# HOW TO USE THE USER APPLICATION WITH PCI-E LINUX DRIVER

## Includes details about Verilog code

Describes how to use a pcie user application (included with the Linux kernel module) to transfer data between DE5-Net board DDR3 and Host PC. Also describes how to run the design on board.

Sachin Kumawat
skumawat@ucdavis.edu

# NOTES ABOUT THE PCI-E DRIVER

The PCI-e Linux driver has been tested working with **Ubuntu 16.04, Kernel version 4.4.0-59-generic**. This driver is **not DMA based** and does not use descriptors to set up the data transfer. Therefore if very large amount of data is transferred, it might keep the Host PC processor busy for the entire duration of data transfer. The data transfer is achieved by writing directly to On-Chip memory via a BAR in blocks of 64KB and a central controller then transfers the data to/from the corresponding location in DDR3. The driver is **not** optimized for high throughput data transfer.

# NOTES ABOUT THE VERILOG DESIGN

- The Verilog design uses **16-bit fixed point** data quantization. Therefore floating point computations are not yet supported. The design is a reference design to study the convolution process on an FPGA and can be used to develop a more practical design. Any overflow will be ignored.
- A central controller initializes the Off-chip SDRAMs with some test data and does a self-check on the computed values by default to verify that the design is functionally correct. Therefore if some custom data is written to the SDRAMs using the User Application, it is expected that the computation self-check will fail as the computed values will be different. The user, in that case, can read back the computed values with the User Application and verify.
- The design uses all the 10 user controllable LEDs and 4 Switches and the 4 Push buttons
- Push Buttons:
    1. BUTTON0: used to reset the controllers and computation modules
    2. BUTTON1 & BUTTON2: used to read timer values off of LEDs and HEX displays
    3. BUTTON3: used to manually reset the PCI-e Hard IP. **CAUTION**: *this will reset the PCI-e link with the host computer and Host kernel will reboot due to that.*
- Switches:
    1. SW0: used to switch between SDRAM & computation status [0] or debug info [1] (see 2.) on LEDs and HEX displays. The Following SDRAM Controller and computation status are displayed:
        - LED0: sdram calibration done
        - LED1: sdram calibration successful
        - LED2: self computation test done
        - LED3: self computation test or sdram read/writes fail
    2. SW1 & SW2: used to select one of the following debug info to be displayed on LEDs and HEX display:
        - Main State [00]
        - Read State [10]
        - Debug message from Central Controller (CC) [10]
        - A Global timer counting the total number of cycles for data transfer and computation [11]
    3. SW3: used to verify the PCI-e functioning [0] and then start the computation [1] once data transfer is done

Refer to the following section on who to run the design and use the buttons and switches to verify successful execution.

# USER APPLICATION

***NOTE***: Please follow the instructions in README file to install the PCI-e Linux driver (which also installs the user application) before proceeding.

The user application interacts with the kernel module to transfer custom data to/from the DE5-Net DDR3 SDRAMs. It can be used to perform the following tasks.

- Set the Read/Write location base Address
- Set the total number of elements or data locations need to Read/Write
- Perform the Read operation and write the read data to a file named *AlexNet_data_rd.txt*
- Perform the Write operation by reading the data from a file name *AlexNet_data.txt*

**Board Setup:**

Before proceeding make sure the board setup is correct to run the design:

- Configure the PCI-e HIP for **x8** operation. Refer the manual to locate and set the corresponding DIP switch
- Keep all the Switches SW0-3 on **low** (logic 0) position

## 1. Data Write

Follow the instruction below to successfully write the data.

- Set the base address as the location to start writing from. For example start writing from $0^{th}$ location in device global memory
- Set total number of 16 bit data elements needed to be written. For example for AlexNet $1^{st}$ layer:
    - IFM: 227 X 227 X 3 (IFM size X #Channels)
    - Weights: (11 X 11 X 3 + 1) X 96 (3D kernels + bias)
    - 227 X 227 X 3 + (11 X 11 X 3 + 1) X 96 = 189531
- Start write. Note that there should be atleast 189531 data values in *AlexNet_data.txt* file

Note: After the process is done, verify that *Write Eplast Timeout = 0* and *Write Pass = 1*. If not, the write process failed for some reason. **Reset** the design using **BUTTON0** (be careful not to press BUTTON3!) and try again. The PCI-e handshake sometimes get stuck in *DDR3_WRITE_FINISH* and *DDR3_READ_FINISH* state due to non-matching clocks.

## 2. Data Read

Follow the instruction below to successfully read the data.

- Set the base address as the location to start reading from. For example, for AlexNet 1st layer, start reading from 189531th location since till 189530, IFM and Weight data is written, so output will be written from 189531 onwards
- Set total number of 16 bit data elements needed to be read. For example for AlexNet 1st layer:
    - OFM: 55 X 55 X 96 = 290400 (OFM size X #channels)
- Start read. Note that there should be enough disk space available

Note: After the process is done, verify that *Read Eplast Timeout = 0* and *Read Pass = 1.* If not, the read process failed. **Reset** the design using **BUTTON0** (again be careful not to press BUTTON3!) and try again.

# STEPS SUMMARY TO RUN THE DESIGN

**NOTE**: only install the PCI-e driver once every time you boot the Host PC. If you need to install it again if you made some changes, either restart the Host PC or reload the kernel. Currently there seems to be some problem with the driver unload, which crashes the kernel if tried to install the same kernel driver module again.

1. Program the FPGA using computer 1 with USB-Blaster cable
2. Reboot the Host PC to train PCI-e link
3. Verify PCI-e Hard IP *alive_led* signal is active by checking if **LED_RJ45_L** is blinking. If the PCI-e link trained well, it should continuously blink. If not, then there might be some problem with the HIP configuration or on-board hardware (check the DIP switch!).
4. Reset the design with BUTTON0
5. Transfer the data using PCI-e user application
6. Switch SW3 to logic high [1]
7. Verify that the computation finished by checking **LED2** is **OFF** (i.e. logically 1). This says that the computation was completed.
    a. If yes, then check if **LED3** is **ON** i.e. logically 0. This says that the computation passed the self-test. If you are writing you custom data, this test will fail and that is expected. To verify the computation in that case, you need to read the data using the user application and then verify yourself
    b. If no, then the computation did not complete successfully. Please reset the design using **BUTTON0 (careful!)** and follow the instructions from **step 4** again to run the design
8. Read the data back to Host PC using the PCI-e user application
9. Verify the computation yourself
10. Reset the design and continue from **step 4** to test again with different data