

EE533 Laboratory #4
NetFPGA Bitfile Generation and Using NetFPGA
Young Cho - youngcho@isi.edu

Please keep the following in mind for your lab submissions

- a) Capture the video of the screen (using free tools like OBS studio) during this process for edited demo video that will be uploaded on youtube
<https://obsproject.com/>
- b) You can extract JPEG from video to use in your report and presentation slides
<https://www.dvdvideosoft.com/guides/free-video-to-jpg-converter.htm>

1. Download and Set up NetFPGA Tool Virtual Machine

- a) Download NetFPGA compiler virtual machine: Fedora 14.ova
<https://drive.google.com/file/d/1IDiZ1afC3v949XWw7ytfwOsu3vyfB4U7/view?usp=sharing>
- b) Start the Virtual Machine Fedora 14
- c) Go to Application -> System Tools -> Terminal
- d) In the terminal, enter 'su' to switch into root user to compile the design into a bitfile

```
[student@fedora ~]$ su  
Password: fighton
```

- e) Change directory into the NetFPGA projects directory with 'cd' command

```
[root@fedora student]# cd /root/netfpga/projects/
```

- f) Copy the reference_router project into your project folder with 'cp' command

```
[root@fedora student]# cp -rf reference_router lab4
```

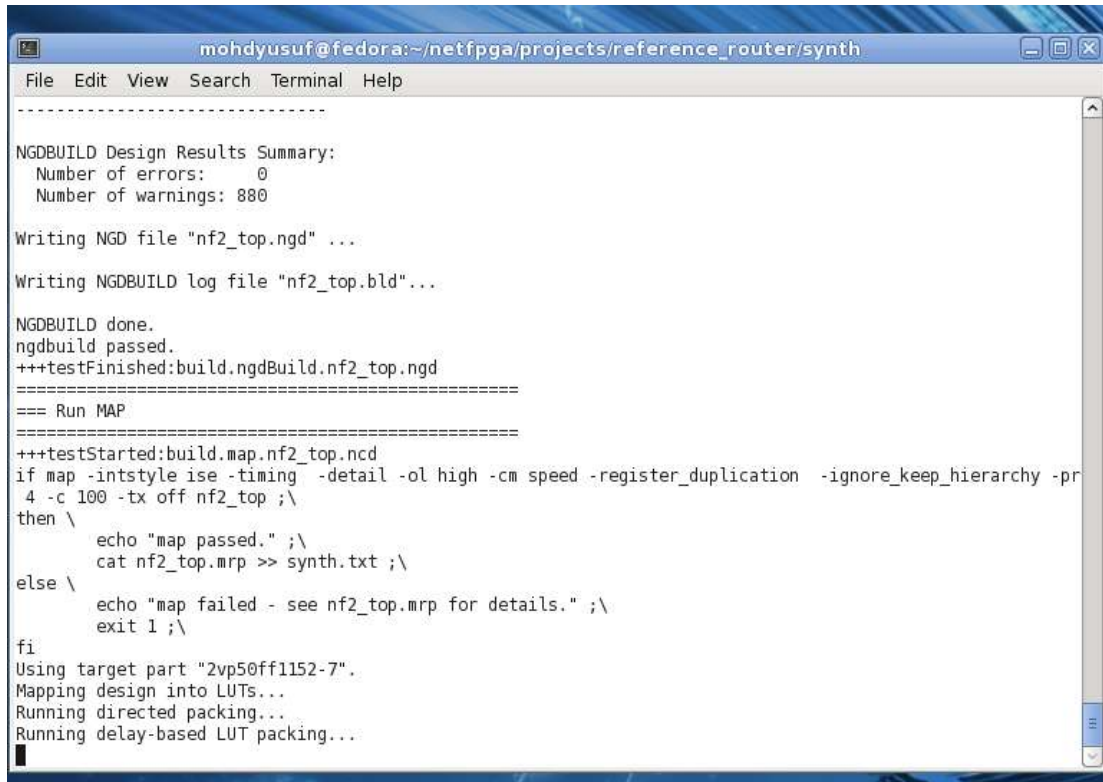
- g) Change directory into a synthesis folder of your project with 'cd' command

```
[root@fedora student]# cd lab4/synth
```

- h) Compile the design by typing 'make'

```
[root@fedora student]# make
```

- i) This should start the compilation and generation of a bitfile for your reference_router project



```
mohdyusuf@fedora:~/netfpga/projects/reference_router/synth
File Edit View Search Terminal Help

-----
NGDBUILD Design Results Summary:
  Number of errors:    0
  Number of warnings: 880

Writing NGD file "nf2_top.ngd" ...

Writing NGDBUILD log file "nf2_top.bld"...

NGDBUILD done.
ngdbuild passed.
+++testFinished:build.ngdBuild.nf2_top.ngd
=====
=== Run MAP
=====
+++testStarted:build.map.nf2_top.ncd
if map -intstyle ise -timing -detail -ol high -cm speed -register_duplication -ignore_keep_hierarchy -pr
  4 -c 100 -tx off nf2_top ;\
then \
  echo "map passed." ;\
  cat nf2_top.mrp >> synth.txt ;\
else \
  echo "map failed - see nf2_top.mrp for details." ;\
  exit 1 ;\
fi
Using target part "2vp50ff1152-7".
Mapping design into LUTs...
Running directed packing...
Running delay-based LUT packing...
```

2. Compile and generate a design bitfile for NetFPGA

- a) Convert Schematics to Verilog code as demonstrated in the following video
http://oasysresearch.com/ise_sch2ver.mkv
- b) Extract ids_hw from lab3_mini_ids_src.zip
- c) Copy all your converted Verilog (*.v) and ip core files (*.xco) into the ids_hw/src
- d) In the NetFPGA project folder in Fedora VM
- e) Copy all of source files within ids_hw into corresponding folders into your NetFPGA project folder in Fedora VM
 - i. Folders 'include' and 'sw' should already be there if you are using a copy of reference_router project
 - ii. Folder 'src' should be created
- f) Then compile the design by 'cd' into the synth folder type 'make'
- g) The make command should generate "*.bit" file which would be your initial Mini-IDS bitfile
- h) Potential problems include broken designs, misplaced files, missing files, poor design with too long critical path

3. Set Up VPN to USC

- a) In order to 'SSH' into NetFPGA machines from your computer, you must either use USC's VPN from off-campus locations or a Secured Wireless connection on campus. One way or another, you must sign in to the campus network through a secured network channel.
- b) To enable VPN on your computer, follow the instructions on the website (<https://itservices.usc.edu/vpn/>) to install Cisco's AnyConnect client software for your Operating System, then sign into USC's VPN network with your username and password.
- c) If VPN is unstable type the following in your local terminal.

```
> sudo apt-get remove network-manager-config-connectivity-ubuntu
```

4. NetFPGA Environment

- a) The first thing you must do is configure the openssh client of your local Linux system to use legacy encryption to allow ssh connection with the NetFPGA servers. The following commands assume that you are using Ubuntu. Other Linux versions may require different commands to do the same. You can search the Internet for the equivalent commands.
- b) Install openssh and nano, if not already installed.

```
> sudo apt-get install openssh-client nano
```

- c) Edit ssh client configuration file using nano editor

```
> sudo nano /etc/ssh/ssh_config
```

- d) In the editor, add the following text at the end of the file then save by typing Control-O. You can learn more about how to use nano from the Internet.

```
HostkeyAlgorithms ssh-dss,ssh-rsa
KexAlgorithms +diffie-hellman-group1-sha1
StrictHostKeyChecking=accept-new
```

- e) Once the file is saved, download the attached bash script “openterm” into your account in your local Linux system.

- f) Then you need to make it an executable file by typing:

```
> chmod a+x openterm
```

- g) You may want to take a look at the script to figure out what it is doing. If not already installed in the system, install gnome-terminal and sshpass.

```
> sudo apt-get install gnome-terminal sshpass
```

- h) Once your VPN connection to vpn.usc.edu is enabled, you execute openterm with your team number and the corresponding password to automatically spawn five terminal windows for your NetFPGA node, and 4 nodes that are connected to NetFPGA.

```
> openterm <TEAM #> <TEAM_PASSWORD>
```

- i) Once you are on the remote servers, you should be able to examine the network configuration. One of the most obvious things that you should find out are IP addresses. You can use the network tools like /sbin/ifconfig to see but please do not attempt to change those settings using the tool since all of the network settings including IP addresses and routes are manually set. Type the following to see the IP addresses for the nodes 0 through 3.

```
> echo $n0
> echo $n1
> echo $n2
> echo $n3
```

- j) When the servers are restarted, the NetFPGAs are initially configured as four input Gigabit Ethernet NICs. One Intel Gigabit Ethernet port from four nodes is connected to these ports.

These NetFPGA ports are assigned IP addresses that belong to the same subnet as the above IP addresses of the nodes. The difference is the last digit of the IP address is 2 instead of 3.

5. NetFPGA-based Linux Kernel IP Router

- a) In this section, we will test the NetFPGA as a network interface card. You'll get familiar with the tools and steps necessary to use the NetFPGA.
- b) On FPGA node, run the following command:

```
> nf_download /home/netfpga/bitfiles/reference_nic.bit
```

- c) 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
```

- d) IP forwarding function of Linux is enabled during the boot. Therefore, your NetFPGA-based reference NIC with Linux IP forwarding will make your server act like a software IP router. Therefore, you should be able to ping all-around.
- e) From n0, you may ping n3 by typing the following in n0 terminal:

```
> ping $n3
```

- f) You may also type the IP address instead of the variable. If your IP address for n3 is 10.0.2.3 then typing the following will do the same task.

```
> ping 10.0.2.3
```

- g) Ensure that you can ping all of the nodes from one another. However, with the Linux kernel as a router, the packets must traverse through the NetFPGA to the CPU over PCI bus and network software stack then back out to the BUS and NetFPGA before being delivered out the destination node.
- h) Now, test the network with a network testing tool called iperf. Learn to configure and use iperf to test the network performance from the Internet. One of the functions that you need to learn is

to use iperf to transmit packets containing specific string patterns and files in their payloads. This function should be used to test your Mini-IDS design.

<https://www.baeldung.com/linux/iperf-measure-network-performance>

- i) Set different nodes as iperf servers and clients to test the network performance. You should use scripts to automate these tests. Collect and tabulate the bandwidth of the network connections and draw your conclusions that you will include in the report.

6. NetFPGA Hardware IP Router

- a) In this section, we will test the NetFPGA as a hardware IP router.
- b) Load the reference router into the NetFPGA. What does the reference router do? How should this help? Run the following command on the fpga node:

```
> nf_download /home/netfpga/bitfiles/reference_router.bit
```

- c) The NetFPGA now needs to know about the routes and IP addresses configured so it can accelerate the routing process. The software calls RKD 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 FPGA node.

```
> rkd &
```

- d) 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.
- e) Now re-run iperf tests. Has the bandwidth changed? If so, why?
- f) 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
```

- g) Launch an iperf server in UDP mode on each node. Write a script 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.
- h) 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?
- i) 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?

7. Initial Attempt in Synthesizing and Testing Your Design

- a) One of the first things 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 a "wire" type variable. Compile the source as you did in Lab 3 and generate the bit file. Then, download the generated bit file into the FPGA node using Linux command line tools like scp. Reconfigure the

FPGA with the generated bitfile and make sure that it is functionally the same as the reference bitfiles.

- b) I recommend that you also develop, simulate, integrate, and 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 them out to the server using the script.
- c) After this, you should incrementally add the components from your mini-IDS design a little bit at a time to compile and test the proper working of your design.
- d) 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.
- e) In the `lab3_mini_ids_src.zip` file you'll find a Perl script called `idsreg`. Copy this file to the `fpga` node and make sure it is executable (you can use `scp` command to copy file from your computer to the nodes).
- f) 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.
- g) 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
- h) 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.
- i) 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.

```
> idsreg reset  
> idsreg pattern ABCDEFG
```

- j) ABCDEFG might be a 7 character string of your choice.

```
> idsreg matches
```

- k) Start an `iperf` server on each node. Start three `iperf` clients on each node. The clients should send their packets to the other nodes. 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.
- l) 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 FPGA node, run `idsreg matches`, and you should see the count of packets dropped.
- m) You will unlikely get this to work by the end of this week. But do the best that you CAN. The goal of the next lab is to get it to work completely and perform various tests.

8. Submit your lab report

- a) Include screenshots of the process
- b) Include detailed GitHub history and descriptions for each members of the team.
- c) Give sufficient evidence that you completed all the tasks successfully.