一种高速数据传输中的软硬件解决方法

余 军 贵州大学电子科学系(550025)

Abstract

Through two aspects, software and hardware ,a instance of DMA transmission and mixed programming applied in high-speed data acquisition system is introduced in this paper.At the same time ,a interface design illustration and a program are presented in it.

Keywords:DMA,interrupt,mixed programming,data transmission

摘 要

本文从软硬件两方面着手,重点介绍了 DMA 传输及混合语言编程应用在高速数据采集过程中的一个实例,并给出了具体实现的接口示意图和程序。

关键词:DMA,中断,混合语言编程,数据传输

在工业控制、检测中,有些对实时性要求很高的应用场合,要求数据采集装置要有较高的采样频率和传输速率。现在市场上有许多高速、高性能、价格低廉的 A/D 转换器都具备较高的 A/D 转换频率,很适合用于高速数据采集。但高速采集来的数据必须及时通过接口传送进计算机并被程序处理,否则会丢失数据,这就对数据传输和处理程序的执行效率提出了较高要求。本文就是从解决硬件传输和软件的执行效率着手,介绍了将 DMA(直接存储器存取)传输和混合语言编程应用在高速数据采集过程中,并在实际应用中取得了较好效果的一个实例。

1 设计要求

在我们的实验中,需要在给定单位时间内对被采集样品输出的不同幅值的离散脉冲信号分别进行计数,计数结果经过程序处理后显示在屏幕上以便对样品进行分析。但由于被采集样品的输出信号周期短、变化快、脉冲宽度窄,因此要求对脉冲幅值进行 A/D 转换的 ADC 要有较高的转换频率,相应的接口电路也要有足够的传输速率以之相匹配。为达到要求,我们选用了转换时间仅为 5μS,转换位数为 10 位的高速、高精度 ADC。经过简单的计算可知,要保证不丢失数据,以之相匹配的接口电路必须具有 400KB/S 以上的数据传输速率。且处理数据的服务程序要有足够快的数据处理速度。否则会因数据不能及时被处理而造成丢失,使信息失真。

2 软硬件上的解决方法

2.1 硬件解决方法

要达到每秒几百 K 的数据传输速率,又要做到接口简单,不影响 CPU 的运行,那微机系统中的 DMA 传输方式是最理想的解决方法。在微机系统中 DMA 传输是不需要 CPU 直接参与的,DMA 控制器具有掌管微机总线的能力,DMA 传输方式具有较高的传输率,可以很容易做到每秒 M 字节以上。且 DMA 控制器与外设间的握手信号很简单,只需要 DMA 请求信号 DREQ 和 DMA 应答信号 DACK,外设也不需要分配 I/O 端口地址。一般微机中都有好几个保留不用的 DMA 通道和中断号,我们的设计直接使用了微机中的这些保留资源,只需一个外中断和一个 DMA 通道。因为外设接口不占用微机的 I/O 端口地址,所以省去了设计端口地址译码器的工作,大大简化了接口电路。如果选用的 ADC 具备三态数据锁存功能可直接连接在总线上,那接口电路上仅需设计一个用于产生中断信号的计数器和一些用于选择微机中断号及 DMA 通道的跳线开关。否则,在接口电路上就得多设计一对像 74LS373 那样的八位三态数据锁存器,用于锁存 A/D 转换输出的数据和连接总线。

把放大后的样品输出脉冲通过比较整形后,就可用来启动 ADC 进行同步转换,只要 ADC 及保持电路有足够高的工作频率,就能保证每个到来的脉冲信号的幅值都能及时被转换为数字量。这样 A/D 转换就不需外界干预,能自动循环进行。每次 A/D 转换完毕后,都会输出转换结束信号 EOC,它可被用作向 DMA 控制器提出传输请求的信号 DREQ,而 DMA 控制器响应请求后发出的应答信号 DACK 和外设读信号 IOR 经相与后,又可用来打开 ADC 内部或外部

的三态数据锁存器，将数据释放到微机数据总线上，并传输到指定的内存缓冲区中，供程序访问。因此在设计中被选用的 DMA 通道应被设置为单字节传输（周期挪用）方式，这样不会影响到 CPU 的运行。整个设计的硬件接口示意图可参见图 1。

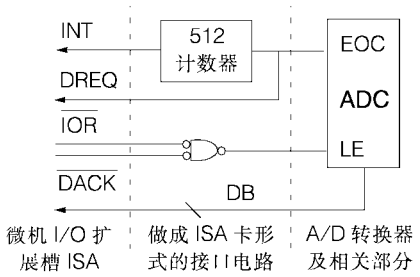
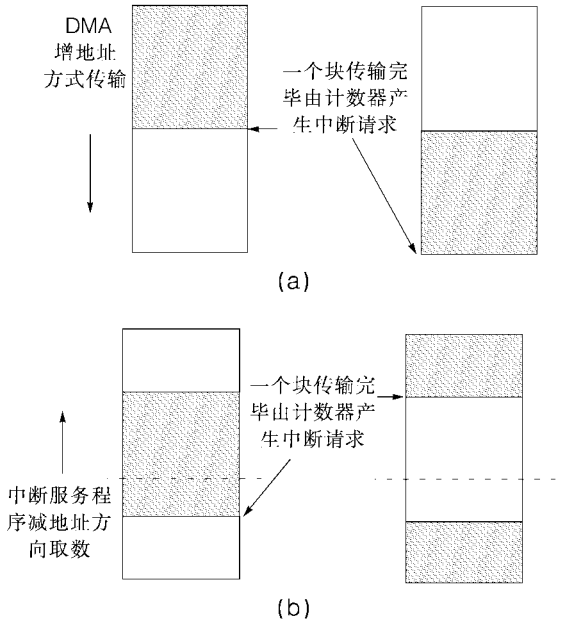


图 1 硬件接口示意图

在接口电路上设计一个计数器的目的是用于在传输完指定字数后，向微机提出中断请求，通过微机响应中断，执行中断服务程序取走已接收的数据。计数器可用 ADC 的转换结束信号 EOC 来计数。ADC 每转换完一个数据，计数器就加一（也可设计成减一）。它表示已有一个采集数据被传送到指定接收缓冲区中。当计数值达到接收缓冲区的一半时（例如将接收缓冲区设为 1024 个字，其一半值为 512），表明接收缓冲区中的前半部分已存放有采集来的数据，这时利用计数器产生一个外中断请求信号 INT 到微机，同时计数器清零，开始新一轮计数。微机响应中断，执行中断服务程序将接收缓冲区中已接收的数据取走，存储到内存中另设的处理缓冲区，供主程序对数据进行处理。选用中断方式，保证了对数据处理的及时性，又提高了 CPU 的工作效率，使 CPU 在大多数时候都可以处理数据，而不必担心数据的传输。这样即使在低档次微机中，设计也能正常工作。另设处理缓冲区可有效防止数据被重复使用，且数据的处理操作和数据的传输操作互不影响。在中断服务程序取走接收缓冲区中的前半部分数据的同时，DMA 传输仍在继续向缓冲区的后半部分传输数据，DMA 传输不会因此受影响。而当后半部分传输完毕，计数器又会产生新的中断请求。由于被选用的 DMA 通道被设置为自动预置，所以在传输完指定个字（即接收缓冲区大小 1024）后，DMA 控制器会自动恢复初值从头开始。其工作方式可参见图 2(a)。

采用这种双缓冲工作方式，保证了 DMA 传输和中断服务程序的取数可同时进行。但有时可能由于某些原因，计数器并不正好在接收缓冲区的一半位置和末尾处发出中断请求信号，而是过了接收缓冲区的一半位置或还未到时就向微机发出中断请求信号 INT，如图 2(b)所示。出现这种错误可在软件上很容易的加以解决，办法是在中断服务程序执行取数操作时，先从 DMA 控制器被选用通道的当前地址寄存器读取当前 DMA 传输的内存地址，以此地址为基础，采用减地址（DMA 对应通道被设置为地址递增方式传输）方式向上取走接收缓冲区的一半数据即可。若取数地址已达到接收缓冲区的首址，则下一个数据从接收缓冲区末尾开始。



注：图中阴影部分表示 CPU 响应中断，通过中断服务将要取走的 DMA 已传输完的数据块，空白部分表示 DMA 正在传输的数据块

图 2 计数器中断请求方式示意图

2.2 软件解决方法

在软件设计上，最好选用执行速度和编译代码质量都较高的程序设计语言。特别是中断服务程序的执行效率，是双缓冲工作能否正常进行的关键，若要是想再进一步提高采集数据的速率，则软件上中断服务程序将是一个很关键的因素。因此在中断服务程序中一般不要涉及太多的处理操作，只设置一个从接收缓冲区取数到处理缓冲区的操作就可以了，而把对数据的处理交由主程序完成。我们选择了 C 语言来编写软件的主要部分。利用在 C 程序中插入行内汇编的形式使用汇编指令编写中断服务程序部分，提高它的执行效率，为以后提高数据传输速率留下空间。混合编程充分发挥了两种语言各自的优势，而采用行内汇编这种方式省去了编写和维护接口代码的工作，便于程序的查阅。在经过软硬件两方面的改进后，我们解决了以前用其它方法中遇到的一提高数据采集速率就丢数的现象。通过实际测试，即使选用更高转换速率的 ADC，将数据采集速率提高一倍，整个设计也无

须做较大改动，只要将接收缓冲区相应增大，传输仍能十分可靠进行，不会丢失数据。

3 程序相关说明

由于设计中选用的 ADC 转换位数为 10 位，并考虑到以后有提高转换位数的可能，因此我们选用微机中能进行字传输的通道。在 PC 系列微机中，有几个保留的 DMA 通道，其中 DMA5、DMA6、DAM7 能进行 16 位的字传输。至于设计中用到的中断，可从微机保留的外中断中选用。为使设计运用灵活，避免和微机中的其它 I/O 设备冲突，硬件方面可在接口卡上设置一个跳线开关，用于跳线选择外中断和 DMA 通道，软件方面可以用程序对话的形式做出以跳线一致的选择。但在下面给出的程序中，为简单起见，直接使用了 DMA5 通道和外中断 IRQ5，至于程序对话形式的选择，可在给出程序基础上适当修改。

微机外中断 IRQ5 采用前沿触发，中断号为 0dh，端口地址为 20h，21h。要注意的是外中断的中断服务程序在结束前，一定要向 20h 端口发一个中断结束信号 EOC(值为 20h)，否则会屏蔽掉低级中断请求。程序中特别设置一个标志变量 int_flag 来反映中断服务程序的执行情况，当中断服务程序执行完取数操作后，将标志置 1，主程序可通过查询该标志了解到是否有新的数据被送到处理缓冲区，主程序在处理完数据后又重新将标志位清 0。被选用的 DMA5 通道被设置为单字传输、DMA 写、地址递增、允许自动预置的工作方式，其端口地址范围是 c0h~deh。DMA 通道的初始化程序由函数 initdma()完成，若函数初始化操作失败，返回非 0 值，否则返回 0 值。

由于 C 程序中插入了汇编指令，因此混合程序要采用 Turbo C 提供的命令行编译方式，用 tcc.exe 编译程序加上参数-B 来编译。下面给出的程序已在 TC2.0 下编译通过。有关对数据的其它处理操作大家可根据需要在给出程序基础上自行设计，在此不做过多叙述。

4 程序实现

```
# define IRQ5 0xd  /*定义 IRQ5 的类型号*/
# define INT5_IMR 0x20  /*定义 IRQ5 的屏蔽字*/
# define EOC 0x20  /*定义中断结束字*/
# define INT_PORT0 0x20  /*定义中断控制器的端口偶地址*/
# define INT_PORT1 0x21  /*定义中断控制器的端口奇地址*/
# define DMA_PORT 0xc0 /*义 DMA 控制器端口地址*/
# define DMA_PAGE 0x8b  /*定义 DMA 页面寄存器端口地址*/
# define DMA_CTRL_WORD 0x55  /*定义 DMA5 工作方式字，单字节传输、地址递增、允许自动预置、DMA 写操作方式、通道 1(第二片 DMA 控制器的通道 5)*/
# define START_DMA 0x1  /*定义启动 DMA 传输的控制字*/
# define END_DMA 0x5/*定义结束 DMA 传输的控制字*/
# define RECEIVE_LENGTH 1024  /*定义接收缓冲区大小*/
# define HALF_RECEIVE 512 /*定义处理缓冲区大小*/
# include "dos.h"
# include "graphics.h"
# include "conio.h"
# include "stdio.h"
unsigned receive_buff[RECEIVE_LENGTH];  /*定义接收缓冲区*/
unsigned sort_buff[HALF_RECEIVE];/*定义处理缓冲区*/
unsigned int_flag;  /*定义中断处理标志变量*/
void interrupt (*oldint)(); /*定义保存原中断向量的变量*/
void installint(void interrupt (* intsev)(),int intnum)
{
 setvect(intnum,intsev);
}
void saveoldint(int intnum)
{
 oldint=getvect(intnum);
}
void sort_upp(void)
{
 register unsigned i,j;
 unsigned min_lable,temp;
 for (i=0;i<HALF_RECEIVE-1;i++)
   {
     min_lable=i;
     for (j=i+1;j<=HALF_RECEIVE;j++)
         if (sort_buff[min_lable]>sort_buff[j])
             min_lable=j;
     temp=sort_buff[i];
     sort_buff[i]=sort_buff[min_lable];
     sort_buff[min_lable]=temp;
   }
} /*将处理缓冲区的数据按从小到大的顺序排列，以便处理显示*/
int initdma(void);
void interrupt getdata(void);
int timer(void);  /*定时函数*/
void dispaly(void);  /*显示处理函数*/
void main(void)
{
 unsigned char int_IMRword;
 if (initdma()) /*初始化 DMA,并判断 DMA 段是否溢出*/
    {
     printf("DMA page overflow error! \a\n");
     exit(1);
    }
 saveoldint(IRQ5);    /*保存原中断向量*/
 installint(getdata,IRQ5);   /*安装中断服务程序向量*/
```

```
 int_IMRword=inportb(INT_PORT1);  /* 读中断屏蔽寄存器值 */
 outportb(INT_PORT1,int_IMRword&(~INT5_IMR));  /* 打开外中断 IRQ5 */
 outportb (DMA_PORT +0x14,START_DMA);      /* 启动 DMA 传输 */
while(!kbhit()&&timer())/* 若定时时间到或按任意键退出 */
       {
        if (!int_flag)
          continue; /* 中断标置是否置位,否则继续等待 */
        sort_upp();  /* 数据排序 */
        display();   /* 进一步处理并显示数据 */
        int_flag=0;      /* 中断标置复位 */
       }
 outportb(INT_PORT1,int_IMRword|INT5_IMR);  /* 关闭外中断 IRQ5 */
 outportb(DMA_PORT+0x14,END_DMA);    /* 结束 DMA 传送  */
 installint(oldint,IRQ5);          /*   恢复原中断   */
}
int initdma(void)
{
 unsigned off,seg;
 long buff;
 unsigned length;
 seg=FP_SEG((void far *)receive_buff);  /* seg=接收缓冲区段地址 */
 off=FP_OFF((void far *)receive_buff);  /* off=接收缓冲区偏移地址 */
 buff=(((long)seg<<4)+(long)off)>>1;  /* 将接收缓冲区物理地址的 A1~A19 位保存到变量 buff 中 */
 disable();  /* 关中断 */
 outportb(DMA_PORT+0x18,0);  /* 清 DMA 控制器中的高低触发器 */
 delay(1);  /* 延时满足 DMA 控制器的定时要求 */
 outportb(DMA_PORT+0x16,DMA_CTRL_WORD);  /* 写 DMA 控制字 */
 delay(1);
 outportb(DMA_PORT+0x4,(unsigned char)(buff&0xff));
/* 写接收缓冲区地址的 A1~A8 位 */
 delay(1);
  outportb (DMA_PORT +0x4, (unsigned char) (buff >> 8&0xff));  /* 写接收缓冲区地址的 A9~A16 位 */
 delay(1);
 outportb (DMA_PAGE,(unsigned char)(buff>>16&0x7));
/* 写接收缓冲区地址的 A17~A19 位到 DMA 页面寄存器 */
 delay(1);
 length=RECEIVE_LENGTH-1;
 outportb (DMA_PORT+0x6,(unsigned char)(length&0xff));
delay(1);
 outportb (DMA_PORT +0x6, (unsigned char) (length >> 8&0xff));/* 将传输字长度分先底后高字节写到 DMA 控制器 */
 delay(1);
 enable();  /* 开中断 */
 if (((buff+length)&0x70000)==(buff&0x70000))
    return (0);
 else return (1);  /* 判断 DMA 是否越界,若没有则返回 0 值,否则返回 1 值 */
}
void interrupt getdata(void)  /* 中断服务程序 */
{
    asm sti  /* 开中断 */
    asm lea bx,_receive_buff  /* 接收缓冲区首址保存到 BX 寄存器 */
    asm lea di,_sort_buff  /* 处理缓冲区首址保存到 DI 寄存器 */
    asm out DMA_PORT+18h,al  /* 清 DMA 控制器中的高低触发器 */
    asm jmp short $+2  /* 延时满足 DMA 控制器的定时要求 */
    asm in al,DMA_PORT+4
    asm jmp short $+2
    asm xchg ah,al
    asm in al,DMA_PORT+4
    asm jmp short $+2
    asm xchg ah,al
    asm mov si,ax  /* 分两次读入 DMA 控制器中的当前地址寄存器中的值并保存到 SI 寄存器中 */
    asm mov cx,HALF_RECEIVE  /* 传输数据长度送到 CX 寄存器 */
    asm dec si
    asm shl si,1
aa: asm cmp si,bx
    asm jnc bb
    asm add si,RECEIVE_LENGTH*2  /* 比较传输地址是否达到接收缓冲区首址,否则从后开始传输 */
bb: asm mov ax,word ptr [si]
    asm mov word ptr [di],ax
    asm dec si
    asm dec si
    asm inc di
    asm inc di
    asm loop aa  /* 循环直到传输完毕 */
    asm mov ax,1
    asm mov word ptr _int_flag,ax/* 置中断传输完标志 */
    asm mov al,EOC
    asm out INT_PORT0,al  /* 送出中断结束标志 */
    asm cli  /* 返回前关中断 */
}
```

参考文献

1 张载鸿.微型机(PC 系列)接口控制教程.清华大学出版社,1994

2 尹彦芝.C 语言高级实用教程.清华大学出版社,1994

[收稿日期:2001.12.24]