
PCI Express on Vertex-7 VC709

Author:

Tran Minh Quang

Matrikel-Nr: 2697987



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1	Introduction	2
2	PCI Express on Vivado Design Suit	2
2.1	PCI IP block	2
2.2	DDR3 SDRAM block	6
3	Xilinx PCI Express DMA Drivers	7
3.1	Install Xilinx PCI Express DMA Drivers	7
3.2	Testing and evaluating the performance	8
3.3	Bitstream preparation	8
3.4	XDMA driver usage	8

References	10
------------	----

1 Introduction

This document will present the steps needed to be able to transfer data between a computer and device via the PCIe express port. The setup consists of 2 main parts: install xdma driver on the host PC and design the corresponding hardware on Vivado. The following chapters will describe these in detail.

2 PCI Express on Vivado Design Suit

2.1 PCI IP block

In this section, a simple PCIe block design on Vivado is presented, which is used to communicate with the Xilinx PCI Express DMA driver on host PC.

As a first step, a new blank project with the board Virtex-7 VC709 Evaluation Platform should be created:

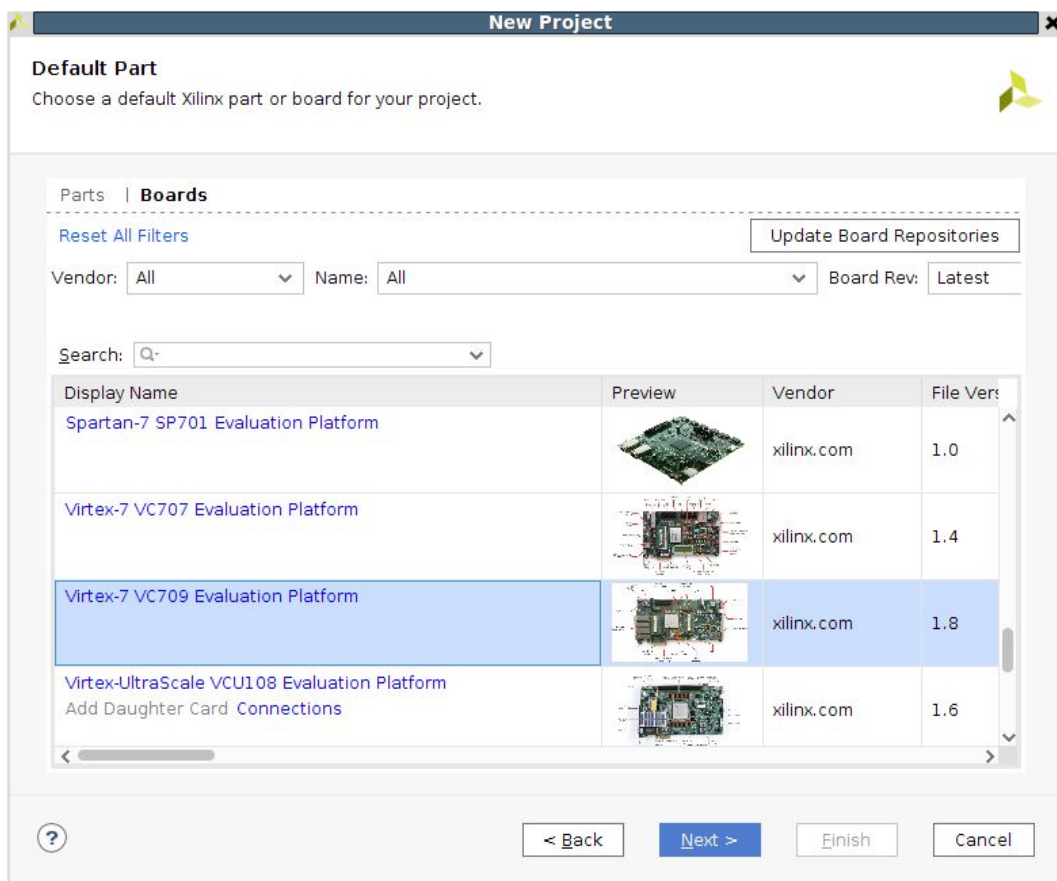


Figure 1: Create new project

Then create a block design and drag PCI Express IP in tab Board to Diagram region. A window will pop up to select the number of lanes for the design. In this example, pci_express x8 is selected:

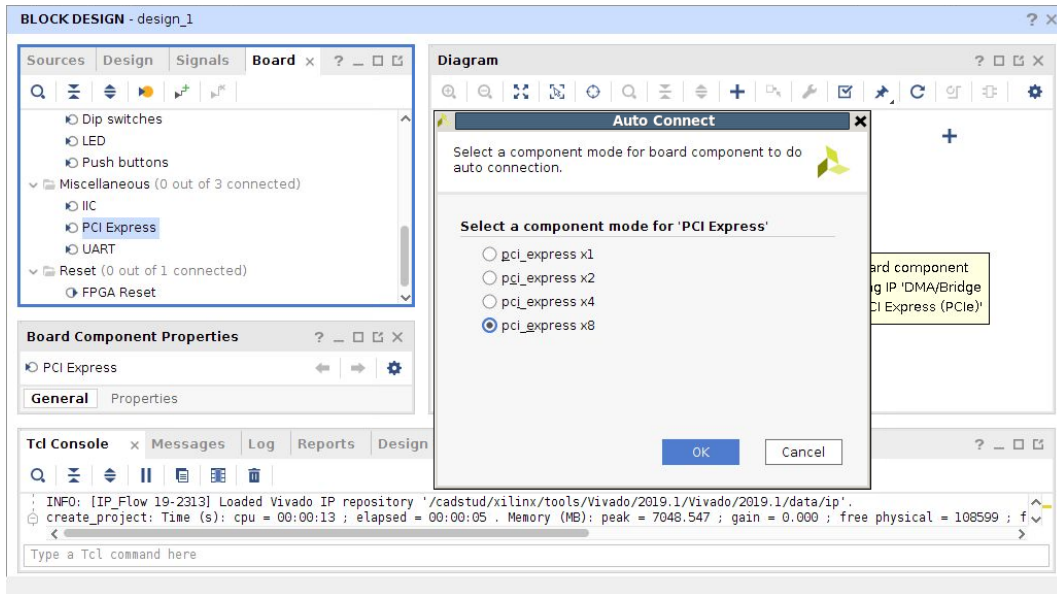


Figure 2: PCI block

Right click on the generated PCIe IP block and chose customize block to configure the IP. In the Basic tab, change the parameters as the following picture:

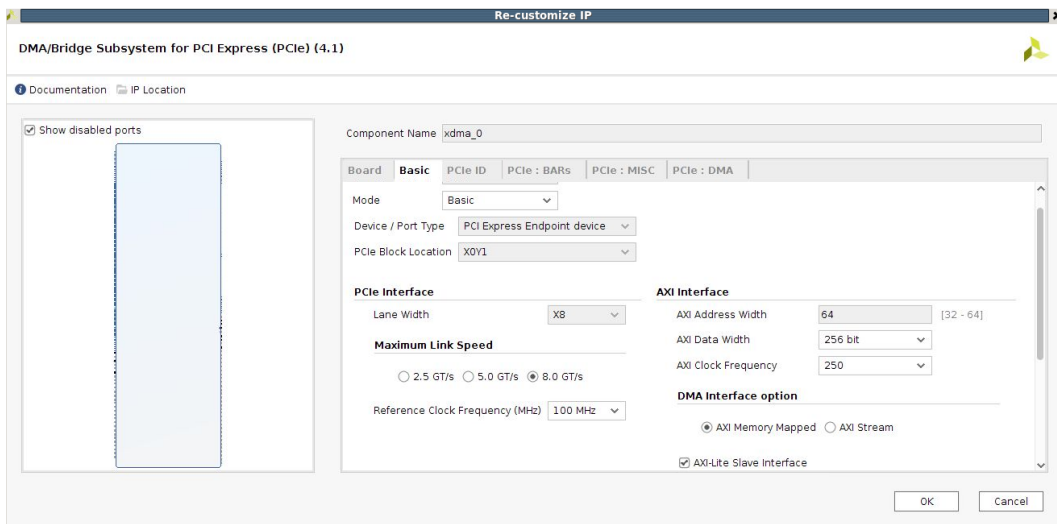


Figure 3: Basic tab configuration

Now move to PCIe ID tab and change Vendor ID to 10EE (This is Xilinx ID) and the device ID to one of those IDs:

```
/** PCIe lane width x8 */
{ PCI_DEVICE(0x10ee, 0x9018), },          /** PF 0 */
{ PCI_DEVICE(0x10ee, 0x9118), },          /** PF 1 */
{ PCI_DEVICE(0x10ee, 0x9218), },          /** PF 2 */
{ PCI_DEVICE(0x10ee, 0x9318), },          /** PF 3 */
```

The device ID is very important. It must be match with predefined IDs in pci_ids.h, which can be found at /dma_ip_drivers/QDMA/linux-kernel/driver/src, to help the driver recognize the hardware (the driver installation is described in the next chapter) . A user-defined ID can also be recognized, but the user has to modify pci_ids.h correspondingly. In this example, the ID 0x9018 is selected:

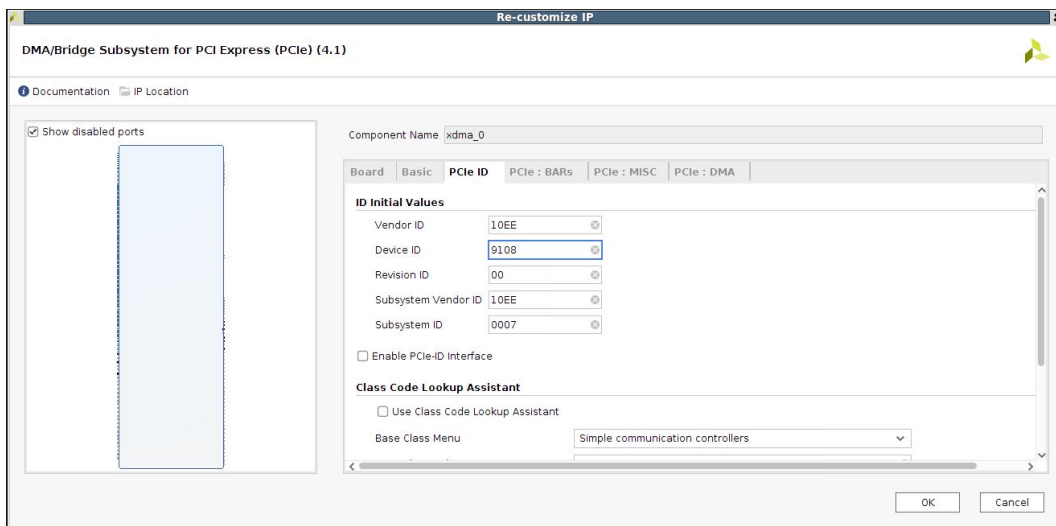


Figure 4: ID configuration

In the second part of PCIe ID tab Class Code Lookup Assistant change the parameters as follow (This configuration is used only for this example, for other applications, it should be changed correspondingly):

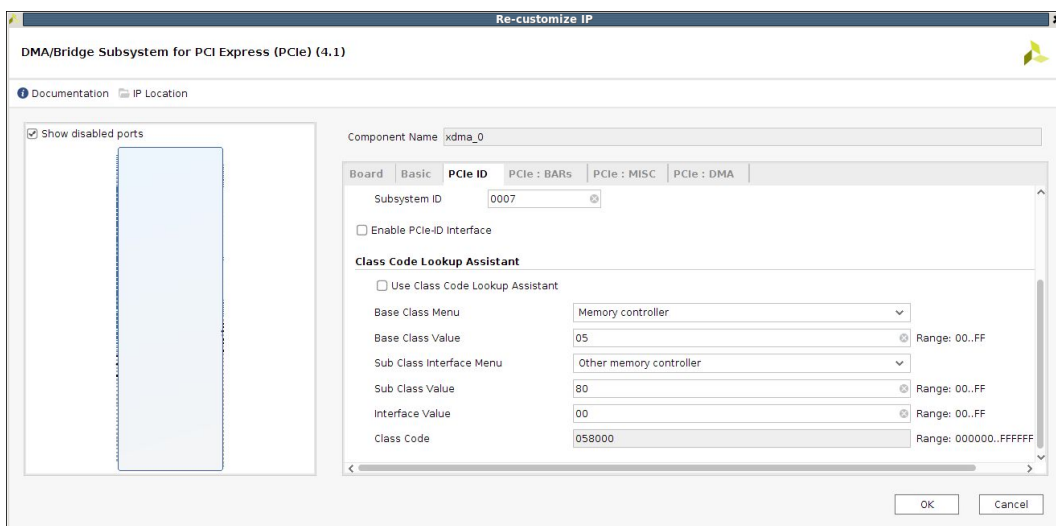


Figure 5: PCIe ID tab part 2

Now move to PCIe : BARs and modify the parameter as following:

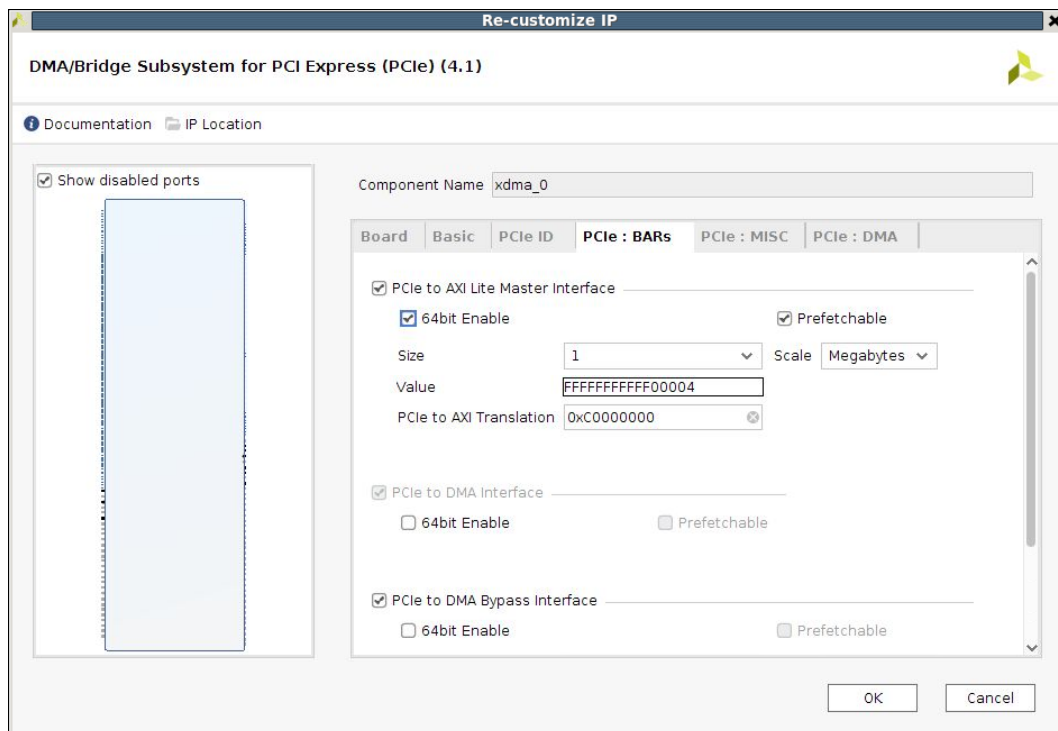


Figure 6: PCI: BARs tab configuration

PCIe : MISC tab is used to configure the interrupt parameters. In this example, all parameters are kept as default. The last tab PCIe : DMA provides parameters to specify the number of channels, which can be used to transfer data in parallel. The PCI IP from Xilinx supports up to 4 channels. The following is an example setup, where one channel is selected:

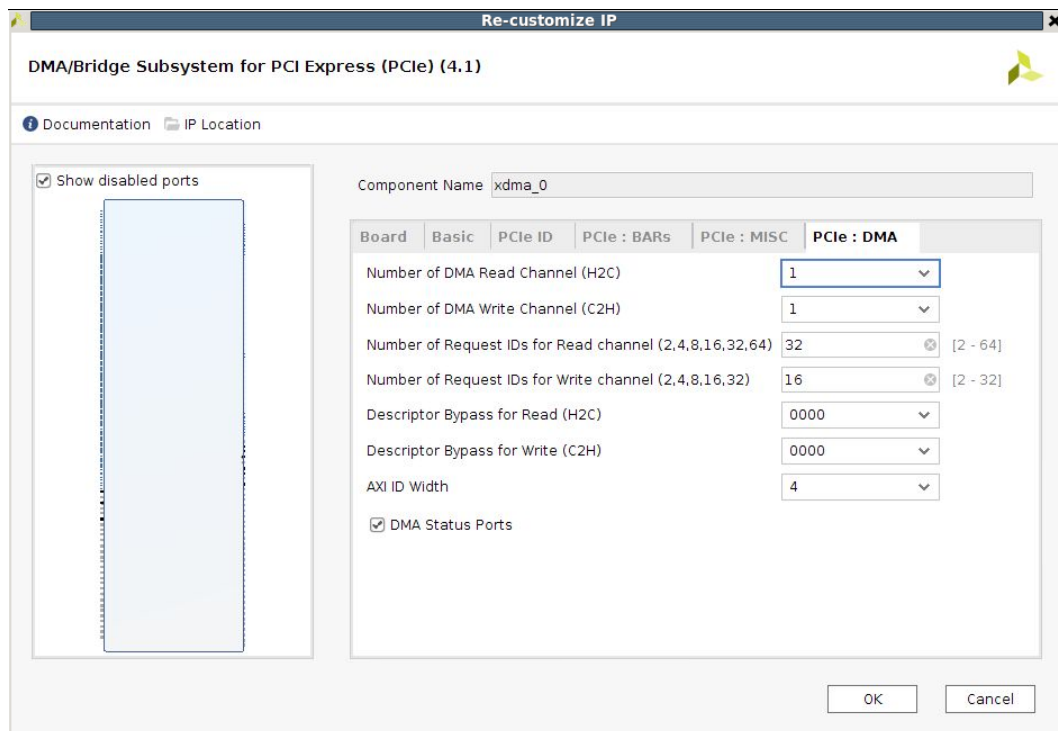


Figure 7: PCI: DMA tab configuration

2.2 DDR3 SDRAM block

A memory interface generator needs to be created to save data on FPGA. Drag the DDR3 SDRAM in Board tab into Diagram region and in the pop-up window select socket j1 for 4GB and socket j1 and j3 for 8GB:

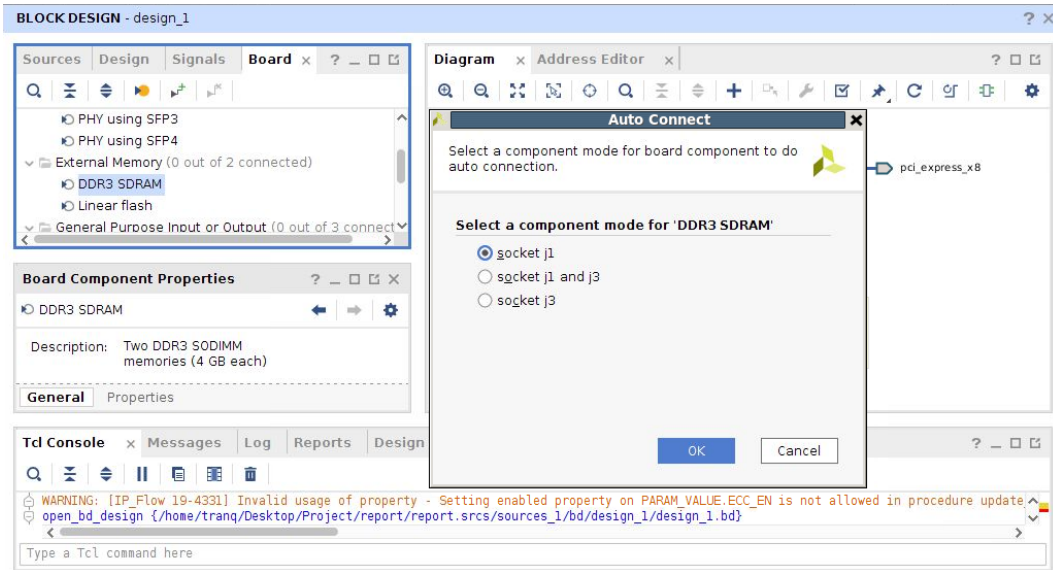


Figure 8: DDR3 SDRAM block

Now run block automation and connection automation to connect all blocks together. The final design is as in the following picture:

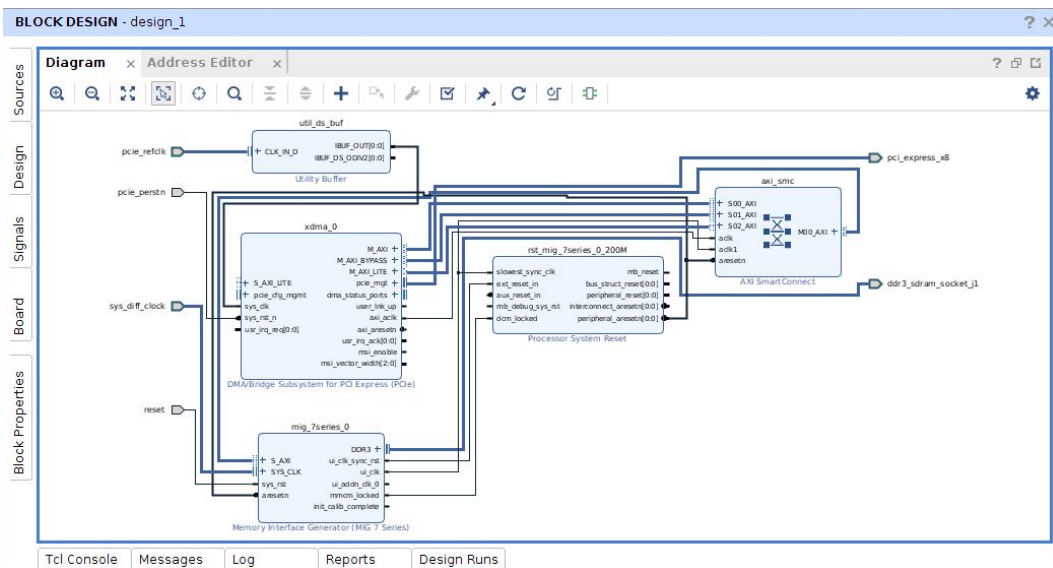


Figure 9: Final block design

The example design is complete. Create the HDL wrapper, generate bitstream, load the bitstream into FPGA and in the next step, the driver needs to be installed on host PC to communicate with FPGA over PCIe.

3 Xilinx PCI Express DMA Drivers

3.1 Install Xilinx PCI Express DMA Drivers

Xilinx provides many reference drivers such as XDMA, QDMA or VSEC can be run on a PCI Express root port host PC to interact with the DMA endpoint IP via PCI Express [2]. In this document, XDAM is used to transfer data between host PC and FPGA Vertex 7 VC709.

The recommended operating system is Ubuntu 18.04 LTS. Some other supported Linux distributions can also be found at[3]: https://xilinx.github.io/dma_ip_drivers/2020.1/linux-kernel/html/system-requirements.html

The first step is to download the software from the following link:

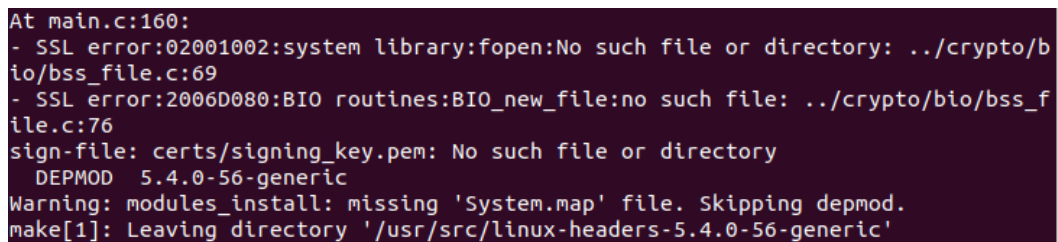
https://github.com/Xilinx/dma_ip_drivers

In the folder downloaded from github there are three drivers: QDMA, XVSEC and XDMA. The Xilinx QDMA can be used with Ultrascale+ devices. The XVSEC driver helps creating and deploying designs that may include the Xilinx VSEC PCIe Features. To communicate with FPGA Vertex 7 VC709 the XDMA driver is used.

From the folder dma_ip_drivers go to directory /XDMA/linux-kernel/xdma and run the following command to compile and install the kernel module driver:

```
$ sudo make install
```

At this step it is very often that this SSL error occurs:



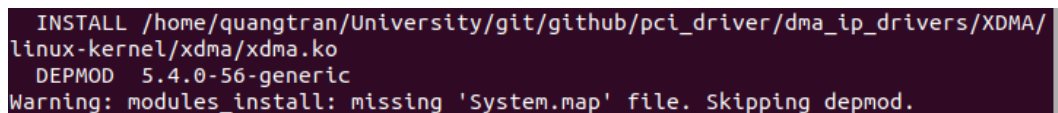
```
At main.c:160:
- SSL error:02001002:system library:fopen:No such file or directory: ../crypto/b
io/bss_file.c:69
- SSL error:2006D080:BIIO routines:BIIO_new_file:no such file: ../crypto/bio/bss_f
ile.c:76
sign-file: certs/signing_key.pem: No such file or directory
DEPMOD 5.4.0-56-generic
Warning: modules_install: missing 'System.map' file. Skipping depmod.
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-56-generic'
```

Figure 10: SSL error

To solve this issue the signing key needs to be generated using the following[1]:

```
$: cd /usr/src/linux-headers-$(uname -r)/certs
$: sudo openssl req -new -x509 -newkey rsa:2048\
-keyout signing_key.pem -outform DER\
-out signing_key.x509 -nodes -subj "/CN=Owner/"
```

Then try sudo make install again. At the end, there will probably the following warning:



```
INSTALL /home/quangtran/University/git/github/pci_driver/dma_ip_drivers/XDMA/
linux-kernel/xdma/xdma.ko
DEPMOD 5.4.0-56-generic
Warning: modules_install: missing 'System.map' file. Skipping depmod.
```

Figure 11: Install warning

But don't worry about it. The driver is installed successfully.

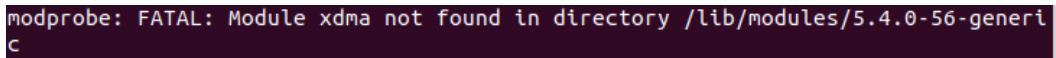
Now change the directory to the /XDMA/linux-kernel/tools and run:

```
$ make
```

Then load the kernel module driver in two steps. Firstly, run the command:

```
$ sudo modprobe xdma
```

At this step, one problem could happen:



```
modprobe: FATAL: Module xdma not found in directory /lib/modules/5.4.0-56-generic
```

Figure 12: Module xdma not found

If it is the case run this command and then try `sudo modprobe xdma` again:

```
$ sudo depmod
```

Secondly, change the directory to `/XDMA/linux-kernel/tests` and use the provided script:

```
$ ./load_driver.sh
```

The installation of XDMA driver is now complete.

3.2 Testing and evaluating the performance

Xilinx XDMA driver provides some script for testing and evaluating the performance at the directory `/XDMA/linux-kernel/tests`:

- `run_test.sh`:
This script runs sample tests on a Xilinx PCIe DMA target and returns a pass (0) or fail (1) result. This script is intended for use with the PCIe DMA example design.
- `perform_hwcount.sh`:
This script runs hardware performance for XDMA for both Host to Card (H2C) and Card to Host (C2H). The result are copied to 'hw_log_h2c.txt' and 'hw_log_c2h.txt' text files. For each direction the performance script loops from 64 bytes to 4MBytes and generate performance numbers (byte size doubles for each loop count). You can grep for 'data rate' on those two files to see data rate values. Data rate values are in percentage of maximum throughput. Maximum data rate for x8 Gen3 is 8Gbytes/s, so for a x8Gen3 design value of 0.81 data rate is $0.81 * 8 = 6.48$ Gbytes/s. Maximum data rate for x16 Gen3 is 16Gbytes/s, so for a x16Gen3 design value of 0.78 data rate is $0.78 * 16 = 12.48$ Gbytes/s. This program can be run on AXI-MM example design.

3.3 Bitstream preparation

To transfer data between PC and FPGA a bitstream of the data is required. One possible way is using the binmake as follow:

```
$ git clone https://github.com/dadadel/binmake
$ cd binmake
$ make
```

The bitstream can be generated by running the following command:

```
$ ./binmake file.txt file.bin
```

For more details, please visit the following link:

<https://stackoverflow.com/questions/8521240/how-to-create-binary-file-using-bash>

3.4 XDMA driver usage

Xilinx XDMA driver provides two basic user application tools can be found at the directory `/XDMA/linux-kernel/tools`:

- DMA to device (`dma_to_device`) is used to perform the host to card data transfers.
usage:

```
$ dma_to_device [OPTIONS]
```

OPTIONS:

```
-d (--device) device node name
-a (--address) the start address on the AXI bus
-s (--size) size of a single transfer in bytes, default 32 bytes
-o (--offset) page offset of transfer
-c (--count) number of transfers, default 1
-f (--data input file) filename to read the data from.
-w (--data output file) filename to write the data of the transfers
-h (--help) print usage help and exit
-v (--verbose) verbose output
```

EXAMPLE:

```
$ dma_to_device -d /dev/xdma0_h2c_0 -f data/datafile.bin -s 16 -a 0x00000000
```

- DMA from Device (dma_from_device) is used to perform the card to host data transfer.
usage:

```
$ dma_from_device [OPTIONS]
```

OPTIONS:

```
-d (--device) device node name
-a (--address) the start address on the AXI bus
-s (--size) size of a single transfer in bytes, default 32 bytes.
-o (--offset) page offset of transfer
-c (--count) number of transfers, default is 1.
-f (--file) file to write the data of the transfers
-h (--help) print usage help and exit
-v (--verbose) verbose output
```

EXAMPLE:

```
$ dma_from_device -d /dev/xdma0_c2h_0 -f data/output_datafile.bin -s 16\
-a 0x00000000
```

The implementation of the applications can be found at the directory /XDMA/linux-kernel/tools in dma_from_device.c and dma_to_device.c. More usage examples could be learned the from the provided script dma_memory_mapped_test.sh, dma_streaming_test.sh, perform_hwcount.sh and run_test.sh, which allocate at /XDMA/linux-kernel/tests.

References

- [1] Ssl error when compile xdma 39. https://github.com/Xilinx/dma_ip_drivers/issues/39.
- [2] Xilinx pci express dma drivers and software guide. <https://www.xilinx.com/support/answers/65444.html>.
- [3] Xilinx pcie drivers documentation. https://xilinx.github.io/dma_ip_drivers/.