DMA/Bridge for PCIe Drivers Overview

AMD
together we advance_

Agenda

- The PCIe DMA Driver
- Accessing and Building the Xilinx Driver

The PCIe DMA Driver

The Xilinx PCI Express DMA IP provides high-performance direct memory access (DMA) via PCI Express. The PCIe DMA can be implemented in Xilinx 7-series XT and UltraScale devices. Xilinx Support Answer 65444 provides drivers and software that can be run on a PCI Express root port host PC to interact with the DMA endpoint IP via PCI Express. The drivers and software provided with the answer record are designed for Linux and Windows operating systems and can be used for lab testing or as a reference for driver and software development. Through the use of the PCIe DMA IP and the associated drivers and software you will be able to generate high-throughput PCIe memory transactions between a host PC and a Xilinx FPGA.

Accessing and Building the Xilinx Driver

The current driver implementation uses the following Kernel functions and must be included in your OS kernel version. The following Linux distributions have been tested:

- Red Hat (RHEL 7)
- Fedora
- CentOS
- Ubuntu

On your host computer, make a temporary directory using mkdir dma_driver and navigate to this directory. Copy the downloaded zip file to the current directory with cp ../Downloads/linux-kernel.zip . (do not forget the period), and unzip the driver zip file. Navigating into linux-kernel, we can open the README to find out how to install the driver. Be aware that the Usage instructions are not exact and there are some additional steps required in between.

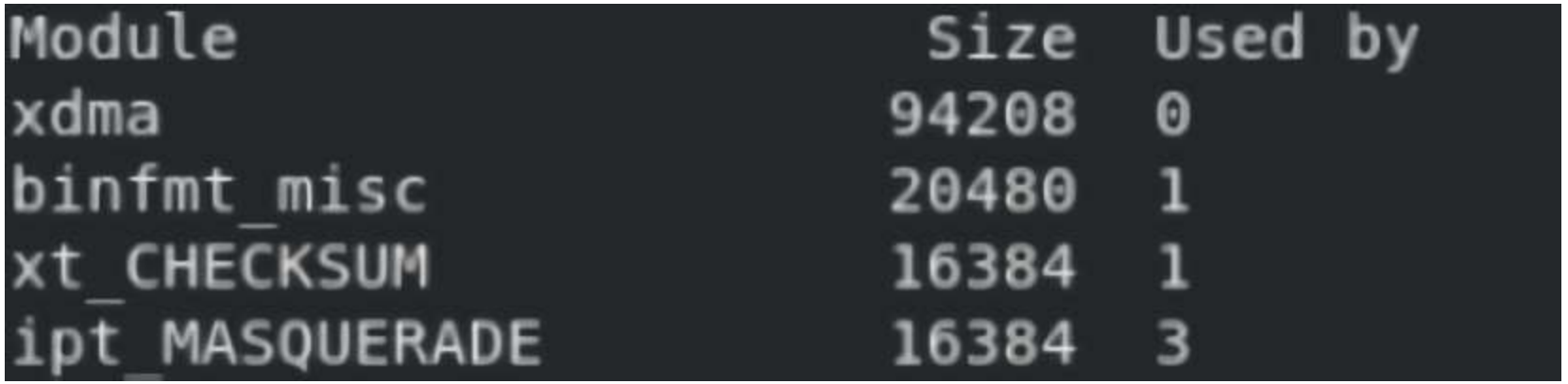
During our testing, after attempting to run the Makefile by using sudo make install in the xdma folder, we found a few errors while compiling. These errors may be fixed in the future, but as of the current XDMA driver version of v2020.1.8, these may prevent you from properly creating the driver:

•Make sure you install the dependencies kernel-devel and elfutils-libelf-devel.

```
Makefile:10: XVC_FLAGS: .
make -C /lib/modules/4.18.0-240.15.1.el8_3.x86_64/build M=/home/gpellot/dma_driver/linux-ker
nel/xdma modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-240.15.1.el8_3.x86_64'
/home/gpellot/dma_driver/linux-kernel/xdma/Makefile:10: XVC_FLAGS: .
  CC [M]  /home/gpellot/dma_driver/linux-kernel/xdma/xdma_mod.o
/home/gpellot/dma_driver/linux-kernel/xdma/xdma_mod.c: In function 'xdma_error_resume':
/home/gpellot/dma_driver/linux-kernel/xdma/xdma_mod.c:299:2: error: implicit declaration of
function 'pci_cleanup_aer_uncorrect_error_status' [-Werror=implicit-function-declaration]
  pci_cleanup_aer_uncorrect_error_status(pdev);
  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cc1: some warnings being treated as errors
make[2]: *** [scripts/Makefile.build:316: /home/gpellot/dma_driver/linux-kernel/xdma/xdma_mo
d.o] Error 1
make[1]: *** [Makefile:1544: _module_/home/gpellot/dma_driver/linux-kernel/xdma] Error 2
make[1]: Leaving directory '/usr/src/kernels/4.18.0-240.15.1.el8_3.x86_64'
make: *** [Makefile:27: all] Error 2
```

Possible Makefile error

•If you encounter this error (implicit declaration of function pci_cleanup_aer_uncorrect_error_status), open xdma_mod.c in your editor of choice and replace pci_cleanup_aer_uncorrect_error_status(pdev) for pci_aer_clear_nonfatal_status(pdev). Save the file.
•There are other errors present in libxdma.c and xmda_mod.c. These files can be updated from this pull request here.
Once all errors have been fixed (there may be more or less depending on the version, check the Github repo), make the driver with sudo make install and run sudo make in the tools folder. Load the driver by navigating to the tests folder, making the tests executables (we opted to test load_driver with chmod +x 'load_driver.sh') and running sudo ./load_driver.sh. Check that the driver is loaded into the kernel with lsmod.



Checking the kernel modules

We can test the driver using the same process by running sudo ./run_test.sh. When we ran the test, we encountered an error on line 28. To fix this, open the run_test.sh file, and on line 28, change if [$channelId == "1fc"]; then to if [“$channelId” == "1fc"]; then. From here, you will be able to connect your physical PCIe device to the host machine and run each test to check that the host can identify the PCIe Endpoint.

```
Pushed a login request to your device...
Success. Logging you in...
Error at line 94, file reg_rw.c (2) [No such file or directory]
./run_test.sh: line 28: [: ==: unary operator expected
Error at line 94, file reg_rw.c (2) [No such file or directory]
./run_test.sh: line 28: [: ==: unary operator expected
Error at line 94, file reg_rw.c (2) [No such file or directory]
./run_test.sh: line 28: [: ==: unary operator expected
Error at line 94, file reg_rw.c (2) [No such file or directory]
./run_test.sh: line 28: [: ==: unary operator expected
Info: Number of enabled h2c channels = 0
Error at line 94, file reg_rw.c (2) [No such file or directory]
Error at line 94, file reg_rw.c (2) [No such file or directory]
Error at line 94, file reg_rw.c (2) [No such file or directory]
Error at line 94, file reg_rw.c (2) [No such file or directory]
Info: Number of enabled c2h channels = 0
Info: The PCIe DMA core is memory mapped.
Error: No PCIe DMA channels were identified.
```

Error in run_test.sh


AMD