与 PCI 总线填充数据的速度相同，可以实现全猝发传送。当 S5933 将最后一个数据填到 FIFO 时，它将撤消 REQ#信号，释放 PCI 总线。只有当 FIFO 管理条件满足时，S5933 才重新设置 REQ#信号，重新请求 PCI 总线。

对于 PCI 写传送（传送数据从外加接口到 PCI），写操作执行直到下列情况发生：MWTC 寄存器为零、外加接口→PCI FIFO 空、PCI 总线仲裁器撤消 S5933 的 GNT#信号或 AMWEN=0 无效。如果传送寄存器不为 0、GNT#保持有效、AMWEN=1，FIFO 将继续写数据到 PCI 总线直到外加总线→PCI 总线 FIFO 缓冲器数据空。如果 PCI 总线从 FIFO 提取数据速度与外加总线填充数据的速度相同，可以实现全猝发传送。当 S5933 将最后一个数据填到 FIFO 时，它将自动撤消 REQ#信号，释放 PCI 总线。只有当 FIFO 管理条件满足时，S5933 才重新设置 REQ#信号，请求 PCI 总线。

3. 配置

FIFO 配置可以在系统初始化阶段和操作期间进行。在初始化阶段定义 FIFO 硬件接口和寄存器访问模式；在操作阶段定义 FIFO 指针的前移条件、endian 转换和总线主控能力。NV 存储器 45H 单元的 bit7 决定是否使用地址寄存器和传送字节计数器控制数据传送。一旦配置信息从 NV 存储器下装，总线主控初始化模式就不能改变。在操作期间，对总线主控地址寄存器和传送字节寄存器的访问不能在 PCI 总线和外加总线间切换。

1）通过 PCI 启动 FIFO 总线主控传送

(1) 定义中断能力：INTCSR bit15=1，允许当读字节计数器到 0 时产生中断；bit14=1，允许当写字节计数器到 0 时产生中断。

(2) 复位 FIFO 标志：MCSR bit26=1，复位外加接口→PCI FIFO 状态标志；bit25=1，复位 PCI→外加接口 FIFO 状态标志。

(3) 定义 FIFO 管理机制：MCSR bit13=1，PCI→外加接口管理机制；bit9=1，外加接口→PCI 管理机制。

(4) 定义读写优先级：MCSR bit12=1，写优先；bit8=1，读优先。

(5) 定义源/目的地址（32 位）：MWAR 总线主控写地址；MRAR 总线主控读地址。

(6) 定义传送字节数：MWTC 总线主控写传送字节数（26 位）；MRTC 总线主控读传送字节数（26 位）。

(7) 允许总线主控：MCSR bit14=1，允许 PCI→外加接口 FIFO 成为总线主控；bit10=1，允许外加接口→PCI FIFO 成为总线主控。

2） 通过外加总线启动 FIFO 总线主控传送

(1) 定义传送计数能力：AGCSTS bit28，允许传送计数。

(2) 定义中断能力：AINT bit15=1，允许当读字节计数器到 0 时产生中断；bit14=1，允许当写字节计数器到 0 时产生中断。

(3) 复位 FIFO 标志：AGCSTS bit26=1，复位 PCI→外加接口 FIFO 状态标志；bit25=1，复位外加接口→PCI FIFO 状态标志。

(4) 定义 FIFO 管理机制：MCSR bit13=1，PCI→外加接口管理机制；bit9=1，外加接口→PCI 管理机制。

(5) 定义读写优先级：MCSR bit12=1，写优先；bit8=1，读优先。

(6) 定义源/目的地址（32 位）：MWAR 总线主控写地址；MRAR 总线主控读地址。

(7) 定义传送字节数：MWTC 总线主控写传送字节数（26 位）；MRTC 总线主控读传送字节数（26 位）。

(8) 允许总线主控：MCSR bit14=1，允许 PCI→外加接口 FIFO 成为总线主控；bit10=1，允许外加接口→PCI FIFO 成为总线主控。

第 4 章 系统电路实现

4.1 系统概述

论文设计的数据采集系统基于 PCI 总线，使用 AMCC 公司的 S5933 实现 PCI 协议的执行。为了简化电路板设计，也为了使该电路板更具有通用性，板上电路不包括一些用户调理电路，如滤波、衰减等，因为这部分电路不需软件支持，可作为一个独立的部分使用。

该电路板是针对噪声测量而设计，因为噪声测量传感器（电容传感器）的频率范围一般为 20~70000Hz，常用的传声器如 CH33 为 20~12500 Hz，故根据香农采样定理（采样频率 f_s 必须大于模拟信号频带的最高频率 f_a（包括噪声在内）的两倍，即 $f_s>2f_a$）而选用了 ADI 公司的 A/D 转换芯片 AD678，它的最高采样频率可达 200KSPS。模拟信号经过 A/D 转换后输出的数据，送入 FIFO 缓存器，由 S5933 控制，通过 DMA 方式送入主机存储区，系统框图如图 4-1 所示，具体电路见附录 1。

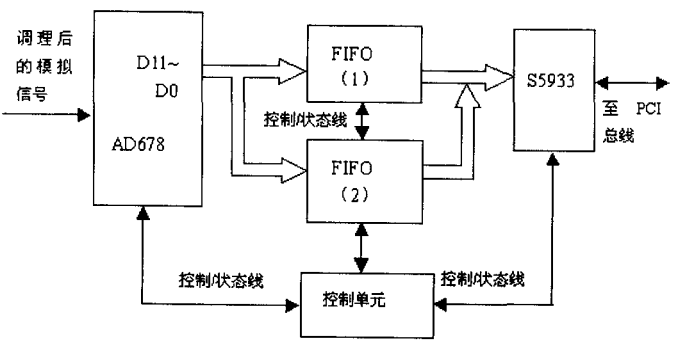


图 4-1 数据采集系统框图

由于 S5933 的 FIFO 数据传送通道控制逻辑简单，可完全用硬件实现，并且传送数据的速度完全满足要求，所以本系统采用 S5933 的 FIFO 通道传送数据。经用户调理后的模拟信号被送入 A/D 转换器，输出 12 位的数据，送入 FIFO 缓冲器，由于 A/D 转换器的转换速度很低，所以该处 FIFO 的主要作用

是实现数据采集的连续性，然后数据被送到 S5933 的内部 FIFO，最后由 DMA 方式将数据读到主机存储区。PCI 接口芯片在前面已有详细的说明，下面将详细讨论该系统电路的其他部分[22-26]。

4.2 A/D 转换器 AD678

本系统采用的 A/D 转换器是 AD678。AD678 是完全的、多功能的 12 模数转换器，它内部包含一个采样/保持放大器，一个与微处理器兼容的总线接口，一个参考电压和时钟发生电路。

AD678 提供了可选的数据接口方式，16 位数据接口既可以通过一次读取 16 位数据操作来实现，也可以通过读两次 8 位数据来实现。数据形式对于无极性模式来说是直接的二进制数据，对于双极性模式来说，是两个二进制的补数。输入电压可达 10V，电压满带宽为 1MHz，满线性带宽为 500kHz。高输入阻抗允许 A/D678 直接与被测信号相接，而不需要经过信号衰减[27]。

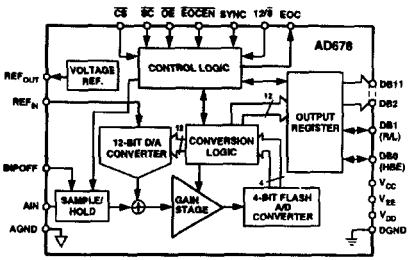


图 4.2 AD678 功能模块图

AD678 的功能模块图如图 4.2 所示，它有两种封装形式，DIP（双列直插式）和 JLCC（J 型引线陶瓷片式载体）。以双列直插式为例，部分引脚说明如表 4.1 所示。

表 4.1 AD678 部分引脚说明

符号	说明
AGND	模拟地。
AIN	模拟信号输入

BIPOFF	双极性补偿。连接到 AGND 时，为无极性 10V 输入，直接二进制数据输出；通过 50Ω电阻连接到 REF_{OUT} 时，为±5V 双极性输入，两个二进制补数输出。
$\overline{CS}$	片选信号，低有效。
DB11-DB4	数字 11 位到 4 位。在 12 位模式，这些引脚为输出数据的高 8 位。在 8 位模式，这些引脚提供了 12 位数据的两个字节。
DB3，DB2	数字位 2 和 3。在 12 位模式，这些引脚为输出数据的 3、2 位。在 8 位模式，它们没被定义，应该连接到 V_{DD}。
DB1（R/$\overline{L}$）	在 12 位模式，该引脚为输出数据的 1 位。8 位模式时，右/左调整。连接到 V_{DD}，右调整；连接到 DGND，左调整输出。
DB0（$\overline{HBE}$）	在 12 位模式，该引脚为输出数据的 0 位。在 8 位模式，高字节使能。如果该位为低电平，输出包含高字节；为高则包含低字节。
EOC	转换结束信号。转换开始时，该信号为低电平；转换结束时，该信号为高电平。在异步电路上，ECO 为漏极开路输出，需要一个 3kΩ的上拉电阻。
$\overline{EOCEN}$	ECO 有效使能信号。
$\overline{OE}$	输出使能。$\overline{OE}$ 下降沿有效 DB11-DB0(12 位模式时)，或 DB11-DB4（位模式时）。与 $\overline{CS}$ 相关。
REF_{IN}	参考输入。+5V 输入提供 10V 极限输入。
REF_{OUT}	+5V 参考输出。通常操作中通过 50Ω电阻连接到 REF_{IN}。
$\overline{SC}$	开始转换，低电平有效。
V_{DD}	+5V 数字电压。
SYNC	SYNC 控制。
12/$\overline{8}$	12/8 位格式。接高电平，则输出为并行 12 位；接低电平，则输出为 8 位多元方式。
V_{CC}	+12V 模拟电压。
V_{EE}	-12V 模拟电压。

AD678 在该数据采集系统中的连接方式如图 4.3 所示。

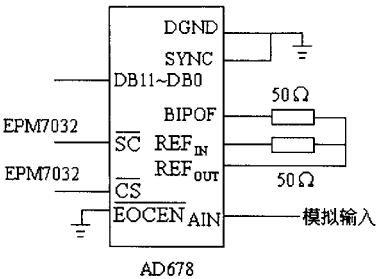


图 4.3 AD678 连接方式

SYNC 端被接到 DGND 端，所以该 AD 转换部分为异步方式，$\overline{SC}$、$\overline{EOCEN}$ 与片选信号 $\overline{CS}$ 无关，$\overline{EOC}$ 为漏极开路输出，它需要 3kΩ 的上拉电阻。

该电路为双极性输入，所以将 BIPOFF 通过 50Ω 电阻连接到 REF_{IN}，REF_{OUT} 也通过 50Ω 电阻连接到 REF_{IN}。

为使用 ECO 控制读写，所以使能 $\overline{EOCEN}$ 端，将其接地。可以通过控制 AD 转换器的的 $\overline{CS}$ 端，来切换 A/D 的工作状态。关于 AD 转换器输出数据的传输将在稍后介绍。

片选信号 $\overline{CS}$ 由 EPM7032 控制，使用 S5933 的 PTNUM[1:0]译码，该处使用的 EMP 内部结构如图 4.4 所示。

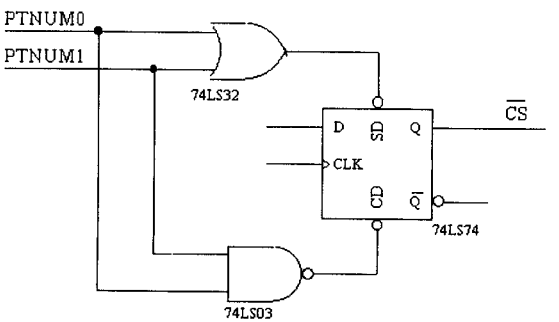


图 4.4 EMP 对 $\overline{CS}$ 的控制

当 PTNUM[1:0]=00 时，74LS74 被清零，Q 端为低电平，所以$\overline{CS}$被有效，开始模/数转换；当 PTNUM[1:0]=11 时，74LS74 被置位，Q 端为高电平，所以$\overline{CS}$无效，结束模/数转换。PTNUM[1:0]=00，01，10，11，分别对应 PCI 基地址空间 1，2，3，4，当对不同的 PTNUM[1:0]空间操作时，PTNUM[1:0]将有不同的输出。所以要得到期望的 PTNUM[1:0]值，只需要对相应的空间做简单的操作即可，如读一个端口。

转换开始信号$\overline{SC}$由可编程逻辑器件 EMP7032 提供一个时钟，该时钟由 33M 的系统时钟经过 8 次分频得到，分频电路由 8 个 D 触发器组成，前一个触发器的输出端 Q 接后一个触发器的时钟端 CLK，第一个触发器的 CLK 端接系统时钟 BPCLK，最后一个触发器的输出端为分频器的输出端，接到$\overline{SC}$端[28]。

AD678 的转换时序图如图 4.5 所示。

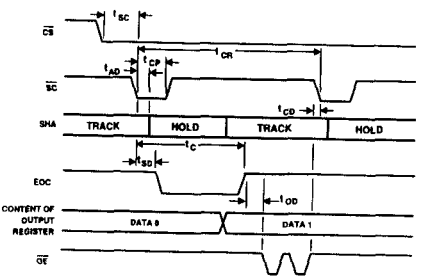


图 4.5 AD678 转换时序图

时间说明：

t_{SC}：$\overline{SC}$延时　　t_{CP}：转换时钟宽度

t_{CR}：转换时间　　t_C：转换时间

t_{AD}：孔径时间　　t_{SD}：状态延时

t_{CD}：转换延时　　t_{OD}：输出延时

4.3 FIFO 缓冲器及其控制

FIFO 是英文“First-In-First-Out”的缩写。在系统设计中，以增加数据传输率、处理大量数据流、匹配具有不同传输率的系统为目标而广泛使用 FIFO

存储器，从而提高了系统性能。FIFO 存储器是以第一个进入其内的数据第一个被移出的形式构成的双口缓冲器，其中一个是存储器的输入口，另一个口是存储器的输出口。对于单片 FIFO 存储器来说，主要有两种结构，主要有两种结构：触发导向结构和零导向传输结构。触发导向传输类型的 FIFO 是由寄存器阵列构成，零导向传输结构的 FIFO 存储器由具有读和写地址指针的双口 RAM 组成[29]。

在本次设计的数据采集系统中采用了两片 IDT72225（1k×18 bit，同步）芯片来缓存数据，进行交替读写数据，实现连续数据采集。外加 FIFO 与 S5933 之间可实现同步猝发数据传送，加快了数据传送速度；另外，若 A/D 转换器件的速度高于 PCI 总线的数据传输速度，因为从 PCI 总线将数据传送到主机存储区的速度总是 A/D 转换器读数据的速度慢，故须先将 A/D 转换器的输出数据存储在 FIFO 芯片中，当达到所需数量后再由计算机读入内存中以进行下一步处理。这样，可以灵活运用 FIFO 芯片构成所需容量、宽度和速度的缓存系统，例如可以进行深度扩展或宽度扩展等。

FIFO 芯片是成系列生产的，由于功能相同，随着存储容量的增加，外部引脚不会变化，这为以后的升级带来方便。例如 IDT72225 的容量为1k×18 bit，IDT72235 为 2k×18 bit，IDT72235 为 4k×18 bit 等[30]。

1 FIFO 芯片说明

FIFO 芯片 IDT72225 为高速同步 FIFO 寄存器，18 位数据输入和 18 位数据输出。数据输入端由写时钟（WCLK）和写使能端（$\overline{WEN}$）控制。当 $\overline{WEN}$ 有效时，数据在时钟的上升沿被写入同步 FIFO。输出端由读时钟（RCLK）和读使能端（$\overline{REN}$）控制，读时钟与写时钟可以同步（单时钟操作），也可异步（双时钟操作）。

IDT72225 有两个确定的状态信号，空标志（$\overline{EF}$）和满标志（$\overline{FF}$）；两个可编程的状态信号，将满标志（$\overline{PAF}$，Almost-Full）和将空标志（$\overline{PAE}$，Almost-Empty）；一个半满信号（$\overline{HF}$）。

IDT72225 有三种封装形式，TOFP、PLCC 和 PGA。

部分引脚说明见表 4.2：

表 4.2 IDT72225 部分引脚说明

符号	说明
D0－D17	18 位数据输入
Q0－Q17	18 位数据输出
$\overline{RS}$	复位信号，低电平有效。在系统加电后，启动写之前

$\overline{WEN}$	写使能端，低电平有效。
WCLK	写时钟输入端。
$\overline{REN}$	读使能端，低电平有效。
RCLK	读时钟输入端。
$\overline{LD}$	有效时，可从偏移和深度寄存器中读写低 12 位数据。
$\overline{FL}$	深度扩展时，先加载的设备 $\overline{FL}$ 端接地，其余接高电平。宽度扩展和单设备时，该端接地。
$\overline{OE}$	数据输出使能端，低电平有效。该端为高电平时，输出数据线处于高阻状态。
$\overline{EF}$	空标志，低电平有效。有效时，FIFO 空，可向 FIFO 写数据，$\overline{EF}$ 为高电平时，FIFO 中有数据，该端与
$\overline{FF}$	满标志，低电平有效。有效时，FIFO 满，不可向 FIFO 其写数据，$\overline{FF}$ 为高电平时，FIFO 中数据未满，该端与 WCLK 同步。

2 接口电路实现

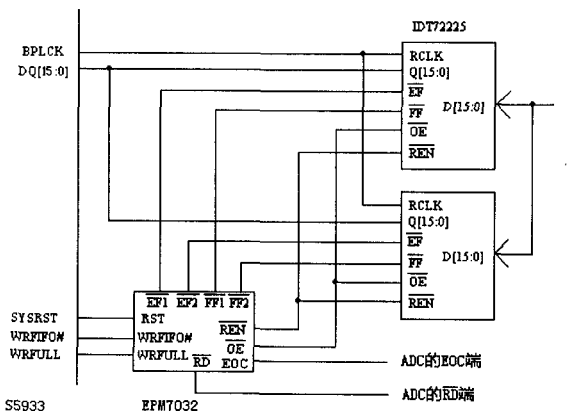


图 4.6 接口电路

S5933 与外加 FIFO 的接口电路如图 1 所示。系统的 A/D 转换器是 12 位的 AD678。每片 FIFO 只用到数据线的低 16 位，即 D[15:0]和 Q[15:0]，并且 FIFO 数据输入的高四位被接地。在读写逻辑单元的控制下，A/D 转换器先向

FIFO1 写数据，若 FIFO1 被写满后，A/D 转换器则向 FIFO2 写数据，同时 FIFO1 转读状态，S5933 将缓存在 FIFO1 的数据读到主机存储区。由于 AD678 的数据转换速度最高只有 200kSPS，在将 FIFO1 读空后，FIFO2 尚未被写满，这期间为等待状态。当 FIFO2 被写满后，将 A/D 转换器切换到向 FIFO1 写数据，同时 FIFO2 进入读状态。所以 FIFO1 和 FIFO2 的读写切换是靠它们的满状态信号 $\overline{FF}$ 来控制完成的。

S5933 的同步 FIFO 数据传送是与 PCI 时钟同步的，所以可以利用板上时钟为外加 FIFO 提供读写时钟。但是接口逻辑必须使 WRFIFO#、RDFIFO#的使能和 DQ[31:0]数据线与 BPCLK 同步。采用 AMCC 公司的 S4402 时钟发生器，可产生多个 BPCLK 时钟的复本，以用来为外加 FIFO 提供读写时钟。从 BPCLK 输入到 BPCLK 的复本输出 S4402 之间的时间间隔低于 1ns。

外加 FIFO 芯片与 S5933 之间通过一片 ALTERA 公司的 7032 实现控制逻辑[31]。7032 的内部逻辑如图 4.7 所示：

在这个数据采集系统中，只需要将数据从外加总线传送到 PCI 总线，所以数据传送只在一个方向上发生，对于 S5933 的控制信号和状态信号，只用到 WRFIFO#和 WRFULL。

在初始状态，FIFO 的复位端被 S5933 的输出复位端有效，FIFO 复位后，使 $\overline{EF}$ 为低，$\overline{FF}$ 为高，即 IDT72225 内部尚无数据。由于 D 触发器 D1 的清零端和 D2 的置位端被连接到 S5933 的输出复位端，这样 FIFO 被复位的同时，D1 被清零，D2 被置位。D1 的 Q 端为低，D2 的 Q 端为高，所以 A/D 转换器的 $\overline{RD}$ 端被有效（低电平），FIFO1 的写使能端 $\overline{WEN1}$ 有效（低电平），此 A/D 转换器向 FIFO1 写数据。由于 D1 的 $\overline{Q}$ 端为高电平，故这时 $\overline{OE1}$、$\overline{REN1}$ 和 WRFIFO1 均无效（高电平）；由于 $\overline{EF2}$ 为低，$\overline{OE2}$、$\overline{REN2}$ 和 WRFIFO2 均无效（高电平），所以 WRFIFO#无效，不能向 S5933 传送数据[32]。

当 FIFO1 被写满时，$\overline{FF1}$ 由高变低，产生一个下降沿脉冲，经过反相器后变为上升沿，输入 D 触发器 D1 的时钟输入端，D1 在时钟上升沿将 D 端的数据传送到 Q 端，Q 端因此变为高，$\overline{Q}$ 为低，$\overline{EF1}$ 早已变高，$\overline{OE1}$、$\overline{REN1}$ 和 WRFIFO1 有效，从而 WRFF 为低电平，AD678 每次转换结束后，EOC 为高电平，WRFIFO#有效（低电平）；$\overline{FF1}$ 变低的同时将 D 触发器 D2 清零，D2 的 Q 端变低，A/D 转换器的读使能端 $\overline{RD}$ 继续被有效，$\overline{WEN1}$ 为高，无效，$\overline{WEN2}$ 为低，有效；$\overline{Q}$ 端为高电平，使 $\overline{OE2}$、$\overline{REN2}$ 和 WRFIFO2 均无效（高电平）。这时 A/D 转换器向 FIFO2 写数据，FIFO1 的数据被读到 S5933 的内部 FIFO，然后通过 DMA 方式被送到主机存储区（S5933 为主设备）。

FIFO1 的数据被读空后，$\overline{EF1}$ 变为低电平，WRFIFO1 无效，从而使 WRFIFO 无效，停止读数据，而 A/D 转换器继续向 FIFO2 写数据，直到 FIFO2

被写满，$\overline{FF2}$ 由高变低，经反相器变为上升沿脉冲，使 D2 触发器的 Q 端变高，$\overline{Q}$ 端为低，此时，$\overline{OE2}$、$\overline{REN2}$ 和 WRFIFO2 有效，$\overline{WEN2}$ 为高，无效，$\overline{WEN1}$ 为低电平，有效，$\overline{RD}$ 继续有效。现在 A/D 转换器向 FIFO1 写数据，S5933 从 FIFO2 读数据。FIFO2 被读空后，$\overline{EF2}$ 为低，使 WRFIFO2 无效，等待 FIFO1 被写满，依次循环。

每次 AD678 转换时，EOC 为低电平，当转换完成后，EOC 都变为高电平，由此产生一个上升沿，经过一个反相器后，变为下降沿，可连接到 AD678 的 $\overline{OE}$ 端，作为 AD678 输出的使能信号；再经过一个反向器后便可作为外部 FIFO 的写时钟 WCLK。

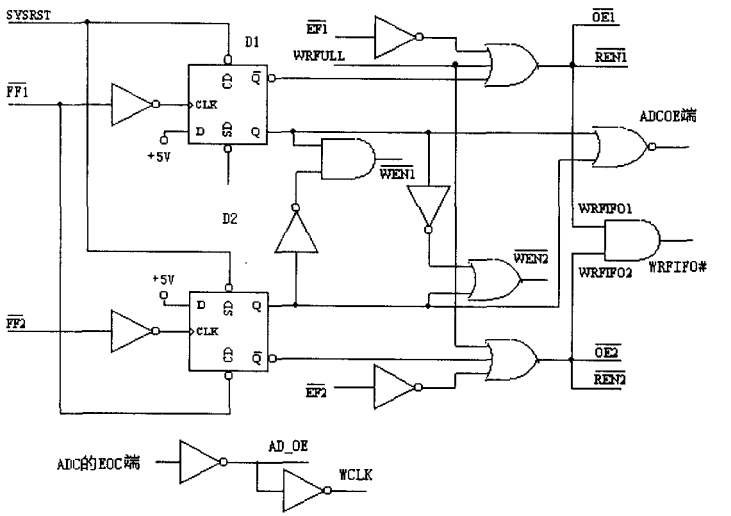


图 4.7 EPM7032 控制读写

若规定长度的数据块采集完毕，由 S5933 向 CPU 申请中断。数据采集在硬件控制单元的控制下自动进行，主机只需要对系统做必要的初始化与数据存储工作，因此提高了数据传输的速度。

图 4.7 仅为读写逻辑的示意图，实际写入可编程逻辑器件的逻辑要做一些相应变化，即必须将接触发器时钟端的控制信号与仅一个与门或或门相关，否则系统将认为该时钟为公共时钟而出错。

EPM7032 属于 ALTERA 公司的 MAX7000 系列可编程逻辑器件，它有 600

EPM7032属于ALTERA公司的MAX7000系列可编程逻辑器件，它有600个可用门，32个宏单元，2个逻辑阵列块，36个用户I/O。对可编程逻辑器件可利用硬件描述语言VHDL/AHDL进行设计。Altera公司的MAX+PLUS II开发系统是一个完全集成化、可以在多个平台上运行的可编程逻辑设计环境[33-35]。

MAX+PLUS II的设计流程由以下几部分组成：

1. 设计输入

MAX+PLUS II的设计输入有图形输入、文本输入和由第三方EDA tool生成的EDIF网表输入等。输入方法不同，生成的文件格式也不同。

2. 编译

MAX+PLUS II编译器可以检查项目中的错误并进行逻辑综合，将项目最终设计结果加载到Altera器件中去，并为模拟和编程产生输出文件。

3. 模拟仿真

设计人员可以通过一个项目来证明所设计的项目功能是否正确。模拟允许把项目在编程到器件之前进行全面检测，以确保它在各种可能的条件下能有正确的响应。在模拟过程中，需要给MAX+PLUS II模拟器提供输入变量，模拟器将利用这些输入信号来产生输出信号（与可编程逻辑器件在同一条件下产生的信号相同）。

根据所需信息的种类，设计人员可用MAX+PLUS II进行功能模拟或时序模拟。功能模拟仅是测试项目的逻辑功能，而时序模拟不仅测试逻辑功能，还测试目标器件最差情况下的时间关系。

4. 器件编程

Altera公司器件的编程方法有许多种，如用计算机及Altera专用编程电缆进行配置、用Altera专用串行EPROM进行配置和用通用EPROM进行配置，可根据具体情况选择使用。

在本数据采集系统中采用的是电缆线配置，因为这种方法配置方便，便于修改。使用的电缆线为并口下载电缆ByteBlaster，它包括以下几部分：与PC机并口相连的25针插座头、与PCB板插座相连的10针插头、25针到10针的变换电路。下载过程完全由MAX+PLUS II软件控制。

第 5 章 驱动程序设计

5.1 驱动程序概述

VxD（Virtual x Driver）是被用来管理系统资源（硬件或者软件）的可执行的二进制代码，通过设备驱动程序，多个进程可以同时使用这些资源，从而可以实现多进程并行运行。VxD 运行于 0 特权级别上，其在内存中的位置也是处在操作系统保护的空间之内的。VxD 不仅指能够虚拟化某一硬件设备的驱动程序，也包括那些作为驱动程序却不虚拟化设备的驱动程序，也有些 VXD 与设备没什么关系，它仅向其它的 VxD 或是应用程序提供服务。VxD 可以随 VMM 一起静态加载，也可以根据需要动态加载或卸载。

5.1.1 VxD 的文件结构

通常设备驱动程序由五个段组成，它们分别是：VXD_CODE、VXD_DATA、VXD_ICODE、VXD_IDATA 和 VXD_REAL_INIT。VXD_CODE 是保护模式下的代码段，一般这个段包括设备驱动程序的控制程序、回调过程、服务程序和 API 接口函数，这个段也命名为_LTEXT；VXD_DATA 是保护模式下的数据段，一般包括设备驱动程序的描述块 DDB（Device Descriptor Block）、服务表以及一些全局变量，这个段也命名为_LDATA；VXD_ICODE 是保护模式下的初始化段，包括一些初始化时使用的服务程序和过程，虚拟机管理（VMM）在初始化结束之后就将这个段丢弃，这个段也命名为_IDATA；VXD_REAL_INIT 包括实模式下初始化数据和代码，虚拟机管理器（VMM）最先就是装入这个代码段，在继续加载其它 VxD 之前，VMM 先调用这个过程，并在该过程返回之后将这个段丢弃，这个段又命名为_RTEXT。除了实模式下初始化数据和代码段外，其它四个段都是 32 位保护模式下的平板模式（flat model）的段，意味着这些过程和数据都有 32 位偏移量[36,37]。

5.1.2 VxD 的数据结构

VMM 是通过 VxD 的设备描述块 DDB 来识别的。DDB 向 VMM 提供了 VxD 的设备控制程序的地址，还向应用程序和其他的 VxD 提供了入口点[38]。VMM 利用这个设备控制程序将 VM 及 Windows 自身的状态通知给 VxD，然后 VxD 通过相应的工作响应这些事件。下面给出 DDB 数据结构的具体信息。

```
typedef struct tabDDB{
DWORD DDB_Next;                    //指出下一个 DDB 的地址，从而将所有
                                   //VxD 的 DDB 连接起来
WORD DDB_SDK_Version;              //该 VxD 所使用的 SDK/DDK 的版本号
WORD DDB_Req_Device_Number; //设备 ID，可使用 UNDEFINED_DEVICE_ID
BYTE DDB_Dev_Major_Version;        //VxD 主版本号
BYTE DDB_Dev_Minor_Version;        //VxD 的次版本号
WORD DDB_Flags;                    //DDB 标致位
BYTE DDB_Name[8];                  //VxD 的名字
DWORD DDB_Init_Order;              //指定 VxD 的初始化顺序
DWORD DDB_Control_Proc;            //设备控制程序的地址
DWORD DDB_V86_API_Proc;            //V86 API 程序的入口地址
DWORD DDB_PM_API_Proc;             //保护模式 API 程序的入口地址
DWORD DDB_V86_API_CSIP;            //V86 入口点的 CS:IP
DWORD DDB_PM_API_CSIP;             //保护模式入口点的 CS:IP
DWORD DDB_Reference_DATA;              //实模式初始化代码设置的参考数据
DWORD DDB_VXD_Service_Table_Ptr;   //VXD 服务表的地址
DWORD DDB_VXD_Service_Table_Size;  //VXD 服务表中提供的 VXD 服务的数目
}DDB;
```

设备控制程序是 VxD 的主要入口地址。在实模式初始化后，所有的从 VMM 发出的到 VxD 的调用都使用这个入口。VMM 通过消息通知 VxD，而 VxD 则根据不同的消息进行不同的响应。设备 ID 用来帮助 VMM 识别 VxD。VMM 或其它的 VxD 通过 VxD 的 ID 识别该 VxD，进而调用该 VxD 的服务。每一个 VxD 都有一个初始化顺序值用来指定 VMM 初始化 VxD 的顺序。初始化顺序值越小的 VxD 就越先初始化。如果两个或多个 VxD 具有相同的初始化顺序值，则 VMM 根据它们在 System.ini 或注册表中出现的顺序进行初始化。如果一个 VxD 不需要指定初始化顺序，则可以使用 UNDEFINED_INIT_ORDER[39]。

5.1.3 VxD 的消息处理

VMM 提供了大量的消息与 VxD 进行通信。在 Windows 系统下编程序本质上就是写消息处理程序与系统进行通信。下面简单介绍在该数据采集卡的

驱动程序中用到的几个主要的消息。

1 PNP_NEW_DEVNODE

该消息通知可动态加载 VxD 新的设备节点被添加到设备树中。

该消息的处理原型是：VOID OnPnpNewDevnode (DWORD node,DWORD loadType)；其参数 node 表示新设备节点的句柄；loadType 表示节点的类型，它可以取 DLVXD_LOAD_DEVLOADER（设备加载程序）、DLVXD_LOAD_DRIVER（设备驱动程序）和 DLVXD_LOAD_ENUMERATOR（设备枚举类型）。

2 SYS_DYNAMIC_DEVICE_INIT

该消息通知可动态加载的 VxD 并作初始化。它与静态加载 VXD 的 DEVICE_INIT 消息非常相似。

该消息的处理程序的原型为：BOOL OnSysDynamicDeviceInit(void)；当该处理程序返回布而真时表示初始化成功，否则表示初始化失败，系统将 VxD 从内存中移去。

3 SYS_DYNAMIC_EXIT

该消息通知可动态加载的 VxD 即将从内存中移去。该消息仅仅传递给以前靠 SYS_DYNAMIC_DEVICE_INIT 加载的 VxD。

该消息的处理程序的原型为：BOOL OnSysDynamicDevice Exit(void)；当该处理程序返回布而真时表示 VxD 能够安全卸载，否则将阻止系统卸载 VxD。

4 WIN32_DEVICEIOCONTROL

该消息有针对地传给指定的 VxD 响应 Win32 应用程序调用 CreateFile 或 DeviceIoControl 函数。CreateFile 函数的第一个参数必须为\\.\name，这里 name 表示 VXD 的名字。

该消息的处理程序的原型为：DWORD OnW32DeviceIoControl(PIOCTLPATAMS p)；其参数 p 指向一个 IOCTLPARAMS 结构的指针。IOCTLPARAMS 的结构具体信息如下所示。

```
struct tagIOCTLParams
{
PCLIENT_STRUCT dioc_pcrs;          //指向客户寄存器的指针
VMHANDLE   dioc_hvm;               //虚拟机句柄
```

```
DWORD       dioc_VxdDDB;          //指向 VXD 的 DDB 结构
DWORD       dioc_IOCtlCode;       //IO 控制指令
PVOID       dioc_InBuf;           //指向输入缓冲区的指针
DWORD       dioc_cbInBuf;         //输入缓冲区的大小
PVOID       dioc_OutBuf;          //指向输出缓冲区的指针
DWORD       dioc_cbOutBuf;        //输出缓冲区的大小
PDWORD      dioc_bytesret;        //指向一个表示返回字节数的双字
OVERLAPPED*dioc_ovrlp;            //指向一个 OVERLAPPED 结构
DWORD       dioc_hDevice;         //设备句柄
DWORD       dioc_ppdb;
}IOCTLPATAMS;
```

所有 VxD 都应该实现对 p->dioc_IOCtlCode = DIOC_OPEN 和 DIOC_CLOSEHANDLE 这两种情形的处理。而其它则根据用户的要求在 VXD 中对应用程序给出的 IO 控制指令进行响应处理[40-41]。

5.1.4 VxD 的运行机制

VxD 有静态和动态之分，对于静态 VxD 必须静态加载，对于动态 VxD 必须动态加载。动态 VxD 可以通过几个 API 函数，即 CreateFile（加载 VxD）、DeviceIOControl（与 VxD 进行交互处理）和 CloseHandle（卸载 VxD）使 VxD 运行起来。该数据采集卡的 VxD 就是动态 VxD，因为它是即插即用的，以后将详细介绍。

静态加载 VxD，通常 VMM 通过以下几个步骤实现：

1 加载 VxD 的实模式初始化段并且调用它的实模式初始化过程，这个过程可以通过返回在 AX 的值决定是否阻止该 VxD 的继续加载或是否继续启动系统，同时在实模式初始化过程中可以为设备保留独占使用的页面，指定需要实例化的数据。

2 加载 VxD 的其余 32 位平板保护模式段并且丢弃实模式初始化段。

3 VMM 传送 SYS_CRITICAL_INIT 消息给 VxD 的设备控制过程。

4 VMM 传送 DEVICE_INIT 消息给 VxD 的设备控制过程。

5 VMM 传送 INIT_COMPLETE 消息给 VxD 的控制过程。

6 丢弃保护模式初始化数据段和代码段，释放这些内存做其他用。在 INIT_COMPLETE 消息后就不能访问这些初始化段。

为了使静态加载的 VxD 运行起来，即如何使系统启动时加载运行该 VxD，必须修改注册表或 SYSTEM.INI 文件，两者改一个即可。修改注册表就是在 VxD 主键下新建一个与即将加载的 VxD 同名的子键，并在该子键对应的右窗口上新建二进制数据项和字符串数据项，以给出该 VxD 的信息。修改 SYSTEM.INI 文件，就是在该文件的[386Enh]小节下面添加一行：device = name.vxd 就可以了，name 为需要加载的 VxD 的名字。

5.2 数据采集卡驱动程序的实现

本论文所设计的数据采集卡工作过程如下，首先初始化系统，由软件设置工作过程中所需的参数；然后启动 A/D 转换，由于 A/D 转换的速度低于总线的数据传输速度，所以先将数据存储在外部 FIFO 存储器，为了保持数据的连续性，系统使用两片 FIFO 轮流传送数据，由 A/D 转换器到外部 FIFO 的数据传送和从外部 FIFO 到 S5933 之间的数据传送使用硬件实现，前文已讲述；当 S5933 的内部 FIFO 有四个或四个以上字节时请求总线，启动总线 DMA，将数据传送到主机存储区，用户所需的数据采集完毕后，系统申请中断，并将所采集到的数据传送的应用程序，便于用户对数据进行处理。

该数据采集卡的驱动程序主要包括以下几个模块：即插即用、中断处理和 VxD 与 Win32 应用程序之间的通信。下面将详细叙述。

5.2.1 即插即用

即插即用（PnP）是微软公司为了使新硬件设备的安装和配置更加容易而采取的一种策略。即插即用既需要硬件支持（既设备可以自身识别并通过使用标准的软件接口来配置从而代替了跳线配置或专有接口配置），也需要软件支持（一个可以分配系统资源的操作系统，如 I/O 地址、IRQ 和驱动程序等能够从操作系统中获得其资源分配）[42]。

在即插即用系统中所有配置管理是由配置管理器控制的。配置管理器生成三个主要的数据存储器：设备节点、设备节点树和注册表。配置管理器由四个主要的软件元来工作：枚举器、仲裁器、设备安装器和设备驱动器.

硬件树是配置管理器在内存中生成和保持的一种结构。硬件树中存储了计算机中所有设备的配置信息。配置管理器使用它为每一个设备配置合适的资源如 IRQ、I/O 端口甚至不可共享资源如 SCSI 识别器。设备节点是硬件

树的一个特殊项。每一个设备节点包含一个独一无二的设备标识符和一个资源需求表。

INF 文件是 Windows 即插即用标准的重要部分。它为设备提供了一个显示给用户的设备描述和一个设备驱动程序的安装脚本。INF 文件能够支持复杂的安装方案，但是大多数的开发者只需要处理那些基本的。一个基本的驱动程序安装方案包括：设备识别、从驱动程序盘中复制驱动程序文件、设备资源需求以及添加 DevLoader 注册表进入项以便使该设备被枚举到时能够装载驱动程序。开发人员可以使用任何的文本编辑器来创建 INF 文件。不过 VtoolsD（2.03 版本）提供了一个叫做 INF Editor 的工具，可以通过它用分层的方法来定位和编辑 INF 文件，本数据采集卡驱动程序的 INF 文件见附录 2。

PnP VxD 的动态安装是一个复杂的过程。当一个枚举器识别出一个特殊的设备时，枚举器向设备管理器传递设备 ID 号，并要求配置管理器为设备创建一个“devnode”（设备节点）。配置管理器在注册表 KNLM\ENUM 主键下，为设备建立一个 ASCII 码设备 ID 号的硬件主键。这个硬件主键包括一个 Driver 值来指向 HKLM\SYSTEM\CURRENTCONTROLSET\SERVICE\CLASS 下的 software 主键。Software 主键包括一个 Devloader 值。然后配置管理器通过这个 Devloader 值来动态安装 VxD。其结果是 VxD 收到了一个 Sys_Dynamic_Device_Init 信息。大多数驱动程序 VxD 对 Sys_Dynamic_Device_Init 处理程序作最小的处理，可能是做某些一次性的初始化及从处理器返回 TRUE（Carry clear）表示成功。一个设备 VxD 通常只有在收到 PNP_New_Devnode 信息时才调用配置管理器服务。设备装载程序 VxD 在从 Sys_Dynamic_Device_Init 返回 TRUE 值后将接受一个 PNP_New_Devnode 信息。

在通过特定的 Devloader 注册值安装完 VxD 后，配置管理器通过给 VxD 发送 PNP_New_Devnode 信息告诉 VxD 设备节点装载了它。这个信息有两个相关参数：Devnode（在 EBX 中传递）和一个 reason code（在 EAX 中传递）。这个 reason code 要么是 DL_LOAD_DEVLOADER，DL_LOAD_DRIVER，要么是 DL_LOAD_ENUMERATOR。简单情况下，由 Devloader 安装的 VxD 就是一个真正的驱动程序 VxD，VxD 的 PNP_New_Devnode 信息处理器将首先收到 DL_LOAD_DEVLOADER 服务的 reason code，因为配置管理器仅仅知道这个 VxD 是一个设备装载程序。响应这个 reason code，VxD 将调用

CM_Register_Device_Driver 服务来使设备管理器知道，这个 VxD 不仅是设备装载程序，也是真正的设备驱动程序。复杂情况下，设备装载程序 VxD 和驱动程序 VxD 是分开的，将首先安装设备装载程序 VxD，然后将接收一个带有 DL_LOAD_DEVLOADER reason code 的 PNP_New_Devnode 信息。作为响应，一个真正的设备装载程序 VxD 使用配置管理器服务来安装真正的 VxD。在安装完驱动程序 VxD 后，配置管理器将向驱动程序 VxD 发送它本身的 PNP_New_Devnode 函数信息，这次带有 DL_LOAD_DRIVER 函数信息，驱动程序 VxD 通过调用 CM_Register_Device_Driver 函数来响应[43]。

在配制管理器安装完所有的 PnP 设备驱动程序 VxD 之后，它调用仲裁器来为所有的 PnP 设备配置资源。一旦仲裁器完成这些配置后，配置管理器将通知每一个驱动程序 VxD 它可以使用已分配好的配置。一个 VxD 通过它的配置处理器函数接收这个通知。VxD 的配置处理函数必须符合这个接口：

(1) 配置回调服务调用的接口函数：

CONFIGRET CM_HANDLER ConfigHandler(CONFIGFUNC cfFunc,SUBCONFIGFUN scfSubFunce,DEVNODE dnDevNode,DWORD dwRefData,ULONG ulFlags);

当通知一个 VxD 重新分配配置时，配制管理器在 cfFunc 中设置 CONFIG_START 参数。在 CONFIG_START 处理过程中，一个 VxD 通过调用另一个配置管理函数，即 CM_Get_Alloc_Log_Conf，可获取以分配给它的配置，如设备中断号，基地址寄存器等[44]。

(2) CM_Get_Alloc_Log_Conf 服务调用函数

CONFIGRET CM_Get_Alloc_Log_Conf(PCMCONFIG pccBuffer , DEVNODE dnDevNode,ULONG ulFlags);

这个函数通过 ulFlags 参数值可检索已分配配置或启动配置，在 CMCONFIG 结构中返回配置。

5.2.2 中断处理

Windows 中由 VPICD(Virtual Programmable Interrupt Controller Device)来管理所有的硬件中断。如果一个 VxD 为中断进行了注册，VPICD 就将这个中断交给它处理。

VtoolsD 提供两种硬件中断类：VhardwareInt 和类 VsharedHardwareInt。VhardwareInt 实现对某个 IRQ 端口的虚拟化，并处理该 IRQ 端口上的硬件中断，对其进行处理。一个 VxD 一旦控制了某个 IRQ，其他 VxD 便不能对它

虚拟化[45-46]。

类 VSharedHardwareInt 是类 VhardwareInt 的派生类。但 VsharedHardwareInt 允许一个 IRQ 被多个 VxD 虚拟化，即多个 VxD 共享一个中断。PCI 设备分配的硬件中断号是共享中断，操作系统给本数据采集卡分配的中断号是 10。类 VSharedHardwareInt 的中断通知事件处理函数 OnSharedHardwareInt()是 BOOL 型。

1 类 VSharedHardwareInt 的主要成员函数

(1) VSharedHardWareInt::VSharedHardInt(int rq, DWORD flags, DWORD timeout, PVOID refdata)

irq：指明被虚拟的 IRQ，取值 0~15。

flags，timeout，refdata 一般均置为 0。

该构造函数创建一个 VSharedHardwareInt 类实例。

(2) VSharedHardWareInt::~VsharedHardInt()

该函数清除一个 VSharedHardwareInt 类实例。

(3) BOOL VSharedHardWareInt::OnSharedHardwareInt(VMHANDLE HVM)

该函数返回 TRUE 表示中断处理成功，返回 FALSE 表示中断处理失败。

PCI 设备分配的硬件中断号是共享中断所以必须使用类 VSharedHardwareInt 的中断通知事件处理函数 OnSharedHardwareInt()。

2 使用类 VsharedHardwareInt

当 IRQ 中断发生时，某个 VxD 的 OnSharedHardwareInt()成员函数被触发，OnSharedHardwareInt()处理了这次 IRQ 中断，它返回 TRUE 告知 VPICD 中断已被处理，VPICD 将不再通知其他共享此硬件中断的 VxD 进行处理，若 OnSharedHardwareInt()忽略了这次 IRQ 中断，它应该返回 FALSE 告诉 VPICD，VPICD 将调用下一个 VxD 的 OnSharedHardwareInt()成员函数，直到 IRQ 中断被处理了或者所有 VxD 的 OnSharedHardwareInt()成员函数都被调用了为止。

本系统的硬件中断是由 DMA 数据传输时，计数器减为 0 触发的。在中断处理函数 OnSharedHardwareInt()中，首先要查询是否是设备本身产生的中断，由于 PCI 设备共享中断号，可能会导致 VxD 拦截了其它设备产生的中断[47]。判断为是后，还需要继续判断是否由 DMA 传输完毕后产生的中断，若

是，则要向用户程序传递所采集的数据。另外，中断产生的原因也可能是目标设备无效或主设备无效，可编写相应的处理代码，具体代码请参考附录 3。

5.2.3 VXD 与 Win32 应用程序之间的通信

VxD 虽然运行在 Ring0 级，但是它可以提供与应用程序之间的接口服务，使应用程序具有处理硬件的能力。Windows 允许 VxD 和应用程序之间进行双向通信：应用程序对 VxD 的通信和 VxD 对应用程序的通信。就作者编写的该数据采集卡的驱动程序来说，应用程序必须向 VxD 传送命令，VxD 要将采集到的数据传送到应用程序。

1 Win32 应用程序对 VxD 的通信

在 Windows 中，Win32 应用程序对 VxD 的通信方法只有一种，就是利用 Microsoft 设备输入输出控制函数 DeviceIoControl()来实现。

在 Win32 应用程序中，首先用 CreateFile()函数加载、打开 VxD 获得 VxD 的设备句柄，格式如下：

```
HANDLE
hDevice=CreateFile("\\\\.\\NAME.VXD",0,0,0,CREATE_NEW,FILE_FLAG_DE
LETE_ON_CLOSE,0)
```

其中 VxD 的路径及文件名格式为"\\\\.\\NAME.VXD"，NAME.VXD 为 VxD 的文件名，如作者编写的 VxD 的文件名是 Wahpnp.vxd，系统加载 VxD 文件所默认的路径：首先是当前目录，然后才是 C:\Windows\System 目录。

如果 CreateFile()的返回值为 INVALID_HANDLE_VALUE，则可以调用 GetLastError()获得错误消息，通常有两种错误消息，一为找不到要加载的 VxD，二为虽然有 VxD 存在，但系统不支持 DeviceIoControl()的要求。

如果 CreateFile()成功，Win32 应用程序就可以调用 DeviceIoControl()函数与 VxD 进行通信了，此函数定义如下：

```
BOOL DeviceIoControl(
HANDLE hDevice,                  //用 CreateFile()函数加载 VxD 所获得的设
备句柄
DWORD dwIoControlCode,     //应用程序调用 VxD 的命令代码
LPVOID lpInBuffer,          //应用程序传递给 VxD 的数据缓冲地址
DWORD nInBufferSize,       //应用程序传递给 VxD 的数据缓冲字节数
LPVOID lpOutBuffer,         //VxD 的返回数据所存放的缓冲地址
```

DWORD nOutBufferSize, //VxD 的返回数据所存放的缓冲字节数

LPDWORD lpBytesReturned, //VxD 实际返回数据的字节数

LPOVERLAPPED lpOverlapped //一个指向 OVERLAPPED 结构的地址，同步时置为 NULL

);

通常需要创建一个头文件，来定义应用程序调用 VxD 的命令代码。定义命令代码格式如下：

#define X CTL_CODE(FILE_DEVICE_UNKNOWN , 1 , METHOD_NETHER , FILE_ANY_ACCESS)

其中 X 为应用程序调用 VxD 的命令代码，若要定义多个命令代码，只需要将第二个参数依次加 1 即可。作者创建了 READ 命令代码。

在 VxD 中，与 DeviceIoControl() 函数相对应的是 DWORD VDevice::OnW32DeviceIoControl(PIOCTLPARAMS pDIOParams)函数。它处理 W32_DEVICEIOCONTROL 控制消息。当 Win32 应用程序动态加载 VxD、动态卸载 VxD 以及调用 DeviceIoControl()函数时，OnW32DeviceIoControl()函数被触发，返回 DEVIOCTL_NOERR(0)表示成功，其他值表示失败[48-49]。参数 pDIOCParams：为指向 IOCTLPARAMS 结构的指针，该结构定义如下：

typedef struct tagIOCTLParams

{

PCLIENT_STRUCT dioc_pcrs;

VMHANDLE dioc_hvm;

DWORD dioc_VxDDDB;

DWORD dioc_IOCtlCode;//应用程序调用 VxD 的命令代码

PVOID dioc_InBuf; //应用程序传递给 VxD 的数据缓冲地址

DWORD dioc_cbInBuf; //应用程序传递给 VxD 的数据缓冲字节数

PVOID dioc_OutBuf; //VxD 的返回数据所存放的缓冲地址

DWORD dioc_cbOutBuf; //VxD 的返回数据所存放的缓冲字节数

PDWORD dioc_bytesret; //VxD 实际返回数据的字节数

OVERLAPPED* dioc_ovrlp; //一个 OVERLAPPED 的结构地址，同步时置为 NULL

```
DWORD        dioc_hDevice;
DWORD        dioc_ppdb;
}IOCTLPARAMS,*PIOCTLPARAMS;
```

其中 dioc_IOCtlCode：为应用程序调用 VxD 的命令代码，VxD 根据它的值进行相应的处理：在 OPEN 命令代码中，申请 DMA 缓冲区，设置启动 DMA 传输(下面小节中将具体叙述)；在 READ 命令代码中，将取得应用程序的 ring0 层句柄，取得应用程序传递给 VxD 的数据缓冲器地址（下面小节中将具体叙述）以及非屏蔽物理中断；在 CLOSEHANDLE 命令代码中，释放 DMA 缓冲区，屏蔽物理中断。后面的六个参数分别与前面介绍 DeviceIoControl()函数中的参数相对应。

2 VxD 对 Win32 应用程序的通信

VxD 对 Win32 应用程序的通信共有三种方法，其中有两种只适应于 VxD 对 Win32 应用程序的通信，它们是：一使用 APC(asynchronous procedure)异步过程调用，二使用 WIN32 事件。二者均依赖 VxD“唤醒”一个 Win32 应用线程。异步过程调用比较简单；使用 WIN32 事件相对来说编程复杂一些，但它的效率高于异步过程调用。还有一种通信方法，就是给应用程序发送消息，它既可以用于 VxD 对 Win32 应用程序的通信，也可以用于 VxD 对 Win16 应用程序的通信。这种方法适用于 VxD 对以消息处理的应用程序的通信。在编写数据采集卡的驱动程序时，只用到的以 APC 方式将采集到的数据从 VxD 传递到 W32 应用程序。

应用程序首先将 VxD 要回调的应用程序的入口地址传递给 VxD，然后应用程序执行 SleepEx()使此线程处于警觉状态（alertable wait）状态。这样当 VxD 捕捉到一些特定的事件发生时，便可调用 VWIN32_QueueUserApc()函数给 Win32 应用程序发消息，即可触发 Ring3 层应用程序的回调函数。VxD 可以把要传递的数据传递给应用程序，或仅回调应用程序一下，通知应用程序进行相应的处理，这个过程就叫做异步调用。只有当应用程序的线程处于警戒状态时，VxD 才能用 VWIN32_QueueUserApc()函数触发应用程序的回调函数。

在应用程序中，VxD 回调应用程序函数的格式如下：

```
DWORD WINAPI CallBackAPC(PVOID param)
{
```

```
..//进行相应的处理，如 printf("%X\n",*(DWORD*)param);
  return 0;
}
```

其中 param 为 VxD 传递给应用程序的参数的首地址指针。

VWIN32_QueueUserApc()函数的格式如下：

VOID VWIN32_QueueUserApc(PVOID Pr3Proc , DWORD Param , THREADHANDLE hThread);

参数：

Pr3Proc：为 VxD 回调应用程序函数的入口地址，如 CallBackApc。

Param：为 VxD 传递给应用程序的数据的入口地址，如 DMABufferLinear。

hThread：为回调的应用程序的 Ring0 层的线程句柄，它由 VxD 在 W32_DeviceIoControl()函数中调用 Get_Cur_Thread_Handle()函数获得的。

5.2.4 DMA 编程

DMA（Direct Memory Access）技术是一种由 DMA 控制器控制完成的存储器与外部设备或存储器之间大数据量传输的方法，也称直接存储器存取方法。它有两个优点：一是相对于处理器执行程序实现外设与存储器之间的数据传输而言，DMA 传输速率要更高一些；二是当需要把一个外设的大量的数据送到指定内存时，它可以自动完成传送任务，这样就使处理器节省了大量对外设查询的时间，从而提高了系统的整体性能[50]。

DMA 有两种类型：系统 DMA 和总线主控 DMA。总线主控 DMA 通常用在 PCI 设备中。

该数据采集系统中，S5933 作为总线主控设备接口，允许使用总线地址寄存器和主控字节计数器实现 DMA 传送。该 DMA 是总线 DMA，它的传输由该系统的 DMA 控制器完成，驱动程序要做的便是申请缓冲区，将缓冲区的物理首地址和传输字节数分别写入 DMA 控制器的相应寄存器中，再置为位 DMA 启动位，便可启动 DMA 操作。

对于总线 DMA 的缓冲区要求：（1）物理上连续；（2）固定的和页面锁定的。

申请物理连续的缓冲区的函数为：

```
PageAllocate(nPages, PG_SYS,  0, 0, 0, 0x100000, &PhysAddr,
```

PAGECONTIG | PAGEFIXED | PAGEUSEALIGN, &hMem,

&DMABufferLinear);

该函数申请定位在4KB的边界上和物理地址在4GB以下的缓冲区。对主要参数说明如下：

nPages：申请的页面数。

PG_SYS：允许VxD在硬件中断期间访问缓冲区，不管中断发生时是哪个VM正在执行。

PhysAddr：

PAGECONTIG：表示物理页面连续。

PAGEFIXED：页面在固定的线性地址处被锁定

PAGEUSEALIGN：设置了PAGEFIXED，必须设置该标志。

DMABufferLinear：返回所申请的物理缓冲区的地址所映射的线性地址，因为程序不能直接对物理地址操作，必须使用线性地址。

hMem：所申请的缓冲区的句柄。

在结束数据传输后，必须释放申请的缓冲区。可使用以下语句：

PageFree(hMem, 0);

启动DMA传输前，必须设置以下寄存器：

总线主控控制/状态寄存器MCSR：FIFO管理，FIFO有四个或以上空单元请求总线，bit9=1；写传送允许，bit10=1；写优先 bit8=1；复位外加接口到PCI的FIFO状态标志位，bit26=1 。

中断控制/中断状态寄存器INTCSR：写传送完中断，bit14=1；指针前移条件byte0，bit27bit26=00；endian 不转换，bit25 24=00。

主控写地址寄存器MWAR：设置采集数据在内存中的地址。

主控写传送计数器MWTC：设置采集数据块的字节数。

关于DMA编程的详细代码请见附录4。

5.3 驱动程序调试

驱动程序的调试通常用SoftICE，也可以使用Debug Monitor。

Debug Monitor是NuMega VtoolsD提供的开发工具之一，它可以用来观察VxD在debug状态下的输出流，它不但可以观察应用程序加载VxD时输出的debug信息，并且可以启动和终止VxD，以观察debug信息，该软件使用非常简单。

SoftICE 是 NuMega 公司开发的功能非常大的调试软件。它结合了硬件调试器的强大功能和符号调试程序的易用性，能够显示程序的原代码，允许通过符号名访问局部和全局的数据。

使用 SoftICE 调试 VxD，首先要激活 symbol loader，选择菜单 File 中的 Load 装入 VxD；然后选择菜单 Module 中的 Translate；最后在菜单 Edit 中的 SoftICE Initialization Settings 选择 symbols，装入 nms 文件。重新呼出 SoftICE 就可以看到源代码了，利用 SoftICE 命令在源代码中设置断点并退出 SoftICE，再运行应用程序时，就会在断点处呼出 SoftICE，显示执行信息。

该驱动程序被应用程序动态加载后，工作状态可由 Debug Monitor 观察到，如图 5.1 所示。

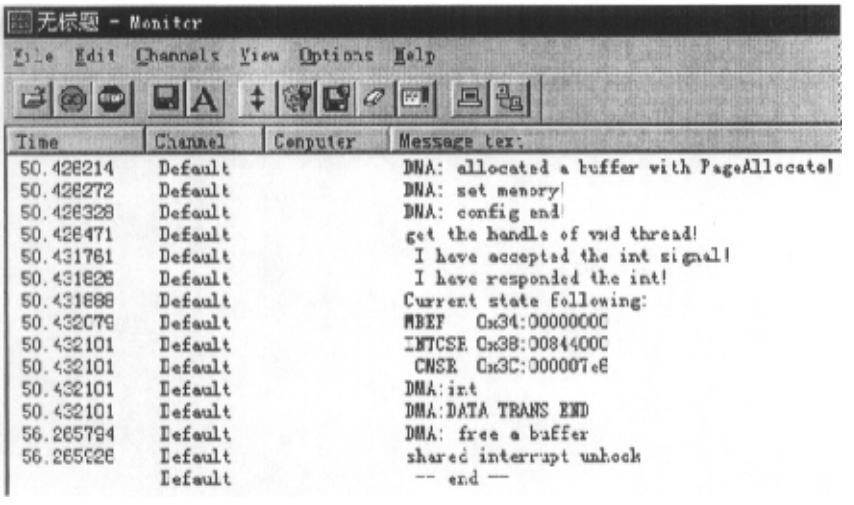


图 5.1 VxD 工作状态

5.4 结论

该数据采集系统是通过信号发生器输出的信号进行测试的。由于系统为双极性输入，范围在-5V~+5V 之间，所以输入电压的最大幅值不应超过 5V。

应用程序为多线程的，当接收到 VxD 传来的数据后，画出信号的波形图，并计算出最大值与最小值。当输入 4V 的直流信号时采集到的数据幅度图见图 5.2；当输入最大幅值为 4V，频率为 40k H_z 的正弦信号时，采集到的数据幅度图见 5.3。由于测试时是用两根导线将信号接入系统的，高频信号易被干扰

而失真。

直流输出信号的幅度图象是在横轴为 360 个象素长度的坐标轴上画出 1024 个采集点得到的，将横坐标做了转换，相当于将图象沿横轴压缩到原来的$\frac{1}{3}$长度。

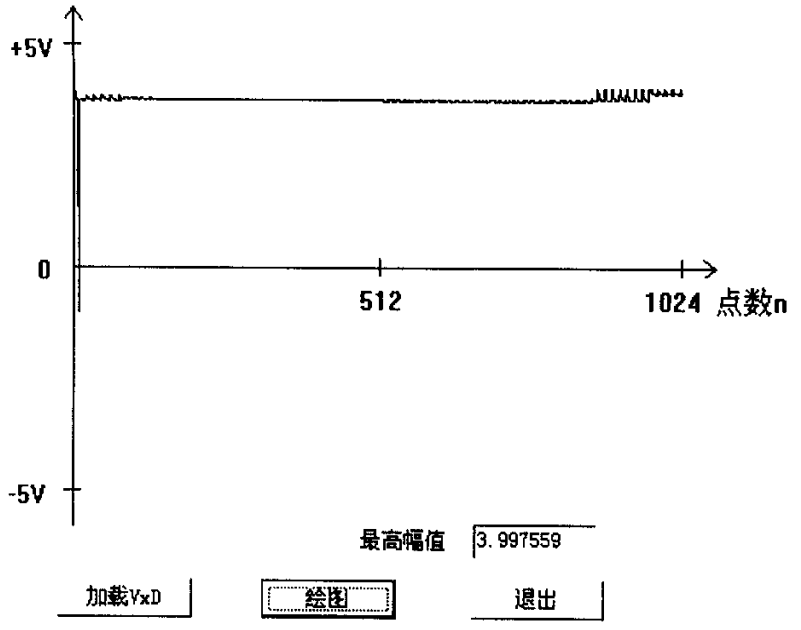


图 5.2 采集到的 4V 直流信号

正弦信号幅度图的绘制跟直流信号的方法一样，但是由于正弦波形太密集了，所以只画出了奇数点的图象。

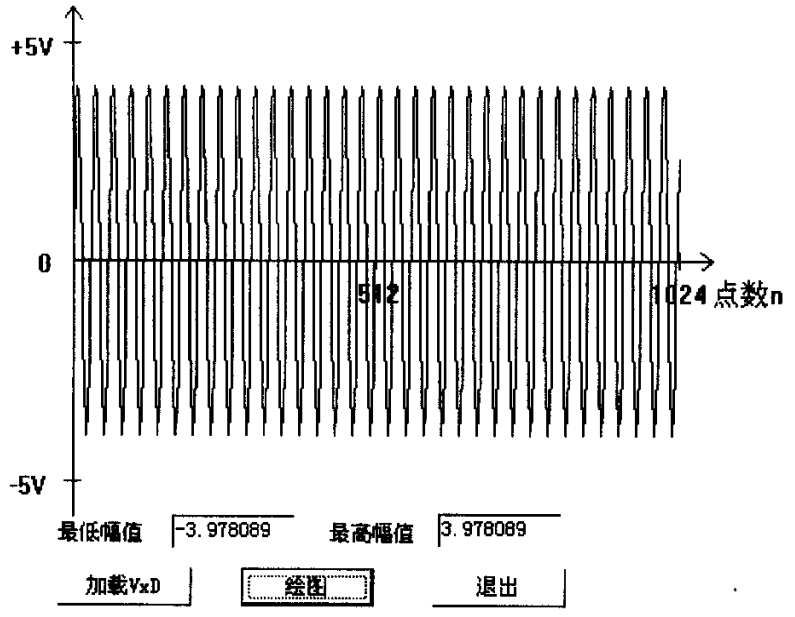


图 5.3 采集到的 40k H_z 正弦信号

将所采集数据的的波形展开，即只画出前面 128 个点的波形，如图 5.4 所示。

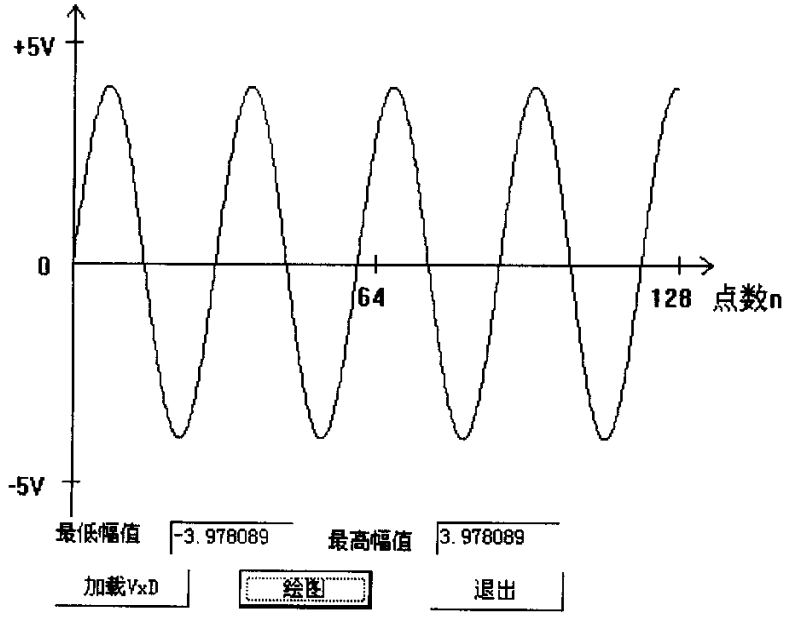


图 5.4 采集到的 40k H_z 正弦信号

结 论

数据采集是自动化测试系统的最重要的组成部分，它涉及到传感器、前期信号处理、模数转换、数据存储与传输、微机总线接口技术以及设备驱动程序等多项关键技术，它为测试系统提供可供分析的数据。数据采集的精度和速度一直是测试人员所关心的问题，采集数据的精度和速度直接影响到测试系统的效果，决定了测试系统的成败。

本论文采用新型微机总线 PCI 总线，设计了数据采集卡。信号前期处理部分包括放大（或衰减）和滤波，这部分电路简单，并且由于智能传感器技术的发展，很多传感器已自带放大或衰减网络，也为了增强数据采集卡的通用性，作者在设计中省略了前期信号处理部分（用户可根据测量对象制作相应的前期信号处理电路），而将重点集中在 PCI 总线接口及硬件驱动程序编写上。

系统采用 PCI 接口芯片 S5933 执行 PCI 协议，S5933 提供三种数据传输方式：PASS-THRU、邮箱和 FIFO，本论文采用 FIFO 通道。由于 S5933 内部 FIFO 只有 8 个双字的深度，系统使用两片外加 FIFO IDT72225（1k×18bit ）通过双缓冲保证了数据采集的连续性。

模拟信号经 A/D 转换后，数据首先存入外加 FIFO，再由控制逻辑将数据传输给 S5933，最后由 DMA 方式传给主机。A/D 转换器 AD678 的最高频率可达 200kSPS，由 S5933 系统的 33M 系统时钟经过 8 次分频可给 AD678 提供采样时钟和写外部 FIFO 的时钟。外部 FIFO 的数据可用同步方式传到 S5933。这两部分控制电路均通过可编程逻辑器件实现。当采集完所需数量的数据后，系统产生中断，由中断处理程序将数据传给用户程序。

由于 Windows 的保护机制，使用户必须编写专门的硬件驱动程序实现对硬件的控制。作者利用 VtoolsD 编写 Win98 下的设备驱动程序 VxD，该 VxD 主要包括以下几部分：取得资源配置、中断处理和 VxD 与 Win32 应用程序之间的通信。资源配置由操作系统完成，为了控制硬件，必须取得这些配置(中断号，寄存器地址等)，这部分在系统初始化中实现；VxD 与 Win32 应用程序之间的通信主要是用户向 VxD 传送命令；中断处理包括查询中断、将数据传

给用户程序和清除中断。

对于该数据采集卡的硬件和软件，作者做了大量的工作，使系统性能达到预期的效果，根据作者在研究中得到的启发和发现的问题，具体将该数据采集系统还需优化的方面总结如下：

1. 本系统是单通道数据采集，若要设计为多通道数据采集系统，必须在系统中加如另外的“AD678+2×IDT72225”模块，数据传输控制与原来完全相同，但还必须加上通道选择部分，该部分可用软件和可编程逻辑器件实现，相应地可采用大容量的可编程逻辑器件。由于模拟开关的低速与模拟干扰，PCI 系统中不再使用。

2. 本系统的信号输入端口采用的是 9 针串口，要接入信号时必须用导线接入，由于串口本身的性能和导线的非屏蔽性，对于高频信号，必将受到干扰，这也是作者考虑不周之处。所以必须将输入端口改善为易于接入屏蔽电缆线的端口以防止信号传输过程中受到干扰，如类似于音频插头的端口等。

3. 本设只完成了 Win98 下的设备驱动程序编写，考虑到 Windows2000 与 WindowsNT 用户，必须编写 Windows2000 与 WindowsNT 下的驱动程序。作者建议使用 WDM，因为 Windows 系列与 Windows NT 最终统一起来，VxD 将被 WDM 取代。

总之，通过作者一年多的努力与导师的指导，已经基本掌握了 PCI 系统开发技术以及 PCI 总线驱动程序的开发技术。虽说系统不够完善，还存在一些问题，但毕竟关键性技术已解决了，相信在今后的努力中，我们会逐步解决这些问题，并优化数据采集系统。

致 谢

本论文是在我的导师一高品贤教授的指导下完成的，导师对本论文的选题、研究内容、乃至具体的章节都进行了详细的指导。导师严谨的治学态度、渊博的知识及孜孜不倦的教诲使我收益匪浅。并且，导师为我的论文的完成提供了经费支持以及相关仪器等便利条件。在此，作者向导师表示深深的谢意，并感谢导师在我本科与研究生期间多年来给予的关心和帮助。

还要特别感谢的两位是我的师兄孟劲松和好友张治。孟师兄在完成本论文期间提出了许多有益的建议和观点，并无私地传授了很多宝贵的经验。感谢张治一直以来对我的关心和照顾，他在繁忙的学习中抽出时间从论文的整体思想到具体的工作都给予了很大的帮助。当然，论文的顺利完成也离不开其他一些好友和同学的帮助与鼓励，他们是：孙美玲、宋九鹏、张丽平、李文迎等，在此不一一列举，我衷心地感谢我的朋友们。

非常感谢父母的鼓励和支持，是他们给我增添了前进的动力和拼搏的勇气。

参考文献

[1] 王承等. 虚拟仪器—现代化仪器发展的新阶段. 测控技术. 2001, 20（10）

[2] The Future of Virtual Instrumentation[J]. SENSORS,1997, (7)

[3] Interoperability Comes to Data Acquisition[J].SENSORS,1999,(8)

[4] 潘丽丽, 高品贤. 基于 PCI 总线数据采集系统的研究[J]. 电子技术应用. 2001, (9)

[5] 刘晖等译. PCI 系统结构(第四版)[M]. 电子工业出版社, 2000

[6] INTERNET 站点 http://www.AMCC.com

[7] 高光天. 模数转换器应用技术. 科学出版社,2001

[8] 王学良, 来忠信. 基于 PCI 总线控制器的高速大容量实时数据采集[J]. 光学 精密工程, 2000(2)

[9] 金明, 罗飞路, 朱霞辉. FIFO 芯片在高速系统中的应用[J]. 电子技术应用.1998（3）

[10] 苏庆会, 王梅花. FIFO 在计算机高速采集系统中的应用[J]. 电子计算机与外部设备. 1999(9)

[11] 孙守阁, 徐勇. Windows 设备驱动程序内幕[M] . 清华大学出版社, 2000

[12] 杨强, 李堂秋. Win 9X 虚拟设备驱动程序编程指南. 清华大学出版社, 1999

[13] PCI Local Bus Specification 2.1. Portland PCI SIG(PCI Special Interest Group), 1995

[14] 李贵山, 戚德虎. PCI 局部总线[M]. 西安电子科技大学出版社, 2000

[15] 薛兆玉, 谭贵珍. PCI 总线[J]. 微计算机应用. 1997, 18(3)

[16] Bernie Osenthal and Ron Sartore. Designing PCI-compliant Master/Slave Interface for Add-On cards. EDN March 30, 1995

[17] PCI System Design Guide, Revision 1.0. Architecture Eevelopment Lab Intel Corporation, 8 September,1993

[18] AMCC S5933 PCI Controller Data Book. San Diego:AMCC(Aplied Micro Circuits Corporation),1998

[19] 蔻小明. S5933 PCI 总线控制器及其应用. 电子技术应用[J]. 1998, (12)

[20] INTERNET 站点 http://www.AMCC.com

[21] 陈利学等. 微机总线与接口设计[M]. 电子科技大学出版社, 1998

[22] 周智敏等. 超高速数据采集技术研究与系统实现. 系统工程与电子技术[J]. 1998(11)

[23] 张旭东等. 基于 PCI 接口的多通道高速数据采集系统. 数据采集与处理[J]. 2000, 15(2)

[24] 王岩,李建国等.基于 IBM-PC 机的超高速数据采集系统的研究[J].华中理工大学学报,1994(8)

[25] 沈兰荪. 高速数据采集系统的原理与应用. 北京:人民邮电出版社, 1995

[26] 姚志峰等. 基于 Windows95 的高速数据采集. 测控技术[J]. 1999, 18(4)

[27] Analog Devices, Inc. AD678 Datasheet, 1998

[28] 李守成, 张志波. 数字电子技术基础[M]. 第三版. 成都: 西南交通大学出版社, 1994

[29] 陈征. FIFO 缓冲存储器的结构及应用[J]. 汕头大学学报, 1998(1)

[30] INTERNER 站点 http://www.IDT.com

[31] INTERNER 站点 http://www.Altera.com

[32] 班荣峰. 基于专用芯片 S5933 的 PCI 总线接口设计. 电子技术应用,1998(11): 55-57

[33] ALTERA Cop. ALTERA News & Views Feb, 1995

[34] 赵曙光等. 可编程逻辑器件原理、开发也应用[M]. 西安: 西安电子科技大学出版社, 2001

[35] 姚嘉, 施旭霞. CPLD 在超高速数据采集系统中的应用. 电子技术应用[J]. 1998, (10)

[36] 武安河, 周利莉. Windows 设备驱动程序(VxD 与 WDM)开发实务[M]. 电子工业出版社, 2001

[37] Ruediger R.Ashe. The Little Device Driver,Microsoft Development Network Technology group

[38] 刘其锋.WIN95 下虚拟设备驱动程序设计开发. 电子技术应用[J]. 2000, (4)

[39] 陈性元等. Windows 的 VxD 技术分析. 小型微型计算机系统[J]. 2000,

21(6)

[40] 彭礼孝. 虚拟设备驱动程序开发起步与进阶[M]. 人民邮电出版社, 2000

[41] Microsoft Windows95 Device Driver Kit. Microsoft Corporation, 1997

[42] 洪志敏. “即插即用”技术的讨论. 计算机应用[J]. 2000, (6)

[43] KAREN HAZZAH. Windows VxD 与设备驱动程序权威指南[M]. 孙喜明译.第二版. 中国电力出版社, 2001

[44] 李海. PCI 设备 Windows 通用驱动程序设计. 电子技术应用[J]. 2000, (1)

[45] 李博, 鲍超. Windows98 下硬件中断驱动程序的开发. 电子技术应用[J]. 2000, (5)

[46] 王庆辉等. 用 VxD 编写 Windows 95 中断处理程序的方法. 光电工程[J]. 1999, 26(3)

[47] 赵敏等. Win 95 下 A/转换中断的编程与实现. 计算机自动测量与控制[J]. 2000, 8(1)

[48] 陈阵等. Win9x 通用数据采集设备驱动程序开发[J]. 计算机应用研究. 2000, (6)

[49] 张维铭. 用 VtoolsD 开发 Windows95 虚拟设备驱动程序[J]. 微型机与应用,1998(8)

[50] 徐晓东,杨振坤. WIN95 下 DMA 方式编程. 计算机系统应用[J]. 1999, (6)

[51] 刘乐善等. 微型计算机接口技术原理及应用[M]. 武汉:华中理工大学出版社. 1996.5

[52] 阮德生. 自动测试技术与计算机仪器系统设计[M]. 西安:西安电子科技大学出版社.1997.

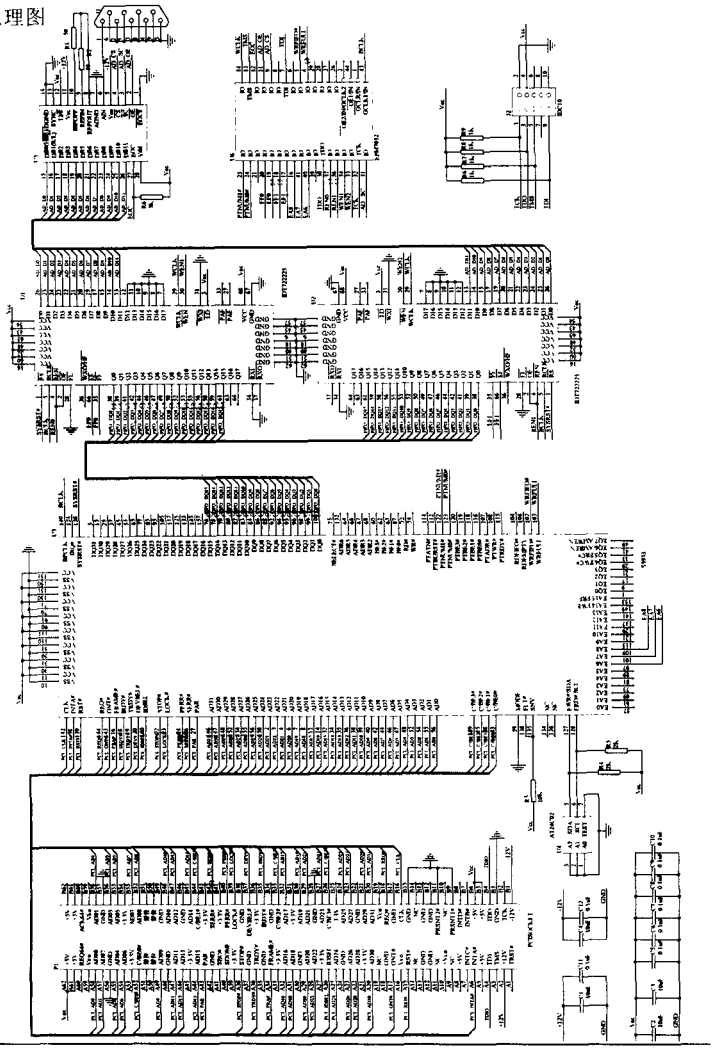
[53] 马明建, 周长城. 数据采集与处理技术[M]. 西安:西安交通大学出版社. 1999.4

[54] David J. Kruglinski, Scot Wingo, George Shepherd. Programming Viaual C++ 6.0. Microsoft Press.1999.

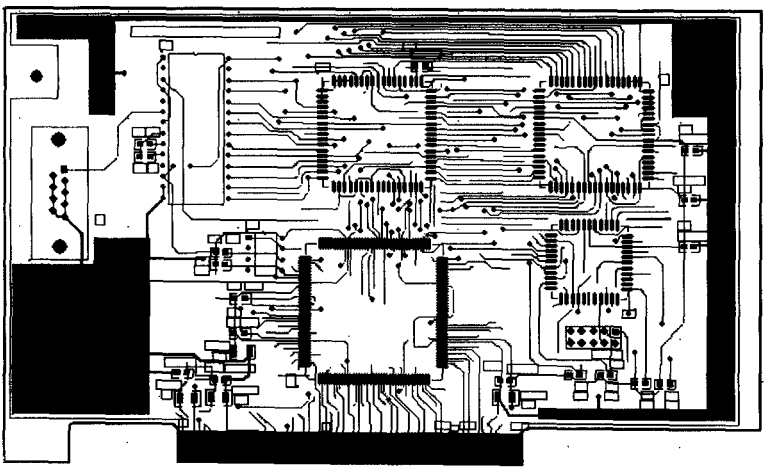
[55] Davis Chapman. 学用 Visual C++ 6.0. 骆长乐译. 北京：清华大学出版社.1999.9.

附录 1 系统电路图

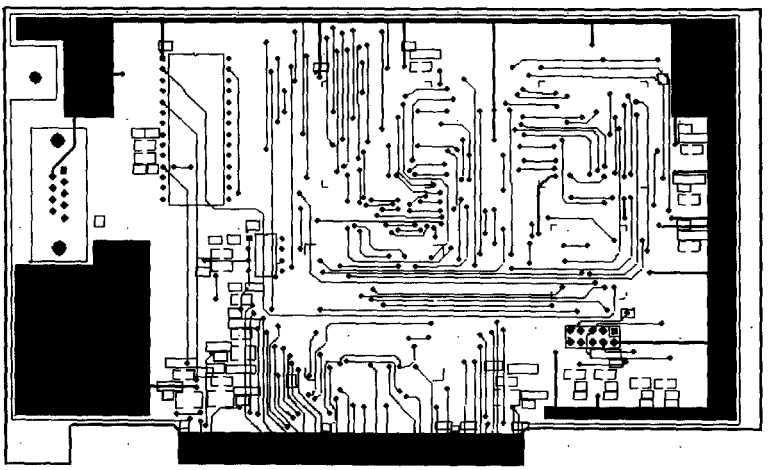
1 原理图



2 顶层印制板图



3 底层印制板图



附录 2 驱动程序的 INF 文件

```
[Version]
Signature=$CHICAGO$ //Win98 系统
Class=Unknown          //其他设备
Provider=%String0%

[ClassInstall]
[DestinationDirs]
DefaultDestDir=11   //表示 C:\Windows\System 目录
CopyFiles_WAH5933=11

[Manufacturer]
%String1%=SECTION_0

[SECTION_0]
%String2%=WAH5933,PCI\VEN_10&DEV_5933

[WAH5933]
CopyFiles=CopyFiles_WAH5933
AddReg=AddReg_WAH5933

[CopyFiles_WAH5933]
Wahpnp.vxd

[AddReg_WAH5933]
HKR,,DevLoader,0,Wahpnp.vxd

[ControlFlags]

[SourceDisksNames]
1="PCI 数据采集卡","",0

[SourceDisksFiles]
Wahpnp.vxd=1

[Strings]
String0="测试技术与仪器"
String1="PLL"
String2="Wahpnp";数据采集卡"
```

附录 3 中断处理代码

```
io=ioport+INTCSR;
sint=_inpd(io);
if((sint & 0x800000)==0x800000)
{
 //判断是否我的 device 产生的中断,bit23=1,中断有效
 //如果是，则继续判断是否由 DMA 产生的，或其由它原因产生
 if ((sint&0x40000)==0x40000) {//DMA 结束中断 bit18=1:写传送完成时中断
 io=ioport+INTCSR;
 _outpd(io,sint&0xff04ffff);      //屏蔽掉该中断
 VWIN32_QueueUserApc(CallBackApc, (DWORD)DMABufferLinear, TheThread);
 //触发应用程序回调函数
}
else if ((sint&0x300000)!=0) {   // DMA 放弃中断
io=ioport+INTCSR;
_outpd(io,sint&0xf03fffff);      //屏蔽掉上述中断
}
return TRUE;
}
sendPhysicalEOI();
```

附录 4 DMA 传输代码

```
    Bufalloc();
    io=ioport+INTCSR;
    _outpd(io,0x003f0000);    // 清除所有中断源

    io=ioport+MWAR;          //io:基地址
    _outpd(io,PhysAddr);     //设置 DMA 地址
    io=ioport+MWTC;
    _outpd(io,4096);          //设置 DMA 传输字节数
    io=ioport+INTCSR;
    _outpd(io,0x00004000);   //(0c004000) bit25 24=00:endian 不转换
                        //bit27 26=00:指针前移条件 byte0    bit14=1:写传送完成时中断
    io=ioport+MCSR;
    _outpd(io,0x04000700); //bit10=1:ADDON-PCIFIFO 写传送允许 bit26=1:读标志复位
                          // bit8=1:写优先  bit9=1:FIFO 有四个或以上空单元请求总线
VOID Bufalloc(void)
{
    BOOL status;
    nPages=20;
    status=PageAllocate(nPages, PG_SYS, 0, 0,      0, 0x100000,  &PhysAddr,
        PAGECONTIG | PAGEFIXED | PAGEUSEALIGN, &hMem, &DMABufferLinear);
    if (status == FALSE){
        bUsingBuffer=FALSE;
        dprintf("DMA: cannot get locked memory\n");
    }
    else
    {
        bUsingBuffer=TRUE;
    }
}
```

攻读硕士学位期间发表的论文

[1] 潘丽丽，高品贤. 基于 PCI 总线数据采集系统的研究. 电子技术应用.2001,

[2] 潘丽丽，高品贤. 外加 FIFO 与 PCI 总线控制器 S5933 接口的实现. 测控技术.（已录用）