EE533 Laboratory #7
NetFPGA Integration in DETER

Young Cho - youngcho@isi.edu

1) Using the NetFPGA with DETER

In this lab we will use the NetFPGA as a network interface card and as a router. You'll get familiar with the tools and steps necessary to use the NetFPGA with DETER. This is a team laboratory.

- Create an experiment with the NS file give as .ns file content at the end of this document
- Login to `control.<EXP-NAME>.USCEE533.isi.deterlab.net` where `<EXP-NAME>` is the name you gave the experiment in step 1.
- The CentOS images on DETER have all the necessary tools installed to use the reference designs with the NetFPGA. There are 10 machines with NetFPGAs on DETER. You'll notice in the NS file that the NetFPGA is treated as a router and the computer containing the NetFPGA is called the parent, named control in our NS file. However, due to the split nature of the NetFPGA machines, the NetFPGA Ethernet interfaces (called nf2c[0,1,2,3]) are not automatically configured. Also, the DETER software does not always map the interfaces in the same order, so we must do some detective work to get started. Take a look at the e-mail you received from DETER when your experiment was running, here is the important snippet:

```
Physical Lan/Link Mapping:
ID          Member          IP             MAC                  NodeID
----------- --------------- -------------- -------------------- --------
lan0        n0:0            10.1.0.3       00:15:17:57:c8:aa    pc024
                                           2/1 <-> 7/47         Cisco4
lan0        nfrouter-1:0    10.1.0.2       00:4E:46:32:43:03    nf067
                                           3/0 <-> 9/4          Cisco4
lan1        n1:0            10.1.1.3       00:15:17:57:c6:ca    pc039
                                           2/1 <-> 1/11         Cisco4
lan1        nfrouter-1:1    10.1.1.2       00:4E:46:32:43:02    nf067
                                           2/0 <-> 9/3          Cisco4
lan2        n2:0            10.1.2.3       00:15:17:57:c6:f2    pc010
                                           2/1 <-> 2/39         Cisco4
lan2        nfrouter-1:2    10.1.2.2       00:4E:46:32:43:01    nf067
                                           1/0 <-> 9/2          Cisco4
lan3        n3:0            10.1.3.3       00:15:17:57:c3:96    pc023
                                           2/1 <-> 7/43         Cisco4
lan3        nfrouter-1:3    10.1.3.2       00:4E:46:32:43:00    nf067
                                           0/0 <-> 9/1          Cisco4
```

- The lines we are interested contain `nfrouter-1:X`. These lines show the mapping between the needed IP address and the MAC address of the particular NetFPGA ethernet port (of which there are 4). In looking at the MAC address, the last octet corresponds to the port number, i.e. 00 is interface nf2c0 and 01 is nf2c1 and so on. Look at your e-mail, figure out the mapping between IP address and the nf2cX and then write this down.

- Now, back on the control node run the following command:

```
$> sudo /usr/local/netfpga/lib/scripts/cpci_reprogram/cpci_reprogram.pl --all
```

This command is important, and MUST be run after every reboot of the computer containing the NetFPGA.

- Now, configure the IP address for all of your NetFPGA interfaces, for the example above (i.e. the number “03” in MAC address “00:4E:46:32:43:03” for lan0 corresponds to nf2c3 and that MAC should have IP address “10.1.0.2” assigned to it.

  ```
  $> sudo ifconfig nf2c3 10.1.0.2/24 up
  ```

  Do this for all four ports (with the corresponding IP addresses).

  At this point you shouldn't be able to ping any of the nodes from the control PC, nor should any of the nodes be able to ping each other. Test this. This is because we've reset the NetFPGA and there is no bitfile loaded, so nothing happens.

- Run the following command:

  ```
  $> nf_download /usr/local/netfpga/bitfiles/reference_nic.bit
  ```

  You should see output similar to below:

  ```
  Found net device: nf2c0
  Bit file built from: nf2_top_par.ncd;HW_TIMEOUT=FALSE
  Part: 2vp50ff1152
  Date: 2010/ 7/26
  Time: 17:29:31
  Error Registers: 0
  Good, after resetting programming interface the FIFO is empty
  Download completed - 2377668 bytes. (expected -1).
  DONE went high - chip has been successfully programmed.
  CPCI Information
  ----------------
  Version: 4 (rev 1)
  Device (Virtex) Information
  ---------------------------
  Project directory: reference_nic
  Project name: Reference NIC
  Project description: Reference NIC
  Device ID: 1
  Version: 1.1.0
  Built against CPCI version: 4 (rev 1)
  Virtex design compiled against active CPCI version
  ```

  Now we should be able to ping all-around. Ensure that you can ping all of the nodes from one another. Right now we are still using the Linux kernel as a router and packets must traverse through the NetFPGA, over the PCI bus and then back again before being delivered out the proper port. On n0 and n1 we need to increase the TCP window sizes to increase performance.

- On n0 run the following commands:

  ```
  $> sudo su - (to get a root shell)
  root@n0: # echo 1048576 > /proc/sys/net/core/rmem max
  root@n0: # echo 1048576 > /proc/sys/net/core/wmem max && exit
  Then on n1, run the following commands:
  $> sudo su - (to get a root shell)
  root@n1: # echo 2097152 > /proc/sys/net/core/wmem max
  ```

```
root@n1: # echo 2097152 > /proc/sys/net/core/rmem max && exit
```

- Explain in your report what these commands do. How should they help, why do we need them? Now use iperf in TCP mode with the server on n1 and the client on n0 to measure the TCP throughput. How much bandwidth do you get? Does this seem low?
- Now we will load the reference router into the NetFPGA. What does the reference router do? How should this help? Run the following command on the control node:

  ```
  $> nf download /usr/local/netfpga/bitfiles/reference_router.bit
  ```

  The NetFPGA now needs to know about the routes and IP addresses configured so it can accelerate the routing process. There is a tool to do this for you, it will load all of the needed information from the Linux kernel into the routing table of the NetFPGA. Run the following command on the control node:

  ```
  $> /usr/local/netfpga/projects/router_kit/sw/rkd &
  ```

  All you should see is something like: [1] 3782. The ampersand at the end of the command places the program in the background immediately. The number is the PID of the rkd program, in case you need to kill it.

- Now re-run the TCP iperf test. Has the bandwidth changed? If so, why?
- Let's explore the routing performance a little more. Go back to the reference nic design:

  ```
  $> sudo killall rkd
  $> nf download /usr/local/netfpga/bitfiles/reference_nic.bit
  ```

- Launch an iperf server in UDP mode on each node. Write a script that will run on users.isi.deterlab.net that will let you launch an iperf client on each node such that each port on the NetFPGA has about 1Gbit of traffic in each direction. Write the script so it launches the client in the background so each client starts more-or-less simultaneously. Use 512 byte packets. After you get it working, use at least 30 seconds per test.
- Make sure to run the test several times and take the average results. Look at the output of the servers to see how much performance you are able to attain. What is the total bandwidth you are able to observe through the NetFPGA? Why does using small packets stress the system?
- Load the reference router.bit file again and start the rkd daemon. Run the same iperf test. What do you observe? Is it fair to say that the NetFPGA can route IP traffic bi-directionally at line-speed for a total of 4Gbps of cross-wise bandwidth?
- Now we want to make the NetFPGA easier to use. Look at the documentation for the tb-set-node-startcmd and write a script that performs the following actions: (store it in your directory on users.isi.deterlab.net so it is accessible when the node is started)

  1) Parse the tbreport.log file in /proj/USCEE533/exp/<EXP-NAME>/tbdata to determine the NetFPGA port to IP address mapping
  2) Run the required ifconfig commands to configure the nf2cX interfaces
  3) Run the cpci reprogram.pl script
  4) Download the reference nic.bit file to the NetFPGA

- You should test the script by swapping out and then swapping in the experiment. If your script runs properly the reference NIC should be running and all nodes should be able to ping at the outset.

2) Synthesizing and Testing Custom NetFPGA bitfiles

- One of the first thing to do is insert a passthrough ids.v into the NetFPGA reference NIC or Router package that simply connects all of the modules' input to output with "wire" type variable.
- Compile the source as you did in Lab 6 and generate the bitfile
- Go through the above steps to make sure that the resulting bitfile is functionally identical.
- Then modify the source codes and xml files to add SW/HW registers.
- In the lab6_mini_ids_src.zip file you'll find a Perl script called idsreg. Copy this file to the control node and make sure it is executable.
- Examine the script to interpret what it is doing.
- The script simply constructs register bits out of what a developer might assign as commands and data. Then the content of register is transferred to NetFPGA to be interpreted according to the developer's rules by the hardware design.
- Once you come to understand what the script is doing, modify it according to your needs.
- One of the first thing you will need to do is to change the address in the script to point to the netfpga compiler assigned register address(es) found in generated files in lib folder.
- Modify the script to write and read to/from your registers
- One the simple design might be write an 8-byte text into a mapped "write" register and wire up the hardware to reverse the order of the bytes and write the new 8-byte value into another mapped "read" register that you can read using the script.
- Recompile the design and test to make sure that interface is working as intended.
- I would also recommend that you also develop, simulate, integrate, then test a logic analyzer built using a block RAM. You should be able to use your register interface to control the function of the logic analyzer to record any internal signals and dump it out to the server using the script.
- After this, you should incrementally add the components from your mini-IDS design little bit at a time to compile and test proper working of your design.

3) Synthesizing, Testing, and Debugging the mini-IDS

- Generate a bit-file for mini-IDS to use on DETER using the VM.
- Use the .ns file in the appendix to create an experiment. Instead of loading the reference router, load your custom generated bit-file using nf_download
- Make sure the interfaces are configured properly and start rkd. Make sure the nodes can ping each other.
- Using your idsreg, you might want to try the following. In order for these commands to work correctly, you may have to change the script to support your underlying hardware design.

```
Run $>idsreg reset
Run $>idsreg pattern ABCDEFG
## where ABCDEFG might be a 7 character string of your choice.
Run $>idsreg matches You should see count = 0x00000000
```

- Start an iperf server on each node.
- Start 3 iperf clients on each node. The clients should send their packets to the other nodes. For two of the iperf clients on each node send "good" packets (i.e. the packets do not contain the string). On one iperf client use the -I option to include the string you chose above. This will create "bad" packets.
- You should observe that the packets from your "bad" iperf clients do not get through. Observe the difference between TCP and UDP mode. In TCP mode your "bad" client should connect, but no data packets should arrive. In UDP mode, no packets should reach the server from the "bad"

client. Use tcpdump to verify this fact. Include a snippet to show that each server is only receiving packets from two clients.

- On the control node run $>idsreg matches and you should see the count of packets dropped.

4) Submission and Demonstration

Answer the following questions in your report:

1. There is a bug when the mini-IDS is passing traffic at near gigabit speeds. What is the bug?
2. Explain in brief how the mini-IDS works and how it interacts with the other modules in the NetFPGA. Be **clear and concise!** Write as if you are describing the mini-IDS project on a webpage where other NetFPGA developers will read about your design.
3. Explain the pattern matching algorithm.
4. Draw a high-level design of the user_data_path.v and ids.v Verilog files. The figure should depict the communication between the components.
5. A major part of the assignment will be in demonstration of your system. Please be complete in going through your design in your youtube demo video.

Please include the following in your report:

- Schematics
- Generated Verilog

Appendix: Lab #8 NS file

```
# EE533 Lab 7 NS File . Creates a network with 4 nodes and a NetFPGA host.
source tb_compat.tcl
set ns [ new Simulator ]

set nfrouter [ $ns node ]
tb-set-hardware $nfrouter netfpga2

set control [ $ns node ]
tb-bind-parent $nfrouter $control

# Create end nodes
set n0 [ $ns node ]
set n1 [ $ns node ]
set n2 [ $ns node ]
set n3 [ $ns node ]

# Put all the nodes in a lan
set lan0 [ $ns make-lan " $nfrouter $n0" 1000Mb 0ms ]
tb-set-ip-lan $n0 $lan0 10.1.0.3

set lan1 [ $ns make-lan " $nfrouter $n1" 1000Mb 0ms ]
tb-set-ip-lan $n1 $lan1 10.1.1.3

set lan2 [ $ns make-lan " $nfrouter $n2" 1000Mb 0ms ]
tb-set-ip-lan $n2 $lan2 10.1.2.3

set lan3 [ $ns make-lan " $nfrouter $n3" 1000Mb 0ms ]
tb-set-ip-lan $n3 $lan3 10.1.3.3

$ns rtproto Static
$ns run
```