基于Zynq平台PCIE高速数据接口的设计与实现

杨亚涛[1]，张松涛[1]，李子臣[1,2]，张明舵[1]，曹广灿[1]

(1. 北京电子科技学院通信工程系 北京 丰台区 100070；2. 北京印刷学院教务处 北京 大兴区 102600)

【摘要】为有效利用PCIE接口的优异传输性能，便于外设与主机进行高速通信，该文基于Xilinx公司Zynq系列芯片，设计实现了多通道、高速度的PCIE接口。利用Zynq-7000系列芯片的FPGA+ARM架构，在PCIE2.0硬核的基础上设计了DMA硬核控制器、设备驱动和应用程序。经测试，在PCIE×1通道和单逻辑通道时，传输速度能达到3.36 Gbps，相比现有设计提升近20%。方案设计完备，具有较强扩展性，为设计基于PCIE接口的外接设备提供了重要参考，具有广阔应用前景。

关 键 词 ARM; DMA; FPGA; 硬件设计; PCIE

中图分类号 TP391 文献标志码 A doi:10.3969/j.issn.1001-0548.2017.03.008

Design and Implementation for High Speed Data Transfer Interface of PCI Express Based on Zynq Platform

YANG Ya-tao[1], ZHANG Song-tao[1], LI Zi-chen[1,2], ZHANG Ming-duo[1], and CAO Guang-can[1]

(1. Department of Communication Engineering, Beijing Electronic Science & Technology Fengtai Beijing 100070;

2. Office of Educational Administration, Beijing Graphic Communication Daxin Beijing 102600)

Abstract In order to use the excellent transmission performance of peripheral component interconnect express (PCIE) interface to carry out high-speed communication between peripherals and the host effectively, a multi-channel and high-speed PCIE interface scheme is designed based on Zynq family chip of Xilinx. The DMA hardcore controller, device driver and application program are designed on the basis of PCI express 2.0 hardcore using the FPGA+ARM architecture of Zynq-7000 platform. Test results show that the transfer rate in our project almost can reach 3.3 Gbps in PCIE×1 lane and single logical channel, which increases about 20% compared with the traditional existing designs. Our scheme owns higher expansibility and wide application prospect, it is an important design reference for external equipment and PCIE interface.

Key words ARM; DMA; FPGA; hardware design; PCI express

作为第三代总线和接口标准，PCIE(PCI Express)的目的是取代作为第二代总线标准的PCI总线，并在物理结构、带宽和服务质量3个方面做出了较大的改进和革新。与PCI总线相比，PCIE总线具有以下特点[1-2]：1) 采用差分串行传输方式，一条PCIE通道(PCIE×1)由两对差分信号线来实现发送和接收；2) 具有很好的灵活性，可以根据实际需要配置成×1，×2，×4，×8，×16和×32通道模式；3) 采用点到点连接，提高数据传输率，保证设备的带宽；4) 以数据包的形式传输，保证数据传输的完整性和可靠性。

鉴于其优异的传输性能，围绕PCIE接口的设计方案不断被提出。文献[3]提出了一种高吞吐量和低延时的PCIE接口设计方案，用于安全云存储的场景中，不过方案是基于PCIE1.1规范设计的，不符合当前PCIE2.0或PCIE3.0规范的趋势；文献[4]提出了一种应用于吉字节数据传输的DMA-PCIE架构方案，虽然数据传输速率较高，但方案描述简单，无法表征设计的完整性，参考性较差；文献[5]在文献[4]的方案上进行了扩充说明，给出了多种模式下的设计思路和测试结果，但可扩展性和移植行较差，不便二次开发。鉴于现有方案的这些问题，本文以Xilinx公司Zynq-7000系列全可编程片上系统为设计平台，基于PCIE2.0的IP核，利用其ARM+FPGA的新型架构，设计实现PCIE高速数据传输接口。

1 Zynq-7000系列芯片

Zynq-7000系列基于Xilinx全可编程片上系统架构设计，包括以双核ARM Cortex™-A9处理器为核

收稿日期：2016－01－24；修回日期：2016－06－19
基金项目：国家自然科学基金(61370188)；中央高校基本科研业务费专项资金(2017XK01)
作者简介：杨亚涛(1978-)，男，博士，副教授，主要从事宽带密码通信和网络安全方面的研究.

心的PS(processing system)部分和28 nm工艺的PL(programmable logic)部分。Zynq-7000 All Programmable SoC平台的功能框图[6]如图1所示。

PS部分在每一个Zynq-7000芯片中都是一样的，包括4个主要模块：应用处理单元(application processor unit, APU)、存储器接口(memory interfaces)、I/O外设接口(I/O peripherals, IOP)、内部互联模块(central interconnect)。

PL的本质就是Xilinx FPGA(Artix-7或Kintex-7)，主要特征包括：可配置逻辑块(configurable logic blocks，CLB)、36 kB BRAM、数字信号处理单元(DSP blocks)、可配置I/O接口(programmable I/O blocks)、低电压串行收发器、集成PCI Express模块、12-bit模数转换器(XADC)、PL配置模块。

Zynq-7000系列的亮点在于它包含了完整的ARM处理子系统[7]。子系统包含了双核的ARM Cortex™-A9处理器，集成了内存控制器和大量的外设，使Cortex™-A9的核在Zynq-7000中可以完全独立于可编程逻辑单元。另外，FPGA部分与ARM处理单元紧密结合，有超过3 000个内部互联，连接资源非常丰富，可提供100 Gb/s以上的内部带宽[8]。在I/O接口方面，FPGA的优点是I/O可以充分自定义，并集成了高速串行口(multi gigabit transceiver , MGT)和模数转换器(XADC)。这样优异的性能，使Zynq-7000系列芯片成为众多高性能设计的选择，为方案提供了可扩展性和配置性[9]。

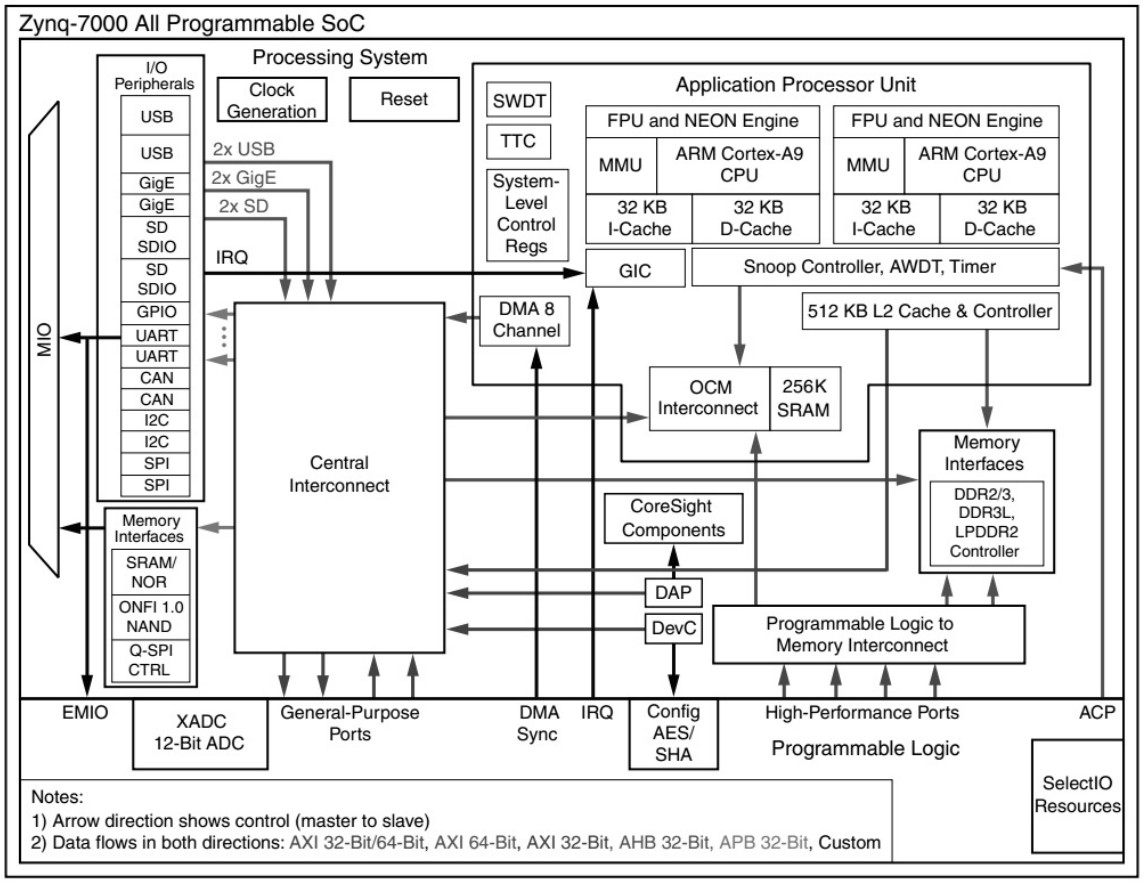

图1　Zynq-7000全可编程片上系统总览

2　总体设计方案

基于以XC7Z030芯片为核心的硬件板卡来设计和验证本文提出的解决方案。板卡设计了PCIE物理接口，用于连接到计算机主板上的PCIE插槽。主机采用Ubuntu操作系统，在此基础上设计和编写PCIE接口设备驱动程序和应用程序，以测试和验证PCIE接口的通信、速率、多通道是否正常等。图2为整个设计的硬件连接方案。

在PL部分，PCIE硬核与直接内存访问(direct memory access, DMA)硬核控制器协同工作，用来访问系统中的存储空间和配置空间；在PS部分，ARM Cortex™-A9处理器与DDR协同工作，用来缓存处理器与PCIE硬核的交互数据。DMA控制器的读/写操作完成后，将中断信号告知ARM Cortex™-A9处理器。整体设计的数据和指令流程如图3所示。

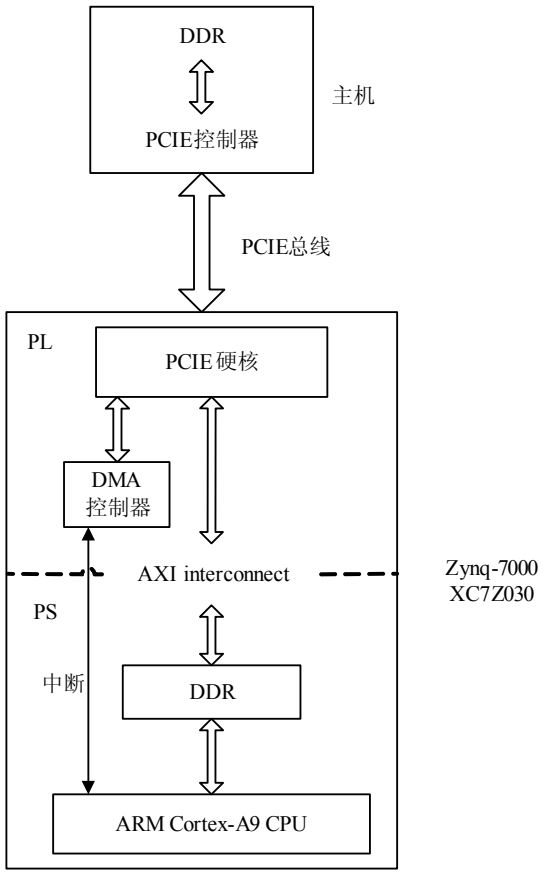

图2 总体方案硬件连接框图

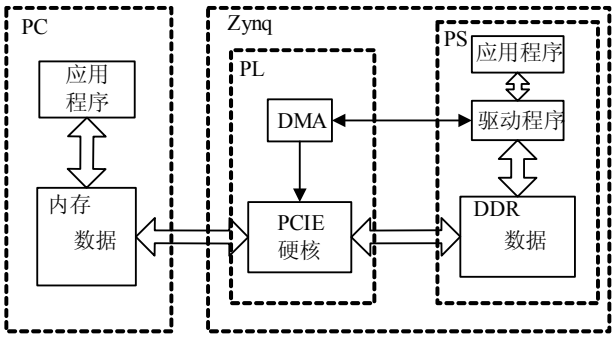

图3 整体设计方案数据流向

3 关键模块设计与实现

3.1 FPGA逻辑部分设计与实现

FPGA逻辑部分的设计基于PCIE硬核实现。PCIE硬核实现了PCIE总线的物理层、数据链路层、配置管理层的协议，上层提供一个事务层接口，用户通过该接口完成应用逻辑，实现对PCIE设备的控制与访问[10]。

为确保设计的正确性和高效率，FPGA逻辑部分采用Xilinx公司在Vivado2014.4中提供的IP核，主要包括ZYNQ7 processing system、axi_master_pcie_v1.0、AXI Interconnect、axi_slave_reg_cfg_v1.0，同时辅以Processor System Reset、AXI BRAM Controller、Block Memory Generator等IP核模块，一起实现PCIE硬核逻辑与处理系统(PS)、PCIE物理接口的连接和通信。

PCIE总线串行传输的特性是通过报文的形式进行数据传输，每个数据报文在PCIE的事务层被封装成一个或者多个事务处理层数据包(transcation layer packet, TLP)，PCIE设备之间则通过这些数据包进行通信。由于TLP中包含TLP前缀、TLP头以及TLP摘要等信息，因此，当设备在进行单次数据传输(每个报文数据负载长度为1)时PCIE总线的性能优势并不明显，其传输速度甚至还不如PCI总线。为了得到更高的传输效率，在使用PCIE总线进行数据传输时往往需要使用DMA的传输方式[11]。

设计采用DMA方式，用于访问计算机系统中的存储器空间和配置空间。该方式在实现高速外部设备和主存储器直接之间的数据交互时，不需要CPU的直接参与。使用这种方式可以减少CPU的占用率，大大提高数据吞吐率和系统性能。DMA方式又分为两种类型：一种称为系统DMA(system DMA)方式，该方式通过系统主板上的DMA控制器来控制总线的数据传输，支持这种方式的系统和设备很少，因此并不常用；另一种方式称为总线主控DMA(bus master DMA)，这种方式由接口卡上的逻辑电路完成DMA控制器，实现数据的传输。设计采用的即是总线主控DMA方式，其基本步骤如下：1) 软件向DMA控制器写入内存传输首地址、数据传输长度、读/写使能标志等信息，之后等待DMA控制器的指令；2) DMA控制器根据接收到的信息生成相应的读/写请求。如果是读请求，则向根联合体(root complex, RC)发送存储器读请求TLP，RC根据读请求中的内存起始地址和数据大小，读出内存数据，组成读完成TLP返回DMA控制器；如果是写请求，向RC发送存储器写请求TLP，将数据写入内存的预定空间；3) DMA控制器的读/写操作完成后，通过中断机制告知软件，软件将读/写使能标志清零[5][11]。DMA读/写流程如图4和图5所示。

DMA控制器与软件之间的状态/控制信息的交换通过特定的寄存器完成，这些寄存器处于基地址寄存器(base address register, BAR)所存储的基地址空间的范围内。通过读写这些寄存器的值，DMA控制器和软件可以判断对方的状态，以此决定下一步的动作。

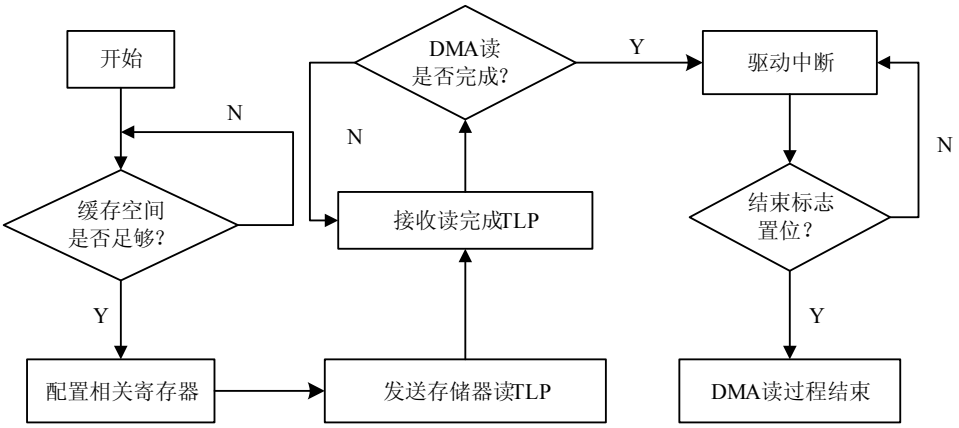

图4　DMA读流程

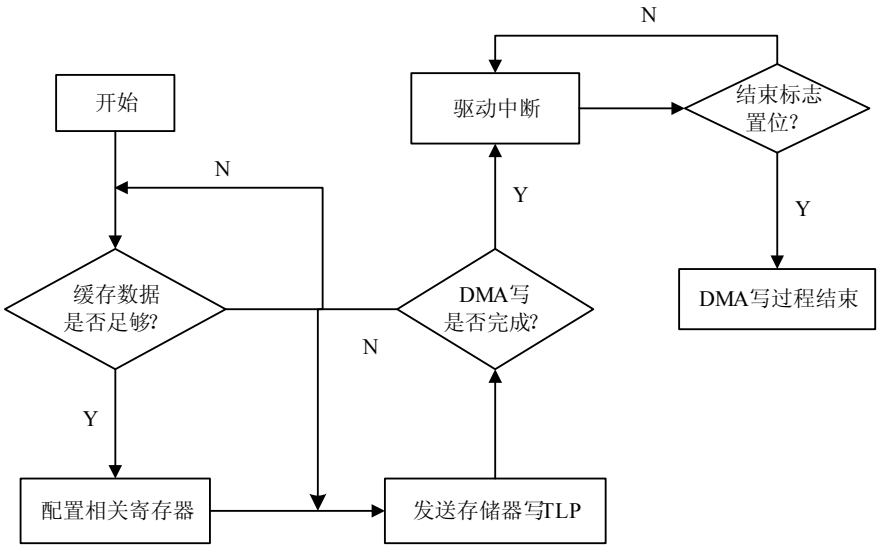

图5　DMA写流程

3.2　ARM处理器部分功能实现

鉴于Zynq芯片ARM+FPGA的新型架构，方案特使用ARM处理器来进行Zynq内部指令和数据的调度和处理，以充分利用芯片的优异性能，同时满足处理敏感数据的安全需求。ARM Cortex^{TM}-A9 CPU使用Xilinx公司提供的专用嵌入式操作系统，此部分的准备工作在本文中不再详述。

ARM Cortex^{TM}-A9处理器中运行了嵌入式Linux操作系统，将用于数据存储的DDR模块识别为处理器的外接设备，但需要对应的驱动才能正常工作，故需要编写DDR驱动程序。DDR作为PCIE硬核和ARM处理器的共用数据存储空间，对两部分来说是透明的，既能接收来自FPGA的控制命令和数据存取请求，又能将地址映射到ARM处理器的寄存器中，作为嵌入式系统的外接设备使用。

驱动程序的编写主要包括以下几个部分：

设备初始化：static int pcie_ddr_a9_ device_init (struct pcie_ddr_a9_dev_t *p device)；

file_operation结构在字符设备驱动程序中占有重要地位，它的成员函数提供了驱动程序与操作系统内核的接口。file_operation结构定义如下：

```
struct file_operation pcie_ddr_a9_fops={
.owner =THIS_MODULE,
.read =pcie_ddr_a9_read,
.write =pcie_ddr_a9_write,
.open =pcie_ddr_a9_open,
.release =pcie_ddr_a9_release,
.poll =pcie_ddr_a9_poll,
.unlocked =pcie_ddr_a9_ioctl,
};
```

对上述函数的调用分别如下所示：

static int pcie_ddr_a9_open(struct inode*inode, struct file *filp)；

static ssize_t pcie_ddr_a9_read(struct file *filp,

char_user *buf, size_t count, loff_t *f_pos)；

static ssize pcie_ddr_a9_write(struct file *filp, const char_user *buf, size_t count, loff_t *f_pos)；

static unsigned int pcie_ddr_a9_poll(struct file *filp, struct poll_table_struct *wait)；

static long pcie_ddr_a9_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)；

int pcie_ddr_a9_release(struct inode *inode, struct file *filp)；

使用这些函数基本完成了驱动的功能，达到了操作系统对DDR读写控制的目的。

3.3 Linux系统下驱动程序和应用程序设计与实现

在开源的Linux系统环境下，设计驱动程序和应用程序来完成计算机主机和PCIE接口设备的数据传输[12]。在应用程序中定义单逻辑和多逻辑通道。采用单逻辑通道，可以减少采用多逻辑通道时切换和调度带来的时间损耗；设计多逻辑通道，可以确保不同类型的任务通过不同逻辑通道传输，提高了事务处理效率和传输速率。

驱动程序的编写主要包含以下几个部分：

设备号的获取：

Static const struct pci_device_id ids[]= { {PCI _DEVICE(0x10EF,0x7030) },{0 } }

设备初始化：cdev_init(&ape->cdev, &alt_fops);

ape-cdev.owner=THIS_MODULE；

注册字符设备到系统：rc=cdev_add(&ape->cdev,ape->devno, DEV_NUM);

If(rc<0){

printk(KERN_ALERT “PCIE APE EP: cdev add error\n”);

goto err_add_cder;

} printk(KERN_DEBUG “probe( )successful.\n”);

注销字符设备：cdev_del(&ape->cdev)

释放字符设备：unregister_chrdev_region (ape ->devno, DEV_NUM)

file_operation结构定义如下：

struct file_operation alt_fops={

owner: THIS_MODULE, read: alt_read, write: alt_write, poll: alt_poll, open: alt_open, release: alt_release}

对读写函数等的调用分别如下：static ssize_t alt_read( ); static ssize_t alt_write( ); static unsigned int alt_poll( ); static int alt_open( ); static int alt_release( )。

驱动申请中断：request_irq(irq_line, altpciechd-ma_irs,IRQF_SHARED,ape->devname,(void*)ape);

驱动释放中断：free_irq (ape->irq_line, (void*) ape);

以上这些部分完成了驱动程序的主要功能。

应用程序完成对设备进行多次读写和测试读写速度的功能，主要功能代码如下所示：

打开设备：fd=open(“dev/pcie0_0”,0_RDWR);

获取当前时间：gettimeofday(&otv, NULL);

向设备写入数据：ret=write(fd, src, data_size);

从设备读出数据：read_num=read(fd, ack, read_sum);

比较读写数据是否一致：ret=memcmp(src, ack, read_sum);

计算读写数据量(字节)：wsum+=data_size; rsum+=read_num;

获取读写完成后的当前时间：gettimeofday (&ntv, NULL);

计算和输出读写时间：printf(“write_speed =%fKBps\n”,(1000.0*wsum/((ntv.tv_sec*1000.0-otv.tv_sec * 10 0 0.0)+(ntv.tv_usec- otv.tv_usec) / 10 00.0))/1024);

printf(“read_speed=%fKBps\n”,(1000.0*rsum/ ((ntv.tv_sec*1000.0-otv.tv_sec*1000.0)+(ntv.tv_usec-o tv.tv_use c)/1000.0))/1024);

PCIE接口的读写速度就是通过上面一系列的函数和运算得到的。

4 系统测试

测试环境及条件：1) 装载Ubuntu操作系统的台式机；2) 功能完备的提供PCIE接口的硬件板卡，并插装到台式机主板的PCIE插槽上；3) 嵌入式Linux内核、文件操作系统、设备树等文件完备，确保硬件板卡上的Zynq芯片能正常启动；4) 驱动程序及应用程序正确编译和安装。

测试步骤及测试项目：1) 加载驱动；2) 执行测试程序，分别测试单通道和多通道(逻辑)时的数据传输速率。

测试结果比较对象：理论上PCIE2.0(×1)支持5.0 GTps/Lane的传输速度，但由于采用8b/10b编码方式以及数据包头和控制数据包的占用，有效数据速率会低于理论值的80%，即不会达到4 Gbps/Lane (512 MBps/Lane)。

如图3所示，测试数据流向的数据传输速度，得到的一些典型值如表1所示。

表1　整体方案数据传输速度测试结果

测试项	命令帧数(数据块数量)	测试时数据长度	单逻辑通道/MB/s	四逻辑通道/MB/s	逻辑通道0/MB/s	逻辑通道1/MB/s
硬件板卡：数据读写循环1 000次计算一次	1	1.5 KB 即1 536 Byte	86	100	52	52
	7		275	200	158	159
	64		417	259	185	185
	128		430	274	218	217
	200		431	281	224	225
	220		419	282	235	225
	255		427	283	233	223

表1中所说的通道，指的是应用程序中定义的逻辑通道，用于不同应用模块与PCIE物理接口的通信。逻辑通道0和1即是四个逻辑通道中的两个。测试时，数据读写循环1 000次计算一次传输速度，以保证数据的可靠性；不断增加每次写入接口的数据块的数量，观测其对传输速度的影响；对数据进行分包处理，以每包1.5 K，即1 536 Byte的大小传输数据。由测试结果可以看出，在进行大包数据的读写时，传输速度不断提高，逐渐靠近512 MBps，可见提高单次传输数据量的大小，可有效提高接口通路的利用率。在采用PCIE×1物理通道情况下，单次传输的数据量较小时，单逻辑通道和四逻辑通道的传输速率各有高低。但在传输的数据量较大时，单逻辑通道速率明显高于多逻辑通道(多线程和多进程)速率，原因在于省去了多逻辑通道传输时的调度和控制过程。

由文献[4-5]给出的公式可知，接口通路的数据吞吐量，在协议开销、有效载荷等基本一致的情况下，与物理通道的数量成正比，因此在相同工作模式下，比较×1物理通道的传输速度即能得到多物理通道传输性能的优劣。

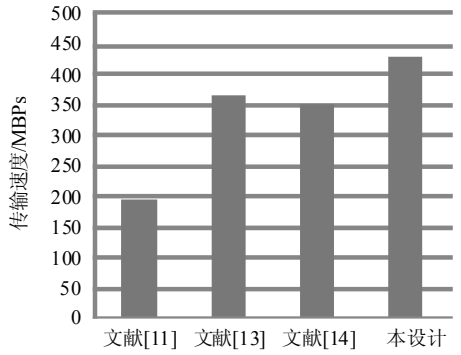

图6　4个设计方案的接口通路传输速度对比

就典型值(PCIE×1物理通道和单逻辑通道时)431 MB/s(约3.36 Gbps)而言，相较于文献[11]和文献[13-14]中给出的测试结果，提高近20%。图6为4个设计方案的接口通路传输速度测试结果对比。

文献[11]给出了接口通路在4-Lane、读写数据包大小在8 KB时的传输速度，与该条件比较，本设计传输速度有25%以上的提升；文献[13]给出了传输系统的最高传输速率，在相同工作模式下对比本设计有18%左右的提升；与文献[14]给出的测试数据相比，本设计有21%左右的提升。同样，在多物理通道时，本设计依然有着较好的传输速度。

对于在应用时选择单逻辑通道还是多逻辑通道的问题，应依据实际需求做出判断，在传输速度和要求多模块同时工作两个因素中权衡。

5　结 束 语

为有效利用PCIE接口的高速传输性能，使用ARM+FPGA架构芯片Zynq-7000系列，设计实现了多通道PCIE高速传输接口。通过设计DMA硬核控制器、PCIE总线驱动、DDR驱动以及应用测试程序，完整实现了PCIE接口通路。该方案更具可扩展性和应用前景，传输速度能达到3.36 Gbps，相比现有设计提升近20%，为设计基于PCIE接口的其他应用奠定了基础。下一步，将对设计进行优化，通过降低协议开销、优化数据调度方式等措施来进一步提高接口传输速率。

参 考 文 献

[1] PCI-SIG. PCI Express base specification revision 2.1[M]. Beaverton, USA: [s.n.], 2009.

[2] 邹晨, 高云. 基于FPGA的PCIe总线DMA传输的设计与实现[J]. 电光与控制, 2015, 7: 84-88.
ZOU Chen, GAO Yun. Design and implementation of DMA transmission with PCIe bus based on FPGA[J]. Electronics Optics & Control, 2015, 7: 84-88.

[3] CHEN Yong-zhen, WANG Yi , HA Ya-jun. sAES: a high thro-ughput and low latency secure cloud storage with pipelined DMA based PCIe interface[C]//2013 International Conference on Field-Programmable Technology (FPT). Kyoto, Japan: IEEE, 2013: 374-377.

[4] ROTA L, CASELLE M, CHILINGARYAN S, et al. A new DMA PCIe architecture for gigabyte data transmission[C]//2014 19th IEEE NPSS Real Time Conference (RT). Nara, Japan: IEEE, 2014: 1-2.

[5] ROTA L, CASELLE M, CHILINGARYAN S, et al. A PCIe DMA arc-hitecture for multi-gigabyte per second data transmission[J]. IEEE Transactions on Nuclear Science, 2015, 62(3): 972-976.

[6] HAN Tai-Lin, CAI Hua, LIU Guang-wen, et al. The face detection and location system based on Zynq[C]//2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). Xiamen, China: IEEE, 2014: 835-839.

[7] ROLAND D, LUKAS S. Towards evolvable syste- ms based on the Xilinx Zynq platform[C]//2013 IEEE International Conference on Evolvable Systems (ICES). Singapore: IEEE, 2013: 89-95.

[8] JOAO S, VALERY S, IOULIIA S. Comparison of on-chip communications in Zynq-7000 all programmable systems-on-chip[J]. IEEE Embedded System Letters, 2015, 7(1): 31-34.

[9] SANTHOSH K R, OSCAR P, JAVIRE A M, et al. An energy efficient hybrid FPGA-GPU based embedded platform to accelerate face recognition applica-tion[C]//2015 IEEE COOL Chips XVIII. Yokohama, Japan: IEEE, 2015: 1-3.

[10] 许峰. 基于Virtex-6的PCI Express高速采集卡设计[J]. 现代电子技术, 2012, 16: 79-81, 85.
XU Feng. Design of PCI Express high-speed acquisition card base on virtex-6[J]. Modern Electronics Technique, 2012, 16: 79-81, 85.

[11] KAVIANIPOUR H, MUSCHTER S, BOHM C. High performance FPGA-based DMA interface for PCIe[J]. IEEE Transactions on Nuclear Science, 2014, 61(2): 745-749.

[12] BOUGIOUKOU E, NTALLA A, PALLI A, et al. Prototyping and performance evaluation of a dynamically adaptable block device driver for PCIe-based SSDs[C]// 2014 25th IEEE International Symposium on Rapid System Prototyping (RSP 2014). New Delhi: IEEE, 2014: 51-57.

[13] 高俊, 杨灿美, 杜学亮. 基于PCIe总线的多路实时传输系统设计[J]. 电子技术应用, 2015, 2: 65-67, 71.
GAO Jun, YANG Can-mei, DU Xue-liang. Design of multi-channel real-time transmission system based on PCIe bus[J]. Applica-tion of Electronic Technique, 2015, 2: 65-67, 71.

[14] AN W, JIN X, DU X, et al. A flexible FPGA-to-FPGA communication system[C]//2016 18th International Conference on Advanced Communication Technology (ICACT2016). Pyeongchang: IEEE, 2016: 586-591.

编 辑 叶 芳