USB主机控制器的设计

许双燕，石江宏
（厦门大学 信息科学与技术学院，福建 厦门 361005）

摘要：讨论在SoPC（System on a Programmable Chip）系统中设计USB主机接口设备的一般方法，着重阐述主机控制器的驱动程序开发。利用Xilinx公司的EDK软件在ML405开发板上搭建一个基于PowerPC的片上系统，设计EZ-Host的USB主机控制器的Linux驱动程序，使系统具有USB主机功能，能够和各种USB设备进行通信，实现SoPC系统上基于Linux的USB接口的扩展，对于开发其他USB主控制器驱动具有一定借鉴意义。

关键词：USB；USB主机控制器；EZ-Host；Linux；SoPC
中图分类号：TN92　　**文献标识码**：A　　**文章编号**：1674-6236（2010）01-0035-03

Design of USB host controller

XU Shuang-yan，SHI Jiang-hong
（*School of Information Science and Technology*，*Xiamen University*，*Xiamen* 361005，*China*）

Abstract：The general development method of USB host interface based on SoPC system is discussed，and the driver program development of host controller is introduced in detail.A PowerPC-based system on chip is built up by using Xilinx′s EDK software in the ML405 development board.Linux driver program of USB host controller of EZ-Host is designed.The system is capable of communicating with a variety of USB devices，achieves extension of the Linux-based USB interface.The design is helpful to the development of other USB host drivers.

Key words：USB；USB host controller；EZ-Host；Linux；SoPC

USB具有简单、标准的连接方式、支持热插拔等诸多优点，因此已成为流行的接口技术。USB是典型的主/从结构的总线标准，即只有USB主机才能与USB设备连接。USB总线与计算机系统的接口部分是主机控制器，它可以被看作一个硬件、固件和软件的综合体[1]。主机控制器实现主机与设备之间的电气和协议层匹配，主要包括：串并转换、帧起始、数据处理、协议使用、传输错误处理、远程唤醒、根Hub、主机系统接口等功能。USB设备之间通过USB Hub连接，主机控制器和USB设备之间一般通过根Hub相连。通常主机控制器提供与根Hub相关的状态查询和控制单元[2]。当有设备插入时，在枚举过程中，主机控制器驱动通过查询和控制单元应答设备伪装成一个Hub，所以通常称此Hub为虚拟根Hub。

这里利用EDK软件搭建一个基于PowerPC的片上系统，实现了SoPC系统上基于Linux的USB接口的扩展，使系统具有USB主机功能，能够和各种USB设备进行通信。

1　开发环境

目前Linux 2.6内核中的USB支持3种主控制器接口：通用主控制器接口（UHCI）、开放控制器接口（OHCI）及增强主机控制接口（EHCI）。在嵌入式系统中，如果处理器集成有USB主机控制器，则可直接引出USB主控端口；而未集成USB主机控制器的处理器则需使用USB主控器件，从总线上扩展USB主机接口[3]。

这里所采用的开发环境是Xilinx公司的ML405开发板。开发板上核心FPGA采用Xilinx的XC4VFX20-FF672器件，其内置1个PowerPC硬核，2个以太网MAC层控制器。开发板上还带有64 MB的DDR SDRAM，10/100/1000以太网端口、带主机/设备端的USB接口器件（CY7C67300）等。

EZ-Host（CY7C67300）是Cypress半导体公司的全速低耗多端口主机/外设控制器，该器件可方便接至高性能CPU上完成USB主机控制器端的功能；拥有16位RISC指令处理器，可作为协处理器使用或单独使用；同时支持USB的OTG协议，拥有2个可独立配置并各带有2个端口的USB串行接口引擎（SIE）；既可用作主机，又可用作外设，并支持多达4个主机端口。另外，该器件拥有一个可编程I/O接口模块，可供各种接口编程使用，可编程实现HPI、HSS、SPI等接口模式。当EZ-Host控制器作为USB主机控制器时，一般采用HPI主机端接口（Host Port Interface）接口模式[4]。

2　硬件设计

Xilinx公司提供一个IP核opb_epc外设控制器（external peripheral controller），为OPB总线与外部同步或异步外围设备之间的数据传送提供一个通用接口，可方便实现处理器对于外设的控制。一个opb_epc最多可接4个外设，且每个外设

收稿日期：2009-07-23　　稿件编号：200907080
基金项目：福建省重大专项项目（2007HZ0003）
作者简介：许双燕（1986—），女，福建泉州人，硕士研究生。研究方向：无线通信技术。

可独立配置成同步或异步模式,其时序参数(如建立时间、保持时间、访问时间周期等)都可由用户设置。opb_epc 通过 OPB 总线接收处理器的读写指令,在相应外设接口产生相应的访问周期,从而实现处理器对外围设备的控制[5]。这里使用 opb_epc 模块作为控制器,实现 PowerPC 与 EZ-Host 的主机控制器的接口通信,嵌入式硬件系统架构如图 1 所示。

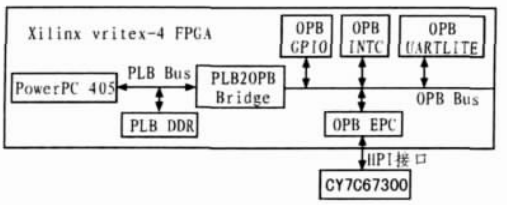

图 1 嵌入式硬件系统架构

ML405 板上的 EZ-Host 控制器工作在异步模式,因此 opb_epc 需配置为支持异步外设模式。这里使用 PowerPC 控制 USB 接口,因此 EZ-Host 工作于协处理器模式,并通过 HPI 接口与外设控制器 opb_epc 相连。

3 驱动程序设计

3.1 USB 主机端的软件结构

Linux USB 主机驱动协议栈由 3 部分组成:USB 主机控制器驱动(HCD)、USB 驱动(USBD)和各种不同的 USB 设备类驱动,如图 2 所示。

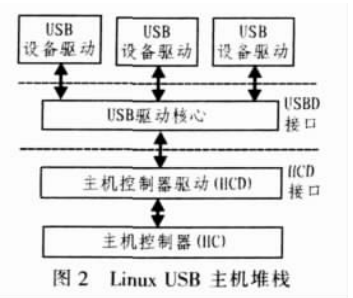

图 2 Linux USB 主机堆栈

USB 设备类驱动(如插入主机的 U 盘、鼠标、键盘等设备驱动)是最终与应用程序交互的软件模块,负责建立虚拟连接、配置,与设备进行通信,将数据集成一个 USB 请求块(URB),然后通过 USB 驱动(USBD)提供的编程接口将 URB 发送到 USBD。USBD 部分是整个 USB 主机驱动的核心,USBD 完成以下工作:USB 设备的枚举和配置,根据需要装载或卸载设备驱动程序,向上为设备驱动程序提供编程接口,向下为主机控制器驱动提供编程接口,实现与设备驱动程序、主机控制驱动程序的通信。

处于最底层 USB 主机控制器驱动(HCD)是 USB 主机直接与硬件交互的软件模块。HCD 作为底层硬件的驱动程序,一方面控制和管理底层硬件,负责将 USB 事务发送给 USB 主机控制器,并最终将串行数据发送到电缆上;另一方面为上层的 USB 系统软件提供统一接口 HCI (Host Controller Interface),将各种不同的 HC 映射到 USB 系统。HC 一般都集成有 Root Hub 的功能,HCD 也要实现 Root Hub Port 访问。

USBD 部分由操作系统实现,一般不需要用户修改。USB 设备类驱动,对于常用的设备 Linux 内核中有较成熟的驱动。针对特定的主机控制器硬件应该实现 HCD 部分,以解决基本的通信问题。故这里主要介绍 EZ-Host 主机控制器驱动(HCD)的设计。

3.2 EZ-Host 主机控制器驱动(HCD)设计

开发过程主要针对 EZ-Host 主机控制器编写 USB 主机控制器驱动程序。该驱动程序是嵌入式 Linux 开发平台下 USB 协议栈和 EZ-Host 主机控制器的一个接口,其作用类似于 Linux 中由 Intel 制定的 UHCI 标准,其硬件设计比较简单,但软件较为复杂。

USB 主机控制器的驱动(HCD)在 USB 子系统中的功能主要有:硬件初始化,为上层(USBD)提供调用接口,管理根 Hub,完成数据传输以及中断处理。根据主机控制器驱动(HCD)在整个 USB 子系统中的功能,可将 EZ-Host HCD 分为 HCD 接口、HCD 初始化、数据传输、中断处理、读写操作、主机协议等模块。HCD 接口模块表现为一套 API 函数,通过这一套 API 函数使 HCD 与 USBD 进行通信[6]。图 3 为 EZ-Host 主机控制器驱动模块结构。

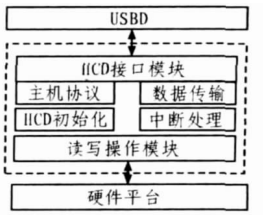

图 3 EZ-Host 主机控制器驱动模块结构

1)初始化。该初到始化涉及到复位 EZ-Host 控制器,并将其初始化到一个已知的状态;初始化必要的 USB 数据结构并为其分配空间;注册 USB host driver 和 USB host bus interface 到 USB host core;注册 USB host core 的中断服务程序;为每一个主端口建立一个虚拟根 Hub,并且注册根 hub 到 USB host core。2)中断处理。EZ-Host 主控制器中断采用电平触发,当中断服务程序注册到 USB 子系统后,EZ-Host 主控制器开始处理中断。3)传输数据。传输处理程序在初始化的过程中注册到 USB 主端子系统,它由 USB host core 唤醒并配置外围设备,发送块数据,或确认块数据的接收。4)接收数据。接收处理程序处理数据包的接收,它由中断处理程序唤醒。数据接收处理程序询问 EZ-Host 主控制器是否有接收错误,如果没有错误,则接收处理程序从 EZ-Host 主控制器的缓冲区中提取数据并将数据存储到一个数据结构,然后接收程序将数据传送到 USB host core,等待进一步处理。5)主机协议实现。HCD 从 Linux USB 协议栈接收并解析 USB 请求,然后建立基于该请求的 USB 交互(transactions),该交互被合理调度安排并发送到 USB 总线上。

3.3 HCD 的关键接口设计

实际工作过程中,应用程序通过文件系统接口访问相应的 USB 设备类驱动程序和 USBD;USB 设备类驱动程序则通

过 USBD 提供的相关接口(USBDI)将数据请求包传递给 USBD;USBD 通过 HCD 提供的接口(HCI)进一步将数据包传递给 HCD;HCD 最终将数据发送到 USB 总线。

主机控制器驱动中,最重要的接口是主机控制器驱动 HCD 与 USBD 之间的接口。在 Linux 内核中,用 usb_hcd 结构体表示 USBD 接口,用来描述主机控制器(HC)的基本信息、硬件资源、状态描述和用于操作主机控制器的 hc_driver 等。其中 usb_hcd 中的 hc_driver 成员非常重要,它包括具体用于操作主机控制器的钩子函数。在 Linux 内核中,使用如下函数创建 HCD:

struct USB_hcd *USB_create_hcd (const struct hc_driver *driver, struct device *dev, char *bus_name);

struct hc_driver 可看作 USBD 模块定义的需要底层主机控制器驱动实现的接口,通过实现这些接口,USBD 可将更上层软件的请求传递给 HCD 以及 HC,HC 及 HCD 完成后,也会通过这些接口通知 USBD。

这里在 EZ-Host 主控制器驱动中定义一个结构体 struct usb_hcd c67x00_hcd,用于描述 EZ-Host 的基本信息、硬件资源、状态描述,定义 struct hc_driver c67x00_hc_driver 来描述用于操作主机控制器的钩子函数,其结构体如图 4 所示。

```
static struct hc_driver c67x00_hc_driver = {
    .description = "c67x00-hcd",
    .product_desc = "Cypress C67X00 Host Controller",
    .hcd_priv_size = sizeof(struct c67x00_hcd),

    /* * generic hardware linkage */

    .irq = c67x00_hcd_irq,
    .flags = HCD_USB11 | HCD_MEMORY,

    /* basic lifecycle operations */

    .start = c67x00_hcd_start,
    .stop = c67x00_hcd_stop,

    /*managing i/o requests and associated device resources */

    .urb_enqueue = c67x00_urb_enqueue,
    .urb_dequeue = c67x00_urb_dequeue,
    .endpoint_disable = c67x00_endpoint_disable,

    /* scheduling support */

    .get_frame_number = c67x00_hcd_get_frame,
    /* root hub support */
    .hub_status_data = c67x00_hub_status_data,
    .hub_control = c67x00_hub_control,  };
```

图 4　struct hc_driver c67x00_hc_driver 结构体

c67x00_hub_start()启动 HCD 主控制器,c67x00_hub_irq() 实现其中断控制处理,c67x00_hub_status_data (), c67x00_hub_control()实现对虚拟根集线器的控制, c67x00_hub_enqueue(), c67x00_hub_dequeue()实现对 USB 请求(URB)进行排队,对 URB 进行调度。根据 hcd 和 endpoint 的信息,安排 URB 的 schedule 到 c67x00,该 URB 的传输完成后,会调用 urb->complete ()通知 USBD。

4　测试结果

在 ML405 开发板上实现了 USB 主机控制器的开发,使系统具有 USB 主机功能。在开发板上分别插入 USB 键盘、USB 鼠标、U 盘进行测试,内核识别信息输出如图 5 所示。

```
/ # [   67.284923] usb 1-1: new low speed USB device using c67x00 and address 2
[   67.612804] usb 1-1: configuration #1 chosen from 1 choice
[   67.769722] input: Logitech Optical USB Mouse as /class/input/input8
[   67.855085] input: USB HID v1.10 Mouse [Logitech Optical USB Mouse] on usb-c67x00_sie-1
[   67.954074] usb 1-1: Product: Optical USB Mouse
[   67.993525] usb 1-1: Manufacturer: Logitech
[   82.985894] usb 1-1: USB disconnect, address 2

/ # [  125.236873] usb 1-1: new low speed USB device using c67x00 and address 3
[  125.625895] usb 1-1: configuration #1 chosen from 1 choice
[  125.887784] input: Dell Dell USB Keyboard as /class/input/input9
[  125.888348] input: USB HID v1.10 Keyboard [Dell Dell USB Keyboard] on usb-c67x00_sie-1
[  125.994596] usb 1-1: Product: Dell USB Keyboard
[  126.834518] usb 1-1: Manufacturer: Dell
[  140.826068] usb 1-1: USB disconnect, address 3

/ # [  349.692859] usb 1-1: new full speed USB device using c67x00 and address 4
[  350.027148] usb 1-1: configuration #1 chosen from 1 choice
[  350.110894] scsi1 : SCSI emulation for USB Mass Storage devices
[  355.213426] scsi 1:0:0:0: Direct-Access              USB 2.0          2.00 PQ: 0 ANSI: 2
[  355.382938] sd 1:0:0:0: [sda] 514304 512-byte hardware sectors (263 MB)
[  355.464553] sd 1:0:0:0: [sda] Write Protect is off
[  355.507343] sd 1:0:0:0: [sda] Assuming drive cache: write through
[  355.634152] sd 1:0:0:0: [sda] 514304 512-byte hardware sectors (263 MB)
[  355.714327] sd 1:0:0:0: [sda] Write Protect is off
[  355.757297] sd 1:0:0:0: [sda] Assuming drive cache: write through
[  355.830526]  sda:(7)usb-storage: queuecommand called
[  355.920469]  sda1
[  355.954088] sd 1:0:0:0: [sda] Attached SCSI removable disk
[  356.022062] sd 1:0:0:0: Attached scsi generic sg0 type 0

/ # mount -t vFat -o iocharset=cp936 /dev/sda1 /mnt/usb/
/ # cd /mnt/usb/
/mnt/usb # ls
RAKE安收机2.ppt   recycler          u-boot22.zip      iptables_1.4.1
book              spidev0.1         u-boot.zip
```

图 5　USB 接口测试

从图 5 中可看出,系统可以方便与大容量存储类(Mass Storage 类) USB 接口、人机接口类 HID(Human Interface Device)USB 接口进行通信,进行正常读写操作,实现了系统的 USB 接口扩展。

5　结束语

详细介绍在 SoPC 平台上进行 USB 主机控制器的硬、软件设计。针对 EZ-Host 器件,详细介绍其 USB 主控制器的 Linux 驱动开发过程及主要的接口设计,对于 USB 的主机控制器的驱动开发有一定参考价值。设计的重点和难点主要集中在主机控制器器件的驱动程序开发的环节上,但 Linux 作为开源系统,在开发设备驱动程序时有着其他嵌入式系统不可比拟的优势,大量的开放源码无疑可以大大加快开发的进程并使得其应用更加的广泛。因此,USB 作为一种新型的高速外设总线,在嵌入式 Linux 领域有着广阔的应用前景。

参考文献:

[1] 杜春雷. ARM 体系结构与编程[M].北京:清华大学出版社, 2003.

[2] 孙 琼.嵌入式 Linux 应用技程序开发详解[M].北京:人民邮电出版社, 2006.

[3] 刘 淼.嵌入式系统接口设计与 Linux 驱动程序开发[M].北京:北京航空航天大学出版社, 2006.

[4] Cypress Semiconductor Corporation. EZ-Host programmable embedded USB host/peripheral controller [EB/OL].2003.http://www.cypress.com.

[5] Xilinx 公司.OPB external peripheral controller (EPC) v1.00a[EB/OL]. 2006.http://www.xilinx.com.

[6] Cypress Semiconductor Corporation.Linux USB driver user's guide CY7C67200/300[EB/OL].2003.http://www.cypress.com.