USB 2.0 设备控制器 IP 核的 Verilog HDL 设计

周芳, 吴宁

(南京航空航天大学信息科学与技术学院, 210016, 南京)

[摘要] 介绍了一种设计 USB 2.0 设备控制器 IP 核的方法. 着重分析了 UTMI 接口、协议层、存储器接口、仲裁器及控制和状态寄存器等几个结构模块及其设计. 使用上述方法在 Xilinx ISE 软件平台上, 实现了 USB 2.0 设备控制器 IP 核的 Verilog HDL 语言代码及其验证.

[关键词] USB, IP 核, Verilog HDL

[中图分类号] TN402, [文献标识码] B, [文章编号] 1672-1292-(2003)04-0066-05

0 引言

USB 是近年来应用在 PC 领域的新型接口技术, 在嵌入式系统中的应用发展迅速. USB 能够用简便有效的方法与多种类型的外设通信, 从而导致了 USB 接口的设计和编程复杂. 为了降低设计者的开发难度, 使用专用的 USB 控制器已成为首选方案. 控制器必须知道如何检测并对 USB 端口的事件做出反应. 目前市场上供应的 USB 控制器主要有两种: 一种是 MCU 集成在芯片里面的; 另一种就是纯粹的 USB 接口芯片, 仅处理 USB 通信, 由一个外部的微控制器(MCU) 来管理 USB 控制器的寄存器、设备描述符的获取和数据包的交换等. 本文所设计的控制器属于后者. IP(Intellectual Property) 核是指将一些在数字电路中常用但比较复杂的功能块设计成可修改参数的模块. 将 USB 接口控制器设计成 IP 核形式, 使用 IP 核将节约芯片设计人员设计、调试和使用 IC 的时间, 极大地缩短产品开发周期. IP 核设计可分为行为、结构和物理 3 个不同级别, 分别对应描述功能行为的"软核(soft IP core)"、完成结构描述的"固核(firm IP core)"和基于物理描述并经过工艺验证的"硬核(hard IP core)"3 个层次. 这相当于集成电路(器件或部件) 的毛坯、半成品和成品的设计技术. 本文所介绍的 USB Control IP 核还只处于软核阶段, 以 Verilog HDL 语言描述文本的形式提交使用, 并经过仿真验证, 使用者可以用它综合出正确的门级网表.

1 控制器结构原理

本文所设计的 USB 2.0 设备控制器可用来进行 USB 协议处理和数据交换, 完成 USB 通信. 控制器的主要设计思想完全基于 USB 2.0 的协议[1], 它负责把 USB 总线上的差分信号进行 NRZI 解码和位解填充, 再经串并转换后得到的分组拆装, 有用的数据放入 RAM 区; 或者把设备要传给主机的数据(已放在 RAM 区) 组装成协议所规定的分组形式, 再通过一个差分驱动电路经过串行化、位填充和 NRZI 编码后输出到 USB 总线上.

根据 USB 2.0 设备控制器所要实现的功能, 控制器可以划分为以下几个模块: UTMI(USB Transceiver Macrocell Interface)、UTMI 接口、协议层 PL(Protocol Layer)、存储器接口和仲裁器、控制和状态寄存器及功能接口. 其结构框图如图 1 所示. 其中, UTMI 部分作为 USB 2.0 的模拟前端, 用作差分信号的 NRZI 编/解码、位处理和串并转换. 为了降低设计难度, 避免涉及到模拟设计, 这部分可以选用已有的成品. 本文

收稿日期: 2003-09-09.

作者简介: 周芳, 女, 1979-, 南京航空航天大学信息科学与技术学院硕士研究生, 主要从事数字系统设计与计算机应用的学习与研究.

通讯联系人: 吴宁, 女, 1956-, 硕士, 南京航空航天大学信息科学与技术学院教授, 主要从事电路与系统的研究.

所用的UTMI 芯片是Agere Systems 公司的USS2x1WA 芯片.

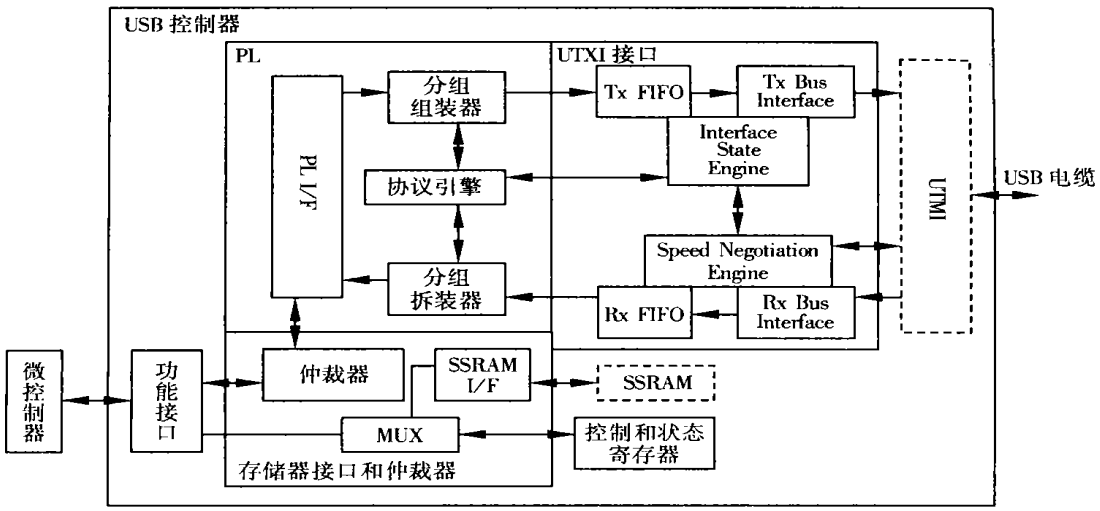

图 1　USB 2.0 控制器结构

2　控制器与外部芯片的接口

控制器与外部芯片的接口有 3 种: 与单口同步静态存储器(SSRAM) 之间的接口; 与微控制器之间的功能接口; 与外部 UTMI 之间的接口.

控制器所需要的外部存储器采用标准的单口 SRAM, 容量为 $2^{15} \times 16\,\text{bit} = 64\,\text{kB}$. 信号接口包括读写信号(SSRAM_ WE 和 SSRAM_ RD) 、16 位数据线 SSRAM_ DATA 及 15 位地址线SRAM_ ADDR.

与微控制器之间的接口信号由 16 位数据线 DATA、16 位地址线 ADDR、读写请求信号、中断请求信号以及 DMA 请求和响应信号(DMA_ REQ 和 DMA_ ACK) 组成. 其中 16 位地址线中低 15 位 addr[14: 0] 直接用于存储器 SSRAM 的地址, 可以支持 64 kB 的 RAM. 第 16 位地址线用来选通 SSRAM 和 USB 控制器内部的寄存器, addr[15] 为高时选择缓冲存储器, 否则选择内部寄存器.

控制器与 UTMI 的接口符合 UTMI 规范[2] 定义. UTMI 接口信号包括: USB 总线状态信号 LINESTATE[1: 0] 、用来设置 UTMI 工作模式的一些信号、与发送数据相关的信号(TxValid、TxReady 等) 、与接收数据相关的信号(RxActive、RxValid、RxError 等) 以及 8 位双向数据线、60 MHz 时钟信号. 根据 USB 2.0 的协议, 数据传输速率是 480 Mb/ S, 而 UTMI 提供的是 8 位并行数据, 所以 60 MHz 的时钟信号完全满足数据传输速率要求.

3　控制器设计

3.1　UTMI 接口

UTMI 接口部分负责 USB 各状态之间的转换及保存发送和接收到的数据. 其中 Tx FIFO 和 Tx Bus Interface 存放和发送已经组装好的分组数据; Rx FIFO 和 Rx Bus Interface 存放和接收 UMTI 送来的 USB 总线上 8 位数据分组; 接口状态引擎 ISE 负责跟踪 USB 总线状态, 控制正常操作、挂起、重新开始和复位等状态之间的转换; 速度识别引擎 SNE 决定 USB 设备工作在全速模式(480 Mb/ s) 或高速模式(12 Mb/ s) .

USS2x1WA[3] 提供的 LINESTATE[1: 0] 引脚用于侦测 USB 总线上的状态, 当 LINESTATE[1: 0] 为“00”时, USB 总线处于 SE0 状态; 为“01”时处于 J 状态; 为“10”时处于 K 状态; 为“11”处于 SE1 状态. ISE 通过读该引脚的状态来进行 USB 控制器状态之间的转换[1] .

在 ISE 中设计一状态转换机, 其转换机制是根据 USB 2.0 协议来设计的, 主要有 4 个状态: 上电/ 功

能复位、正常操作、挂起和复位. 各状态之间的转换关系如图2所示. 仅当ISE处于正常操作状态时, 控制器才可以和UMTI之间进行数据分组的接收和发送. 复位状态时, 控制器可以进行高速、全速模式之间的转换. 进入挂起状态后, 控制器内部停止工作, 并通知UMTI, 等待USB总线上送来的唤醒信号, 从而退出挂起状态.

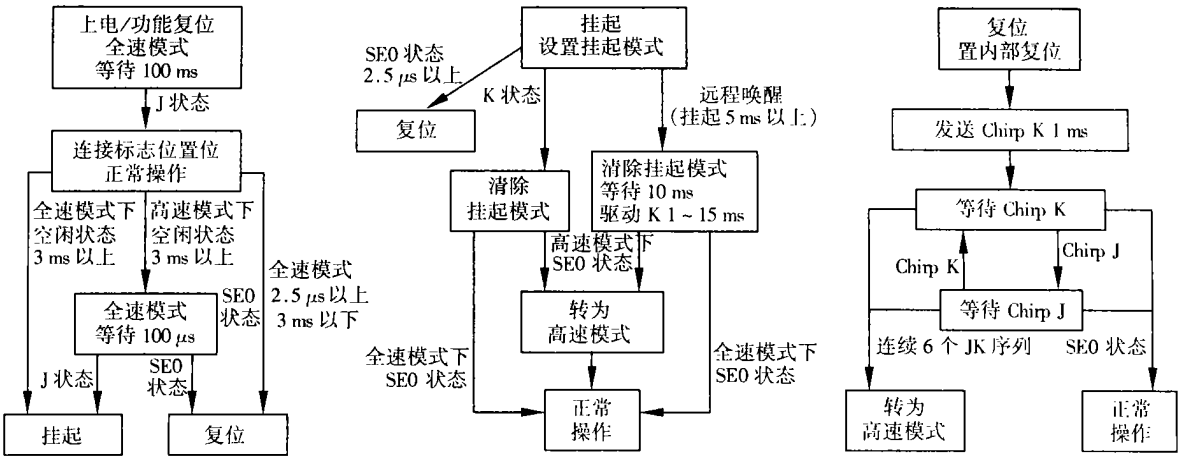

图2　ISE状态机转换图

3.2　协议层PL

控制器的核心逻辑是PL(Protocol Layer)模块, 负责管理所有USB数据I/O和控制通信. 其中协议引擎(Protocol Engine)处理所有标准的USB握手信号和控制通信. 分组组装器组装分组并送入输出FIFO, 先组装分组头, 插入适当的PID(分组标识)和校验和, 然后加入数据域. 分组拆装器先解码出PID和序列号以及校验和, 再从8位PID取低4位(或高4位取反)得到PID[3:0], 通过USB 2.0协议的PID类型定义译码出PID名, 判断是Token分组(OUT、IN、SOF和SETUP)还是DATA分组(DATA0、DATA1、DATA2和MDATA). 其Verilog HDL语言描述如下:

```
assign  pid_TOKEN= pid_SETUP| pid_PING | pid_OUT| pid_IN| pid_SOF;
assign  pid_DATA= pid_DATA0| pid_DATA1| pid_DATA2 | pid_MDATA;
```

若是Token分组, 则将后续的16位数据分别放入两个8位临时寄存器token0和token1, 并取出分组中的7位地址、4位端点号及5位CRC校验码.

如果检验校验码正确, 将地址、端点号和帧号等放入相应寄存器. Token类型如果是IN, 就执行组装分组并发送分组; 如果是OUT就拆卸接收到的数据分组. 对于其他不支持的Token则视为错误处理:

```
assign pid_error= pid_NACK| pid_STALL| pid_NYET| pid_PRE| pid_ACK| pid_ERR| pid_SPLIT;
```

如果校验码出错则不进行Token的解码, 等待下一个Token的到来.

若是DATA分组, 则紧接着PID的是最大载荷为1 024字节的数据和16位CRC16校验码. 对数据的处理先写入端点寄存器, 然后通过内部DMA操作写入SSRAM.

3.3　控制和状态寄存器

寄存器是控制器内部不可缺少的重要部分, 为了存放控制器中各状态信息和事件数据, 并和外部微控制器进行一些控制参数的交互, 有必要定义一些寄存器[4]. 该控制器中所定义的寄存器均为32位, 所以访问寄存器的地址addr是4的倍数, addr[1:0]始终为0, 由addr[8:2]7根地址线来访问这些寄存器. 内部寄存器有两种, 一种是针对整个控制器状态的寄存器, 包括总线状态、传输模式、中断设置等; 另一种是针对每一个端点情况的寄存器. 这里详细介绍端点寄存器的设计.

数据的传输实际上是通过端点(Endpoint)进行, 控制器通过写端点寄存器配置端点. 该控制器最多

可有 16 个端点,每个端点有相应的 4 个寄存器:Epn_ CSR、Epn_ INT、Epn_ BUF0 和 Epn_ BUF1(这里 n 是指端点号),格式如表 1 所示. 地址 addr[8:4] 用来选择端点号,其值(10 进制)从 4 到 19 分别代表 Epn(n = 0~ 15). addr[3:2] 指定寄存器类型.

CSR 寄存器指定端点的工作模式并且向控制器报告指定端点的状态. 通过读 Ep_ CSR[31:30] 这 2 位可以知道下次所要处理的缓冲器;CSR 为"00"时,指 Buffer0;为"01"时,指 Buffer1. Ep_ CSR[27:24] 指定了端点类型和传输类型,其类型编码如表 2 所示. Ep_ CSR[21:18] 指定端点号,共可以有 16 个端点. Ep_ CSR[15] 为 DMA 使能位,置"1"时允许外部 DMA 操作,否则禁止 DMA 操作.

表 1　端点寄存器格式

	addr[3:2]	31 30	17 16 15	0
Epn_ CSR	00	Config/Status Bits		
Epn_ INT	01	Interrupt Mask	Interrupt Source	
Epn_ BUF0	10	U Buffer0 Size	Buffer0 Pointer	
Epn_ BUF1	11	U Buffer0 Size	Buffer0 Pointer	

表 2　类型编码表

Ep_ CSR[27:26]	端点类型	Ep_ CSR[25:24]	传输类型
00	控制端点	00	中断传输
01	IN 端点	01	同步传输
10	OUT 端点	10	批量传输
11	保留	11	保留

当控制器收到中断时,读中断源寄存器(Ep_ INT[6:0])来判断中断源和产生的原因. 如 Ep_ INT[2] 定义为该控制器接收到不支持的 PID 而产生的中断:Ep_ INT[2] = pid_ error. Ep_ INT[4] 和 Ep_ INT[3] 分别表示 Buffer1 和 Buffer0 的满或空的状态位. Buffer0 和 Buffer 1 用来存储临时数据,用它们作为专用的输入/输出缓冲器可以提高 USB 的数据吞吐能力,双 Buffer 能够减少微控制器和驱动软件之间的延迟. Epn_ BUFn[31](标记缓冲器是否被使用过)在使用后被控制器置 1,在清空或重填充该缓冲器后,控制器清除该位. 该位初始化时为 0. Epn_ BUFn[30:17] 指定缓冲器能容纳的字节数. Epn_ BUFn[16:0] 是缓冲器的指针,装载存储器 SRAM 中数据的地址. 控制端点(Endpoint0)比较特殊,它既要接收也要发送数据,对于控制端点,Buffer0 用于 OUT 缓冲器,Buffer1 则是 IN 缓冲器. 从 SETUP 和 OUT 分组来的数据,写入 Buffer0,IN 分组的数据则是从 Buffer1 中获取.

3.4　存储器接口和仲裁器

存储器接口和仲裁器负责仲裁内部 PL 和外部微控制器对存储器的访问,两者均通过采用内部 DMA 方式访问 SSRAM. 当外部总线有访问 SSRAM 的请求时,且 PL 没有请求访问存储器,控制逻辑引用 Verilog HDL 语言描述如下:

```
assign wsel= (wreq | wack) & ! mreq;
always @ (wsel or wdin or mdin)
    if(wsel)  sram_ dout= wdin;
    else      sram_ dout= mdin;
always @ (wsel or wadr or madr)
    if(wsel)  sram_ adr= wadr;
    else      sram_ adr= madr;
always @ (wsel or wack or wwe or wreq or mwe or mcyc)
    if(wsel)  sram_ we= wreq & wwe;
    else      sram_ we= mwe & mreq;
```

其中 wreq、wack 分别为外部总线和存储器之间的请求和响应信号,wdin、wadr 和 wwe 分别是外部总线给出的数据、地址和写信号,mreq 是内部 DMA 向存储器发送的请求信号,mdin、madr 和 mwe 分别是内部 DMA 给出的数据、地址和写信号.

由控制逻辑可看出,内部 DMA 操作的优先级比外部总线高.

4 结论

文中提出的USB 2.0功能控制器的实现方案,用Verilog HDL语言实现代码,已在XILINX公司的FPGA Virtex XCV300-6fg456中用Xilinx ISE软件通过了RTL级的行为设计,综合出正确的门电路级网表. FPGA的规模是32万门,1 536个CLB(可配置逻辑单元). 该控制模块占用2 050个Slice(66%),使用了1 697个Slice触发器(27%)和3 047个4输入LUT表(49%),整个FPGA的速度可达到56.870 MHz[5]. 该方案实现的控制器便于修改且易于实现,可作为一个功能模块嵌入到SOC中,可在各种情况下最大限度地灵活设计片上系统.

[参考文献]

[1] Compag, Hewlet-Packard, Intel, *et al*. Universal Serial Bus Specification Revision 2.0 [DB/OL]. http://www.usb.org/developers/docs/usb_20.zip, 2002-12-21/2003-08.

[2] Intel Corporation. USB 2.0 Transceiver Macrocell Interface Specification Version 1.05 [DB/OL]. http://www.intel.com/technology/usb/download/2_0_Xcvr_Macrocell_1__05.pdf, 2001-03-29/2003-08.

[3] Agere Systems Inc Semiconductors. USS2x1 Data Sheet [DB/OL]. http://www.agere.com/client/docs/PB00029-5.pdf, 2002-10/2003-08.

[4] Rudolf Usselmann. USB Function IP Core Rev 1.5 [DB/OL]. http://www.opencores.org, 2002-01/2003-08.

[5] Xilinx Inc. ISE 4 User Guide [DB/OL]. http://www.xilinx.com/support/sw_manuals/xilinx6/index_htm, 2001/2003-08.

USB 2.0 Device Controller IP Core For Verilog HDL Designs

Zhou Fang, Wu Ning

(College of information science & technology, NUAA, 210016, Nanjing, PRC)

Abstract: A USB 2.0 device controller IP core design method is introduced in this paper. The function of controller is first summarized, and internal structure modules are explained. Then the controller's interface with other external IC is introduced. At last UTMI interface, PL, Memory Interface and Arbiter, Control/status registers, and realize the controller IP core by verilog HDL in Xilinx ISE are analyzed and designed.

Key words: USB, IP core, Verilog HDL

[责任编辑: 严海琳]