P51 - Lab 1, Introduction to NetFPGA

Dr Noa Zilberman

Lent, 2017/18

The goal of this lab is to introduce you to NetFPGA, and to provide hands on experience both in using the platform and developing for it.

1 Background

The demand-led growth of cloud computing and datacenter networks has meant that many constituent technologies are beyond the budget of the research community. In order to make and validate timely, relevant research contributions, the wider research community requires accessible evaluation, experimentation and demonstration environments with specification comparable to the subsystems of the most massive datacenter networks.

The NetFPGA is an open platform enabling researchers and instructors to build high-speed, hardware-accelerated networking systems. The most prominent NetFPGA success is OpenFlow, which in turn has reignited the Software Defined Networking movement. NetFPGA enabled OpenFlow by providing a widely available open-source development platform capable of line-rate operation and was, until its commercial uptake, the reference platform for OpenFlow.

NetFPGA enables rapid prototyping of high bandwidth devices, using flexible, open-source IPs. Specifically, we use NetFPGA SUME, an open-source FPGA-based PCIe board, designed for the research community. NetFPGA SUME has I/O capabilities for 100Gbps operation as a networking device, stand alone computing unit or for test and measurement.

2 Development Machines

During the course you will use an assigned development machine. Each pair will be assigned a different machine. All the machines are located in the Practical Classroom (SW02). This provides access to the machines, so you can change the physical connectivity according to the experiment.

You will interact with the machines via ssh:

1. On a computer in the Practical Classroom, log in using your own UIS credentials.

2. `ssh -X root@<hostname>.nf.cl.cam.ac.uk` and enter the password. Hosts ending in .cl.cam.ac.uk are permitted to ssh into these machines. `-X` enables X11 forwarding, allowing you to run graphical applications.
   To ssh to the machines from outside the lab, follow the instructions on `https://www.cl.cam.ac.uk/local/sys/ssh/`.

Hostname	IP Address
nf-test101	128.232.82.84
nf-test108	128.232.82.78
nf-test110	128.232.82.93
nf-test111	128.232.82.81

Important: The IP addresses noted above should not be used for anything except for communication with the machines. The network interfaces assigned for the tests use different IP addresses.

3 Test Setup

Each of the development machines is equipped as follows:

- NetFPGA SUME board, used as the development platform
  NetFPGA network ports are marked *nf0* to *nf3*.
- Solarflare network interface cards (NIC) - SFN6122F
  Solarflare network ports are marked *slf0* and *slf1*.
- Optical fibers - duplex fibres with separate strands for transmitting and receiving.
- Optical 10G transceivers (SFP+) - converting electrical signals to optical signals and vice versa.

The network port markings noted above apply only to Figure 1. The name of the port on each machine may differ, (e.g.,*eth1*).
In a typical setup, each NIC port will be connected to a NetFPGA SUME port, e.g. slf0 to nf0, and slf1 to nf1.

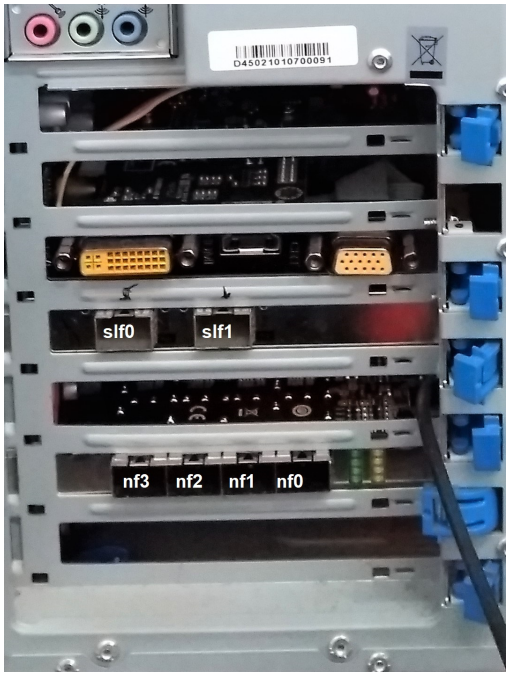

Figure 1: Development Platform

4 Saving Your Work

Make sure to frequently back up your work.
The most recommended way to back up your work is using frequent commits to a git repository. Please do not push any changes, data or results directly to the NetFPGA repository. You can fork the repository to your own user and push changes there. If you would like to suggest a correction or an enhancement to the code, please follow the instructions in:
`https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/Contributing-Code`.

To copy a remote directory onto your local machine:
`sftp root@<hostname>.nf.cl.cam.ac.uk` and `get -r <directory>`.
There are also other ways to copy a remote directory, you are welcome to use those as well.

5 Using NetFPGA

This section provides step-by-step instructions how to access and test a NetFPGA design. To this end, we will be using the Reference Switch design studied in class.

5.1 Accessing and programming the board

1. Login to the development machine:

   ```
   $ ssh root@<hostname>.nf.cl.cam.ac.uk
   ```

2. ```
   $ cd ~/P51/NetFPGA-SUME-live/tools/
   $ vim settings.sh
   ```

3. Make sure that NF_PROJECT_NAME is set to "reference_switch"

4. Load the environment settings:

   ```
   $ source settings.sh
   ```

5. Compile the driver (one time only):

   ```
   cd $DRIVER_FOLDER
   make
   ```

6. Compile register access application (one time only):

   ```
   cd $APPS_FOLDER
   make
   ```

7. Program the NetFPGA board:

   ```
   $ cd $SUME_FOLDER/tools/scripts
   $ ./run_load_image.sh $NF_DESIGN_DIR/bitfiles/reference_switch.bit
   ```

   Note that you may need to reset the machine if this is the first time the board is programmed after power up. This allows the PCIe bus to properly identify and enumerate the board. The first time the board is programmed after reset (not power up!), you may see a message "rmmod: ERROR: Module sume_riffa is not currently loaded". It can be ignored.

8. The board is now programmed and ready.
   ```

5.2 Simple Debug

- Check if the board is recognized by the host:

  ```
  lspci -v |grep Xilinx
  ```

  The expected result is:

  ```
  01:00.0 Memory controller: Xilinx Corporation Device 7028
  Subsystem: Xilinx Corporation Device 0007
  ```

- Check if simple register access works:

  ```
  cd $APPS_FOLDER
  ./rwaxi
  ```

  The expected result is:

  ```
  WARNING: using default test address 0x44020000
  READ  0x44020000 = 0x0001da02
  ```

  Note that the return value may change - 0x0001da02 is the ID of the output port lookup module. You module(s) may use a different value.

- Reading all the design registers:

  ```
  cd $APPS_FOLDER
  make register_read
  cd $NF_DESIGN_DIR/sw/host/apps/
  ./register_read.sh
  ```

- Reading a specific register (e.g. address 0x44010004):

  ```
  cd $APPS_FOLDER
  ./rwaxi -a 0x44010008
  ```

- Writing to a specific register (e.g. address 0x44010010, value 0xabcdabcd):

  ```
  cd $APPS_FOLDER
  ./rwaxi -a 0x44010010 -w 0xabcdabcd
  ```

- Making sure that register access reaching the module:
  To make sure that a register access reaches the hardware and is not replied by some cache, most NetFPGA modules implement a "FLIP" register (typically at offset 0xc). The FLIP register returns the inverse of the value written to it. For example:

```
cd $APPS_FOLDER
./rwaxi -a 0x4401000C -w 0x55555555
./rwaxi -a 0x4401000C
```

Will return:

```
READ  0x4401000c = 0xaaaaaaaa
```

- Making a file (e.g., sh) executable:

```
chmod +x <filename>
```

5.3 Testing a design

This section covers simple design testing of the NetFPGA platform, focusing on functionality. Performance testing will be discussed later in the course.

For each NetFPGA design, functional tests need to be written. The tests are located under *$NF_DESIGN_DIR/test*. Each test has a dedicated folder called *hw/sim/both_major_minor*, for example *both_simple_broadcast* or *both_learning_sw*. The test itself is written in python, in a file called *run.py*. The NetFPGA test environment (python based as well) calls this file when invoked.

The NetFPGA Reference Switch is a *Learning Switch*, meaning that forwarding is done based on MAC addresses that the switch sees in incoming packets and associates with ports. For example, is a packet with source MAC address $aa:bb:cc:dd:ee:ff$ is received on Port 1, the switch will save in its lookup table an entry equivalent to "$aa:bb:cc:dd:ee:ff$, Port 1", and the next time a packet arrives with a destination MAC address $aa:bb:cc:dd:ee:ff$ it will know to send it to port 1. When the destination MAC address is not in the lookup table, the switch uses broadcast: it sends the packet on all ports (except for the incoming port).

The test *both_simple_broadcast* tests the switch's broadcast operation, and the test *both_learning_sw* tests both broadcast and learning. Read the file *run.py* under each test to learn how exactly it is done.

The following describes the steps for running a simulation, without GUI:

```
cd $SUME_FOLDER/tools/scripts
./nf_test.py sim --major learning --minor sw
```

The following describes the steps for running a simulation, with GUI (Vivado xsim):

```
cd $SUME_FOLDER/tools/scripts
./nf_test.py sim --major learning --minor sw --gui
```

Note that using xsim is not mandatory. You can also change the environment and use Modelsim, and a license for that is available.

The following describes the steps for running a hardware test:

1. Connect port 0 of the NIC to port 0 of NetFPGA using a fibre.
2. Connect port 1 of the NIC to port 1 of NetFPGA using a fibre.
3. Check what are the interfaces names of your NIC (These can also be configures/modified, but it is not mandatory):

   ```
   ifconfig -a
   ```

4. Update interface names in the project test configuration files:

   ```
   vim $NF_DESIGN_DIR/test/global/setup
   vim $NF_DESIGN_DIR/test/connections/conn
   ```

   Note that *connections/conn* file reflects the physical (external) connectivity between the NIC's ports and NetFPGA's ports.

5. Run the hardware test:

   ```
   cd $SUME_FOLDER/tools/scripts
   ./nf_test.py hw --major learning --minor sw
   ```

The Reference Switch also works as a normal switch, enabling to connect multiple devices. For the following exercise you should work with another team, so two machines are used. We refer to those as Machine A and machine B.

1. On Machine A, Connect port 0 of the NIC to port 0 of NetFPGA using a fibre.
2. On Machine B, Connect port 0 of the NIC to port 1 of NetFPGA on Machine A using a fibre.
3. On Machine A, configure NIC port 0 (lets assume it is called eth1):

   ```
   ifconfig eth1 10.0.0.100 up
   ```

4. On Machine B, configure NIC port 0 (lets assume it is called eth1):

   ```
   ifconfig eth1 10.0.0.101 up
   ```

5. Check that the connectivity between both machines works. From Machine A ping Machine B:

   ```
   ping 10.0.0.101
   ```

6. You can also check the same in the opposite direction. From Machine B ping Machine A:

   ```
   ping 10.0.0.100
   ```

6 Building a project

The following steps are typically required when building a project:

- Compiling an IP core:

  ```
  cd $IP_FOLDER/<ip core name>
  make
  ```

  This step is required only for new IP cores or when changes are made to the tcl file of the core. There is no need to run make if only the HDL files were modified.

- Compiling CAM/TCAM cores:
  Follows the instructions on `https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/NetFPGA-SUME-TCAM-IPs`
  The CAM core is required for building the NetFPGA project. You only need to run this step once.

- Compiling all cores and building libraries:

  ```
  cd $SUME_FOLDER
  make
  ```

  This step if typically required only once: after the git repository is cloned or pulled. It is also required if the *make clean* command was called.

- Building a project:

  ```
  cd $NF_DESIGN_DIR
  make
  ```

  The result of this step is the programming (bit) file.

7 Useful links

- NetFPGA Repository: `https://github.com/NetFPGA/NetFPGA-SUME-live/`
- NetFPGA Wiki: `https://github.com/NetFPGA/NetFPGA-SUME-public/wiki`
- NetFPGA registration page: `https://netfpga.org/site/#/SUME_reg_form/`