P51 - Lab 2
Introduction to NetFPGA and P4-NetFPGA - Part 2

Dr Noa Zilberman

Lent, 2018/19

The goal of this lab is to introduce you to the NetFPGA register infrastructure, as well as to the test environment, providing hands on experience in NetFPGA and P4-NetFPGA development.

1 Development Machines

This week you will continue and use the machines assigned to you last week. All the machines are located in the Practical Classroom (SW02).

1. On a computer in the Practical Classroom, log in using your own UIS credentials.
2. From Moodle, download the private key.
3. Limit the permissions of the private key: `chmod 600 p51_key`
4. `ssh -X root@<hostname>.nf.cl.cam.ac.uk -i p51_key`. Hosts ending in .cl.cam.ac.uk are permitted to ssh into these machines. `-X` enables X11 forwarding, allowing you to run graphical applications. `-i` is the private ssh authentication key.

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-test103	128.232.82.63
nf-test104	128.232.82.64
nf-test102	128.232.82.62
nf-test108	128.232.82.68
nf-test110	128.232.82.70
nf-test111	128.232.82.71

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.

2 Practical Instructions

This section provides step-by-step instructions how to add a new register to your design. To this end, we will be using the (verilog) Reference Switch design studied in class.

2.1 Accessing the board

1. Login to the development machine:

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

2. Pull the latest NetFPGA release:

   ```
   cd ~/NetFPGA-SUME-live/
   git pull
   ```

3.
   ```
   cd tools
   vim settings.sh
   ```

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

5. Load the environment settings:

   ```
   source settings.sh
   ```

6. Compile all cores:

   ```
   cd $SUME_FOLDER
   make
   ```

2.2 Adding a new register

The goal of this exercise is to add a register that counts the number of packets of size 64B. Once you complete this exercise, try adding a second register that sets the packet size to monitor.

The following instructions use the spreadsheet.

You can alternatively use the csv file, combined with the script *csv_gen.py*. The file is located under *SUME_FOLDER/tools/infrastructure*. Then skip to step 7.

1. In Libreoffice, set security to medium.

2. Open $IP_FOLDER/switch_output_port_lookup_v1_0_1/data/module_generation.xls

3. If needed, change block name to match your module name (for sub-module this is optional)
4. Delete all indirect registers (and others you dont want) (note potential issues in some releases)
5. Change OS to Linux
6. Press Generate Registers. You may need to copy locally *SUME_FOLDER/tools/infrastructure/regs_template.txt*.
7. From console, run:

   ```
   python regs_gen.py
   cp *.v $IP_FOLDER/switch_output_port_lookup_v1_0_1/hdl
   ```

8. Copy data files to the data folder:

   ```
   cp <*.tcl,*.h,*.txt> $IP_FOLDER/switch_output_port_lookup_v1_0_1/data
   ```

9. Go to the core's folder:

   ```
   cd $IP_FOLDER/switch\output_port_lookup_v1_0_1/
   ```

10. Open the HDL code of the core:

    ```
    vim hdl/switch_output_port_lookup.v
    ```

11. Add the following lines to the code, starting line 370 (this is a hint, you can also use your own code):

    ```
    reg next_is_new;
    always @(posedge axis_aclk) next_is_new <= #1 ~resetn_sync ? 1'b0 :
         (s_axis_tvalid && s_axis_tready) ? s_axis_tlast : next_is_new;
    ```

12. Copy the lines from the template file to switch_output_port_lookup.v
13. Add in switch_output_port.v support for the register functionality
14. Compiling your IP core:

    ```
    cd $IP_FOLDER/switch_output_port_lookup_v1_0_1/
    make
    ```

15. Update the functional test to read (or write) your register:

    ```
    cd $NF_DESIGN_DIR/test/both_learning_sw
    vim run.py
    ```

16. Run a simulation of your design and test that it works:

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

2.3 Coding and adding registers to the P4 module

The goal of this exercise is to add write a P4-based calculator program, including the use of registers. The switch acts as the calculator, receiving a packet and operating on its contents. The operations that need to be supported are:

- ADD - add two operands and return the result.
- SUBTRACT - subtract two operands and return the result.
- ADD_REG - add an operand to the current value stored in a register on the switch and return the result.
- SET_REG - set the value of a register on the switch.
- LOOKUP - lookup the given key in a table on the switch and return the result.

The following packet header will be used in this exercise:

```
header Calc_h {
bit<32> op1; //First operand
bit<8> opCode; //Operation to perform
bit<32> op2; //Second operand
bit<32> result; //Result, optional
}
```

The Calc_h header comes after the Ethernet header.
Once the switch has calculated the result, the packet is returned *to the sender.* This means that the source and destination MAC addresses need to be swapped.

1. Make sure to use the P4-NetFPGA flow:

   ```
   vim ~/.bashrc
   ```

   Go to line 103. change the line to:

   ```
   source /opt/Xilinx/Vivado/2018.2/settings64.sh
   ```

   Exit the ssh connection and connect again to the machine. Note that sourcing bashrc again (without exiting current session) will lead to problems.

2. Change the project's Makefile to point to the correct P4 source:

   ```
   cd $P4_PROJECT_DIR/src
   vim Makefile
   ```

   In line #39, change $P4_PROJECT_NAME_solution.p4 to $P4_PROJECT_NAME.p4

3. Write your code:

   ```
   vim switch_calc.p4
   ```

4. If needed, add entries to the tables you defined in the P4 program:

   ```
   cd $P4_PROJECT_DIR/src
   vim commands.txt
   ```

5. Run the P4-SDNet compiler to generate the resulting HDL and an initial simulation framework:

   ```
   cd $P4_PROJECT_DIR && make
   ```

6. Run a simulation of your code in SDNet:

   ```
   cd $P4_PROJECT_DIR/nf_sume_sdnet_ip/SimpleSumeSwitch
   ./vivado_sim.bash
   ```

   The following describes the steps for running a simulation of the P4 code in the SDNet environment, with GUI:

   ```
   cd $P4_PROJECT_DIR/nf_sume_sdnet_ip/SimpleSumeSwitch
   ./vivado_sim_waveform.bash
   ```

7. Generate the scripts that configure the table entries :

   ```
   cd $P4_PROJECT_DIR && make config_writes
   ```

8. Wrap SDNet output in wrapper module and install as a NetFPGA library core:

   ```
   cd $P4_PROJECT_DIR
   make uninstall_sdnet && make install_sdnet
   ```

9. Set up the simulation environment:

   ```
   cd $NF_DESIGN_DIR/test/sim_switch_default
   make
   ```

   The test code can be viewed and edited in run.py.

10. Run the simulation:

    ```
    cd $SUME_FOLDER/tools/scripts/
    ./nf_test.py sim --major switch --minor default
    ```

To run a simulation with GUI (Vivado xsim), replace the last stage with:

```
cd $SUME_FOLDER/tools/scripts
./nf_test.py sim --major switch --minor default --gui
```

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

11. Run a simulation of registers access:

    ```
    cd $NF_DESIGN_DIR/test/sim_switch_ctrlWrites && make
    cd $SUME_FOLDER/tools/scripts
    ./nf_test.py sim --major switch --minor ctrlWrites
    ```

3 Building a project

The following steps are typically required when building a project. We recommend that you follow them to test your design on the hardware:

1. 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.

2. 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.

3. Compiling all cores and building libraries (excluding CAM):

   ```
   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.

4. Building a project:

   ```
   cd $NF_DESIGN_DIR
   make
   ```

   The result of this step is the programming (bit) file. *This step takes 45 minutes or more. Do not run it during class.*

5. Follow the instructions for hardware testing from Lab 1.

4 Common Issues

- Problem: Vivado's GUI does not open.
  Solution: 1) make sure to ssh using -X.
  2) edit ~/.bashrc. comment the line *export DISPLAY=:0*.

- Problem: Receiving extra or unexpected packets in the hardware test.
  Solution: Try resetting the machine, this is sometimes a problem in Scapy. If the problem persists, this is likely as the network manager is not disabled and/or the network is not configured properly. Follow the steps in `https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/Reference-Operating-System-Setup-Guide`, under the *Network Configuration Manager* section. In particular, make sure that your NIC's MAC address appears on the list of non managed interfaces (/etc/NetworkManager/NetworkManager.conf) and that **all** your NIC's host interfaces are set to manual (/etc/network/interfaces).

5 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/`
- P4-NetFPGA Repository: `https://github.com/NetFPGA/P4-NetFPGA-live`
- P4-NetFPGA Wiki: `https://github.com/NetFPGA/P4-NetFPGA-public/wiki`
- P4-NetFPGA registration page: `https://goo.gl/forms/h7RbYmKZL7H4EaUf1`
- P4-NetFPGA online tutorial: `https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Tutorial-Assignments`