文章编号:1671-4598(2018)03-0158-04 DOI:10.16526/j.cnki.11—4762/tp.2018.03.040 中图分类号:TN919 文献标识码:A

CY7C68013A 端口模式下数据传输模块的设计

刘正发[1,2], 韦 飞[1], 冷 双[1], 王永松[1]
(1. 中国科学院 国家空间科学中心, 北京 100190; 2. 中国科学院大学, 北京 100049)

摘要: Cypress 公司的 USB2.0 控制芯片 CY7C68013A 可配置成 3 种接口模式: 端口、从属和 GPIF 主控模式; 从属模式和 GPIF 主控模式实现了 USB 内部数据缓冲与外部设备之间的无缝连接, 常用于高速实时传输外设; 而对于低速实时传输和高速非实时传输外设, 由 CPU 控制的端口模式则提供了有效的传输方案; 针对于端口模式文献介绍较少的现状, 同时为帮助开发者更好地理解 USB 数据传输过程, 提出了端口模式下数据传输模块的设计方法; 以芯片 CY7C68013A 为核心, 通过描述 USB 数据传输过程, 给出了端口模式下数据传输模块的通信协议设计、硬件设计、固件程序设计、驱动程序设计和上位机界面程序设计; 模块实现了计算机与外设数据的可靠传输, 测试表明, 满足了数据传输系统要求; 采用命令/响应式传输方式, 保障了数据稳定可靠传输, 具有很高的使用价值, 同时为其它接口模式的开发提供了借鉴意义。

关键词: CY7C68013A; 端口模式; 数据传输

Design of Data Transmission Module Based on the "Ports" mode of CY7C68013A Chip

Liu Zhengfa[1,2], Wei Fei[1], Leng Shuang[1], Wang Yongsong[1]
(1. National Space Science Center, Chinese Academy of Sciences, Beijing 100190, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Three interface modes, Ports, Slave FIFO and GPIF Master, are available for CY7C68013A which is a Cypress' s USB2.0 microcontroller. In Slave FIFO mode and GPIF Master mode, the data FIFOs in the USB connect directly to outside interface. And the both mode are usually used for a peripheral which requires high—speed and real—time data transmission. On the other hand, the Port mode where CPU participate provides an effective method for a peripheral which requires low—speed and real—time or high—speed and non—real—time data transmission. Because the Port mode was less paid attention in the previous literatures, and in order to help developers to better understand the details of USB data transmission, this paper introduced the design of data transmission module in detail. This module chose the chip CY7C68013A as the core of the design. Firstly, it described the USB data transmission process. And then, it focused on communication protocol design, hardware design, firmware design, driver software design, and host application design, which are covered in the module design. Reliable data transmission between the computer and peripherals had been achieved by this module. And the test results showed that this module can meet requirements for data transmission system. This module can ensured data be transmitted reliably with the use of command/response transmission protocol, and had high application value. And it had referential value for other interface modes design.

Keywords: CY7C68013A; ports mode; data transmission

0 引言

基于 USB 接口的数据传输系统因其连接方便、配置灵活、即插即用和支持热插拔等优点, 在数据采集、自动控制和电子测量等领域得到了广泛地应用[1]。Cypress 公司的 CY7C68013A 是一款高速的 USB2.0 接口控制芯片[2], 因其高度的集成性, 而得到了广泛的应用。

CY7C68013A 支持 3 种接口连接模式: 端口模式、通用可编程接口 GPIF 主控模式和从器件 FIFO 模式。在端口模式中, 所有的 IO 引脚均为通用的 IO 口, 固件程序简单, 数据传输需要 CPU 参与, 传输速率较低。后两种模式, 外设直接连接内部端点缓冲 FIFO, 不需要 CPU 参与, 具有很高的传输速率。GPIF 主控模式, CY7C68013A 作为主控制器, 固件程序复杂。从属 FIFO 模式, CY7C68013A 作为从器件被外部控制器访问, 固件程序简单, 但需要外部控制器的时序控制, 增加了硬件电路的复杂性。综合以上 3 种模式的优缺点, 在传输速率要求不高的场合, 端口模式可作为一种有效的传输方式[3]。但对于端口模式, 文献中介绍较少[3-4]。针对这种情况, 本文将介绍 CY7C68013A 端口模式下数据传输模块的设计, 并重点介绍其通信协议设计和软硬件设计。

1 系统总体设计

系统的任务是设计一个通用的数据传输模块, 来实现计算机与低速实时传输和高速非实时传输外设之间的数据交互。对应的系统连接框图如图 1 所示。上位机通过 USB 控制器接收外设接口电路发送的数据, 并进行显示和保存, 实现对外设的实时监测; 同时向外设发送命令数据, 实现对外设的功能控制。对应各部分的具体功能描述如下:

上位机界面程序: 控制系统的工作过程, 显示、保存接收的外设数据, 并向外设发送命令数据。

USB 接口电路: 实现 PC 机与 USB 接口控制芯片的物理

收稿日期:2017-07-06; **修回日期**:2017-08-29。
基金项目:科技部重大科学仪器开发专项(2012YQ130125)。
作者简介:刘正发(1990-),男,山西运城人,硕士研究生,主要从事地面检测系统方向的研究。

连接。

USB 接口控制芯片：采用 CY7C68013A 的端口模式实现与外部设备之间数据双向传输。

串行 EEPROM：采用 24LC00 提供设备的 VID/PID，用于设备的枚举和重枚举。

外设接口电路：实现模块与外设电路的连接，主要包含端口模式下与外设之间的通信协议信号。

其中，外设接口电路直接决定此数据传输模块的可靠性，是设计的重点，后面会给出详细的设计介绍。

2　硬件设计

2.1　CY7C68013A 芯片

CY7C68013A 在单芯片上集成了 USB2.0 收发器、智能串行接口引擎、增强型 8051 微处理器、16 KB 软配置 RAM、4 KB 端点缓冲区 FIFO 和可编程外设接口，主要用于 USB 主机与外设之间的数据传输。其中串行接口引擎用于完成大部分 USB2.0 协议的处理工作，通过 USB 收发器与 USB 主机进行数据交互；可编程外设接口用于 USB 设备与外设之间的连接；4KBFIFO 则作为数据缓冲区，传输中的数据缓存，同时增加了吞吐量。

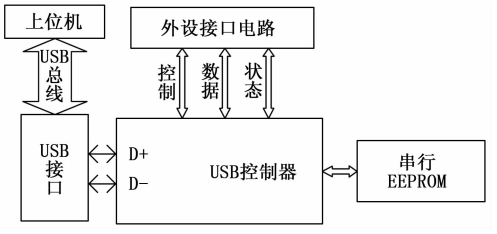

图 1　系统连接框图

设计中，USB 和外设之间通信的控制和状态信号由 IOA 和 IOD 接口完成，IOB 接口则用于通信的数据总线。同时，设置 EP2－FIFO 为 USB 发送数据缓冲区，EP6－FIFO 为接收数据缓冲区。

2.2　USB 数据传输过程

端口模式下 USB 数据传输过程：

USB 数据发送时，CPU 按照预定的通信协议读取 IO 端口的数据，并将数据存放在预发送的 IN 端点缓冲区，将预发送的字节数写入对应的端点字节计数寄存器中，在接收到主机 IN 请求后，USB 内核将 IN 端点缓冲区的数据传输到主机，主机将接收到的数据存放在内存中，并通过读取内存来保存和显示接收的数据。

USB 数据接收时，主机将预发送的数据写入内存中，并发送 OUT 请求，USB 内核在有空闲 OUT 端点缓冲区时接受 OUT 请求，并将接收的数据写入预定义的 OUT 端点缓冲区，CPU 通过读取对应的端点字节计数寄存器来确定接收的字节数，随后按照预定义的通信协议将数据发送到 IO 端口，供外设读取。

2.3　通信协议设计

端口模式下的数据传输是通用 IO 口来实现的，为实现数据的可靠通信，必须对这些通用 IO 口赋予实际的意义，即制定通信协议。通信协议如下：

IOB7－IOB0 被定义为 8 位的数据信号；WRB（IOD［0］）与 WRB _ ACK（IOA［1］）、RDB（IOA［2］）与 RDB _ ACK（IOD［2］）被定义为单字节传输的请求/响应信号；IOA［7：3］与 WR _ ACK（IOA［0］）、IOD［7：3］与 RD _ ACK（IOD［1］）被定义数据传输的请求/响应信号。主要的工作过程如下：

单字节数据请求/响应式传输过程：在 USB 向外设发送单字节数据时，USB 首先将 RDB 信号置 1，向外设发送写数据请求，在接收到外设发送的写应答信号 RDB _ ACK 为 1 后，将输出端点缓冲区 EP2FIFO 中的单字节数据写入 8 位的数据端口，并撤销请求信号 RDB 为 0，通知外设读取数据，当检测到 RDB _ ACK 信号为 0 时，表明外设成功接收数据，开始准备发送下一字节数据。同理，在 USB 接收外设发送的单字节数据时，USB 在接收到外设请求信号 WRB 为 1 后，将写应答信号 WRB _ ACK 置 1，通知外设发送数据，在检测到 WRB 信号为 0 时，将数据端口数据写入数据端点缓冲区 EP6FIFO，完成后将 WRB _ ACK 置 0，通知外设数据接收成功，开始准备接收下一字节数据。

数据传输过程同样采用上述的请求/响应式传输方式，采用 RD 和 IOA［7：3］作为 USB 接收外设数据的控制信号，同理采用 IOD［7：3］和 WR _ ACK 作为 USB 向外设发送数据的控制信号。IOA［7：3］和 IOD［7：3］在作为数据接收应答和数据发送请求信号的同时，还可利用其不同逻辑组合来实现不同类型数据的传输，最多可支持 31 种不同数据传输。

如上所述，采用请求/响应式数据传输方式，保障了数据传输的连续性和可靠性，不仅适用于低速传输外设，同样适用于高速非实时传输外设。同时，借助不同的逻辑组合实现不同类型的数据传输，拓展了可适用范围。

2.4　外设接口电路设计

根据上述 USB 与外设之间的通信协议，对应的外设接口电路设计图如图 2 所示。采用端点 EP2 为发送缓冲，端点 EP6 为接收缓冲。

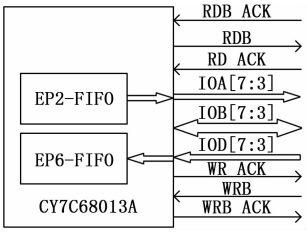

图 2　外设接口电路设计图

3　软件设计

系统软件设计包括固件设计、驱动程序设计和上位机界面程序设计三部分。这三部分之间的关系如图 3 所示，应用程序通过 CyAPI.lib 库对 Win32 子系统进行 Win32 API 调用，用于访问设备驱动程序，从而与固件程序进行通信。

3.1　固件程序设计

端口模式下的数据传输需要固件程序的参与，所以固件程序主要实现两方面的功能：一方面用于响应 USB 主机请求，另一方面用于实现 USB 设备与外设之间的数据通信。

设计中的固件程序设计是借助 Cypress 公司提供的 EZ－USB FX2LP 软件开发包完成的。开发包中的固件框架提供了完备的框架函数和程序代码，极大地简化和加速 USB 外设的开发。框架流程如图 4 所示，主要包括 USB 设备初始化、设

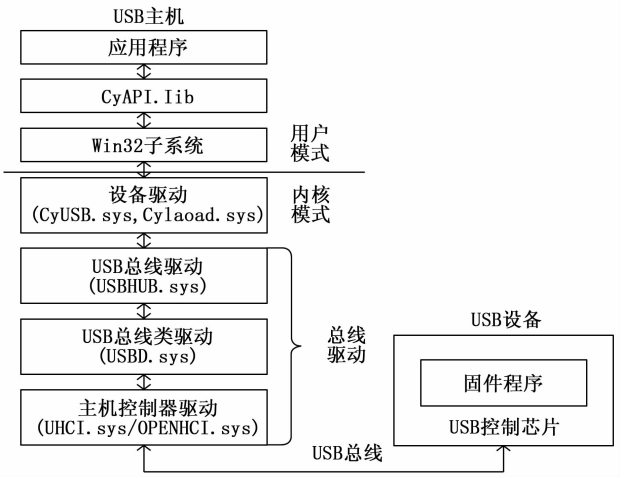

图 3 系统软件组成

备重枚举、外设功能函数、USB 主机请求响应和电源管理等，对于框架流程在文献［5］中有详细描述，这里不再赘述。由于固件框架中已经包含了 USB 主机请求响应函数，故在固件程序设计中主要完成 USB 设备与外设之间的数据通信，包括初始化程序设计和通信程序设计两部分。

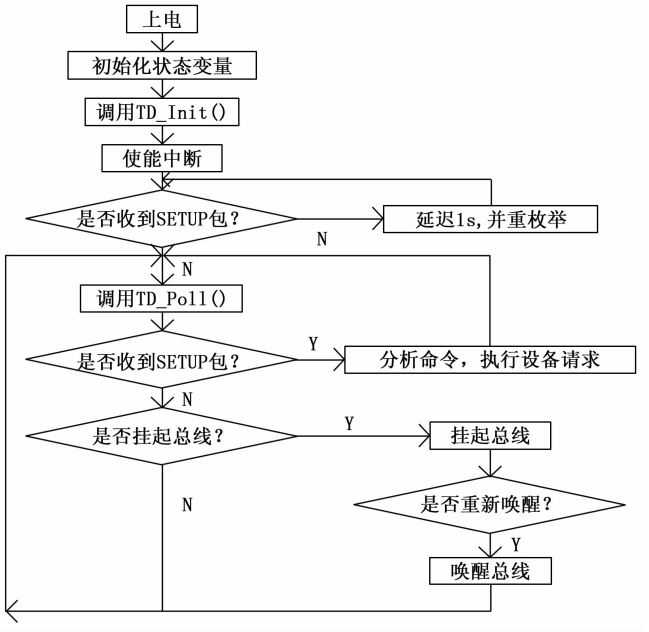

图 4 固件框架流程图

3.1.1 主要寄存器及功能描述

端口模式下，与固件程序相关的寄存器及功能描述如表 1 所示。CPUCS 用于选择 CPU 工作时钟，可选时钟频率为 12/24/48 MHz，设计中为提高传输速率、增加吞吐量，选择时钟频率为 48 MHz。IFCONFIG 用于选择端口模式。EPxCFG 用于配置端点方向和对应缓冲区大小，配置端点 EP2 为 OUT，EP6 为 IN；为增加吞吐量，将端点 EP2 和 EP6 均设为 4 倍缓冲，同时设置端点 EP4 和 EP8 为无效。AUTOPTRSETUP 用于配置自动指针，通过使能自动指针和地址自加，实现对数据缓冲区的快速访问。关于其他寄存器的位定义和功能描述可参考文献［2］。

表 1 与固件程序相关的寄存器及功能描述

寄存器	描述
CPUCS	CPU 控制和状态寄存器
IFCONFIG	接口配置寄存器
EPxCFG	端点 2,4,6,8 配置寄存器
EP2468SAT	端点 2,4,6,8 空/满状态标志寄存器
EPxBCL	端点 2,4,6,8 低 8 位字节计数器寄存器
EPxBCH	端点 2,4,6,8 高 8 位字节计数器寄存器
OEx	设置端口(A,B,C,D,E)引脚的输入/输出方向
IOx	端口(A,B,C,D,E)状态
AUTOPTRSETUP	自动指针设置寄存器

3.1.2 初始化程序设计

初始化程序设计是指对变量、USB 内部状态和相关寄存器的初始化，主要包括选择时钟频率、设置接口模式、端点和端口以及使能自动指针等操作。对应软件程序在框架函数 TD_Init（）中完成，其主要代码如下：

```
TD_Init(void)
{  ---
CPUCS = 0x12;//设置 CPU 频率为 48MHz
SYNCDELAY;
IFCONFIG |= 0x40;//选择端口模式
SYNCDELAY;
EP2CFG = 0xA0;//配置 EP2 为 4 倍的 OUT
端点,端点大小为 512 字节
SYNCDELAY;
EP6CFG = 0xE0;//配置 EP6 为 4 倍的 IN 端点,端点大小为 512
字节
SYNCDELAY;
EP4CFG = 0x02;//配置 PE4 端点无效
SYNCDELAY;
EP8CFG = 0x02;//配置 EP8 端点无效
SYNCDELAY;
OEA =0x00;//配置 IOA[7:0]为输出端口
SYNCDELAY;
OED =0x00;//配置 IOD[7:0]为输入端口
SYNCDELAY;
AUTOPTRSETUP |= 0x01;//使能自动指针
---
}
```

3.1.3 通信程序设计

根据 2.3 节描述的通信协议，对应通信程序主要工作流程如图 5 所示，对应软件程序在框架函数 TD_Poll（）完成。下面给出 USB 作为接收端，接收外设发送的数据，并提交给上位机的主要程序代码。而 USB 作为发送端与其作为接收端类似，这里不再给出详细的程序代码。

```
void TD_Poll(void){  if(IOD & 0x80)//根据 IOD[7:3]判断不同
的数据请求类型
{  if(!(EP2468STAT & bmEP6FULL))//判断 EP6 IN 端点缓
冲是否已满,若不满,则接收数据
{  AUTOPTRH1= MSB( &EP6FIFOBUF );//将 EP6FIFOBUF
地址赋值给自动指针
AUTOPTRL1= LSB( &EP6FIFOBUF );
IOA |= 0x01;//IOA[0]表示 WR_ACK,将其置 1,同意外设发送
数据
```

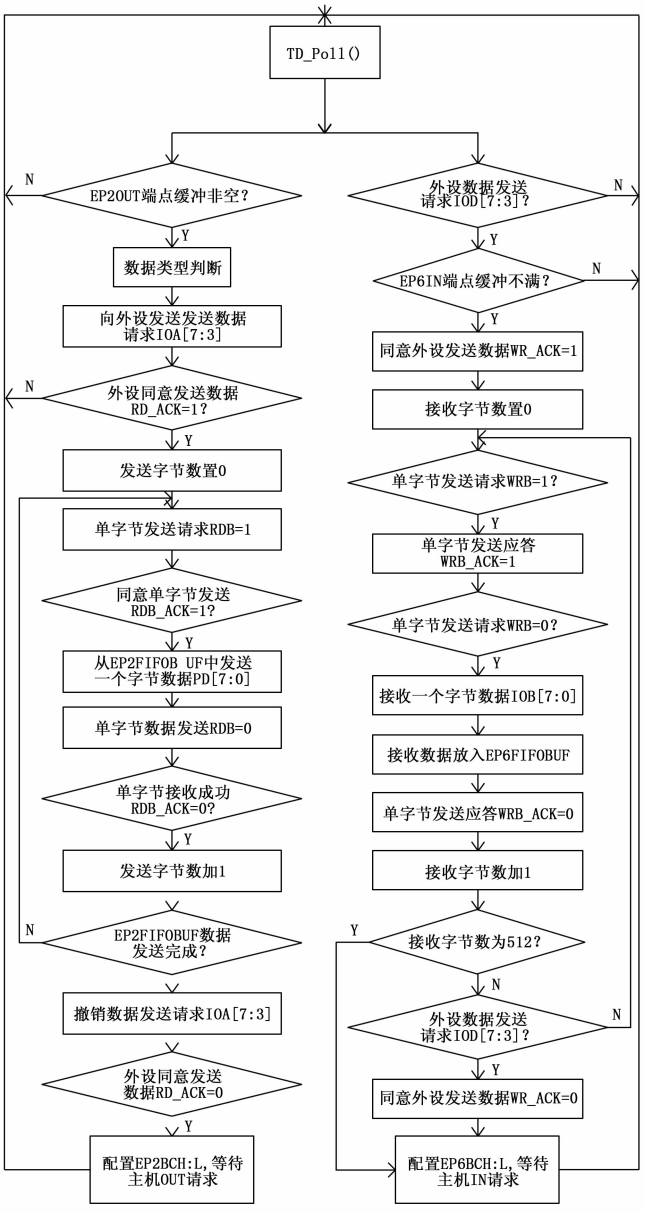

图 5　通信程序主要工作流程

```
for( i = 0x00; i < 512;)//接收字节数 i 初始化
{   while(! (IOD& 0x01));//IOD[0]表示 WRB,检测单字节发送请求
IOA |= 0x02;//IOA[1]表示 WRB_ACK,将其置 1,同意单字节发送请求
while(IOD& 0x01);//IOD[0]表示 WRB,判断是否可以读取数据
EXTAUTODAT1 = IOB;//读取端口数据 IOB[7:0]
IOA &= 0xFD;//IOA[1]表示 WRB_ACK,将其置 0,为接收下一字节做准备
i++;//接收字节数加 1
if(IOD & 0x80)//根据 IOD[7:3]判断数据发送请求是否结束
;//继续循环
else
{   IOA&= 0xFE;//IOA[0]表示 WR_ACK,将其置 0,为接收下一数据做准备
break;//跳出循环,结束接收数据操作
}
}
EP6BCH = i/256;//向上位机提交 IN 包
SYNCDELAY;
EP6BCL = i%256;
}
}
}
```

3.2　驱动程序设计

驱动程序保证了应用程序对 USB 设备的正常访问。Windows 的 USB 驱动遵循 Win32 驱动模式，采用分层驱动模型，包括设备驱动层和总线驱动层。其中设备驱动包含 USB 通信协议细节，用于实现应用程序与 USB 设备通信；总线驱动由根集线器驱动、总线类驱动和主机控制器驱动组成，用于传递总线通信，并最终实现 USB 主机与 USB 设备连接。图 3 表明了对应驱动在 USB 通信中的连接关系。其中总线驱动由操作系统提供，不需要开发者编写，故在驱动程序设计中只需完成设备驱动程序设计。

Cypress 公司提供了通用的 USB 驱动程序，包括固件下载驱动 CyLoad. sys 和通用驱动 Cyload. sys 两部分。其中固件下载驱动用于固件程序下载，通用驱动则用于主机与固件之间的通信。固件下载驱动在 USB 设备自枚举之后进行固件下载，然后按照固件程序进行设备重枚举，重枚举之后在通用驱动下完成 USB 设备通信。设计中，为确保 USB 设备具有唯一的设备 VID 和 PID，将对应设备信息文件 CyLoad. inf 和 CyLoad. inf 文件中的 VID/PID 均设置为 0x04B4/0x00F0。

3.3　上位机界面程序设计

上位机界面程序一方面向外设发送命令数据；另一方面接收外设数据，并进行显示和保存。Cypress 公司提供的 USB 主机控制函数库 CyAPI. lib 可实现 Visual C++环境下对 USB 设备的读写。在使用 Cypress 公司提供的驱动程序的基础上，只需在主机程序中加入头文件 CyAPI. h 和库文件 CyAPI. lib，然后便可以调用相应的控制函数。

界面程序主要分为四部分：1）USB 设备连接检测和选择。通过 Isopen 来检测 USB 设备连接，并根据设备 PID/VID 选择需要通信的 USB 设备。2）数据接收操作。通过“Start”按钮，开启数据接收主进程，并利用多线程实现数据的实时接收、显示和保存。再次点击此按钮，结束数据接收操作。3）数据发送操作。将预发送的数据以文件的形式保存，通过组合框控件，选择相应的数据发送操作，同时记录已发送的数据，以备后期查看。4）辅助功能。辅助功能主要包括数据保存时间间隔设置、接收数据文件保存目录设置和发送数据编辑等。表 2 对实现界面程序主要操作和功能的方法作了详细介绍。

4　实验结果与分析

在模块测试中，将 FPGA 发送的数据作为外设数据来源，FPGA 以 00－3F 的递增数据作为一个数据包，循环发送数据。利用此传输模块来接收 FPGA 发送的数据包，接收保存的数据包内容如图 6 所示。表明系统可正确保存接收到的数据。数据传输中界面状态，编辑框实时显示接收的数据，满足了系统接收数据显示的功能要求。实验结果表明，可以准确无误地实现接收数据的显示和保证操作，满足外设数据传输要求。

（下转第 166 页）