研究与设计

基于USB总线的焊接过程瞬态信号检测

朱六妹,李广辉,杨先林,王 伟
(华中科技大学 现代焊接技术研究中心,湖北 武汉 430074)

摘要:介绍了基于USB2.0总线高速实时数据采集系统的硬件系统和工作原理,阐述了USB固件程序编写方法以及在VC++6.0环境下编写客户端驱动程序和应用软件的设计思路,实现了焊接过程瞬态电流电压信号的高速采集、实时显示、数据存储和分析。

关键词:数据采集;USB;固件程序;WDM驱动程序;多线程
中图分类号:TG409 **文献标识码**:A **文章编号**:1001-2303(2007)01-0035-07

Signal detection based on USB2.0 bus in welding process

ZHU Liu-mei, LI Guang-hui, YANG Xian-lin, WANG Wei
(Welding Research Institute, HuaZhong University of Science and Technology, Wuhan 430074, China)

Abstract: This paper introduces a high speed and real time data acquisition system based on USB2.0 bus. It systematically discusses the achievement of the hardware and software of the system, and explains the programming methods of USB firmware, USB client driver and software. It realizes real time display and data conversation of the signal of current and voltage generating from the welding process on PC.

Key words: data acquisition; USB2.0; firmware; WDM driver; multi-thread

焊接过程包含丰富的电、光、声等信息,如何实现高速数据采集、存储和显示这些信号,是分析和控制焊接过程的技术关键。常用的方法是通过PC机上的数据采集卡实现数据采集,但其存在采集速度较低、数据点丢失、在一些电磁干扰性强的测试现场无法专门对其做电磁屏蔽而使数据失真、安装麻烦、价格昂贵、可扩展性差等缺点。采用微控制器(嵌入式单片机、DSP、工控机)对动态过程进行数据采集,通过计算机接口将数据传至PC机进行在线或离线的分析、加工和处理,可以充分发挥微控制器和PC机的技术特点,是目前较为先进的信号处理技术。USB(Universal Serial Bus通用串行总线)是为解决计算机外设种类日益增加与有限的主板插槽和端口之间的矛盾,由Intel、Microsoft等多家公司提出制定的工业总线规范[1],目前,USB2.0支持的最高传输速率可达480 Mbps,很好地解决了主机与外设之间的数据传输瓶颈问题。USB作为较新的计算机接口技术,以其高速、即插即用、数据传输可靠和兼容性好、低成本等突出优点,得到越来越广泛的应用。

本研究结合USB接口技术,采用数字信号处理器DSP作为核心控制器,提出了一个基于USB 2.0总线的实时数据采集系统的具体方案。该系统实时性好、功能强,能实时采集16路模拟信号,具有最高2 MHz的采样频率和10位的分辨率,能满足各种弧焊过程瞬态信号的工业现场检测和分析。

收稿日期:2006-07-28;**修回日期**:2006-12-05
基金项目:国家高技术研究发展计划(863计划)资助项目——高温材料等离子激光复合快速成型技术(2003AA305800)
作者简介:朱六妹(1947—),女,上海人,副教授,主要从事焊接过程智能控制以及CAD设计工作。

1 检测系统硬件

1.1 系统结构和工作原理

系统结构如图1所示,由传感器(如霍尔电流、电压传感器)接收电路、信号调理电路、A/D转换电路、DSP控制器和USB接口组成。USB2.0协议由USB接口芯片完成。USB固件程序嵌入到DSP主循环数据采集程序中,其作用是当PC枚举(USB检测过程)、配置USB外设时,DSP接收和发送相关的USB设备信息;配置USB设备成功后,根据接收到的命令进行相应的操作。当USB接口芯片检测到PC起动的某一传输请求时,便通过中断方式将此请求通知DSP,DSP访问USB接口芯片相应的状态寄存器,获得与此传输有关的各种参数,并根据具体传输参数对USB接口芯片的控制器和数据寄存器进行相应操作,以完成与PC机数据传输的请求。

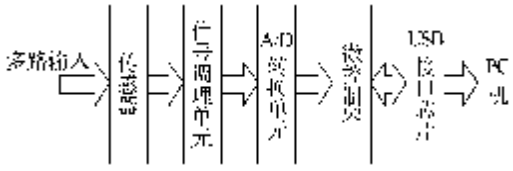

图1 系统工作原理

Fig.1 Structure block diagram of system

1.2 系统硬件设计

采用PHILIPS公司封装了USB2.0协议的接口芯片ISP1581,其价格低、功能强,具有完整的USB2.0规范。既支持高速(480 Mbps)操作,又支持全速(12 Mbps)操作,在底层(硬件电路)完成了USB2.0/1.1物理层(包括模拟收发器、串行接口引擎、串并数据转换等)和数据协议层(通信规则)的任务,并实现了连同端点EP0(用于访问设置缓冲器)在内的16个USB端点的共同协作[2]。ISP1581通过一个高速的通用并口与DSP实现高速通信,内部通用DMA模块使得数据流方便集成,能适应大多数设备类规范的设计。

微控制器选择TI公司定点DSP芯片TMS320LF2407A,其内嵌了数字马达、位移控制功能,运算速度为40 MIPS,具有32 kB的FLASH存储空间,提供了高性价比的可编程解决方案,并内嵌了256 B大小的引导ROM以方便对电路进行编程;内带高性能的10 bit的A/D,提供16路模拟信号输入,最小转换时间500 ns[3],所以无需外扩模数转换器。

TMS320LF2407A与USB接口芯片ISP1581的硬件连接如图2所示。

TMS320LF2407A DSP控制器的模拟数字转换模块提供了20个用于与外部电路接口的引脚,其中,ADCIN00~ADCIN15共16个引脚用于模拟信号的输入;其他4个引脚分别为VREFHI、VREFLO、VCCA和VSSA。模拟数字转换模块正常工作时需要小于等于3.3 V的基准电压,该电压由外部电源提供,直流基准电压可通过基准电压输入引脚VREFHI和VREFLO得到;而引脚VCCA和VSSA必须分别接到3.3 V直流电源和模拟地上,连至VCCA和VSSA的模拟引线应该尽可能短,以使两者正确匹配,同时还应该使用其他减噪技术以确保精确转换。下列方程近似给出了转换后得到的数字结果[4]

$$\text{数字结果}=1\,023\times(\text{模拟输入电压}-U_{\text{REFLO}})/(U_{\text{REFHI}}-U_{\text{REFLO}})$$

电弧电信号在ADC转换之前,要先进行一定的处理,使其满足ADC模块A/D转换器的输入电压条件。电弧电压直接取自焊嘴和工件之间,起弧后一般在0~40 V之间,电流信号由分流器检测,所得信号很小,只有0~75 mV,两信号均不在A/D转换器允许的输入电压(0~3.3 V)范围内。因此必须在ADC之前加信号预处理电路,以完成信号的放大、缩小和隔离功能。

ISP1581采用通用处理器模式,采用8051方式读写选通信号[2],由DSP作为微控制器。两者在选定工作方式下的信号连线如图2所示,ISP1581的16根数据线直接与DSP的数据线相连,存储管理单元MMU和集成RAM作为USB的缓冲区,允许DSP以自己的速率对USB信息包进行读写。ISP1581的片内寄存器映射到DSP的片外地址空间中,DSP通过8位地址线选择要访问的寄存器,在读写信号选通的控制下,利用16位数据线与选定的寄存器交换数据,在访问单字节寄存器时,高字节内容忽略。ISP1581通过中断引脚INT向DSP报告发生的事件。利用D+、D-与主机进行数据交换。

2 系统软件设计

软件系统的设计包括USB固件程序、USB设备驱动程序和界面应用程序三部分。

2.1 固件程序设计

2.1.1 固件工作原理

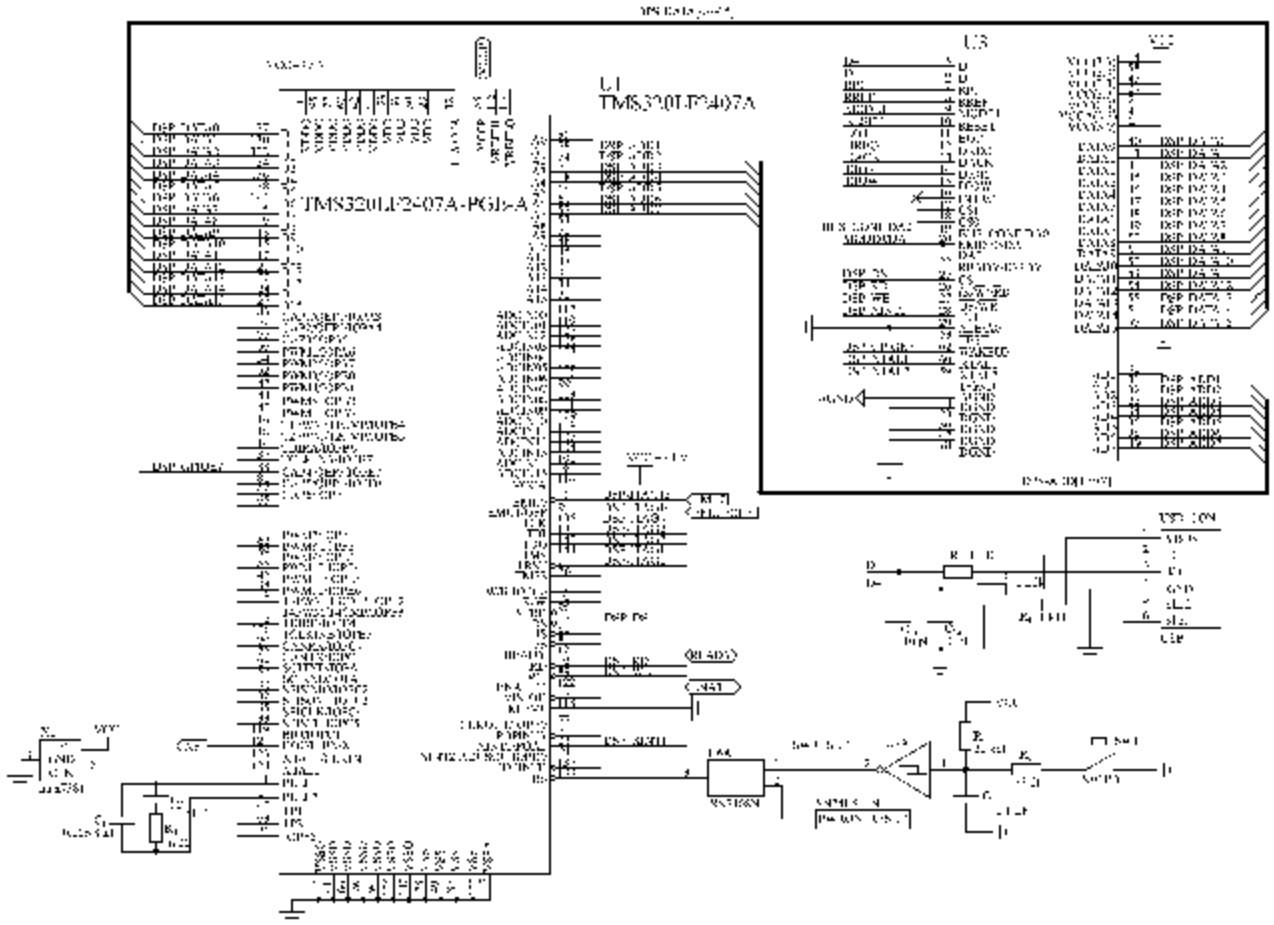

图 2　ISP1581 与 DSP 连接电路

Fig.2　Interface circuit diagram of USB and DSP

ISP1581 固件程序运行在 DSP 上,是设备运行的核心,也是系统设计的重点和难点。固件程序可分为主程序、中断服务程序和请求处理程序三部分[5]。其中,主程序用来循环查询一些标志(如 A/D 转换是否完成、是否进行处理、数据是否存储等),一旦标志有效,就进入相应的子程序进行处理;中断服务程序用于处理 ISP1581 产生的硬件中断,同时根据中断的类型不同将其标志置位或清除;请求处理程序负责完成枚举阶段收到的标准请求以及在正常工作阶段收到的自定义请求等。

2.1.2　固件流程和组成文件

C 语言编制的固件文件结构如图 3 所示。

(1)主循环流程(Mainloop.c)。

主程序流程如图 4 所示。上电后,初始化 DSP 和 ISP1581。然后,主循环程序轮询事件标志,进入相应的子程序进行下一步的处理。

(2)中断服务程序(ISR)流程(Isr.c)。

中断服务流程用以处理由 ISP1581 产生的中

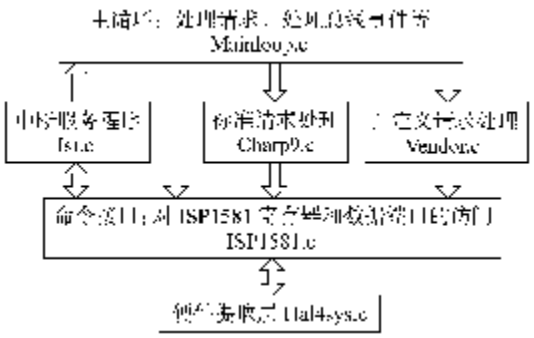

图 3　固件文件结构

Fig.3　Structure of firmware files

断。当有中断产生时,由 DSP 判断中断类型进行判断,通过访问 ISP1581 的中断寄存器,建立正确的事件标志,以通知主循环程序进行处理。

(3)USB 标准请求处理流程(Charp.c)。

在进行数据传输前,主机必须枚举设备。该过程通过给端点 0 发送标准请求的控制传输来实现。USB 标准请求流程如图 4 所示,翻译设备请求类

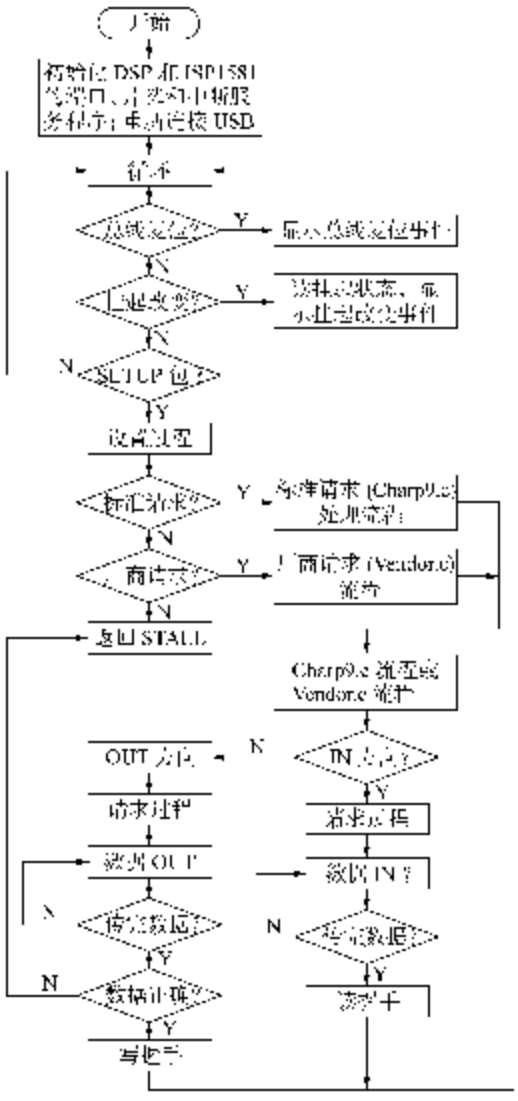

图4 主程序流程

Fig.4 Proceeding of main circuit

型,转到相应的处理子流程。枚举过程如下:

①主机使用默认地址(地址0)读取设备描述符GetDeviceDescriptor;

②主机使用SetAddress为设备分配唯一的地址;

③连续3次调用GetDeviceDescriptor,读取全部设备描述符、配置描述符、接口描述符和端点描述符;

④主机调用GetConfigDescriptor获得设备当前配置;

⑤主机调用GetStringDescriptor获取设备字符串描述符;

⑥读取全部配置描述符后,主机操作系统将找到新设备,并提示安装驱动程序;

⑦在设备能通信前,主机给出SetCongfiguration请求,为设备分配资源,设备调整有关信息后,才能被客户软件利用。

(4)自定义请求处理(Vendor.c)。

自定义请求和USB标准请求一样,都根据控制传输的内容进行相应处理。本系统的固件程序中定义了两个自定义请求,分别为取得固件版本和将DSP采集到的数据以批量传输方式从设备传至主机。

(5)ISP1581.c。

包含了对ISP1581接口进行操作的模块子程序。

(6)Hal4sys.c。

包含了对DSP数据总线、地址总线等硬件接口的操作。

2.2 设备驱动程序设计

2.2.1 驱动程序模型选择

在Windows环境下,设备驱动程序有VxD(Virtual Device Driver)、KMD(Kernel Mode Driver)和WDM(Win32 Driver Model)三类。其中,WDM类型的驱动程序采用了Microsoft目前大力推行的全新的Win32驱动程序模式,可在Windows 98/Me/NT/2000/XP等操作系统平台下运行。USB接口设备驱动程序是基于USB总线驱动程序之上的,它在系统软件中的地位和它支持的电源管理、即插即用新特性,使得它必须采用全新的Win32驱动程序模式实现,即USB接口设备的驱动程序必须是一个WDM驱动程序。

2.2.2 驱动程序开发

USB设备驱动程序的开发工具有Windows DDK,KRFTech公司的WinDriver,Compuware公司的Driver Works等。其中DDK功能强大,但是要求开发人员对操作系统比较熟悉,开发难度比较大;WinDriver使用非常方便,开发周期短,但程序效率低,不能满足性能较高的USB设备。DriverWorks使用比较方便,功能强大,是较为理想的开发工具。本设计选用DriverWorks开发驱动程序,VC++6.0作为程序编辑编译环境,SoftICE作为单机调试环境。

DriverWorks对微软DDK的库函数进行封装,提供了DriverWizard代码生成工具,使得程序编写更加容易,由DriverWinzard生成的驱动程序框架类似VC++的MFC,对其进行少量修改即可使用。DriverWorks提供了KUsbLowerDevice、KUsbPipe和KUsbInterface三个基类支持USB设备驱动程序的编写[6],KUsbLowerDevice类使驱动程序存取设备的

缺省默认控制端点实现设备的信息配置,对设备发送各种控制信息和状态请求;KUsbInterface 类描述了一个 USB 接口,驱动程序使用 KUsbInterface 类的实例来表示接口、管理设备管道的存取;KUsbPipe 类描述了管道的功能,实现数据的传输,在一个确定的配置中,每个管道和特定的接口链接。需要注意的是:驱动程序可以为每个管道声明类实例,但并不为默认控制管道(对应 0 端点)声明实例,而是由类 KUsbLowerDevice 来实现这个管道的功能。

2.3　应用程序设计

2.3.1　VC++6.0 多线程技术

对于本数据采集系统,需要实时观察已采集的数据,但采集速度快、数据量较大,这就需要几秒或更长的时间才能全部显示出来,若此时正在进行数据采集,则数据的更新显示和后台数据采集可能在时间上会产生冲突,影响程序的正常运行[7]。为解决这个问题,在程序中需采用多线程技术,有效利用 CPU 等待时间,加快程序的反应速度,增加执行的效率。用一个辅助数据采集线程负责系统与 USB 设备通信和缓冲区管理,主线程进行用户界面数据更新显示和存储等处理。

辅助线程通知主线程更新数据有两种方法:采用定时器定时查询或者利用消息通知的方法[8]。由于 Windows 系统本身为多任务系统,定时器的控制精度只能达到 55 ms,本系统的采样频率可达到 2 MHz,数据传输速度 24 MB/s(高速),而显示器屏幕的刷新频率(一般为 85 Hz)在一定时间内所能更新的数据点数又有限,采用定时器服务函数必将造成数据点丢失,所以只能采取消息通知的方法。但采用消息机制又容易造成堵塞,主线程可能会处理一些耗时的操作,来不及对消息进行处理。为避免消息阻塞,在程序中设置了几个标志,当有相同自定义消息没有得到处理时,则将采集线程挂起,暂停向主线程发送消息。同时,为了实现任何情况下对数据采集与显示的实时性操作,将系统进程设置为高优先级,数据转储线程设置为高优先级。Win32 对进程和线程的优先级设置提供了相应的 API 函数。

通过利用消息处理机制,在保证数据实时采集的同时,也可保证对数据的显示和存储及时处理,使整个程序尽可能协调运行。

2.3.2　高速数据缓冲存储技术

高速数据采集一般都采用连续采集的方式,在这种情况下,驱动程序将采集到的数据依次转移到用户缓冲区,当用户缓冲区满了之后,下一采集的数据将覆盖用户缓冲区中原来数据,如此循环,因此,在高速数据采集时如何准确、高效地读出并保存用户缓冲区中的数据是一个关键问题。解决方法是采用双缓冲区:假设用户缓冲区分为两个大小相等的缓冲区 1 和 2,当驱动向缓冲区 1 导入数据时,主线程读取缓冲区 2,更新显示;当驱动程序向缓冲区 2 转移数据时,主线程读缓冲区 1,如此循环。程序流程如图 5 所示。

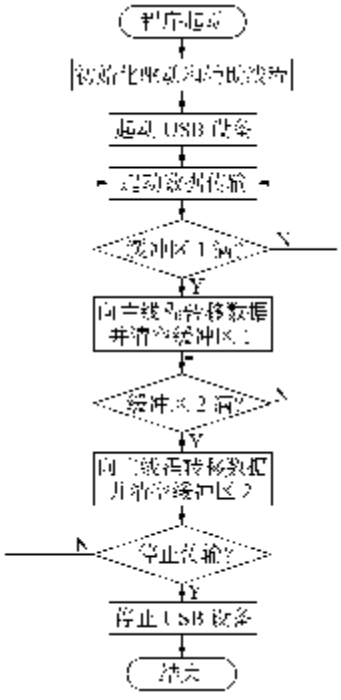

图 5　读数据流程

Fig.5　Receiving data proceeding

2.3.3　实现举例

```
UINT ReadThread(LPVOID pParam)
{//异步打开设备
    HANDLE IOWaiter;
    ULONG  nItems,nBytesRead;
    CString msg;
    const int nDataCount=3000;
    int nCount=0;
    //初始化链表
    U_DataList1.RemoveAll( );
    U_DataList2.RemoveAll( );
```

```
    //创建手动重置事件,异步方式调用 I/O
    IOWaiter=CreateEvent(NULL,TRUE,FALSE,NULL);
    if (IOWaiter=NULL)
{
        ……错误处理
  CloseHandle(m_hDevice);
    return 1;
    }
   OVERLAPPED ol; //OVERLAPPED 结构对象初始化
    ol.Offset=0;
    ol.OffsetHigh=0;
    ol.hEvent=IOWaiter;
    KeepRunning=TRUE;
    ResetEvent(IOWaiter);
//循环读数据
while (TRUE)
{//判断缓冲区 1 是否为空,为空则向其写数据
  if (U_DataList1.IsEmpty( ))
  {
    int nTemp1[nDataCount];
    //DeviceIoControl 异步调用读数据
    if (!DeviceIoControl(m_hDevice,IOCTL_READ_UDATA,
NULL,0,nTemp1,sizeof(nTemp1),&nBytesRead,&ol))
    {
    if (GetLastError( )!=ERROR_IO_PENDING)
    {//返回错误
     ……错误处理
    goto EXIT;       //跳出线程
    }
     }
While (WaitForSingleObject(IOWaiter,100)=WAIT_
TIMEOUT)
{ //正常超时
 if (!KeepRunning)
 {
    CancelIo(m_hDevice);  //取消 I/O 操作
    goto EXIT;
 }
}
for (int i=0;i<=nDataCount;i++)
 { //将 dsp 定点数据数据并存入缓冲区链表 1
  CUData* pUData=new CUData( );
  pUData->m_nData=nTemp1[i];
  U_DataList1.AddTail(pUData);
 }
//向视图发送消息 WM_UPDATEVIEW 更新显示
PostMessage (ViewHwnd,WM_UPDATEVIEW,0,0);
}
 //判断缓冲区 2 是否为空,不为空则向其写数据
 if (U_DataList2.IsEmpty( ))
 {
  int nTemp2[nDataCount];
  //读操作同上
 {略}
 While (WaitForSingleObject(IOWaiter,100)=WAIT_
TIMEOUT)
 { //正常超时
 if (!KeepRunning)
 {
    CancelIo(m_hDevice);  //取消 I/O 操作
    goto EXIT;
  }
 }
for (int i;i<=nDataCount;i++)
 {//将 dsp 定点数据数据并存入缓冲区链表 2
  CUData* pUData=new CUData();
  pUData->m_nData=nTemp2[i];
  U_DataList2.AddTail(pUData);
 }
PostMessage(ViewHwnd,WM_UPDATEVIEW,0,0);
 }
//若缓冲区中数据都不为空,则线程睡眠 10 ms
Sleep(10);
}//退出线程
  EXIT:  CloseHandle(m_hDevice);
         CloseHandle(IOWaiter);
         return 0;
}
```

3 实验结果

CO_2 焊接过程中电流、电压信号的采集和结果瞬时显示界面如图 6 所示。实验条件:I=100 A;U=19 V;焊丝直径 D=1.2 mm;干伸长 L=15 mm;焊速

v=35 cm/min;气体流量 Q=16 L/min。A/D 转换频率 2 MHz,数据传输速度 24 MB/s(高速),软件包括了对 USB 设备的操作、USB 状态显示、信号类型选择、刷新率选择和数据保存选项功能。由于 Windows 系统本身并非实时系统,故没有对信号进行实时分析处理,分析处理算法的复杂性增加了线程运行时间,更可能造成线程阻塞,使程序失去稳定性。图中原始波形毛刺较大,难以观察信号变化,不利于分析和判断信号特征。程序实现了数据保存功能,这样使得数据的过后再现和静态分析更为容易。静态分析模块采用 Matlab 提供的 VC++6.0 编程接口实现了滤波、小波信号特征提取、小波信号降噪等信号分析和处理方法,如图 7、图 8 所示。

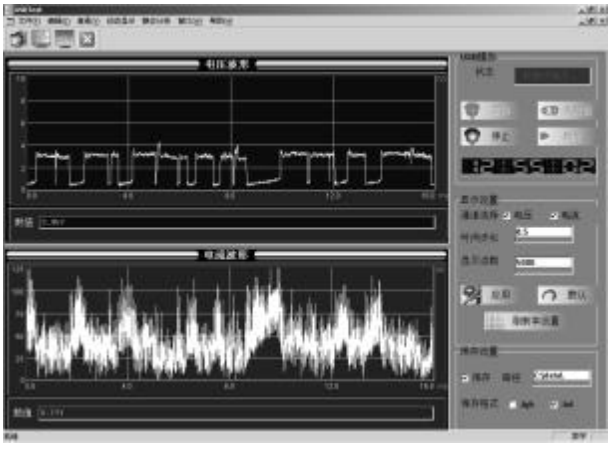

图 6　实时显示界面

Fig.6　Real time display interface

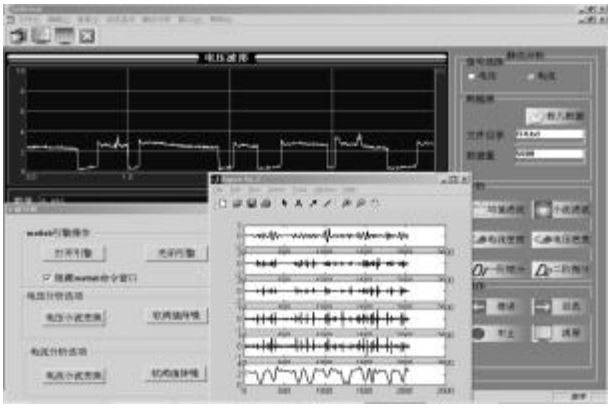

图 7　信息小波特征提取

Fig.7　Distrilled character signal by wavelet

4　结论

(1)通过研究以 DSP 芯片为控制器的 USB 总线数据采集系统,实现了焊接过程的实时监控。数据采集频率可达 2 MHz,数据传输速度可达 24 MB/s,充分发挥了 DSP 高速信号采集和 USB 总线高速数据传输的特性,并将其应用于 CO_2 焊接过程信号检测。

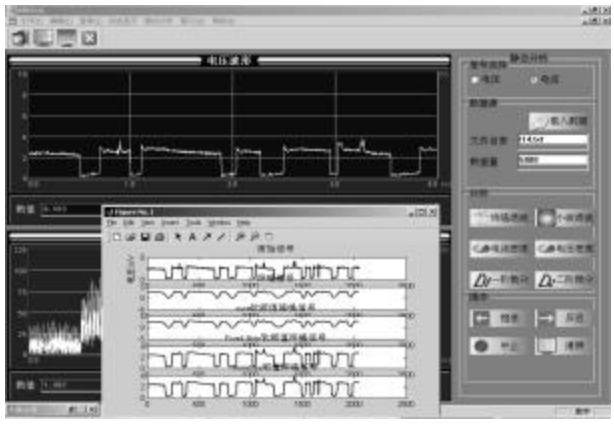

图 8　信息小波降噪

Fig.8　Filter signal by wavelet

(2)在 DSP 内实现了 USB 固件程序,并在上位机利用微软最新驱动程序开发模型 WDM 实现对 USB 设备驱动程序的编写,从而完成了对 USB 设备的配置和数据传输功能。

(3)利用多线程和双缓冲编程技术实现了数据的间隔刷新。上位机应用程序成功完成了数据曲线的动态显示任务,并实现了数据保存、滤波、小波分析等静态分析功能。

(4)该数据采集系统可广泛应用于焊接过程中对电信号的采集,摆脱了只能用数据采集卡或示波器才能对信号进行检测的限制,为焊接过程信号检测提供了新的检测手段

参考文献:

[1] Universal Serial Bus Specification Rev2.0[S].Apr.2000.

[2] Philips Semiconductors Inc.ISP1581 hi-Speed Universal Serial Bus interface device Rev.05[DB/OL].Feb.2003.

[3] 刘和平.TMS320LF240X DSP 结构、原理及应用[M].北京:北京航空航天大学出版社,2002.

[4] 孙　广,白日辉,何建萍,等.短路过渡 CO_2 焊的数字化波形控制[J].电焊机,2004,34(增刊):88-90.

[5] Philips Semiconductors Inc.ISP1581 Programming Guide Rev 1.0[DB/OL].Mar.2002.

[6] 武安河.Windows2000/XP WDM 设备驱动程序开发[M].北京:电子工业出版社,2005.

[7] 刁修民,刘亚斌.在 Visual C++环境下实现高速数据采集的几个问题[J].计算机测量与控制,2003,11(2):131-134.

[8] 乔　林,杨志刚.Visual C++6.0 高级编程技术[M].北京:中国铁道出版社,2000.