第47卷 第4期 计 算 机 工 程 2021年4月
Vol.47 No.4 Computer Engineering April 2021

·体系结构与软件技术· 文章编号：1000-3428(2021)04-0166-07 文献标志码：A 中图分类号：TP311

基于Linux的PXIe可重构仪器设备驱动程序开发

王法臻，崔少辉，王 成

（陆军工程大学石家庄校区，石家庄 050003）

摘 要： PXIe可重构仪器具备多通道并行测试能力，可用于解决共享资源测试系统中的测试资源竞争和死锁等问题。为确保PXIe可重构仪器在国产操作系统下正常运行，在Deepin操作系统下开发PXIe设备驱动程序，实现上位机与仪器设备之间的通信。介绍Linux字符设备驱动程序，基于该驱动类型结构设计PXIe设备驱动的开发流程。在此基础上，通过共享内存映射提高应用程序与驱动程序的数据交互效率，并基于阻塞和中断机制进行直接存储器存取传输。通过Qt Creator设计的图形界面测试程序对驱动程序的运行情况进行检验，测试结果表明，该设备驱动程序运行稳定，数据传输准确可靠，可满足PXIe可重构仪器的通信需求。

关键词： Linux系统；驱动程序；PXIe可重构仪器；共享内存映射；直接存储器存取；中断；Deepin操作系统

开放科学（资源服务）标志码（OSID）：

中文引用格式： 王法臻，崔少辉，王成. 基于Linux的PXIe可重构仪器设备驱动程序开发[J]. 计算机工程，2021，47(4)：166-172.

英文引用格式： WANG Fazhen，CUI Shaohui，WANG Cheng. Device driver development for PXIe reconfigurable instrument based on Linux[J].Computer Engineering，2021，47(4)：166-172.

Device Driver Development for PXIe Reconfigurable Instrument Based on Linux

WANG Fazhen，CUI Shaohui，WANG Cheng

（Shijiazhuang Campus of Army Engineering University，Shijiazhuang 050003，China）

【Abstract】 The PXIe reconfigurable instrument has the ability of multi-channel parallel test，which provides a good solution to the problems of test resource competition and deadlock in the shared-resource test system. In order to realize the smooth operation of the PXIe reconfigurable instrument under the domestic operating system，PXIe device driver is developed under the Deepin operating system to solve the communication problem between the upper computer and the instrument device. Linux character device driver is introduced and the development process of PXIe device driver is designed based on the Linux character device driver structure. On this basis，the shared-memory mapping is used to improve the efficiency of data interactions between application program and driver，and Direct Memory Access（DMA）transmission is implemented based on the block and interrupt mechanism. Qt Creator is used to design the graphic interface test program to verify the working condition of the driver，and the test results show that the device driver runs stably and the data transmission is accurate and reliable，which meets the communication requirements of PXIe reconfigurable instrument.

【Key words】 Linux system；driver；PXIe reconfigurable instrument；shared-memory mapping；Direct Memory Access（DMA）；interrupt；Deepin operating system

DOI： 10. 19678/j. issn. 1000-3428. 0058572

0 概述

PXIe总线是PCIe总线在仪器领域的拓展，通过在PCIe总线基础上附加必要的时钟信号、触发总线、星形总线和本地总线等，可形成PXIe扩展信号[1]。PXIe总线采用点对点的通信方式并利用差分信号进行数据传输，可使数据的传输速率和品质得到大幅提升。

设备驱动程序作为操作系统的重要组成部分，是应用程序与硬件完成数据交互所必需的媒介[2-3]。为满足自主设计的PXIe可重构仪器与上位机之间的通信需求，在完成PXIe接口硬件部分设计的基础

基金项目： 国家自然科学基金（61501493）。

作者简介： 王法臻（1992—），男，硕士研究生，主研方向为装备测试技术；崔少辉，教授、博士；王 成，讲师、博士。

收稿日期： 2020-06-08 **修回日期：** 2020-07-23 **E-mail：** 527316850@qq.com

上，必须开发相应的PXIe设备驱动程序才能实现功能控制指令和测试数据经PXIe总线的传输。因此，设计稳定可靠的驱动程序十分关键。

本文在研究Linux字符设备驱动结构的基础上开发可重构仪器的PXIe设备驱动程序。设计直接存储器存取（Direct Memory Acess，DMA）传输操作流程，并对应用程序与驱动程序数据的交互方式进行优化，从而提高数据传输效率，使PXIe可重构仪器能够在国产操作系统下稳定运行。

1　PXIe可重构仪器

通用自动测试系统（Automatic Test System，ATS）采用共享资源架构，其通过开关系统分配测试通道和资源，易产生开关延迟、测试资源竞争和死锁等问题[4]。多通道可重构仪器具备良好的通用性和灵活性[5-6]，可根据用户需求进行重构从而获得不同的测试能力，且每路测试通道均具有独立完成测试的能力，因此为传统通用ATS存在的问题提供了良好的解决方案。

多通道PXIe可重构仪器以Altera公司的Cyclone IV系列FPGA为核心，配置测量功能电路模块，其32路测试通道均具备数模转换、模数转换、波形产生以及电压比较等功能，各通道可根据测试需求通过重构实现一定的功能扩展。上位机与仪器设备经PXIe总线接口完成功能控制指令及测试数据的传输。由于PXIe与PCIe遵循相同的协议，因此PXIe接口的开发通过FPGA中的PCIe IP硬核实现，配置选择为PCIe 1.1 x4链路，理论上最大传输速率可达1 GB/s。该实现方式相比采用专用芯片进行接口开发的优势在于可简化电路设计同时降低开发成本。

2　Linux设备驱动程序

Linux贯彻“一切皆文件”的思想，因此各类外部设备也都被看作是文件，一般称作设备文件（或设备节点）[7]。用户层应用程序并不能直接访问操作外部硬件设备，需要先通过如open、close、read、write和unlocked_ioctl等系统调用访问设备文件，然后借助Linux中虚拟文件系统（Virtual File System，VFS）所提供的访问接口寻找各个系统调用的响应函数，从而进一步操作外部硬件设备[8]。

设备文件所对应的设备驱动程序负责为操作外部硬件设备提供与系统调用相关联的响应函数，这些响应函数可直接访问外部硬件设备。因此，用户层应用程序必须通过设备驱动程序来实现对外部硬件设备的访问操作。Linux系统下设备驱动程序、应用程序及硬件设备三者之间的关系如图1所示。

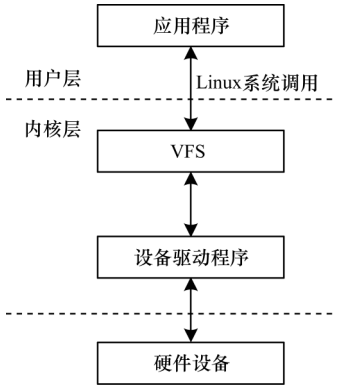

图1　设备驱动、应用程序与硬件设备间的关系

Fig.1　Relationship of device driver，application and hardware device

Linux设备驱动程序按设备类型一般划分为字符设备驱动、块设备驱动和网络接口驱动3类[9]。字符设备以字节为单位传输数据，块设备以块为单位（每块至少512 Byte）传输数据，网络设备以用户数据包形式在网络媒介上传输数据。

PXIe设备驱动程序以字符设备驱动结构为主体框架进行设计开发，因此，需要先对字符设备驱动的相关情况有所掌握。字符设备、字符设备驱动和应用程序之间的结构关系如图2所示，具体描述如下：

1）应用程序通过系统调用经VFS提供的接口访问驱动程序，驱动程序根据应用程序要求进一步操作访问设备。

2）结构体cdev负责描述字符设备驱动，其主要由设备号和文件操作结构体两部分组成。当系统调用宏module_init加载驱动时，即实现该结构体的实例化并完成字符设备的注册。

3）设备号dev_t负责存放驱动程序给设备分配的主设备号和次设备号，主设备号负责标识设备类型，次设备号负责标识同类设备中的特定设备。由于设备号是设备文件在系统中的标志且唯一确定，因此新设备号的申请不能与已有设备号发生冲突。

4）文件操作结构体file_operations作为用户层与内核层交互的接口，是建立起应用程序与驱动程序交互的关键所在。当用户层应用程序使用open、release、unlocked_ioctl等Linux系统调用时，系统经由VFS间接寻找并执行file_operations结构体中所对应的xxx_open、xxx_release、xxx_unlocked_ioctl等操作函数，完成对设备的指定操作。

5）驱动卸载在系统调用宏module_exit时完成，负责设备号的释放和所申请的相关资源并注销cdev结构体。

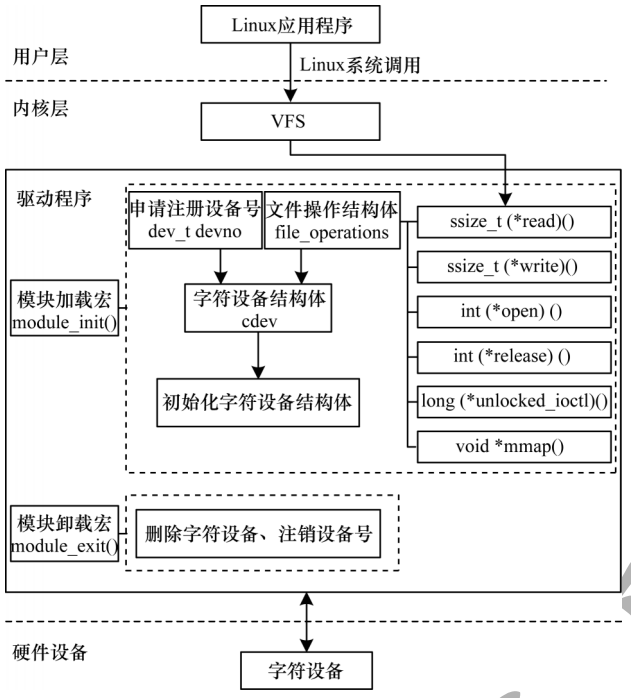

图2 字符设备驱动程序结构

Fig.2 Structure of character device driver

为使仪器能够在国产操作系统下工作运行,驱动程序的开发环境选择国产Deepin 15.11操作系统,Linux内核版本号4.15.0-30-generic,使用系统自带vim编辑器和GCC编译器完成驱动程序代码的编写及编译。

3 PXIe设备驱动设计

PXIe设备驱动程序基于字符设备驱动结构的开发设计流程如图3所示。

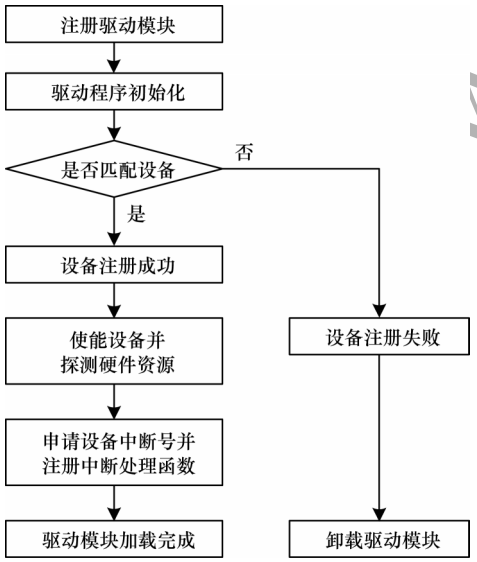

图3 PXIe设备驱动程序设计流程

Fig.3 Procedure of PXIe device driver design

3.1 驱动初始化

驱动程序的初始化主要完成对结构体cdev和结构体pci_driver的初始化,目的在于完成PXIe设备在系统中的注册、硬件资源的获取以及建立与应用程序的交互关系。

cdev结构体已在上文进行相关介绍,主要实现设备在系统中的注册并建立与应用程序的交互关系。设备号dev_t申请成功后,通过调用函数cdev_init初始化cdev中的file_operations结构体,建立其与应用程序的交互关系,通过调用函数cdev_add向系统添加字符设备从而完成设备注册。Linux系统调用所对应的file_operations结构体成员函数设置如下:函数pxie_open和pxie_release负责连接和断开应用程序与驱动程序的连接;函数dma_mmap负责实现内存共享映射以提高应用层与内核层间数据交互速率;函数pxie_read和pxie_write负责对BAR空间上寄存器的访问,函数dma_ unlocked_ioctl用于DMA传输控制指令的发送。

pci_driver结构体实现驱动程序与特定设备的匹配以及硬件资源信息的获取。驱动初始化的实现代码如下:

```
static struct pci_driver pxie_driver={
.name = "PXIE_DEVICE",/*设备驱动名称*/
.id_table = pxie_id,/*设备与驱动匹配*/
.probe = pxie_probe,/*设备资源探测*/
.remove = pxie_remove,/*设备资源删除*/};
pci_register_driver(&pxie_driver);
```

3.1.1 设备匹配

操作系统启动后会自动检测总线上的所有设备,并将厂商号(VendorID)、设备号(DeviceID)等硬件信息记录在pci_dev结构体中。驱动程序即通过这些硬件信息实现与设备的匹配连接。

驱动程序的pci_device_id结构体中包含设备的ID信息,通过宏MODULE_DEVICE_TABLE将其导出至用户空间。在驱动初始化过程中,当调用函数pci_register_driver初始化pci_driver结构体时,系统会自动匹配pci_dev结构体与驱动程序pci_device_id结构体中的设备ID信息。若匹配成功,则激发驱动程序调用pxie_probe函数去探测设备以获取硬件资源信息[10-11]。设备匹配的实现代码如下:

```
static struct pci_device_id pxie_id [] __initdata = {
{PXIE_VENDOR_ID, PXIE_DEVICE_ID, PCI_ANY_ID, PCI_ANY_ID,0,0,0},{0,}};
MODULE_DEVICE_TABLE(pci,pxie_id);
```

3.1.2 设备探测

设备硬件资源的探测通过pxie_probe函数进行,主要完成I/O资源的申请、基址寄存器(Base

Address Register，BAR）空间基地址的获取、设备中断号的申请以及中断处理函数的注册，同时将相关资源信息保存至私有数据结构体private_data中供驱动程序使用。设备探测的实现代码如下：

```
static int pxie_probe (struct pci_dev *dev, const struct pci_device_id *id){
    pci_enable_device(dev);/*使能设备*/
    pci_set_master(dev);
    pci_set_dma_mask(dev,DMA_BIT_MASK(32));
    pci_request_regions(dev,private_data->name);
    /*申请I/O资源，确保该区域不被其他程序占用*/
    private_data->bar0_addr=pci_resource_start(dev,0);
    private_data->bar0_len=pci_resource_len(dev,0);
    private_data->bar1_addr=pci_resource_start(dev,1);
    private_data->bar1_len=pci_resource_len(dev,1);
    private->bar0=ioremap(bar0_addr,bar0_len);
    private->bar1=ioremap(bar1_addr,bar1_len);
    /*BAR物理地址映射为驱动使用的虚拟地址*/
    request_irq (dev->irq, intrpt_handler, IRQF_SHARED, private_data->name,private_data);/*申请中断号，注册中断处理函数intrpt_handler*/
    private->irq=dev->irq;
    ……
    return 0;}
```

PXIe设备共有6个BAR可使用，均可配置为I/O空间或存储器空间，两者在访问方式上有所不同，设备仅使用BAR0、BAR1且配置为存储器空间。在完成BAR物理地址到内核空间虚拟地址的映射后，便可通过函数ioread32/iowrite32访问BAR中相应地址上的配置寄存器实现DMA传输。

3.1.3　设备删除

通过调用函数cdev_del和pci_unregister_ driver分别注销cdev结构体和pci_driver结构体，以实现设备的移除并调用pxie_remove函数删除所申请的硬件资源。设备删除的实现代码如下：

```
static void pxie_remove(struct pci_dev *dev){
    free_irq(dev->irq,private_data);/*释放中断号*/
    iounmap(private_data->bar0);
    iounmap(private_data->bar1);/*解除BAR物理地址映射*/
    pci_release_regions(dev);/*释放I/O资源*/
    pci_disable_device(dev);/*禁用设备*/}
```

3.2　共享内存映射

以DMA方式向设备发送数据，用户层应用程序需要先将数据传输至内核层驱动程序的DMA缓冲区中，再通过对该缓冲区的操作写入设备。以DMA方式从设备中读取数据沿反方向执行。在频繁进行数据传输时，大量数据若通过函数copy_from_user和copy_to_user以拷贝的方式实现用户层和内核层之间的数据交互，会影响整体传输效率[12]，而通过建立共享内存映射可有效解决这一问题[13-14]。

共享内存映射是指将一段物理内存同时映射出用户空间虚拟地址和内核空间虚拟地址，这样应用程序和驱动程序均可对该段物理内存进行访问，从而减少数据拷贝带来的时间开销，提高数据的传输效率。共享内存映射的具体实现由系统调用mmap，通过其在file_operations结构体中的响应函数dma_mmap完成。利用虚拟内存区（Virtual Memory Areas，VMA）结构将驱动程序创建的DMA缓冲区的虚拟地址转换到该段内存的物理地址，然后通过此物理地址为该段内存建立新的页表并映射出起始的用户空间虚拟地址供应用程序使用[15-16]。共享内存映射的实现代码如下：

```
static int dma_mmap(struct file *filp,struct vm_area_stuct *vma){
    unsigned long start,end,size;
    start=(unsigned long)vma->vm_start;
    end=(unsigned long)vma->vm_end;
    size=(unsigned long)(end-start);
    private_data->dma_len= DMA_LENGTH;
    pci_alloc_consistent(private_data->dev,private_data->dam_len,&private_data->dam_bus_addr);
    /*创建DMA缓冲区*/
    for (page=virt_to_page(private_data->dam_kva); page<virt_to_page ((char*) private_data->dma_kva+private_data->dma_len);page++)
    { SetPageReserved(page);}/*建立新的页表*/
    private_data->dma_phy=virt_to_ phys (private_data->dma_ kva);
    pfn=private_data->dma_phy>>PAGE_SHIFT;
    remap_pfn_range (vma,start,pfn,size,vma->vm_page_prot); /*用户空间虚拟地址映射到物理地址*/}
```

用户层应用程序通过mmap系统调用所得到的返回地址就是该段共享内存在用户层的虚拟地址，应用程序通过该地址可直接对DMA缓冲区进行访问。

```
unsigned int *user_dmaaddress=(unsigned int *) mmap(NULL,DMA_LENGTH,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
```

当设备停止使用时，应解除共享内存映射关系并释放所申请的DMA缓冲区。应用程序在release系统调用的响应函数pxie_release中，通过调用函数munmap和pci_free_consistent实现。

3.3　DMA及中断处理

上位机同设备之间的数据传输采用DMA方式实现。由于在FPGA内部已设计实现DMA控制器，因此CPU只需要完成对DMA传输相关寄存器的配置以及中断信号的处理即可，而不必直接参与数据传输，从而有效降低CPU负荷并提高数据的传输速率。DMA传输操作流程如图4所示。

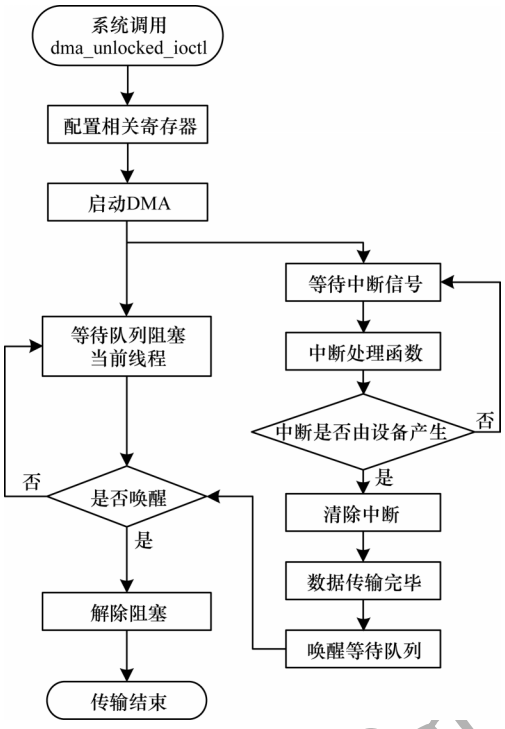

图4 DMA传输操作流程

Fig.4 Procedure of DMA transmission operating

在启动DMA传输前,通过函数iowrite32设置BAR空间上DMA传输首地址、传输长度以及中断服务寄存器,然后设置读写控制寄存器以启动DMA传输。同时,通过等待队列方式阻塞当前进程直至设备向上位机发出中断信号,该信号是数据传输完毕的标志。

由于接口设计采用INTx中断方式,设备可能与其他设备共享同一中断号,系统在实际运行过程所接收到的中断信号可能源自其他设备,因此还需要对中断信号的来源进行判断,即通过函数ioread32检查BAR空间上中断服务寄存器的数值变化。若中断信号确由设备产生,则清除中断信号并唤醒等待队列所阻塞的进程以通知应用程序传输结束。

4 驱动程序的测试

Linux驱动程序以内核模块形式通过静态或动态的方式加载至内核中[17]。由于静态加载方式在内核编译以及重启系统过程中会带来大量的时间开销,因此在驱动程序开发调试阶段采用动态加载方式添加和移除驱动模块。驱动模块的加载与卸载通过命令insmod和命令rmmod实现[18-19]。

4.1 加载与卸载

驱动代码文件pxie_device.c的编译通过vim编辑器编写Makefile文件实现。由于Makefile文件可通过make工具自动完成编译工作,因此只需要通过系统终端在源代码文件所在目录下执行make命令,即可通过GCC编译器生成驱动模块pxie_device.ko。pxie_device.ko通过动态方式加载至内核后需创建相应的设备节点(设备文件),该节点的创建为应用程序提供了访问设备的接口。加载与卸载的实现代码如下:

1.系统终端实现驱动模块加载的Shell命令

sudo insmod pxie_device.ko/*调用驱动模块加载宏module_init*/

sudo mknod /dev/pxie_device c 240 0/*添加设备节点(系统分配的主设备号240,从设备号0)*/

sudo chmod 666/dev/pxie_device/*更改节点用户读写权限*/

lsmod/*打开系统驱动模块显示列表,检查驱动加载是否成功*/

2.系统终端实现驱动模块卸载的Shell命令

sudo rmmod pxie_device/*调用驱动模块卸载宏module_exit*/

sudo rm-rf /dev/pxie_device/*删除设备节点*/

系统驱动模块显示列表如图5所示。

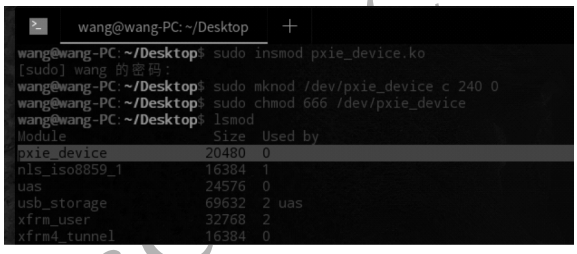

图5 系统驱动模块显示列表

Fig.5 Display list of system driver module

4.2 测试结果

Qt是一个跨平台的C++应用程序开发框架[20],本文通过其集成开发环境Qt Creator以图形化界面的方式开发实现测试程序,如图6所示。测试程序通过对FPGA中FIFO(First Input First Output)存储器进行数据读写传输以验证驱动程序是否符合设计需求,数据校验结果如图7所示。

图6 测试程序界面

Fig.6 Test program interface

图7　数据校验结果

Fig.7　Data verification result

在测试过程中，数据实际传输速率较理论峰值存在一定差距，这主要是由于软件时间花销和协议规范消耗带宽（8b/10b编码以及各协议层数据包封装）所导致。当数据量较小时，这部分花销占用比较大，导致实际测得的传输速率较低；当数据量较大时，这部分花销占用比减小，从而使得传输速率显著提高。

DMA写（设备接收上位机发送的数据）数据传速率可达350 MB/s，DMA读（上位机接收设备发送的数据）数据传输速率可达420 MB/s。在传输相同数据量的情况下，DMA读速率大于DMA写速率，这主要是因为在软件从设备读取数据时，硬件端只需发送存储器写请求TLP即可；而向设备写入数据时，硬件端在发送存储器读请求TLP后，还需要有存储器读完成报文进行应答，从而产生了一些时间花销。

5　结束语

目前，Windows操作系统仍占据国内操作系统绝大部分市场份额，但微软已停止对Win7的技术支持，而Win8/Win10安全性较差，因此，实现面向国产操作系统的软件开发具有重大意义。本文介绍国产Deepin操作系统下PXIe驱动程序的开发方法，对应用程序与驱动程序间的数据交互过程进行优化，同时结合阻塞机制和中断机制设计DMA传输操作流程。测试结果表明，该方法数据传输准确稳定，可满足PXIe可重构仪器的数据通信需求，并且其通用性较好，在工程实践应用中具有一定参考价值。在实际测试系统中往往需要使用多个同类型仪器，下一步将实现应用程序通过单个驱动同时对多个可重构仪器的操作控制及数据传输。

参考文献

[1] ZHANG Qinxin. Design and implementation of high-speed data transmission platform based on PXIe high-speed interface[D]. Chengdu: University of Electronic Science and Technology of China, 2018. (in Chinese)
张沁馨. 基于PXIe高速接口的高速数据传输平台的设计与实现[D]. 成都：电子科技大学，2018.

[2] ZHANG Yifan, HUANG Chao, OU Jiansheng, et al. Research on reliability and correctness assurance methods and techniques for device drivers[J]. Journal of Software, 2015, 25(2): 239-253. (in Chinese)
张一帆，黄超，欧建生，等. 设备驱动程序可靠性和正确性保障方法与技术研究进展[J]. 软件学报，2015，25(2)：239-253.

[3] WAN Maning, HOU Zhihua. Development and implementation of transmission of USB device driver based on Linux operating system[J]. Microelectronics & Computer, 2018, 35(11): 124-127. (in Chinese)
万玛宁，侯志化. 基于Linux操作系统的USB设备驱动程序开发及传输实现[J]. 微电子学与计算机，2018，35(11)：124-127.

[4] GOU Mingze, CUI Shaohui. Design of the PXIe bus reconfigurable test instrument[J]. Process Automation Instrumentation, 2017, 38(11): 68-70, 74. (in Chinese)
苟铭泽，崔少辉. PXIe总线可重构测试仪器设计[J]. 自动化仪表，2017，38(11)：68-70，74.

[5] HONG Wanfan. Design of multifunctional instrument by the reconfigurable technology based on FPGA[D]. Taiyuan: North University of China, 2016. (in Chinese)
洪万帆. 基于FPGA可重构技术的多功能仪器设计[D]. 太原：中北大学，2016.

[6] FU Zhenpeng. Research on PXI reconfigurable instrument for missile tests[D]. Harbin: Harbin Institute of Technology, 2014. (in Chinese)
付振鹏. 面向导弹测试的PXI可重构仪器研究[D]. 哈尔滨：哈尔滨工业大学，2014.

[7] YANG Bingjian, WEI Feng, CHEN Yongzhi. Device driver development of PCIE synchronous clock card based on Linux[J]. Computer Measurement and Control, 2017, 25(1): 98-100, 104. (in Chinese)
杨兵见，魏丰，陈永志. Linux下的PCIE同步时钟卡的设备驱动程序开发[J]. 计算机测量与控制，2017，25(1)：98-100，104.

[8] LIU Kai, HU Ailan. Design of driver based on PCI-E timing card on Linux[J]. Microcomputer & Its Applications, 2015, 34(24): 13-15, 18. (in Chinese)
刘凯，胡爱兰. Linux下基于PCI-E时统卡的驱动程序设计[J]. 微型机与应用，2015，34(24)：13-15，18.

[9] SONG Baohua. Linux device driver development details: based on the latest Linux 4.0 kernel[M]. Beijing: China Machine Press, 2016. (in Chinese)
宋宝华. Linux设备驱动开发详解：基于最新的Linux4.0内核[M]. 北京：机械工业出版社，2016.

[10] CHEN Mengtong, WEI Feng, YANG Bingjian. Driver design of PCI synchronous clock card under Linux[J]. Computer Measurement & Control, 2018, 26(1): 145-148. (in Chinese)
陈梦桐，魏丰，杨兵见. Linux下PCI同步时钟卡的驱动程序设计[J]. 计算机测量与控制，2018，26(1)：145-148.

[11] VENKATESWARAN S. Essential Linux device drivers[M]. Translated by SONG Baohua, HE Zhaoran, SHI Haibin, et al.

Beijing:Posts & Telecom Press,2016. (in Chinese)
VENKATESWARAN S. 精通Linux设备驱动程序开发[M]. 宋宝华,何昭然,史海滨,等译. 北京:人民邮电出版社,2016.

[12] WANG Shiming, YAN Fabao, XU Zhiyong, et al. Design kernel multithread direct storage PCIE driver[J]. Computer Applications and Software, 2016, 33(9): 235-237, 263. (in Chinese)
王仕明,严发宝,徐智勇,等. 内核多进程直接存储PCIE驱动设计[J]. 计算机应用与软件,2016,33(9):235-237,263.

[13] YU Jin, HUANG Hao, ZHU Yu, et al. DBox: high-performance secure boxes for various device drivers of monolithic kernels[J]. Chinese Journal of Computers, 2020, 43(4): 724-738. (in Chinese)
余劲,黄皓,诸渝,等. DBox:宏内核下各种设备驱动程序的高性能安全盒[J]. 计算机学报,2020,43(4):724-738.

[14] BAEK S H, PARK K W. Compatible byte-addressable direct I/O for peripheral memory devices in Linux[J]. Information Systems, 2020, 91: 1-10.

[15] YING Sancong, WANG Mingyin, ZHANG Xing. Key technologies of high-performance PCI driver[J]. Computer Engineering and Design, 2012, 33(6): 2208-2212. (in Chinese)
应三丛,汪明寅,张行. 高性能PCI驱动程序的关键技术[J]. 计算机工程与设计,2012,33(6):2208-2212.

[16] LAO Wei. Linux kernel mmap protection mechanism research[J]. Journal of Information Security Research, 2018, 12(4): 1135-1141. (in Chinese)
劳伟. Linux内核mmap保护机制研究[J]. 信息安全研究,2018,12(4):1135-1141.

[17] WANG Jun, HAN Li, DU Bojun, et al. Design and implementation of driver of Beidou navigation timing card under Linux PCI-E[J]. Computer Measurement & Control, 2016, 24(4): 115-117. (in Chinese)
王军,韩力,杜博军,等. 基于PCI-E总线的北斗导航授时卡Linux驱动设计[J]. 计算机测量与控制,2016,24(4):115-117.

[18] ZHAI Gaoshou, ZHAI Ruixia, LIU Feng, et al. Study and implementation of fault injection method for device drivers[J]. Netinfo Security, 2019, 19(6): 19-27. (in Chinese)
翟高寿,翟瑞霞,刘峰,等. 设备驱动故障注入方法的研究与实现[J]. 信息网络安全,2019,19(6):19-27.

[19] ZHAO Xin, GUO Enquan, LI Xiaojie. Design and implementation of GPIB driver optimal in LINUX system[J]. Computer Measurement & Control, 2020, 28(3): 163-167. (in Chinese)
赵昕,郭恩全,李小杰. LINUX系统下GPIB驱动优化设计与实现[J]. 计算机测量与控制,2020,28(3):163-167.

[20] TAN Jingjia, HE Lesheng, WANG Jun, et al. Design of AXI DMA data transmission and display based on Zedboard[J]. Video Engineering, 2018, 42(6): 41-45. (in Chinese)
谭景甲,何乐生,王俊,等. 基于Zedboard平台AXI DMA数据传输与显示的设计[J]. 电视技术,2018,42(6):41-45.

编辑 金胡考