基于 TM1300 的 USB 接口设计与实现

Design and Implementation of USB Interface Based on TM1300

董文龙　蒋本珊　（北京理工大学 计算机科学工程系　100081）

摘要： USB 接口可分为 USB 主机和 USB 外设，总线采用主从工作模式，主机端控制数据传输，设备端只是被动地响应 USB 主机的访问请求。本文介绍了 TM1300 芯片，给出以 USB 主机控制器 ISP1561 为核心，在 pSOS 操作系统上实现的主机端接口设计方案，介绍了系统硬件设计，并分析了驱动程序的实现方法。

关键词： TM1300　ISP1561　pSOS　接口设计

1　引言

当今，在工业制造、信息通信及消费类电子产品中嵌入式系统得到越来越广泛的应用，同时系统提供多种接口与设备进行通信，通用串行总线（USB）接口就是其中之一。USB 接口提供了内置电源，能智能识别外围设备的插入或拆卸；一个 USB 控制器可以连接多达 127 个外设，而每个外设间距离（线缆长度）可达 5 米，支持多个不同设备的串行连接。由于常见的外设几乎都支持 USB，所以只要增加 USB 接口，就可以完成嵌入式主机与常见外设的通信。

TM1300 是 Philips 公司推出的系列多媒体 DSP Trimedia 的一款 32 位性能优良的多媒体处理芯片。它以多媒体处理和通信功能为主，特别针对数字视频和音频应用进行了结构优化；其核心处理器采用的是 VLIW（Very Long Instruction Word）超长指令字结构，可以在每一个时钟周期内同时进行 5 个操作。TM1300 处理器内部含有 128 个通用寄存器，这些寄存器不是分段的，所有操作都能使用这些寄存器。TM1300 使用 32 位线性寻址，寻址能力达到 4GB，同时为了解决高速 VLIW CPU 和低速 SDRAM 之间的数据交换瓶颈，内部集成了 16KB 的数据高速缓存和 32KB 的指令高速缓存，以确保 VLIW CPU 的全速运行，并且数据高速缓存是双端口的，允许同时进行双向访问。此外，TM1300 的二进制运行代码以压缩的格式存放在 SDRAM 和指令 Cache 中，压缩的代码一方面可以提高指令 Cache 的命中率，另一方面可以减少 Cache 与 SDRAM 之间的数据交换。指令 Cache 中有一个专门的指令解压机构，它负责解压缩指令并以 224 位的数据位宽向 VLIW CPU 提供指令。TM1300 提供 4 个通用的定时器，其中 3 个可被编程用来产生 CPU 时钟周期、数据/指令断点、Cache 跟踪、音频/视频时钟等等。在接口方面，它集成视频输入、输出接口，音频输入、输出接口，PCI 接口以及 JTAG 接口等外围设备，而 USB 主机控制器 ISP1561 内部集成了 PCI 接口，可连接在任何 32 位，33MHz 的 PCI 总线上，本文给出的设计方案能够实现 CPU 与接口芯片的无缝连接。

2　硬件接口设计

目前，市场上主要有三大类 USB 主控制器[4]，一种是支持由 Intel 公司最先提出的通用主控制器接口（Universal Host Controller Interface，简称 UHCI），另一种是支持由微软、康柏和国家半导体公司联合设计提出的开放主控制器接口（Open Host Controller Interface，简称 OHCI），还有一种是支持由康柏、惠普、Intel、lucent、微软、NEC 和 Philips 提出的增强主机控制器接口（Enhanced Host Controller Interface，简称 EHCI）。本设计选用的飞利浦 ISP1561 芯片，是符合 OHCI 和 EHCI 规范的 USB 主控制器。芯片内部通过 PCI 接口与 CPU 相连，并集成了一个根集线器（Root Hub），提供与 USB 电缆的连接。

2.1　Philips 的 ISP1561 芯片

ISP1561 符合 USB2.0 规范，提供的下游端口为 USB 各种速度的传输提供了单芯片解决方案，其主要特点有：

（1）OHCI 主控制器与设备传输速率为 12Mbit/s 或 1.5Mbit/s。

（2）EHCI 主控制器与设备传输速率为 480Mbit/s。当 EHCI 主控制器占用端口时，OHCI 主控制器不允许修改端口寄存器。

（3）动态端口路由策略使得 OHCI 主控制器和 EHCI 主控制器复用同一物理下游端口。两个寄存器位用来控制端口所有权的切换，当系统上电和复位时，OHCI 默认控制所有的下游端口。

（4）集成高速 USB 模拟收发器，通过内部终端电阻直接与 USB 电缆相连。

(5) 对下游端口支持独立供电切换和过流保护切换。

(6) 支持高级电源管理功能,并符合 PCI 总线电源管理接口规范。

每一个主控制器包含各自的片上操作寄存器组,驱动程序需要访问这些寄存器组来实现 USB 的数据传输。对于 OHCI 主控制器(HC),内部有以下几种类型的寄存器:

(1) HC 控制和状态类寄存器(Control and Status Registers)

(2) HC 地址指针类寄存器(Memory Pointer Registers)

(2) HC 计数类寄存器(Frame Counter Registers)

(3) HC 根集线器类寄存器(Root Hub Registers)

对于 EHCI 主控制器,内部有两类寄存器组:

① 一组只读的功能寄存器(Capability Registers)

② 一组可读写的操作寄存器(Operational Registers)

这些寄存器映射到系统可寻址空间的物理内存区,并提供了对 ISP1561 控制所需的各种类型的信息。

2.2 硬件总体框图

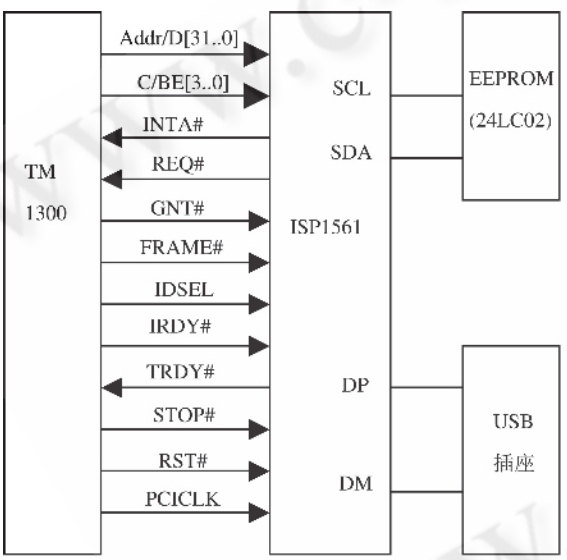

图 1　硬件总体框图

主要信号引脚有:

时钟和复位信号:PCICLK 与 RST#

传输控制信号: FRAME#、IRDY#、DEVSEL#、TRDY#、STOP#、IDSEL

地址和数据信号:A/D[31:0]、C/BE[3:0]、PAR

仲裁信号:REQ#、GNT#

错误类信号:PERR#、SERR#

SCL: I2C 总线时钟信号

SDA: I2C 总线数据信号

DP 与 DM: 实现与 USB 下游端口物理连接的差分数据线

3 驱动程序设计

TM1300 处理器支持 pSOS 实时多任务操作系统内核[2],pSOS 操作系统由美国 ISI(Integrated System Inc)公司开发。它基于开放式操作系统标准,并且针对多媒体应用作了专门的优化。pSOS 集成了一整套嵌入式软件模块、工具和服务,真正实现了抢先式、基于优先级的任务调度及合理的中断处理。

由于采用的芯片不带微控制器内核,所以关于 USB 协议和规范的实现必须通过 TM1300 控制 ISP1561 芯片来完成。为了保证程序的模块化和良好的可移植性,在 pSOS 系统中,驱动程序的设计采用分层结构,每一层都相对独立,并为上一层提供了屏蔽本层具体特征的接口。结构模型如图 2 所示。

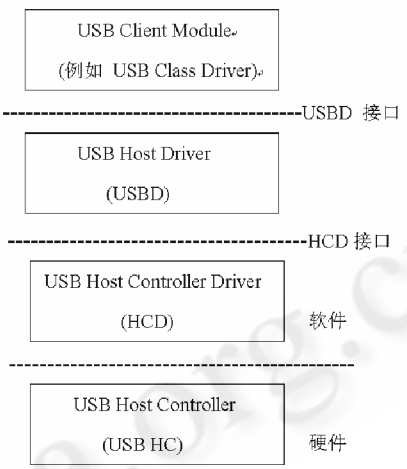

图 2　USB 主驱动栈分层结构

在整个栈结构中,使用数据结构 URB(USB request block)来协调各层的处理,它包含了建立任何 USB 传输所需的设备地址、端点号、数据缓冲区地址、缓冲区长度和传输方向等所有信息,其 C 语言设计如下:

```
typedef struc urb{
  unsigned long pipe; //位 7 表示传输方向,位 8 - 14 表示设备地址,位 15 - 18 表示设备端点
  void * data_buf; //数据缓冲区地址
  int * buf_len; //数据缓冲区长度
  int * data_len; //有效数据长度
  int status; //处理状态
  int timeout; //超时限制
}
```

最底层是硬件接口层,完成硬件上 TM1300 与 ISP1561

的对接,实现 USB 数据传输的机械特性和电气特性。

HCD 和 USBD 提供了基于不同抽象层次的软件界面,它们以一定的协同方式工作以实现 USB 系统的功能。HCD 是 USB 软件中的最下一层,提供了抽象的主机控制器,且对主机控制器所见到的 USB 系统的数据传输进行了抽象。该层驱动程序需实现:

(1) init_isp1561( )。初始化 ISP1561 函数,主要功能是分配数据结构的存储空间、地址指针类寄存器清零、计数器类寄存器设定初始值以及设置控制类寄存器并使能中断;

(2) ohci_interrupt( )。中断处理函数,主要功能是依据不同的中断状态做不同的处理,完成时要写状态类寄存器,清除当前中断状态标志位;

(3) usb_submit_urb( )。发送数据包函数,主要功能是完成数据封包,并请求芯片处理;

(4) usb_return_urb( )。接收数据包函数,负责接收设备传来的数据并提交给上层。

由于 CPU 和主控制器之间通过中断方式来交互,HCD 直接访问硬件,该层需要加载中断服务程序, TM1300 处理器有其自己的加载方式,C 语言伪代码可描述为:

```
setup_interrupt( ) //加载中断服务程序
{
  unsigned char irq =  inPCI_PINC; //定义中断引脚
  intInstanceSetup_t intr; //创建中断实例
  intr. enabled = TRUE;
  intr. handler = ohci_interrupt; //中断服务程序的地址
  intr. priority = intPRIO_2;
  intr. level_triggered = TRUE;
  ret = intOpen( (intInterrupt_t) irq);
  ……
  ret = intInstanceSetup( (intInterrupt_t) irq,&intr);
  ……
}
```

USBD 提供一个抽象的设备,且对 USBD 客户和 USB 设备功能部件之间的数据传输进行抽象。该层驱动程序需实现的功能有:

(1) usb_get_descriptor( ):读取并解析 USB 设备描述符和配置描述符;

(2) usb_set_address( ):为设备分配唯一的地址;

(3) usb_set_config( ):使用默认的配置来配置设备;

(4) 实现 USB 协议标准请求的接口函数,能够和设备交互;如 usb_get_status( ),set_clear_status( )等。

(5) 提供与高层驱动程序的接口。

USB Client 模块在 USB 主驱动栈的顶端,它负责管理连接到 USB 上的不同类型的设备。USB 客户端驱动程序依靠 USBD 来提供与每个设备的通信路径。该层驱动程序主要负责把通信命令以 URB 方式发给设备,并获取设备的工作状态,实现与设备进行数据传输,真正为应用程序提供读写,控制接口等功能。USB 工作组把设备分为不同的类型,如音频类设备,存储类设备,打印类设备等等。不同类型的设备需要用不同类型的命令格式。以存储设备类为例,其命令的数据封装结构 CBW(Command Block Wrapper)可描述为:

```
typedef struct usb_cbw{
  unsigned long signature; //命令块的识别字段
  unsigned long tag; //命令块标志字段,与状态块相对应
  unsigned long data_len; //数据的长度
  unsigned char cmd_len; //命令块长度
  unsigned char flags; //传输方向
  unsigned char operater; //命令代码
} USB_CBW;
```

其中各种命令的描述及含义如表 1[3]。

表 1

命令符	数据传输方向	命令代码	命令块长度	说明
INQUIRY	DATA_IN	0x12	0x06	设备查询
READ_FORMAT_CAPACITIES	DATA_IN	0x23	0x0A	读取格式化信息
READ_CAPACITY	DATA_IN	0x25	0x0A	获取容量信息
READ	DATA_IN	0x28	0x0A	读数据
WRITE	DATA_OUT	0x2A	0x0A	写数据
TEST_UNIT_READY	DATA_OUT	0x00	0x06	设备准备好

传输状态的数据封装结构 CSW(Command Status Wrapper)可描述为:

```
typedef struct usb_csw{
```

(下转第 88 页)

(上接第 91 页)

```
    unsigned long signature; //状态块的识别字段
    unsigned long tag; //状态块标志字段,与命令块相对应
    unsigned char cmd_sts; //返回的状态标志
}
```

由于提供了 USBD 和 HCD 两层协议,USB 客户端只是完成各种命令的解析和封装,这样既简化了驱动程序软件通信协议的复杂度,又使它与硬件的操作隔离开来,大大简化和设备通信的工作。

4 结束语

本文利用 TM1300 内部集成多种接口的特点,给出一种通过 PCI 总线连接 ISP1561 的硬件设计方案。在驱动程序的分层设计中,主要阐述了对 ISP1561 驱动的实现,上层模块的操作最后也都是通过调用该层提供的接口来完成的。本方案现已实现,当 USB 插座接入 USB 设备时,系统能够正确的检测到设备,读出存储在设备中的音频和视频数据,并且在监视器上播放,效果良好。

参考文献

1 Philips. ISP1561 Universal Serial Bus 2.0 PCI host controller Rev. 01 2003 -2.

2 Philips. TM1300 Data Book. 2001.

3 USB Mass Storage Class —Bulk Only Transport Rev. 1. 0 1999 -09.

4 埃科尔逊,USB 大全[M],中国电力出版社,2001 -05。

5 萧世文, USB 2.0 硬件设计[M],清华大学出版社,2002。