

XDMA linux平台调试过程记录

PCIe学习笔记系列：

1. [PCIe基础知识及Xilinx相关IP核介绍](#)

概念了解：简单学习PCIe的数据链路与拓扑结构，另外看看有什么相关的IP核。

2. [【PG054】7 Series Integrated Block for PCI Express IP核的学习](#)

基础学习：关于Pcie IP核的数据手册，学习PCIe相关的IP核的配置参数及其对应的含义。

3. [Xilinx PCIe IP核示例工程代码分析与仿真](#)

基础学习：关于PCIe IP核的仿真，学习PCIe的配置流程以及应用过程。

4. [Xilinx XDMA 例程代码分析与仿真结果](#)

应用学习：关于Xilinx PCIe DMA IP核的仿真，学习 PCIe DMA 的配置过程以及具体的数据传输流程。

5. [XDMA linux平台调试过程记录](#)

应用学习：关于XDMA的实际调试过程，可在此基础上定制自己的需求。

[Xilinx XDMA 例程代码分析与仿真结果](#)分析了对XDMA IP核的读写过程，现在进行实际测试。

1 需要的资料

以调通为目的，需要的准备有：

1. [65444 - Xilinx PCI Express DMA Drivers and Software Guide](#)

Xilinx官网的一个问答，以前叫Answer65444,最近几天网页好像重新排版，统一只有数字代号了。

其中包含Linux和Windows平台下面的XDMA驱动。

- [Linux平台驱动](#)

- [Windows平台驱动](#)

我是在Linux下面进行调试的，Windows也试过了，但是驱动时好时坏，本来工程也是linux平台就暂时不测Windows平台了。

2. 驱动中描述的硬件环境

官方描述支持的环境：

- 系统架构: x86_64

- Linux 内核: 3.10+

- RHEL/CentOS: 7.6, 7.7, 7.8, 7.9, 8.1, 8.2

- Ubuntu: 16.04.5 LTS, 16.04.6 LTS 18.04.1 LTS, 18.04.2 LTS, 18.04.4 LTS, 18.04.5 LTS 20.04 LTS, 20.04.1 LT

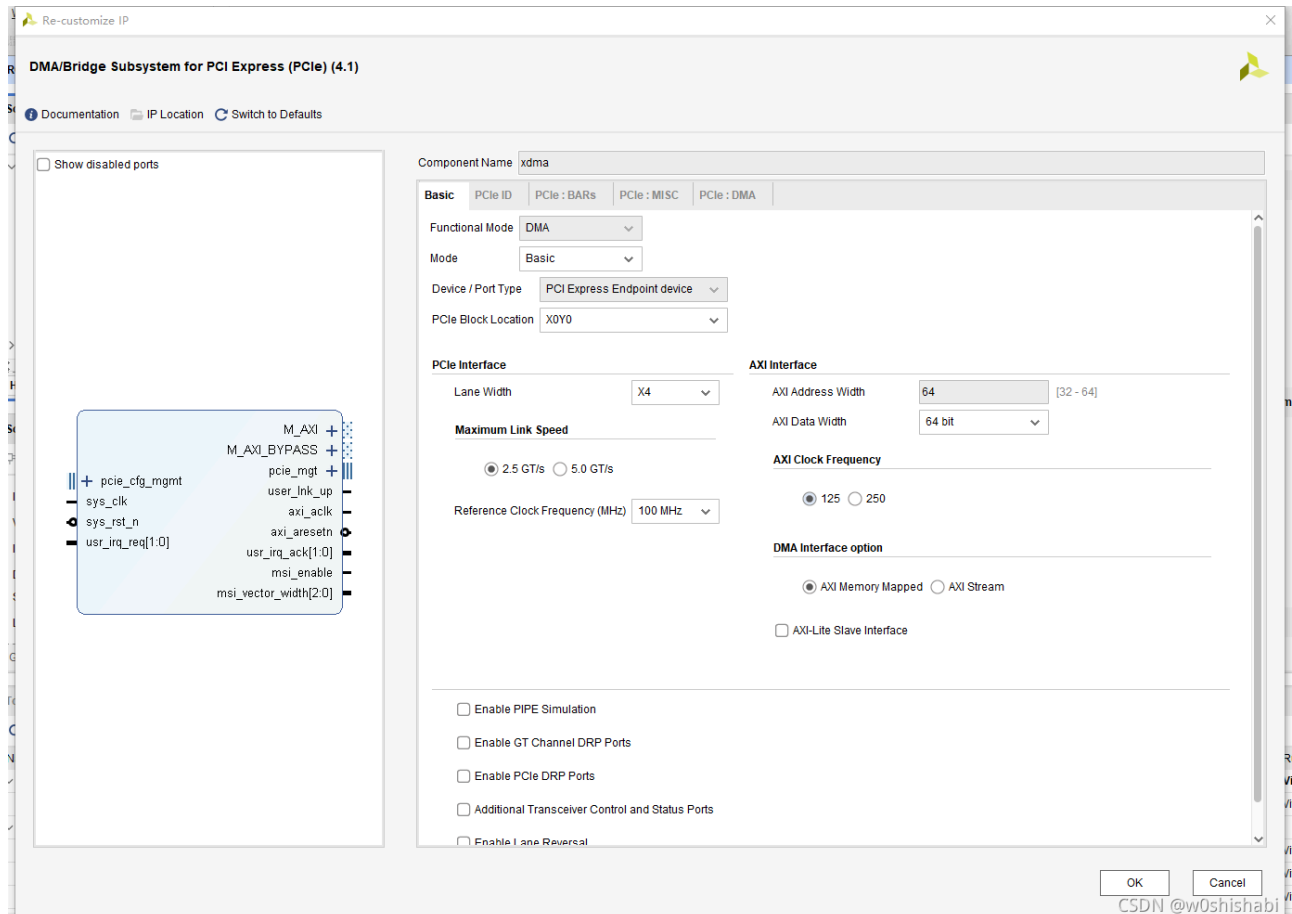
实测基于ubuntu18.04.4的 国产优化过的操作系统 ubuntu-18.04.4-enhanced-amd64 也都是支持的。

2 打开示例工程

2.1 IP核的配置

简单描述一下我的配置：

page1: Basic

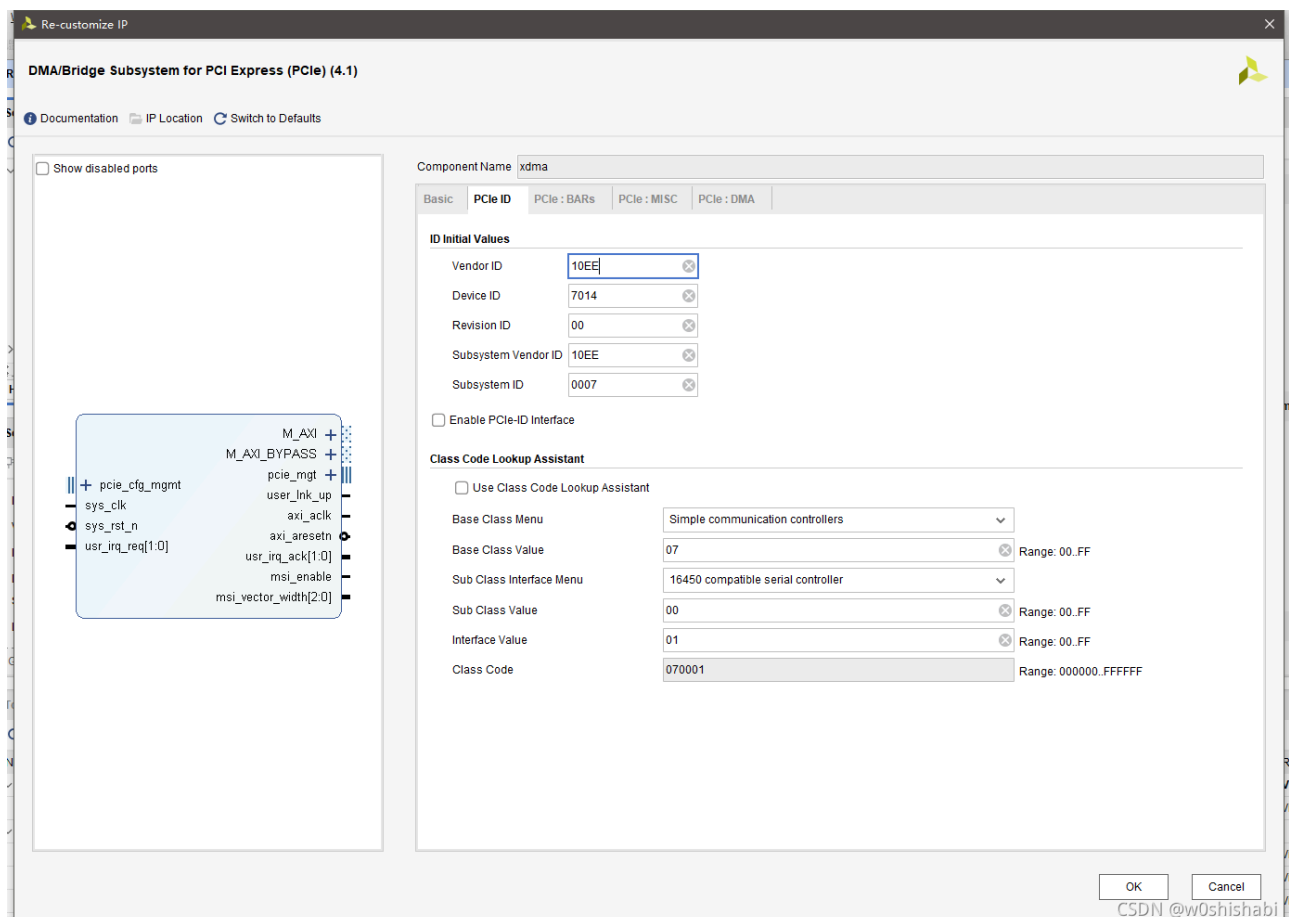


lane width,硬件决定

line speed、AXI DATA WIDTH与AXI Clock Frequency是有制约关系的，这个很好理解，在一定的链路速率下，数据位宽越大，数据对应的时钟就越慢。实际应用再满足链路速率的条件下，选择较小的时钟以满足时序收敛。

下面关于勾选的全取消，满足功能的前提下越简单越好。

page2: PCIe ID



如果要使用Xilinx提供的驱动，供应商ID(Vender ID)不可以改，因为驱动与Xilinx供应商（0x10ee）对应。
Device ID可以改。
其余默认。

【20220117补充】：关于Device ID有效值的问题，可以在“dma_ip_drivers-master\XDMA\linux-kernel\readme.txt”文件中找到答案。

“dma_ip_drivers-master\dma_ip_drivers-master\XDMA\linux-kernel\xdma\xdma_mod.c”文件中通过以下格式来定义此驱动可识别的Device ID：{ PCI_DEVICE(0x10ee, 0x8038), },。我们要新增可识别的Device ID，在代码中加上即可，比如我需要增加Device ID为8030的设备，则在xdma_mod.c中pci_ids数组增加代码：{ PCI_DEVICE(0x10ee, 0x8030), },。注意更改源文件后要重新make编译驱动文件。

所有支持的Device ID如下：

```
static const struct pci_device_id pci_ids[] = {
    { PCI_DEVICE(0x10ee, 0x9048), },
    { PCI_DEVICE(0x10ee, 0x9044), },
    { PCI_DEVICE(0x10ee, 0x9042), },
    { PCI_DEVICE(0x10ee, 0x9041), },
    { PCI_DEVICE(0x10ee, 0x903f), },
    { PCI_DEVICE(0x10ee, 0x9038), },
    { PCI_DEVICE(0x10ee, 0x9028), },
    { PCI_DEVICE(0x10ee, 0x9018), },
    { PCI_DEVICE(0x10ee, 0x9034), },
    { PCI_DEVICE(0x10ee, 0x9024), },
    { PCI_DEVICE(0x10ee, 0x9014), },
    { PCI_DEVICE(0x10ee, 0x9032), },
    { PCI_DEVICE(0x10ee, 0x9022), },
    { PCI_DEVICE(0x10ee, 0x9012), },
    { PCI_DEVICE(0x10ee, 0x9031), },
    { PCI_DEVICE(0x10ee, 0x9021), },
    { PCI_DEVICE(0x10ee, 0x9011), },
    { PCI_DEVICE(0x10ee, 0x8011), },
}
```

```

    { PCI_DEVICE(0x10ee, 0x8012), },
    { PCI_DEVICE(0x10ee, 0x8014), },
    { PCI_DEVICE(0x10ee, 0x8018), },
    { PCI_DEVICE(0x10ee, 0x8021), },
    { PCI_DEVICE(0x10ee, 0x8022), },
    { PCI_DEVICE(0x10ee, 0x8024), },
    { PCI_DEVICE(0x10ee, 0x8028), },
    { PCI_DEVICE(0x10ee, 0x8031), },
    { PCI_DEVICE(0x10ee, 0x8032), },
    { PCI_DEVICE(0x10ee, 0x8034), },
    { PCI_DEVICE(0x10ee, 0x8038), },

    { PCI_DEVICE(0x10ee, 0x7011), },
    { PCI_DEVICE(0x10ee, 0x7012), },
    { PCI_DEVICE(0x10ee, 0x7014), },
    { PCI_DEVICE(0x10ee, 0x7018), },
    { PCI_DEVICE(0x10ee, 0x7021), },
    { PCI_DEVICE(0x10ee, 0x7022), },
    { PCI_DEVICE(0x10ee, 0x7024), },
    { PCI_DEVICE(0x10ee, 0x7028), },
    { PCI_DEVICE(0x10ee, 0x7031), },
    { PCI_DEVICE(0x10ee, 0x7032), },
    { PCI_DEVICE(0x10ee, 0x7034), },
    { PCI_DEVICE(0x10ee, 0x7038), },

    { PCI_DEVICE(0x10ee, 0x6828), },
    { PCI_DEVICE(0x10ee, 0x6830), },
    { PCI_DEVICE(0x10ee, 0x6928), },
    { PCI_DEVICE(0x10ee, 0x6930), },
    { PCI_DEVICE(0x10ee, 0x6A28), },
    { PCI_DEVICE(0x10ee, 0x6A30), },
    { PCI_DEVICE(0x10ee, 0x6D30), },

    { PCI_DEVICE(0x10ee, 0x4808), },
    { PCI_DEVICE(0x10ee, 0x4828), },
    { PCI_DEVICE(0x10ee, 0x4908), },
    { PCI_DEVICE(0x10ee, 0x4A28), },
    { PCI_DEVICE(0x10ee, 0x4B28), },

    { PCI_DEVICE(0x10ee, 0x2808), },

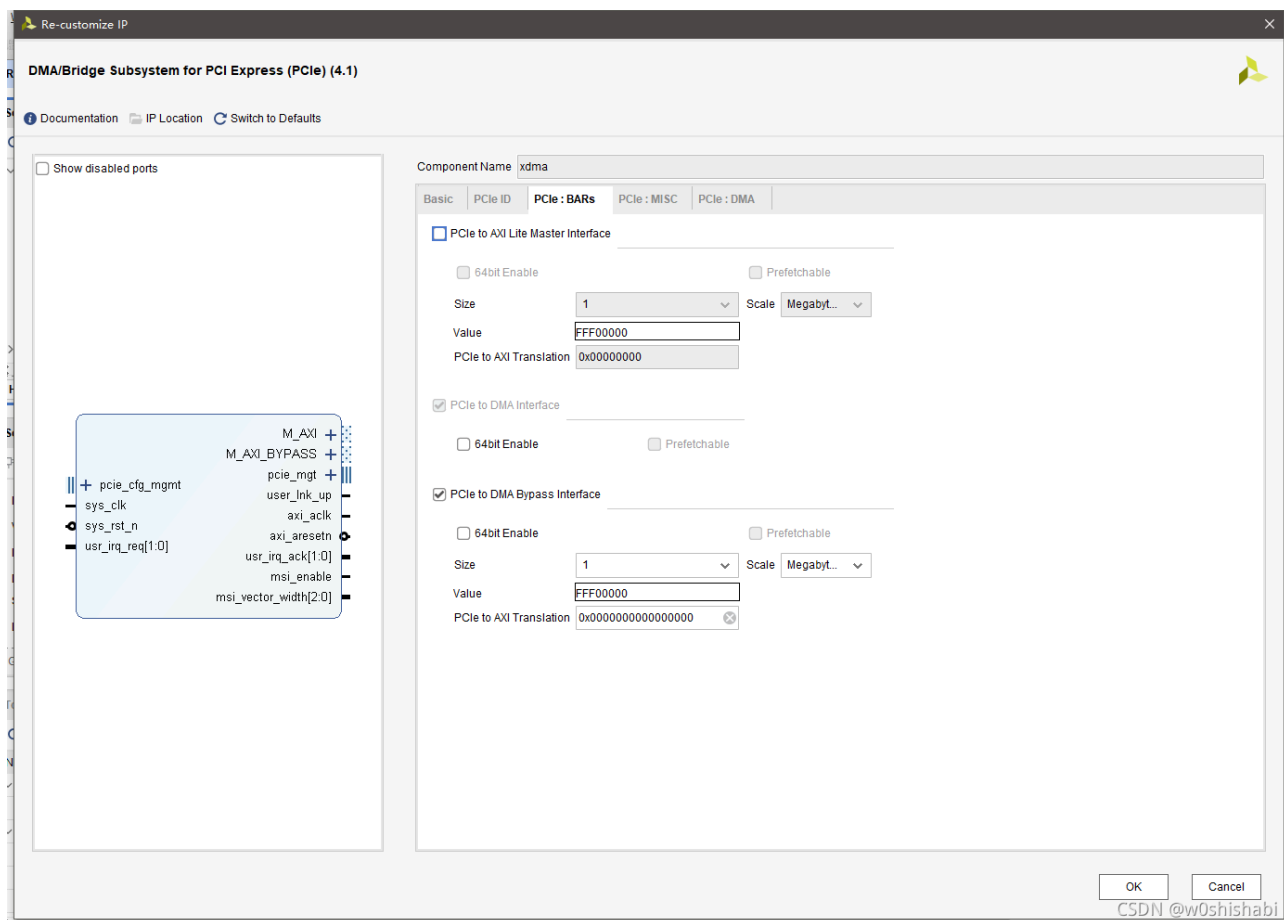
#ifdef INTERNAL_TESTING
    { PCI_DEVICE(0x1d0f, 0x1042), 0},
#endif
    /* aws */
    { PCI_DEVICE(0x1d0f, 0xf000), },
    { PCI_DEVICE(0x1d0f, 0xf001), },

    {0,}
};

```

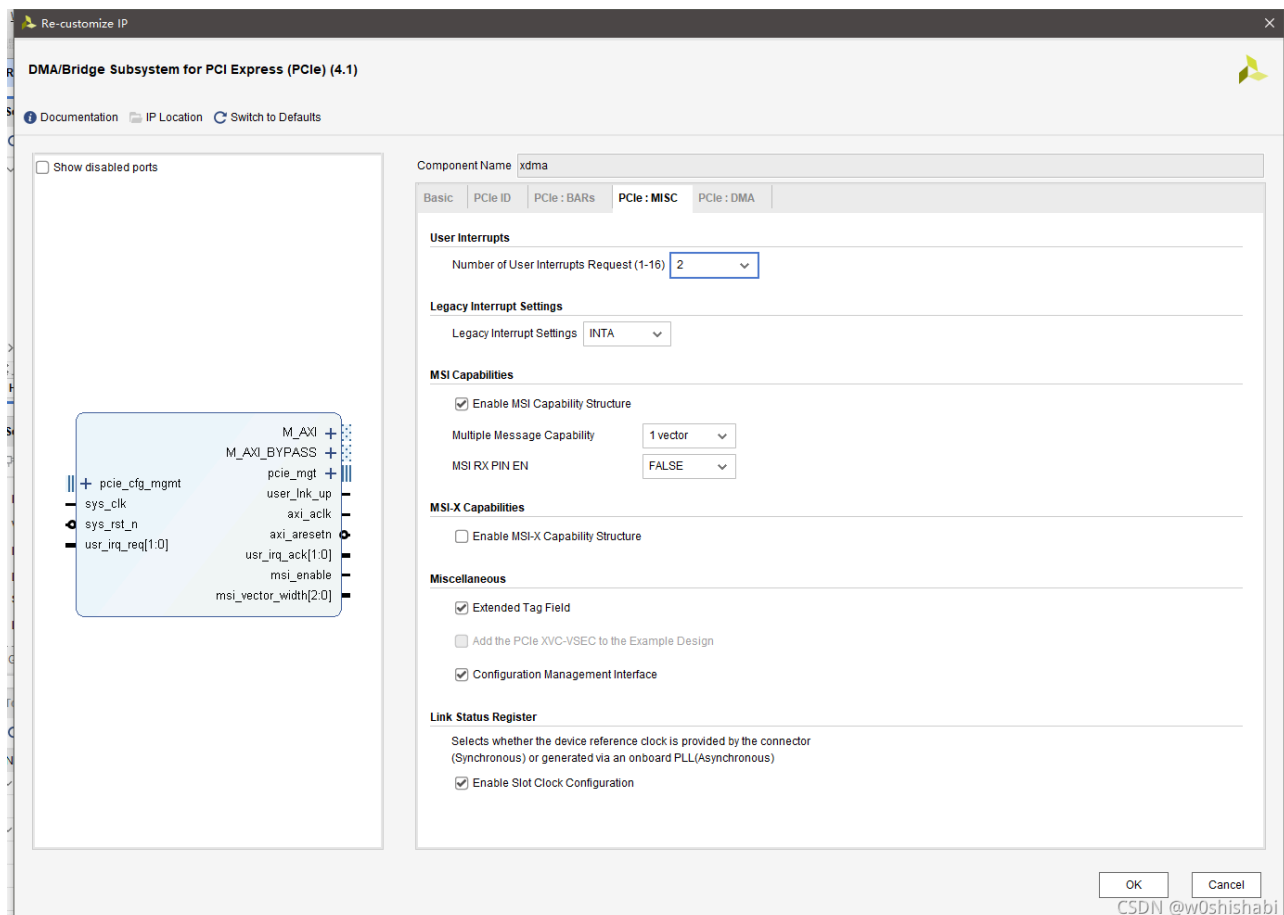


page3: PCIe BARs



基址寄存器的配置（Base Address Register），首先存储PCIe DMA的BAR是默认使能的。然后还使能了Bypass通道。

page4: PCIe MISC

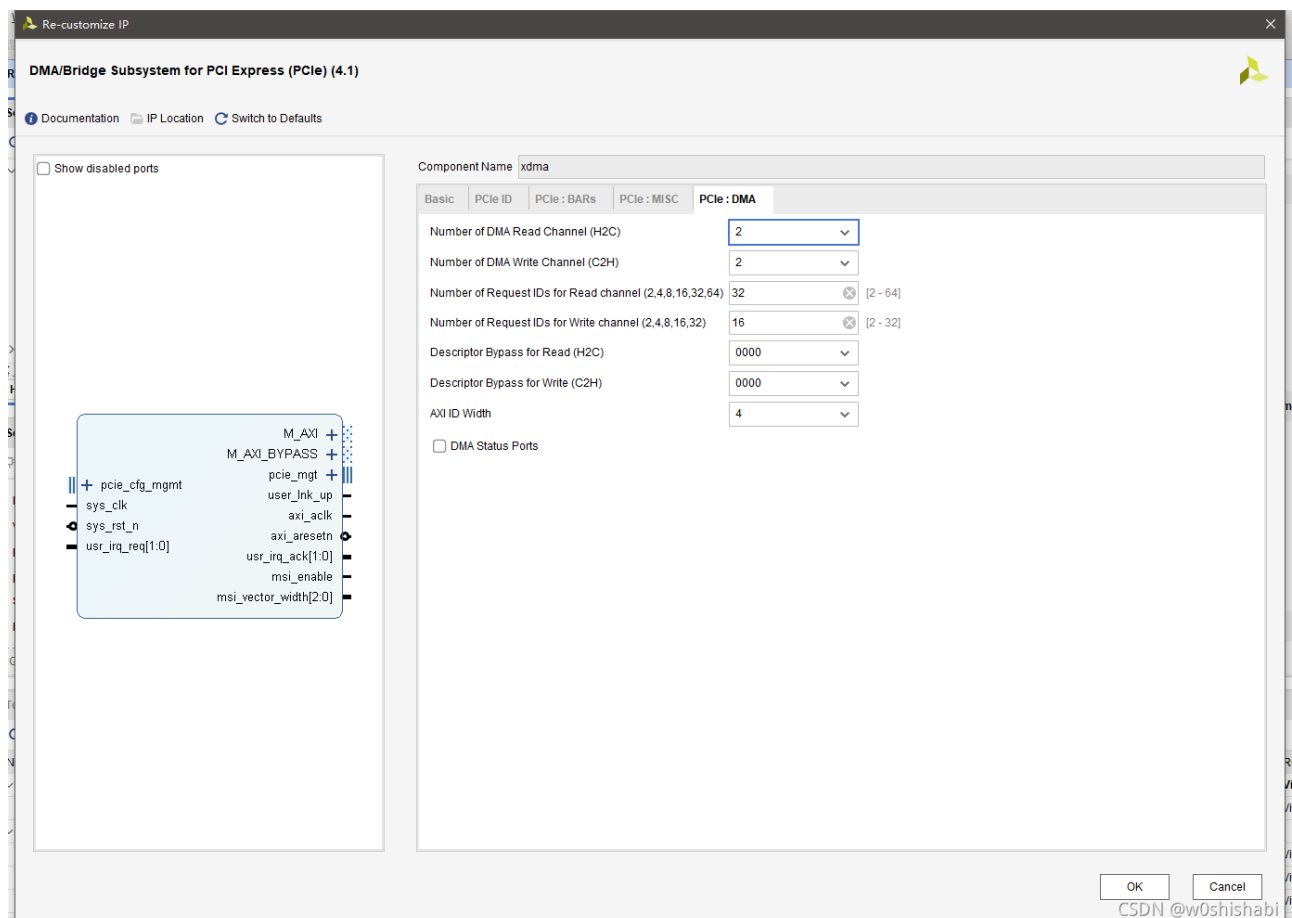


User interrupts设置用户中断的数量，我设置为两个用户中断用于测试。这里的中断完全是我们自己决定含义以及自己使用的，不会影响基本的功能。

MSI与MSI-X不能同时使能。

其他默认即可。

page5: PCIe DMA



读写（H2C,c2H）通道的数量，主要是用于需要同时使用多个通道进行数据传输的场景。其他默认即可。

2.2 打开示例工程

右击IP核，open example design来打开。

推荐先看关于例程的仿真：[Xilinx XDMA 例程代码分析与仿真结果](#)，先对PC配置PCIe endpoint设备的过程，以及DMA的过程有一个详细的了解，对实际调试的过程也会很清晰。

然后根据自己的板子配置好时钟和锁好引脚即可。

关于时钟这里提醒一下例程给出的sys_clk_p 和 sys_clk_n port锁到PCIe接口的时钟就行，时钟由PC的PCIe插槽给。

3 调试过程

主要的使用过程在官方驱动的readme文件中已经说的很清楚了，这里只是把调试的过程记录一下。

下载的驱动文件包包含XDMA、QDMA、XVSEC的驱动。

- QDMA (Queue DMA) IP通过PCI Express提供高性能的直接内存访问(DMA)。PCIe QDMA可以在UltraScale+设备中实现。linux内核驱动和DPDK驱动都可以运行在PCI Express根端口主机PC上，通过PCI Express与QDMA端点IP进行交互。
- Xilinx-VSEC(XVSEC)是Xilinx支持的VSEV。XVSEC驱动程序有助于创建和部署可能包含Xilinx VSEC PCIe特性的设计。特定于供应商的扩展功能(Vendor Specific Extended Capability, VSEC)是PCIe的一个特性。VSEC本身是在FPGA硬件中的PCIe扩展功能寄存器中实现的(软IP或硬IP)。驱动程序和软件的创建是为了与这个硬件实现的特性进行接口和使用。XVSEC驱动程序目前包括MCAP VSEC，将扩展到包括XVC VSEC和NULL VSEC。

3.1 驱动的文件结构

- XDMA驱动用于我们的DMA_Bridge Subsystem for PCI Express IP核，文件的目录结构：

```
XDMA\linux-kernel
├─include
├─tests
│  └─data
├─tools
└─xdma
```

- xdma：包含Xilinx PCIe DMA内核模块驱动程序文件。
- include：包含所需的所有包含文件编译驱动程序。
- tests：包含示例应用程序，以测试提供的内核模块驱动程序和Xilinx PCIe DMA IP。该目录还包含以下脚本和目录。

- load_driver.sh

这个脚本加载内核模块，并创建所提供的软件所使用的必要内核节点。

内核设备节点将在/dev/xdma*下创建。

在/dev/xdma/card*下创建额外的设备节点，以更容易地区分多个PCIe DMA卡。运行此脚本需要root权限。

- run_test.sh

- dma_memory_mapped_test.sh, dma_streaming_test.sh

- data/:

run_test.sh在Xilinx PCIe DMA目标上运行样例测试，并返回通过(0)或失败(1)结果。

这个脚本调用相同目录下的其他两个脚本：

- dma_memory_mapped_test.sh: memory-mapped dma 模式

- dma_streaming_test.sh: streaming dma 模式

data/目录包含二进制数据文件，用于使用run_test.sh将DMA数据传输到Xilinx FPGA PCIe端点设备。

注意：这些脚本仅用于PCIe DMA示例设计。

- perform_hwcount.sh:

该脚本运行主机到卡(H2C)和卡到主机(C2H)的XDMA硬件性

能。结果被复制到' hw_log_h2c.txt' 和' hw_log_c2.txt' 文本文件中。

对于每个方向，执行从64字节循环到4MBytes的脚本，并生成性能结果(对于每个循环计数，字节大小加倍)。

可以在这两个文件上查找' data rate' ，以查看数据率值。数据速率值以最大吞吐量的百分比表示。

x8Gen3的最大数据速率为8Gbytes/s，因此对于x8Gen3设计值为0.81的数据速率为 $0.818 = 6.48\text{Gbytes/s}$ 。

x16Gen3的最大数据速率是16Gbytes/s，所以对于x16Gen3设计值为0.78的数据速率是 $0.7816 = 12.48\text{Gbytes/s}$ 。

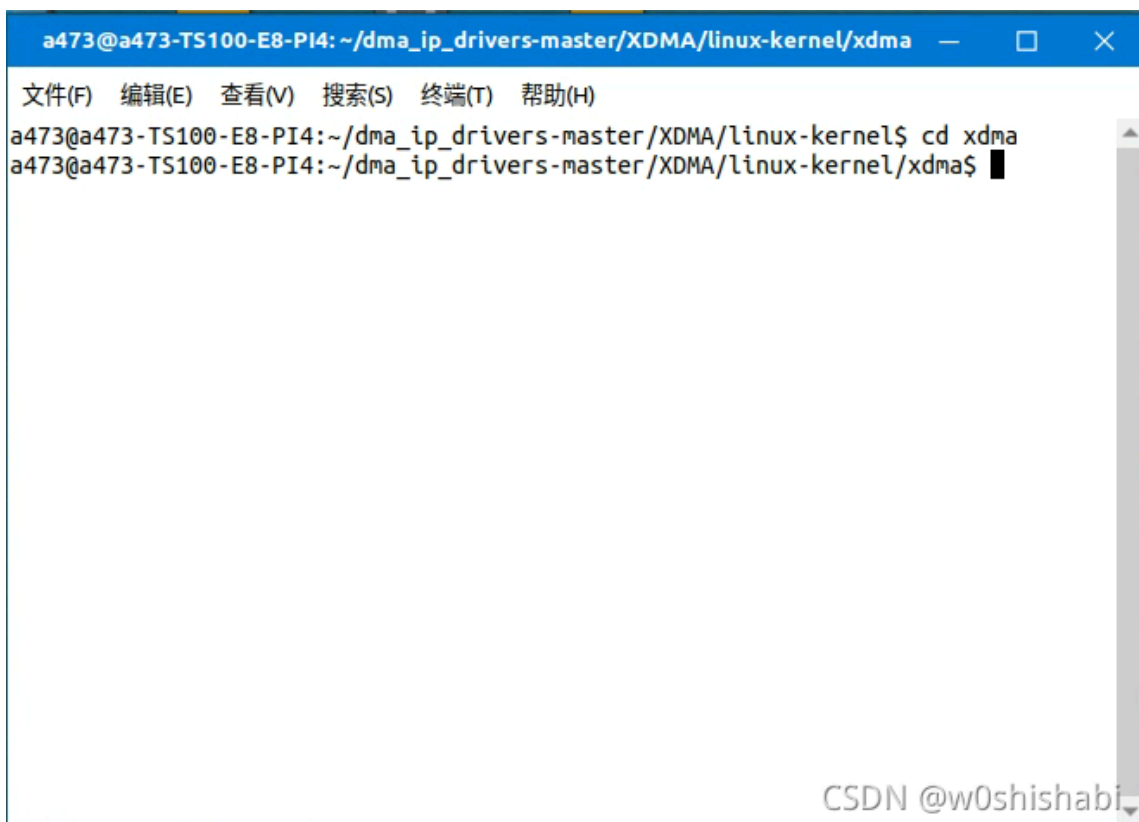
本程序可以在AXI-MM示例设计上运行。

AXI-ST实例设计是一个环回设计，H2C和C2H连接。在AXI-ST示例设计上运行不会生成正确的结果。

如果AXI-ST设计独立于H2C和C2H，则可以生成性能结果。

3.2 用法

1. 切换当前路径到xdma/

A terminal window screenshot showing a user at a473@TS100-E8-PI4 navigating to the xdma directory. The terminal title is 'a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/xdma'. The command prompt shows the user is in the directory ~/dma_ip_drivers-master/XDMA/linux-kernel and has entered 'cd xdma'. The prompt then changes to ~/dma_ip_drivers-master/XDMA/linux-kernel/xdma\$.

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/xdma
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel$ cd xdma
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/xdma$
```

2. 编译与安装内核模块驱动程序
注意这一步需要root权限


```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/xdma
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel$ cd xdma
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/xdma$ make install
Makefile:10: XVC_FLAGS: .
make -C /lib/modules/5.4.0-90-generic/build M=/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma modules
make[1]: 进入目录"/usr/src/linux-headers-5.4.0-90-generic"
/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/Makefile:10: XVC_FLAGS: .
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/libxdma.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma_cdev.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/cdev_ctrl.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/cdev_events.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/cdev_sgdm.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/cdev_xvc.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/cdev_bypass.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma_mod.o
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma_thread.o
LD [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma.o
/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/Makefile:10: XVC_FLAGS: .
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma.mod.o
LD [M] /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma.ko
make[1]: 离开目录"/usr/src/linux-headers-5.4.0-90-generic"
make -C /lib/modules/5.4.0-90-generic/build M=/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma modules_install
make[1]: 进入目录"/usr/src/linux-headers-5.4.0-90-generic"
mkdir: 无法创建目录"/lib/modules/5.4.0-90-generic/extra": 权限不够
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/xdma$ sudo make install
[sudo] a473 的密码:
Makefile:10: XVC_FLAGS: .
make -C /lib/modules/5.4.0-90-generic/build M=/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma modules
make[1]: 进入目录"/usr/src/linux-headers-5.4.0-90-generic"
/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/Makefile:10: XVC_FLAGS: .
Building modules, stage 2.
MODPOST 1 modules
make[1]: 离开目录"/usr/src/linux-headers-5.4.0-90-generic"
make -C /lib/modules/5.4.0-90-generic/build M=/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma modules_install
make[1]: 进入目录"/usr/src/linux-headers-5.4.0-90-generic"
INSTALL /home/a473/dma_ip_drivers-master/XDMA/linux-kernel/xdma/xdma.ko
At main.c:160:
- SSL error:02001002:system library:fopen:No such file or directory: ../crypto/bio/bss_file.c:72
- SSL error:2006D080:BIO routines:BIO_new_file:no such file: ../crypto/bio/bss_file.c:79
sign-file: certs/signing_key.pem: No such file or directory
DEPMOD 5.4.0-90-generic
Warning: modules_install: missing 'System.map' file. Skipping depmod.
make[1]: 离开目录"/usr/src/linux-headers-5.4.0-90-generic"
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/xdma$
```

CSDN @w0shishabi

可以看到，程序在xdma路径下创建了两个文件：xdma.mod.o、xdma.ko
然后执行xdma modules_install 安装xdma.ko文件。
最后的SSL error是证书问题，可以忽略。
如果出现未找到gcc命令，sudo apt-get install gcc 安装gcc即可。

```
a473@a473-ubuntu: ~/桌面/linux-kernel/xdma
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-ubuntu:~/桌面/linux-kernel/xdma$ sudo make install
Makefile:10: XVC_FLAGS: ..
make -C /lib/modules/5.4.0-100-generic/build M=/home/a473/桌面/linux-kernel/xdma
modules
make[1]: 进入目录"/usr/src/linux-headers-5.4.0-100-generic"
arch/x86/Makefile:151: CONFIG_X86_X32 enabled but no binutils support
make[1]: gcc: Command not found
/home/a473/桌面/linux-kernel/xdma/Makefile:10: XVC_FLAGS: ..
CC [M] /home/a473/桌面/linux-kernel/xdma/libxdma.o
/bin/bash: gcc: 未找到命令
scripts/Makefile.build:270: recipe for target '/home/a473/桌面/linux-kernel/xdma
/libxdma.o' failed
make[2]: *** [/home/a473/桌面/linux-kernel/xdma/libxdma.o] Error 127
Makefile:1762: recipe for target '/home/a473/桌面/linux-kernel/xdma' failed
make[1]: *** [/home/a473/桌面/linux-kernel/xdma] Error 2
make[1]: 离开目录"/usr/src/linux-headers-5.4.0-100-generic"
Makefile:27: recipe for target 'all' failed
make: *** [all] Error 2
a473@a473-ubuntu:~/桌面/linux-kernel/xdma$ sudo make install CSDN @w0shishabi
```

3. 切换到路径到tool/
4. 编译提供的示例测试工具。

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/tools
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
At main.c:160:
- SSL error:02001002:system library:fopen:No such file or directory: ../crypto/bio/bss_file.c:72
- SSL error:2006D080:BIO routines:BIO_new_file:no such file: ../crypto/bio/bss_file.c:79
sign-file: certs/signing_key.pem: No such file or directory
DEPMOD 5.4.0-90-generic
Warning: modules install: missing 'System.map' file. Skipping depmod.
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-90-generic”
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/xdma$ cd ../tools/
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ sudo make
cc -c -std=c99 -o reg_rw.o reg_rw.c -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SOURCE
cc -o reg_rw reg_rw.o
cc -c -std=c99 -o dma_to_device.o dma_to_device.c -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FI
LE_SOURCE
In file included from /usr/include/assert.h:35:0,
                 from dma_to_device.c:13:
/usr/include/features.h:184:3: warning: #warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use
_DEFAULT_SOURCE" [-Wcpp]
# warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use _DEFAULT_SOURCE"
cc -lrt -o dma_to_device dma_to_device.o -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SOURCE
cc -c -std=c99 -o dma_from_device.o dma_from_device.c -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARG
E_FILE_SOURCE
In file included from /usr/include/assert.h:35:0,
                 from dma_from_device.c:13:
/usr/include/features.h:184:3: warning: #warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use
_DEFAULT_SOURCE" [-Wcpp]
# warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use _DEFAULT_SOURCE"
cc -lrt -o dma_from_device dma_from_device.o -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SO
URCE
cc -c -std=c99 -o performance.o performance.c -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_S
OURCE
cc -o performance performance.o -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SOURCE
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$
```

5. 加载内核模块驱动程序:

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/tests
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
                 from dma_to_device.c:13:
/usr/include/features.h:184:3: warning: #warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use
_DEFAULT_SOURCE" [-Wcpp]
# warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use _DEFAULT_SOURCE"
cc -lrt -o dma_to_device dma_to_device.o -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SOURCE
cc -c -std=c99 -o dma_from_device.o dma_from_device.c -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARG
E_FILE_SOURCE
In file included from /usr/include/assert.h:35:0,
                 from dma_from_device.c:13:
/usr/include/features.h:184:3: warning: #warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use
_DEFAULT_SOURCE" [-Wcpp]
# warning "_BSD_SOURCE and _SVID_SOURCE are deprecated, use _DEFAULT_SOURCE"
cc -lrt -o dma_from_device dma_from_device.o -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SO
URCE
cc -c -std=c99 -o performance.o performance.c -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_S
OURCE
cc -o performance performance.o -D_FILE_OFFSET_BITS=64 -D_GNU_SOURCE -D_LARGE_FILE_SOURCE
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ s
s: 未找到命令
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ ls
dma_from_device  dma_to_device  dma_utils.c  performance.c  reg_rw
dma_from_device.c  dma_to_device.c  Makefile     performance.o  reg_rw.c
dma_from_device.o  dma_to_device.o  performance  perform_hwcount.sh  reg_rw.o
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ cd ../tests/
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ ls
dma  dma_streaming_test.sh  perform_hwcount.sh
dma_memory_mapped_test.sh  load_driver.sh  run_test.sh
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ sudo ./load_driver.sh
Loading xdma driver...
The Kernel module installed correctly and the xmda devices were recognized.
DONE
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$
```

6. 运行提供的测试脚本以生成基本DMA数据流。


```
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dma_from_device.o dma_to_device.o performance perform_hwcount.sh reg_rw.o
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ cd ../tests/
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ ls
dma_streaming_test.sh perform_hwcount.sh
dma_memory_mapped_test.sh load_driver.sh run_test.sh
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ sudo ./load_driver.sh
Loading xdma driver...
The kernel module installed correctly and the xmda devices were recognized.
DONE
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ sudo ./run_test.sh
Info: Number of enabled h2c channels = 2
Info: Number of enabled c2h channels = 2
Info: The PCIe DMA core is memory mapped.
Info: Running PCIe DMA memory mapped write read test
      transfer size: 1024
      transfer count: 1
Info: Writing to h2c channel 0 at address offset 0.
Info: Writing to h2c channel 1 at address offset 1024.
Info: Wait for current transactions to complete.
/dev/xdma0_h2c_0 ** Average BW = 1024, 14.603329
/dev/xdma0_h2c_1 ** Average BW = 1024, 13.646237
Info: Writing to h2c channel 0 at address offset 2048.
Info: Writing to h2c channel 1 at address offset 3072.
Info: Wait for current transactions to complete.
/dev/xdma0_h2c_0 ** Average BW = 1024, 17.156164
/dev/xdma0_h2c_1 ** Average BW = 1024, 27.980436
Info: Reading from c2h channel 0 at address offset 0.
Info: Reading from c2h channel 1 at address offset 1024.
Info: Wait for the current transactions to complete.
/dev/xdma0_c2h_0 ** Average BW = 1024, 25.924707
/dev/xdma0_c2h_1 ** Average BW = 1024, 34.285332
Info: Reading from c2h channel 0 at address offset 2048.
Info: Reading from c2h channel 1 at address offset 3072.
Info: Wait for the current transactions to complete.
/dev/xdma0_c2h_0 ** Average BW = 1024, 17.888651
/dev/xdma0_c2h_1 ** Average BW = 1024, 15.901855
Info: Checking data integrity.
Info: Data check passed for address range 0 - 1024.
Info: Data check passed for address range 1024 - 2048.
Info: Data check passed for address range 2048 - 3072.
Info: Data check passed for address range 3072 - 4096.
Info: All PCIe DMA memory mapped tests passed.
Info: All tests in run_tests.sh passed.
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$
```

CSDN @w0shishabi

7. 检查驱动程序版本号

```
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ modinfo ../xdma/xdma.ko
filename:
/home/a473/dma_ip_drivers-master/XDMA/linux-kernel/tests/../../xdma/xdma.ko
license:
Dual BSD/GPL
version:
2020.2.0
description:
Xilinx XDMA Reference Driver
author:
Xilinx, Inc.
srcversion:
A948DE35D052B3FE562AE06
alias:
pci:v00001D0Fd0000F001sv*sd*bc*sc*i*
alias:
pci:v00001D0Fd0000F000sv*sd*bc*sc*i*
alias:
pci:v000010EEd00002808sv*sd*bc*sc*i*
alias:
pci:v000010EEd00004B28sv*sd*bc*sc*i*
alias:
pci:v000010EEd00004A28sv*sd*bc*sc*i*
alias:
pci:v000010EEd00004908sv*sd*bc*sc*i*
alias:
pci:v000010EEd00004828sv*sd*bc*sc*i*
alias:
pci:v000010EEd00004808sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006D30sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006A30sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006A28sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006930sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006928sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006830sv*sd*bc*sc*i*
alias:
pci:v000010EEd00006828sv*sd*bc*sc*i*
alias:
pci:v000010EEd00007038sv*sd*bc*sc*i*
alias:
pci:v000010EEd00007034sv*sd*bc*sc*i*
alias:
pci:v000010FFd00007032sv*sd*bc*sc*i*
```

CSDN @w0shishabi

3.3 其他

3.3.1 XDMA驱动安装了哪些设备？

打开PC的/dev路径，看看以xdma开头命名的设备：

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/tests
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$ ls /dev/xdma*
/dev/xdma0_bypass      /dev/xdma0_control    /dev/xdma0_events_14  /dev/xdma0_events_7
/dev/xdma0_bypass_c2h_0 /dev/xdma0_events_0    /dev/xdma0_events_15  /dev/xdma0_events_8
/dev/xdma0_bypass_c2h_1 /dev/xdma0_events_1    /dev/xdma0_events_2    /dev/xdma0_events_9
/dev/xdma0_bypass_h2c_0 /dev/xdma0_events_10   /dev/xdma0_events_3    /dev/xdma0_h2c_0
/dev/xdma0_bypass_h2c_1 /dev/xdma0_events_11   /dev/xdma0_events_4    /dev/xdma0_h2c_1
/dev/xdma0_c2h_0        /dev/xdma0_events_12   /dev/xdma0_events_5
/dev/xdma0_c2h_1        /dev/xdma0_events_13   /dev/xdma0_events_6
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tests$
```

- xdma0_bypass_*: 关于Bypass BAR的相关操作。c2h代表写，h2c代表读。_0/1代表通道号。
- xdma0_control: 关于PCIe 配置空间的读写。
- xdma0_event_*: 16个用户中断，在IP核配置时使能了才有用。
- xdma0_c2h_*: DMA访问。c2h代表写，h2c代表读。_0/1代表通道号。

3.3.2 测试用例的流程？

打开test/run_test.sh文件：

基本流程为：

1. 设置Tool的路径。
2. 设置PCIe DMA传输的大小transferSize和每次数据传输重复的次数transferCount
3. 访问xdma0_control查询哪些通道使能。
4. 查询DMA类型是memory mapped 还是 streaming。
5. 执行DMA测试
 1. 如果是MM类型，执行dma_memory_mapped_test.sh脚本。
 2. 如果是streaming类型，执行dma_streaming_test.sh脚本。

6. 退出

打开test/dma_memory_mapped_test.sh文件：

基本流程为：

1. 并行写入所有启用的h2c Channel。
2. 并行读取所有启用的c2h Channel。
3. 如果可能的话（MM），验证写入的数据与读取的数据是否匹配。
4. 退出。

3.3.3 怎么单独测试内存读写？

在官方测试用例中，可以看到其使用一个名为reg_rw的工具函数。查看一下这个函数的用法：

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/tools
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ ./reg_rw -h

Usage: ./reg_rw <device> <address> [[type] data]
device : character device to access
address : memory address to access
type    : access operation type : [b]yte, [h]alfword, [w]ord
data    : data to be written for a write

a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$
```

可以看到，这个函数需要需要4个参数

- device：字符类型设备。指的3.3.1查看的设备中的字符设备。
- address：需要访问的内存地址。指的是起始字节地址。
- type：访问类型。b:字节，h：半字，w：字。
- data：如果有数据，代表是写操作。如果没有则为读操作。

例子：

1. 往BYPASS BAR指向的内存空间，首地址为0x0的字节写入数据0x11223344：

```
sudo ./reg_rw /dev/xdma0_bypass 0 w 0x11223344
```

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/tools
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ ./reg_rw /dev/xdma0_bypass 0 w 0x11223344
argc = 5
device: /dev/xdma0_bypass
address: 0x00000000
access type: write
access width given.
access width: word (32-bits)
Error at line 94, file reg_rw.c (13) [Permission denied]
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ sudo ./reg_rw /dev/xdma0_bypass 0 w 0x11223344
argc = 5
device: /dev/xdma0_bypass
address: 0x00000000
access type: write
access width given.
access width: word (32-bits)
character device /dev/xdma0_bypass opened.
Memory mapped at address 0x7f0ab1623000.
Write 32-bits value 0x11223344 to 0x00000000 (0x0x7f0ab1623000)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$
```

这里注意，访问系统的字符设备需要root权限。

1. 读取BYPASS BAR指向的内存空间，首地址为0x0，数据格式为halfword：

```
sudo ./reg_rw /dev/xdma0_bypass 0 h
```



```
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ sudo ./reg_rw /dev/xdma0_bypass 0 h
argc = 4
device: /dev/xdma0_bypass
address: 0x00000000
access type: write
access width given.
access width: half word (16-bits)
character device /dev/xdma0_bypass opened.
Memory mapped at address 0x7f265b24c000.
Read 16-bit value at address 0x00000000 (0x7f265b24c000): 0x3344
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$
```

可见读取出的数据为刚才写入的低16位0x3344。读写访问都是对的。

提醒：

有一个细节。reg_rw对内存空间的读访问必须以整word为边界，比如可以以h格式访问地址0，1，2。但是以h格式访问3就会出错。

reg_rw仅可以对非dma通道的BAR字符设备访问。

```
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ sudo ./reg_rw /dev/xdma0_bypass 0 w 0x11223344
argc = 5
device: /dev/xdma0_bypass
address: 0x00000000
access type: write
access width given.
access width: word (32-bits)
character device /dev/xdma0_bypass opened.
Memory mapped at address 0x7fae3e031000.
Write 32-bits value 0x11223344 to 0x00000000 (0x0x7fae3e031000)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ sudo ./reg_rw /dev/xdma0_bypass 2 h
argc = 4
device: /dev/xdma0_bypass
address: 0x00000002
access type: write
access width given.
access width: half word (16-bits)
character device /dev/xdma0_bypass opened.
Memory mapped at address 0x7f99dca45000.
Read 16-bit value at address 0x00000002 (0x7f99dca45002): 0x1122
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ sudo ./reg_rw /dev/xdma0_bypass 3 h
argc = 4
device: /dev/xdma0_bypass
address: 0x00000003
access type: write
access width given.
access width: half word (16-bits)
character device /dev/xdma0_bypass opened.
Memory mapped at address 0x7faabd8a1000.
Read 16-bit value at address 0x00000003 (0x7faabd8a1003): 0xffff
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$
```

3.3.4 在内存读写的过程中，内部AXI4的时序？

这里抓了一下Bypass 通道的AXI4 接口，简单看一下寄存器读写时内部AXI4的时序。关于AXI4，[AXI4协议学习：架构、信号定义、工作时序和握手机制](#)进行了详细的介绍。

关于AXI4规范手册，官网可以下载，或[AMBA AXI 各版本协议规范VersionB、IH10022C、IH10022E、IH10022H、IH10022H_c](#)有提供。

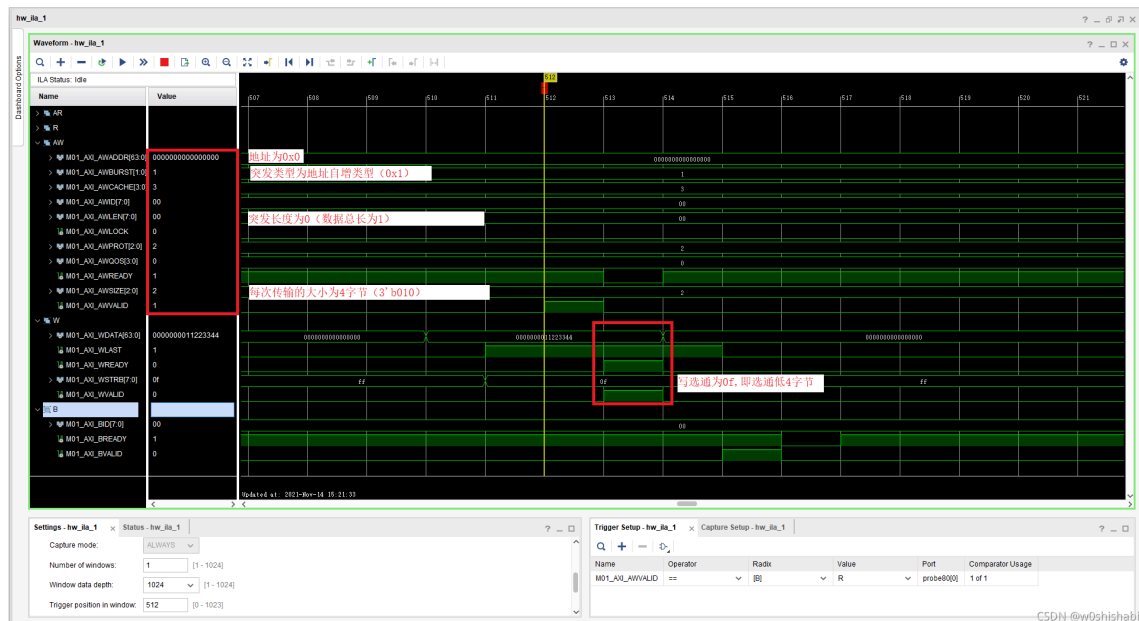
只能说学好AXI4是必要的！

1. 写数据

对应的命令：

```
sudo ./reg_rw /dev/xdma0_bypass 0 w 0x11223344
```

时序图：

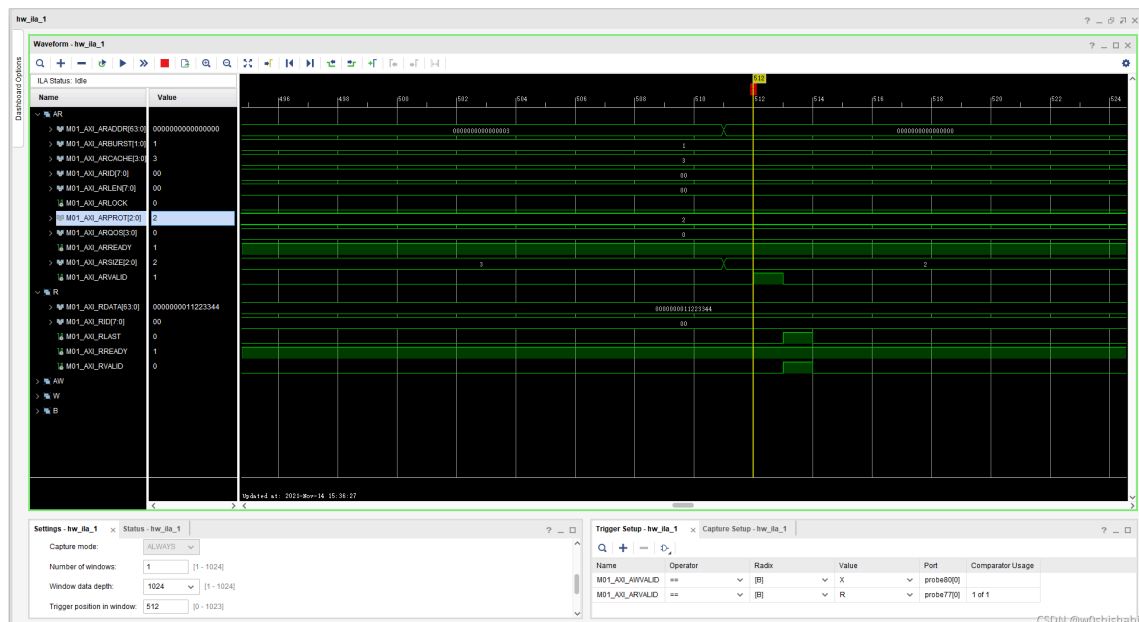


2. 读数据

对应的命令：

```
sudo ./reg_rw /dev/xdma0_bypass 0 h
```

时序图：



从时序图可以体现为什么不能跨整word地址读取，因为无论是按h还是b格式读，起始都是驱动做的数据格式处理，在硬件端都是读取一个word。

3.3.4 其他

1. 怎么查看PC有没有挂上PCIe设备？

lspci指令，如果PC识别到了PCIe设备，可以看到：

```
a473@a473-TS100-E8-PI4: ~/dma_ip_drivers-master/XDMA/linux-kernel/tools
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ lspci
00:00.0 Host bridge: Intel Corporation 4th Gen Core Processor DRAM Controller (rev 06)
00:14.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB xHCI (rev 05)
00:1a.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI #2 (rev 05)
00:1c.0 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #1 (rev d5)
00:1c.1 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #2 (rev d5)
00:1c.2 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #3 (rev d5)
00:1c.3 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #4 (rev d5)
00:1c.4 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #5 (rev d5)
00:1d.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI #1 (rev 05)
00:1f.0 ISA bridge: Intel Corporation C222 Series Chipset Family Server Essential SKU LPC Controller (rev 05)
00:1f.2 SATA controller: Intel Corporation 8 Series/C220 Series Chipset Family 6-port SATA Controller 1 [AHCI mode] (rev 05)
00:1f.3 SMBus: Intel Corporation 8 Series/C220 Series Chipset Family SMBus Controller (rev 05)
01:00.0 Ethernet controller: Intel Corporation I210 Gigabit Network Connection (rev 03)
02:00.0 Ethernet controller: Intel Corporation I210 Gigabit Network Connection (rev 03)
03:00.0 PCI bridge: ASPEED Technology, Inc. AST1150 PCI-to-PCI Bridge (rev 02)
04:00.0 VGA compatible controller: ASPEED Technology, Inc. ASPEED Graphics Family (rev 21)
05:00.0 PCI bridge: ASMedia Technology Inc. ASM1083/1085 PCIe to PCI Bridge (rev 04)
07:00.0 Serial controller: Xilinx Corporation Device 7014
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$
```

CSDN @w0shishabi

看到了slot编号后，可以详细查看一下：

```
a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ lspci -s 07:00.0 -v
07:00.0 Serial controller: Xilinx Corporation Device 7014 (prog-if 01 [16450])
    Subsystem: Xilinx Corporation Device 0007
    Flags: bus master, fast devsel, latency 0, IRQ 42
    Memory at dc200000 (32-bit, non-prefetchable) [size=64K]
    Memory at dc100000 (32-bit, non-prefetchable) [size=1M]
    Capabilities: <access denied>
    Kernel driver in use: xdma

a473@a473-TS100-E8-PI4:~/dma_ip_drivers-master/XDMA/linux-kernel/tools$ a
```