2014年9月13日

**PCIE 数据采集卡数据传输功能模块**

**使用说明书**

[1. 介绍 3](#_Toc382159597)

[2. 驱动软件结构 3](#_Toc382159598)

[3. 用户软件接口 4](#_Toc382159599)

[3.1. open和close 4](#_Toc382159600)

[3.2. 读写缓冲区与读写粒度配置 4](#_Toc382159601)

[3.3. read 6](#_Toc382159602)

[3.4. write 6](#_Toc382159603)

[3.5. ioctl 7](#_Toc382159604)

[3.5.1. PCIE BAR寄存器的读写 7](#_Toc382159605)

[3.5.2. 获取读写缓冲区大小和读写块大小 8](#_Toc382159606)

[3.5.3. 启动DMA 9](#_Toc382159607)

[3.5.4. 关闭DMA 10](#_Toc382159608)

[4. FPGA逻辑接口 10](#_Toc382159609)

[4.1. 读FIFO接口 10](#_Toc382159610)

[4.2. 写FIFO接口 11](#_Toc382159611)

[4.3. BAR空间接口 11](#_Toc382159612)

[4.4. 中断接口 12](#_Toc382159613)

[4.5. 其它接口 12](#_Toc382159614)

[5. 主要操作流程 14](#_Toc382159615)

[5.1. 启动采集卡的读写操作 14](#_Toc382159616)

[5.2. 停止读写 15](#_Toc382159617)

# 介绍

本说明书是为PCIE 数据采集卡的PCIE数据传输功能所作的软硬件的说明，包括软硬件的用户接口以及将PCIE数据传输功能集成进系统时的注意事项。

PCIE数据采集卡采用PCIE2.0X4接口与主机做数据交互。主机是基于Intel X86 的服务器，操作系统为Ubuntu, 主机内安装PCIE转接卡，通过cable连接本数据采集卡。

本数据采集卡的PCIE数据传输功能由软硬件配合完成，软件工作在主机端，硬件功能由采集卡上的Xilinx V6系列FPGA器件完成。

本数据采集卡与主机端传输的最大数据量可达到读写同时400MByte/s。此外，主机端还需要通过PCIE的BAR空间访问采集卡上的其它功能单元。本概要设计将基于以上目标提出软硬件的相关结构，接口，以及使用方法。

# 用户软件接口

## open和close

采集卡设备驱动会在/dev/目录下呈现给用户一个字符设备。用户可以调用标准的Linux系统调用（如open, read, write等）操作字符设备，就像操作普通文件一样，但有些操作上会有些细微区别。

open和close接口和普通文件的打开关闭接口是一样的。在对采集卡做任何操作前需要先open设备，当需要关闭采集卡的功能时需要close设备。

**系统调用open的语法格式**：

*fd = open(pathname, flags);*

这里，pathname是文件名；flags是打开的类型（如读或写）。系统调用open返回一个称为文件描述符的整数，如果失败则返回-1。其它系统调用，如read，write，ioctl等，都要使用系统调用open返回的文件描述符。

**系统调用open举例**：

*fd = open(“dev/pdma”, O\_RDONLY);* //以只读方式打开

*fd = open(“dev/pdma”, O\_WRONLY);* //以只写方式打开

*fd = open(“dev/pdma”, O\_RDWR);* //以读写方式打开

**系统调用close的语法格式**：

*close(fd);*

这里，这里fd是一个已经打开的文件描述符。

**系统调用close举例：**

*close(fd);* //关闭由open系统调用返回的文件描述符fd

## 读写缓冲区与读写粒度配置

由于数据采集卡的A/D部分以固定速率产生/消耗数据，而用户软件读写数据的速率不能与A/D的速率精确同步，为保证读写数据不丢失，驱动会管理一组读写数据缓冲区, 读写缓冲区大小是可以配置的。用户可在驱动模块编译时静态配置，或在加载模块时动态指定。具体配置的大小取决于主机端的内存情况以及实际的需要。建议最小不能小于128MB，在内存足够的情况下，可适当配大一些，注意此配置在驱动加载后，采集卡工作期间不能修改。

**注意，用户程序一旦启动了DMA和数据采集卡的采集功能，数据会不断地在读缓冲区堆积，用户程序必须尽快做read操作，如果长时间不读，读缓冲区会溢出。如果发生缓冲区溢出，相关信息将记录在消息日志中。**

为保证读写带宽能够达到400MB/s，用户读写操作每次需要以固定大小进行，称为读写粒度，比如16KB，64KB，或者512KB等。用户可在驱动模块编译时静态配置读写粒度的大小，或在加载模块时动态指定。注意此项配置必须是4KB的整数倍，最小为4KB，最大不能超过1MB。但为了保证能够达到400MB/s的性能，建议读写粒度最小配置为16KB。

**静态配置的方法：**

修改pdma-dev.h的宏PDMA\_POOL和PDMA\_BLOCK来分别实现对读写缓冲区和读写粒度的静态配置。例如：

*#define PDMA\_POOL "128m"* //将读写缓冲区各自配置为128MB(总共256MB)

*#define PDMA\_BLOCK "16k"* //将读写粒度配置为16KB

**动态配置的方法：**

可以在驱动模块加载时通过pool和block模块参数来分别实现对读写缓冲区和读写粒度进行配置。例如：

*Insmod pdma.ko pool=128m* //将读写缓冲区各自配置为128MB(总共256MB)

*Insmod pdma.ko block=16k* //将读写粒度配置为16KB

*Insmod pdma.ko pool=256m block=64k* //同时指定缓冲区和读写粒度大小

**注意：**

1. 缓冲区的大小必须是读写粒度大小的整数倍。读写粒度大小必须是4KB的整数倍，同时不能大于1MB。
2. 动态配置的会覆盖静态配置。如果模块加载时指定了pool或block，那么静态配置的宏将不再起作用。
3. 缓冲区和读写粒度大小配置（包括静态和动态配置）支持G/g, M/m, K/k，它们分别代表GB, MB和KB；同时也支持十六进制和十进制格式。
4. 读写缓冲区和读写粒度大小在模块加载后，不能动态修改。

## read

Read操作意味着从采集卡上读出一批数据。

**系统调用read的语法格式**：

*number = read(fd, buffer, count);*

这里，fd是由open返回的文件描述符，buffer是用户进程中的一块内存地址。在该调用成功结束时，该地址中将存放所读的数据。count是要读的字节数，number是实际读的字节数。

**注意：**

1. read接口和普通文件的读接口基本相同。但是每次read的粒度需要根据3.2中描述的读写粒度来进行，否则read会返回失败。也就是说用户程序一定要先按照粒度准备好了内存空间再发read操作。
2. 为防止读缓冲区溢出，用户程序需要尽快的将数据从内核缓冲区中读走。
3. 另外，这里的read操作是阻塞的，当内核读缓冲区中没有数据时，读操作会被阻塞，进程开始休眠。当内核缓冲区中有数据时，驱动再将数据拷贝给用户的buffer，之后唤醒用户进程。

## write

write操作意味着向采集卡写入一批数据。

**系统调用write的语法格式**：

*number = write(fd, buffer, count);*

这里，fd是由open返回的文件描述符，buffer是用户进程中的一块内存地址。在该调用成功结束时，该地址中存放的数据将被写到文件中。count是要写的字节数，number是实际写的字节数。

**注意：**

1. write和read类似，也要以一个固定大小做写操作，否则会返回失败。这个大小由3.2中描述的读写粒度配置来决定。
2. 当写缓冲区不满时，写数据会被缓冲区接纳并立刻向PCIE接口转发。当缓冲区满的时候，写操作会被阻塞。

## ioctl

ioctl系统调用主要向用户提供一些对设备的控制。

**系统调用ioctl的语法格式**：

*ret = ioctl(fd, cmd, …);*

这里，fd是由open返回的文件描述符，cmd就是用户程序对设备的控制命令，至于后面的省略号，那是一些补充参数，一般最多一个，有或没有是和cmd的意义相关的。ioctl系统调用在成功时返回0，失败时返回-1。

采集卡设备的ioctl包含以下几方面，每种控制都对应唯一的一个cmd。驱动程序已为用户程序提供了一个pdma-ioctl.h的头文件，里面为每种控制都提供了一个唯一的cmd宏定义，用户程序在使用ioctl系统调用时需要include该头文件。

### PCIE BAR寄存器的读写

设备卡为用户提供了256 Bytes的BAR空间供外部逻辑使用。用户程序可以通过ioctl系统调用直接对该256 Bytes进行读写。

驱动提供对PCIE BAR空间的访问接口，为防止对BAR空间的访问堆积在PCIE链路上影响带宽，对PCIE BAR空间的读写操作每次只能访问一个word (32bits)。

**ioctl定义：**

该ioctl的cmd为PDMA\_IOC\_RW\_REG（在pdma-ioctl.h中定义）；它用到如下结构体作为其参数。

*struct pdma\_rw\_reg {*

*unsigned int type; /\* 0 for read, non-zero for write \*/*

*unsigned int addr; /\* access address, from 0 to 256, must be 4 aligned \*/*

*unsigned int val; /\* read/write value \*/*

*};*

type：用于区分寄存器的读或写，0表示读，非0表示写；

addr: 寄存器的读写地址，范围为0到255，单位为Byte，必须是4对齐；

val: 对于读，当ioctl成功返回时该值中存放从寄存器中读回的值；

对于写，该值表示要写入的值，用户在调用ioctl前需要先初始化它。

**举例：**

以写操作为例。

*struct pdma\_rw\_reg ctrl;*

*ctrl.type = 1;*

*ctrl.addr = 0;*

*ctrl.val = 0x01;*

*ret = ioctl(fd, PDMA\_IOC\_RW\_REG, (unsigned long)&ctrl);*

*if (ret == -1) error handler;*

### 获取读写缓冲区大小和读写块大小

用户可以通过ioctl接口获取内核读写缓冲区的大小，以及读写粒度大小。

**ioctl定义：**

该ioctl的cmd为PDMA\_IOC\_INFO（在pdma-ioctl.h中定义）；它用到如下结构体作为其参数。

*struct pdma\_info {*

*unsigned long rd\_pool\_sz; /\* read pool size \*/*

*unsigned long wt\_pool\_sz; /\* write pool size \*/*

*unsigned int rd\_block\_sz; /\* read block size \*/*

*unsigned int wt\_block\_sz; /\* write block size \*/*

*};*

*rd\_pool\_sz*：读缓冲区的大小；

*wt\_pool\_sz*: 写缓冲区的大小；

*rd\_block\_sz*: 读粒度的大小；

*wt\_block\_sz*: 写粒度的大小；

**举例：**

*struct pdma\_info info;*

*info.type = 1;*

*info.addr = 0;*

*info.val = 0x01;*

*ret = ioctl(fd, PDMA\_IOC\_INFO, (unsigned long)&info);*

*if (ret == -1) error handler;*

### 启动DMA

在开启采集卡的A/D之间需要先启动DMA，驱动将完成DMA初始化的准备工作。

**ioctl定义：**

该ioctl的cmd为PDMA\_IOC\_START\_DMA（在pdma-ioctl.h中定义）；它不需要额外的参数。

**举例：**

*ret = ioctl(fd, PDMA\_IOC\_START\_DMA);*

*if (ret == -1) error handler;*

**注意：**

1. 用户程序在加载完模块后，读写之前要先调用该接口启动DMA；
2. 用户程序在关闭DMA后，读写前也要重新启动DMA。

### 关闭DMA

用户在停止采集卡的A/D之后需要关闭DMA，驱动将对DMA和相关buffer资源做释放。

**ioctl定义：**

该ioctl的cmd为PDMA\_IOC\_STOP\_DMA（在pdma-ioctl.h中定义）；它不需要额外的参数。

**举例：**

*ret = ioctl(fd, PDMA\_IOC\_STOP\_DMA);*

*if (ret == -1) error handler;*

**注意：**

1. 用户程序卸载模块前，要先调用该接口关闭DMA；

## 使用示例

使用示例可参考软件包中的使用说明和代码。

# FPGA中DMA与user逻辑接口

FPGA器件中的DMA模块负责在PCIE接口和采集卡上的user逻辑之间做数据传递，同时也为user逻辑提供PCIE BAR空间接口。

## DMA与user逻辑的TX FIFO接口

DMA从TX FIFO接口中读取数据，向PCIE接口转发。

接口信号：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 宽度(bit) | 描述 |
| tx\_dat\_fifo\_rclk | DMA->USER | 1 | FIFO的读时钟，62.5MHz |
| tx\_dat\_fifo\_rclk\_rst\_n | DMA->USER | 1 | Reset，低有效 |
| tx\_dat\_fifo\_rden | DMA->USER | 1 | FIFO读enable, 高有效 |
| tx\_dat\_fifo\_dout | USER->DMA | 64 | FIFO数据 |
| tx\_dat\_fifo\_empty | USER->DMA | 1 | FIFO 空标识，高有效。 |

另外，建议TX FIFO深度最好>=32

## DMA与user逻辑的RX FIFO接口

DMA从PCIE接口接收数据并向RX FIFO转发。

接口信号：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 宽度(bit) | 描述 |
| rx\_dat\_fifo\_wclk | DMA->USER | 1 | FIFO的写时钟，62.5MHz |
| rx\_dat\_fifo\_wclk\_rst\_n | DMA->USER | 1 | Reset，低有效 |
| rx\_dat\_fifo\_wren | DMA->USER | 1 | FIFO写enable，高有效 |
| rx\_dat\_fifo\_din | DMA->USER | 64 | FIFO数据 |
| rx\_dat\_fifo\_full | USER->DMA | 1 | FIFO 满标识，高有效。 |

另外，建议读RX FIFO深度最好>=32

## BAR空间接口（GR接口）

PCIE协议中提供了BAR空间供用户配置PCIE设备之用。此空间通常仅用来配置系统的寄存器，不适合用作大量数据传输。本设计中提供了512 Byte的BAR0空间，BAR0空间又分为两部分，其中低256Byte供DMA内部使用，高256Byte供FPGA中user逻辑使用。

接口信号：

| 信号名称 | **方向** | 宽度(bit) | 描述 |
| --- | --- | --- | --- |
| clk\_dp | DMA->USER | 1 | GR接口的时钟，62.5MHz |
| dp\_reset\_n | DMA->USER | 1 | GR接口的reset信号，低有效。 |
| gr1\_rd | DMA->USER | 1 | BAR寄存器读使能，高有效。读使能会一直持续有效直到gr1\_rdy信号有效，DMA看到gr1\_rdy信号有效后的下一拍撤掉gr1\_rd信号，user逻辑也要同时撤掉gr1\_rdy信号。读操作应维持至少2拍，第一拍user逻辑对地址译码，第二拍user逻辑使能gr1\_rdy,并给出gr1\_rdata。 |
| gr1\_wr | DMA->USER | 1 | BAR寄存器写使能，高有效。写使能会一直持续有效直到gr1\_rdy信号有效，DMA看到gr1\_rdy信号有效后的下一拍撤掉gr1\_wr信号，user逻辑也要同时撤掉gr1\_rdy信号。User逻辑最快可在gr1\_wr有效的当拍使能gr1\_rdy信号。 |
| gr1\_rdy | USER->DMA | 1 | User逻辑用来对读写寄存器的时序做控制，高有效。如果user逻辑没有做完当前的读写操作，则需要将此信号一直拉在低，当操作完成时，将此信号置高。对一次读写操作，此信号应该只在操作完成时有效一个时钟周期。其它时间始终维持在低。 |
| gr1\_addr | DMA->USER | 6 | 读写操作的地址，可寻址64 word。与gr1\_rd或gr1\_wr同时有效。 |
| gr1\_wdata | DMA->USER | 32 | 写操作数据，与gr1\_wr同时有效。 |
| gr1\_rdata | USER->DMA | 32 | 读操作数据，在读操作时与gr1\_rdy同时有效。 |

寄存器接口工作在clk\_dp时钟。

* 读时序：gr1\_rd与gr1\_addr同时有效，要求user逻辑最快在下一拍准备好reg\_rdata,准备好reg\_rdata的同时使能gr1\_rdy信号。注意，gr1\_rdy信号最快只能在gr1\_rd有效的下一拍使能。
* 写时序：gr1\_wr ，gr1\_addr，gr1\_wdata同时有效。User逻辑保存数据后使能gr1\_rdy信号。
* 读写操作不会同时发生。
* 具体实现可参考zhx\_user\_app.v

## 中断接口（MSI接口）

DMA在处理完每次的数据搬移后会向主机发出MSI中断，中断是通过xilinx的PCIE IP提供的一组cfg\_interrupt\_\*信号实现的，这组信号的时序关系可参考xilinx文档UG517。

本设计没有向user逻辑提供中断接口，如果user逻辑希望向主机发MSI中断，可以在dmac\_top\_wrap的上一层做简单逻辑，在user发送中断的期间先屏蔽DMA的中断请求，待user的中断被PCIE IP接收后，再放开DMA的中断信号。

用户的中断处理程序需自行在pdma-user-intr.c中添加。

## 集成示例

逻辑集成的示例可参考zhx\_user\_app.v。

# 主要操作流程

## 启动采集卡的读写操作

用户程序需要在驱动加载后按照下面流程启动读写操作。

开始(open)

调用ioctl获取读写块大小

启动DMA

启动采集卡采集数据(A/D)

读写设备(read/write)

注意，用户程序在调用ioctl，read和write系统调用前，需要先open设备。

另外上述流程中，“启动采集卡采集数据(A/D)”是用户通过ioctl接口操作BAR寄存器来操作的。执行“启动采集卡采集数据”后，要立刻执行“读写设备”操作，如间隔太久可能导致读数据缓冲区溢出。

## 停止读写

如果用户想停止设备工作或者卸载驱动，需要按照以下流程操作。

停止采集卡采集数据(A/D)

关闭DMA

结束(close)

注意，如果在调用“停止采集卡采集数据”接口时，驱动中还有正在进行的读写操作未完成，这些未完成操作将直接返回失败。

## 使用示例

使用示例可见软件包中的说明文档和测试工具代码。

# 测试结果

本设计在目标软硬件环境下作了读写带宽和数据正确性的测试。

测试环境：

* Intel i7-3770 / DDR3-1660 8GB / 主板 ASUS P8Z77-VLK
* Ubuntu 12.04

测试结果：

在读写粒度为4KB，16KB，64KB的情况下，读写带宽可以同时达到450MB/s以上。当把读写操作设为loop模式时，读回的数据与写入的数据一致。