文章编号:1005－0523(2005)05－0110－04

DSP 平台的 USB 接口设计及系统开发

凌仕勇，黄兆华

(华东交通大学 信息工程学院,江西 南昌 330013)

摘要:介绍一种基于 DSP 平台,以 USB 接口方式实现用于继电器检测与保护装置的通信方式的实现.它采用 Philips 公司的 ISP1581-BD 接口芯片,实现 DSP 数据采集与 PC 机的高速传输.并介绍了在设计开发 USB 外设时主机应用程序及设备驱动程序的方法及代码.

关 键 词:USB 设备驱动程序 ISP1581_BD

中图分类号:TP391.4　　**文献标识码**:A

1 引 言

ISP1581 是一种价格低、功能强的通用串行总线(USB)接口器件,它完全符合 USB2.0 规范,并为基于微控制器或微处理器的系统提供了高速 USB 通信能力.ISP1581 与系统的微控制器/微处理器的通信是通过一个高速的通用并行接口来实现的.它符合现有的大多数器件的分类规格,比如:成像类、海量存储器件、通信器件、打印设备以及人机接口设备.

内部通用 DMA 模块使得数据流很方便的集成.另外,多种结构的 DMA 模块实现了海量存储的应用.这种实现 USB 接口的标准组件使得使用者可以在各种不同类型的微控制器中选择出一种最合适的微控制器.此外,ISP1581 内部还集成了许多特性,包括 SoftConnectTM、低频晶体振荡器和集成的终止寄存器.所有这些特性都为系统大大节约了成本,同时使强大的 USB 功能很容易地用于 PC 机外设.

在应用 USB 设备的开发时,设备驱动的开发是一个难点,很多国内开发者就采用 USB 设备厂商提供的驱动程序作自己的设备驱动,但因设备厂商根本不提供驱动源代码,这样开发者就无从知道具体的控制 IO 及应用接口 GUID,也就无法应用到自己的上位机应用程序中去,且这些驱动仅实现了一些最基本的通信功能,开发者必须自己根据具体应用设计自己的设备驱动.在本系统中,我采用 Driver-Studio 来开发自己的驱动,有效地解决了应用中的诸多实际问题如各种访问控制,上下位机接口的一致性问题等等.

2 系统设计

系统设计包括硬件方案设计与 USB 设备的软件设计.而 USB 设备的软件设计主要包括两部分:一是 USB 设备端的软件,主要完成 USB 协议处理与数据交换(多数情况下是一个中断子程序)以及其它应用功能程序;二是 PC 端的程序,由 USB 通信程序和用户服务程序两部分组成,用户服务程序通过 USB 通信程序与系统 USBDI(USB 设备接口)通信,USB 通信程序开发难度比较大,可用 DDK 或 Driver-Work 等开发工具开发.

2.1 USB 芯片与 DSP 的硬件连接

本方案以 TI 公司的 TMS320C5XX 作为微控制

收稿日期:2005－06－12

作者简介:凌仕勇(1974－),男,江西九江人,硕士研究生,研究方向为人工智能.

中国知网 https://www.cnki.net

器，采用外部供电方式，连接电路如图 1 所示.

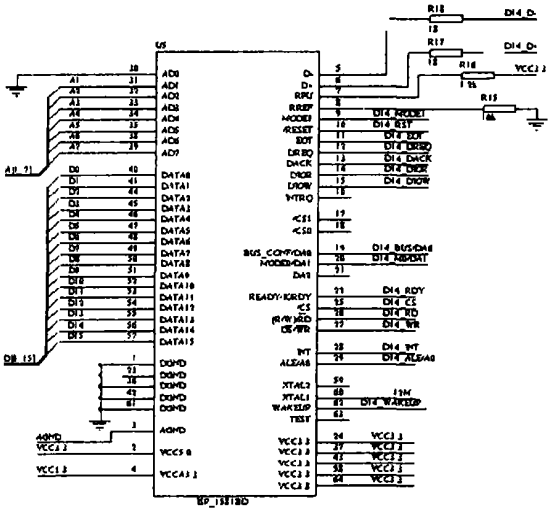

图 1　ISP1581－BD 与 DSP 硬件连接图

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，允许微控制器以自己的速率对 USB 信息包进行读写. D＋，D－信号线上串接 18Ω 电阻. XTAL1 接 12MHz 晶振.

2.2　固件程序设计

由于采用的是不带微控制器内核的 USB 接口芯片，所以关于 USB2.0 协议规范的实现都必须靠 DSP 控制 ISP1581 芯片来完成. 固件的主要设计任务是：在 DSP 的平台上编写程序，以完成 USB2.0 规范所要求的标准请求及用户根据产品需要自己定义的请求. 为了不影响程序的执行效率，本方案采用中断方式完成固件的编写，同时，为了保证程序的模块化及良好的可移植性，在设计中采用分层结构进行固件的编写. 详细方法可参见文献[1].

在本方案中，采用端点 2 以批量方式来与上位机进行通信，采用端点 0 来进行除设备枚举之外的设备命令控制字的处理，这在下面的设备驱动程序中将会谈到. 因为对于继电器来说，有很多标志位需要传送；且因 USB 是单向控制的，故在每次读写数据时均需首先发送一命令控制字，然后再通过 API 读写函数进行双方通信. 所以我在此设一个 IO 请求结构：struct IOREQ｛unsigned short wValue；unsigned short wIndex；｝，wValue 代表 16 位标志位，wIndex 代表需传送的字节数，这个结构命令传送的驱动需要自己在设备驱动程序中实现(通过设备类请求的方式，见 2.3 述)，且在上层应用程序中通过 DeviceIoControl(…)函数，通过驱动的类请求传给 USB 设备端点 0，USB 设备则以类设备请求方式处理这个设备控制命令.

本驱动中采用 0x00 作为 USB 设备写命令的请求类型号，0x01 作为 USB 设备读命令的请求类型号，设备固件中处理 IO 控制请求的代码如下：

```
unsigned char type, req;
type = ControlData.DeviceRequest.bmRequestType & 0x60;//请求类型
req = ControlData.DeviceRequest.bRequest & 0x0F;//请求号
  if (type == 0x00)
    (*StandardDeviceRequest[req])();//调用标准请求
  else if (type ==0x40)//类请求
    {if (req==0x00) Handle_Write_DeviceIo();//处理写控制命令
     if(req==0x01) Handle_Read_DeviceIo();//处理读控制命令
    }
```

2.3　PC 端的 USB 驱动程序设计

DriverStudio 是一套用来简化微软 Windows 平台下设备驱动程序的开发、调试和测试的工具包. DriverStudio 当前的版本包括 DriverWorks、SoftICE 及其它工具模块. DriverWorks：对于 Windows NT 下的 Windows 98 与 Windows 2000/XP 共同支持的 Win32 驱动模型(WDM)设备驱动程序的开发提供完全的支持. DriverWorks 中包含一个非常完善的源代码生成工具(DriverWizard)以及相应的类库和驱动程序样本，它提供了在 C＋＋下进行设备驱动程序开发的支持. SoftICE：SoftICE 是一个功能极其强大的内核模式调试器，它支持在配置一台单独的计算机或两台计算机下进行设备驱动程序的调试. DriverWorks 提供了三个与 USB 直接相关的类：UsbLowerDevice，KUsbInterface 和 KUsbPipe 类，用于实现 USB 设备操作. KUsbLowerDevice 类用于逻辑设备的编程，KUsbInterface 类用于接口的编程，KUsbPipe 类用于管道的编程. 最基本的例程有设备的启动，停止，卸载，读写，设备控制等例程[2].

DriverStudio 本身提供了一个设备驱动的框架，且可自动生成端点 2 的批量传输方式的源代码，故在此不讨论. 对于设备控制命令传送的驱动则需根据实际应用具体实现如下：采用设备类请求函数

```
PURB BuildVendorRequest(
PUCHAR TransferBuffer,//为驱动程序存放传输数据的内存区
ULONG TransferBufferLength,//传输的字节数，对应于类请求的 wLength
UCHAR RequestTypeReservedBits,//为类请求字节中的保留
```

中国知网 https://www.cnki.net

```
位
  UCHAR Request,//具体请求数值,对应于类请求的 bRequest
  USHORT Value,//对应于类请求的 wValue
  BOOLEAN bIn=FALSE,//TRUE 为输入:设备->主机,FALSE 为输出:主机->设备
  BOOLEAN bShortOk=FALSE,//TRUE 表示设备传输的字节数,可少于指定的字节数
  PURB Link=NULL,//为下一个传输的 URB
  UCHAR Index=0, //对应于类请求的 wIndex
  USHORT Function=URB_FUNCTION_VENTOR_DEVICE,//类别的请求
      PURB pUrb=NULL//指向一存在的 URB
  );
```

实现的方法是对于读请求 IOCTL_READ 及写请求 IOCTL_WRITE, 将相应的类别代码通过此成员函数传送给底层驱动,实现 IO 控制命令请求.代码实现如下:

```
NTSTATUS USBDRIVERDevice::DeviceControl(KIrp I)
{
NTSTATUS status;
PURB pUrb;
struct IOREQ * Ioctrl;
switch (I.IoctlCode())
{
case IOCTL_READ://处理 IO 读请求
  PIRP pIrp=(PIRP)I;
  Ioctrl=(struct IOREQ *) pIrp->AssociatedIrp.SystemBuffer;//取缓冲区数据
  pUrb=m_Lower.BuildVendorRequest(NULL,0,0,0,Ioctrl->wvalue,TRUE,TRUE,NULL,
      Ioctrl->wIndex,URB_FUNCTION_VENDOR_DEVICE);//类请求,读请求
  if(pUrb==NULL)
    status=STATUS_INSUFFICIENT_RESOURCES;
  else
      {status=m_Lower.SubmitUrb(pUrb);delete pUrb;}
  break;
case IOCTL_WRITE: //处理 IO 写请求
  PIRP pIrp=(PIRP)I;
  Ioctrl=(struct IOREQ *)pIrp->AssociatedIrp.SystemBuffer;
//取缓冲区数据
  pUrb=m_Lower.BuildVendorRequest(NULL,0,0,1,Ioctrl->wvalue,TRUE,TRUE,NULL,
    Ioctrl->wIndex,URB_FUNCTION_VENDOR_DEVICE); //类请求,写请求
  if(pUrb==NULL)
    status=STATUS_INSUFFICIENT_RESOURCES;
  else
    {status=m_Lower.SubmitUrb(pUrb);delete pUrb;}
  break;
default:
  status = STATUS_INVALID_PARAMETER;
  break;
}
if (status == STATUS_PENDING)
  return status;
else
  return I.PnpComplete(this, status);//成功,提交 URB
}
```

另外,对于设备驱动的安装,还应修改设备配置文件中的安装程序类别和 GUID:Class=USB,ClassGUID={36FC9E60-C465-11CF-8056-444553540000},否则将无法识别出是 USB 设备.

2.4 PC 端的应用程序设计

通过调用 API 函数编写 PC 的应用程序.函数有 CreateFile(),CloseHandle(), DeviceIOControl(),ReadFile(),WriteFile().其中 DeviceIOControl()用于 PC(主机)向 DSP 发送请求,ReadFile()和 WriteFile()分别用于从 USB 设备中读出数据以及向 USB 设备中写入数据.在设计过程中必须注意的问题是:由于 USB 接口是主一从方式的接口,它的一切传输过程都必须通过主机向外设发送请求后才可以开始,所以在使用 ReadFile()、WriteFile()读写数据之前,必须先通过 DeviceIOControl()向 USB 设备发送请求,而且还须采用异步 IO 方式.例如,读数据的部分代码为:

```
struct IOREQ Ioctrl;//IO 请求的结构
OVERLAPPED ov;//异步 IO
memset(&ov,0,sizeof(OVERLAPPED));
ov.hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);//创建一事件
ResetEvent(ov.hEvent);//复位事件
unsigned long nBytes;
Ioctrl.wvalue=xx; Ioctrl.wIndex=xx;// 传送的标志位及要传输的字节数
bResult = DeviceIoControl(hDevice,IOCTL_READ,&Ioctrl,8,NULL,0,&nBytes,&ov);//启动  //读命令,
if (bResult! =TRUE)
  {
  if(GetLastError()! =ERROR-IO-PENDING)//IO 错误
    {
    MessageBox("请求数据传送失败! 已放弃","错误");
```

中国知网 https://www.cnki.net

```
    return 0;
    }
  switch(::WaitForSingleObject(ov.hEvent,20))//等待 20ms
    {
    case WAIT_OBJECT_0:
        if (!::GetOverlappedResult (hDevice, &ov, &nBytes,
TRUE))
          return 0;
          break;
       case WAIT_TIMEOUT://超时错误
        ::CancelIo(hDevice);
            return 0;
    default:
      return 0;
    }
  }
bResult = ReadFile(hDevice,...,&ov);// 从设备读数据
if (bResult! = TRUE)
{...//同上}
```

3　系统测试及结果

继电器测试与保护装置的交流实验部分测试结果如下图所示.实验证明，利用 USB 传输响应时间完全达到设定要求，即使在有 10 路 AD 数据的情况下也反应灵敏.对于本系统，在最后的测试当中，通过 USB2.0 接口，PC 机与 DSP 系统的通信速率最高达到了 850 KB/S 以上(约 7 Mbps).这个速率指的是有效数据传输速率，不包括数据传输联络的头信息部分且扣除了设备中断造成的影响，故此速率令人满意.

以有 10 路 AD 数据的情况下为例，每毫秒采集 10 路 16 位数据所要求的数据传输速率为(10×16×1000)/(8 * 1024)≈20 KB/S(远小于本系统的 850 KB/S)，由此可见，此方案的通信速率可完全满足要求，特别地，用 USB 传输方式，有效的解决了现场作业方便性的问题.

图 2　继电器交流实验模块

参考文献：

[1] 周立功，等. PDIUSBD12 USB 固件编程与驱动开发[M]. 北京：北京航空航天大学出版社，2003.

[2] 武安河，等. Windows 2000/XP WDM 设备驱动程序开发[M]. 北京：电子工业出版社，2003.

[3] 施诺，等[译]. Windows 2000 设备驱动程序设计指南[M]. 北京：机械工业出版社，2001.

USB Interface Designing And System Developing on DSP

LING Shi-yong，HUANG Zhao-hua

(School of Information Eng.，East China Jiaotong Univ.，Nanchanhg 330013，China)

Abstract：This paper introduces a communicating method of relay detection and protection with USB on DSP platform. It fulfils the high speed transmission between DSP and PC under the microchip ISP1581 of Philips. Also it talks about some methods and codes of host machine's application and device driver programs.

Key words：USB Device Driver；ISP1581_BD

中国知网　https://www.cnki.net