EE533: Network Processor Design & Programming

Lab #4: NetFPGA BitFile Generation and Using NetFPGA

Instructor: Prof. Young Cho, PhD

Team Number: **#3**

Project Partners:

Member #1: **Sarthak Jain**

Member #2: **Archit Sethi**

Member #3: **Justin Santos**

Designed, Created, and Submitted by Team #3

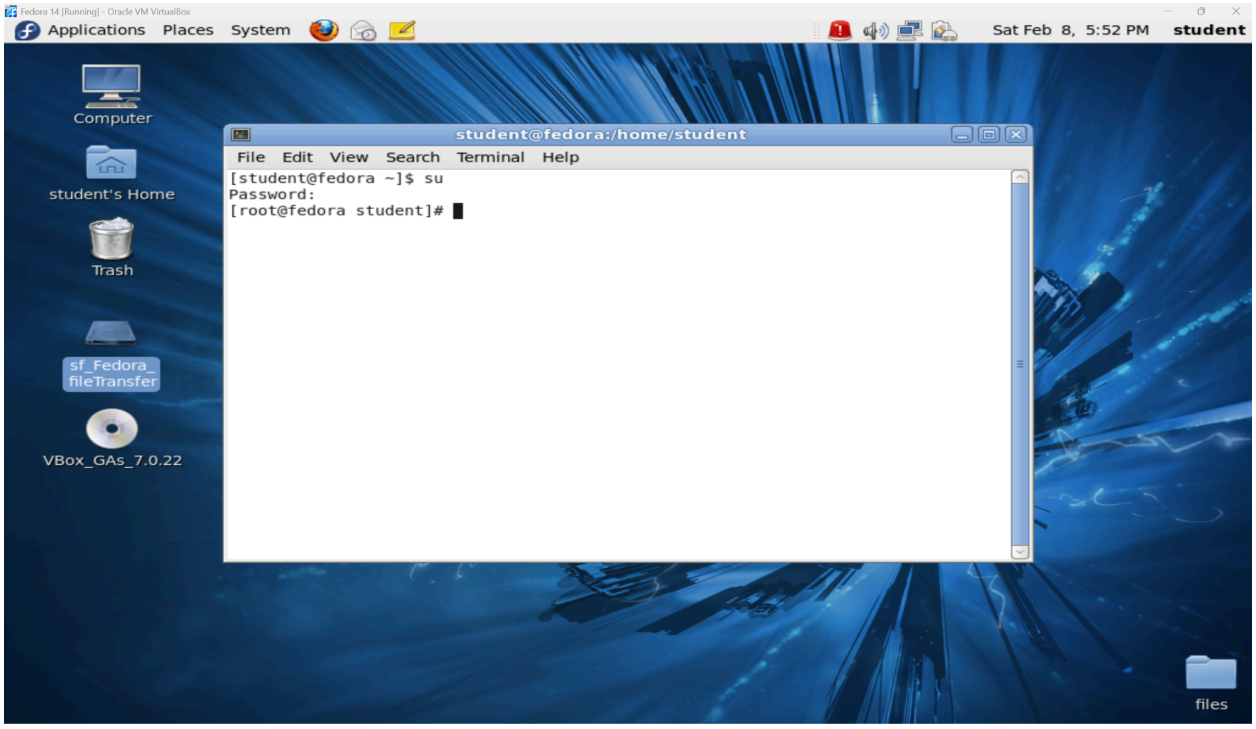
University of Southern California

Los Angeles, CA 90007

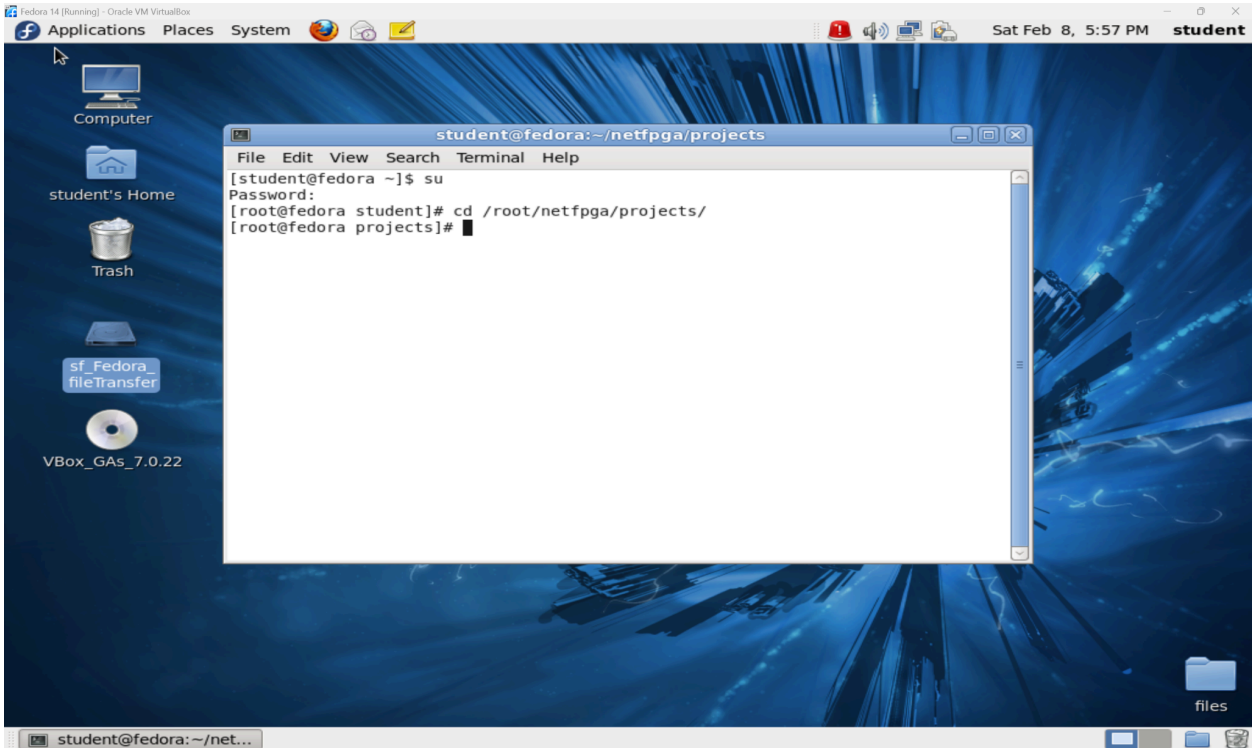
Team #4 GitHub Repository Link: *Team#3 Link*

Part-1: Download and Set-Up NetFPGA Tool Virtual Machine

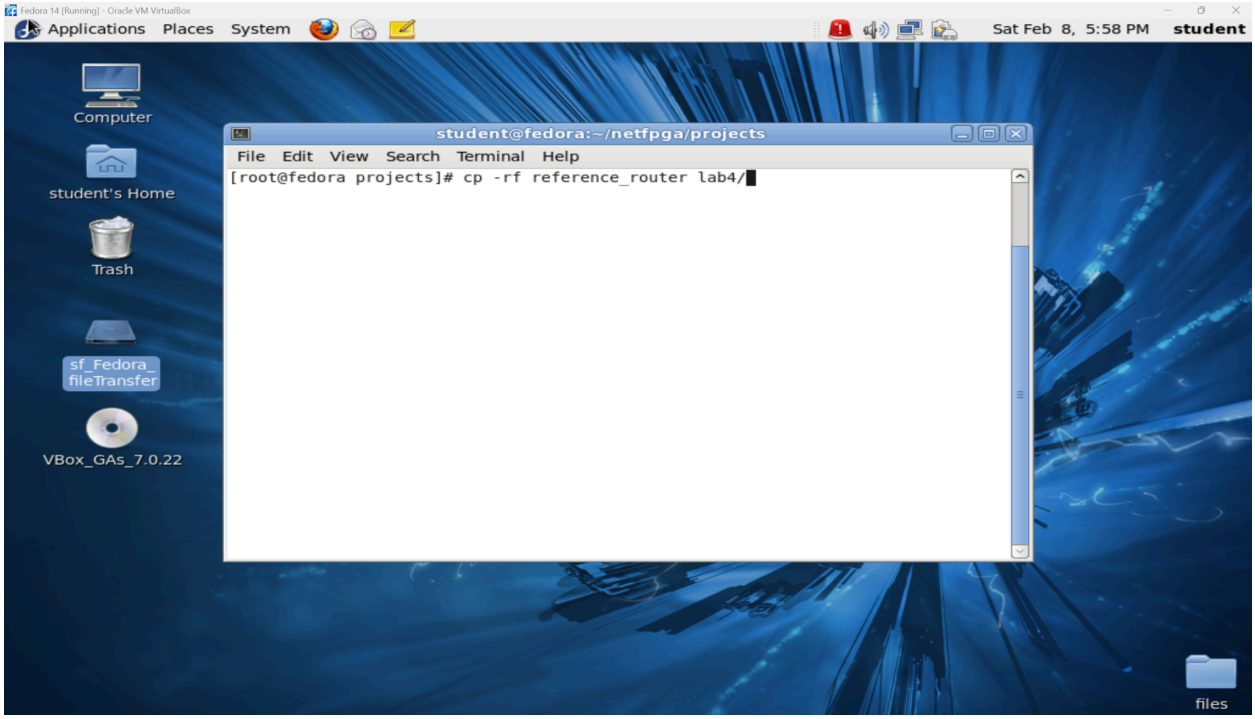
1. Virtual Machine Fedora 14 was successfully downloaded and installed on the VirtualBox software.
2. In the terminal, enter ‘su’ to switch into root user to compile the design into a bitfile.



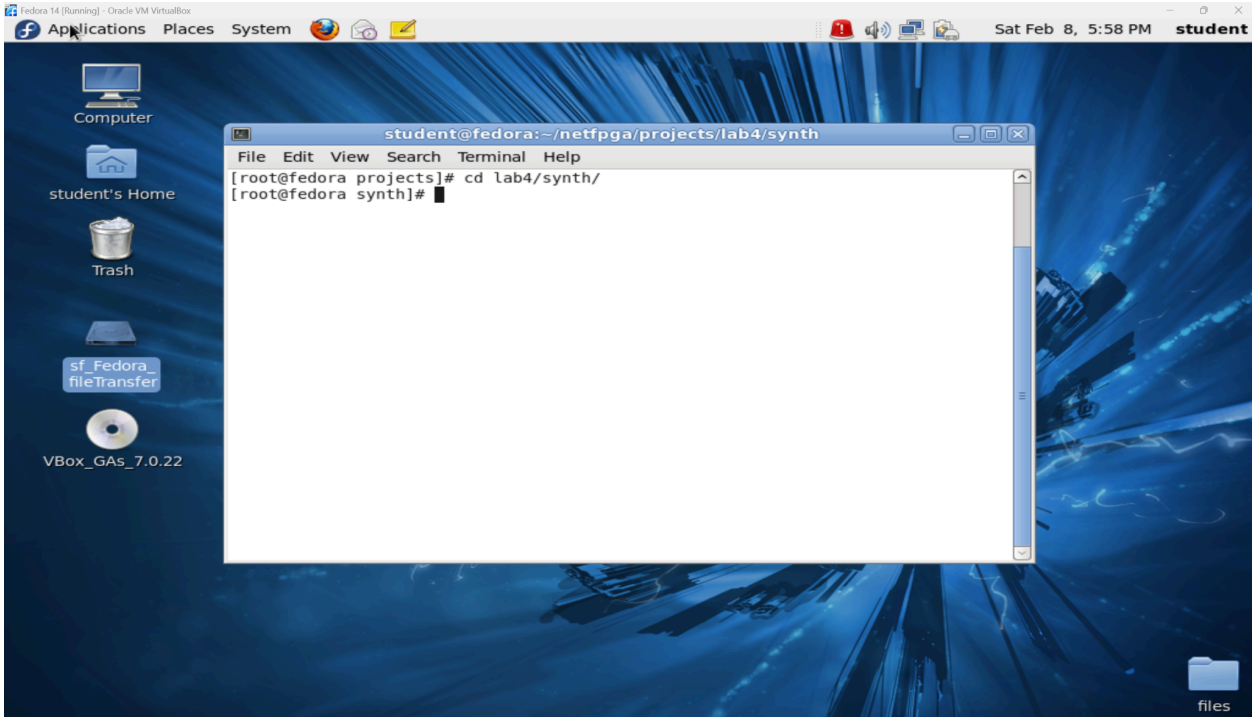
3. Change directory into the NetFPGA projects directory with ‘cd’ command.



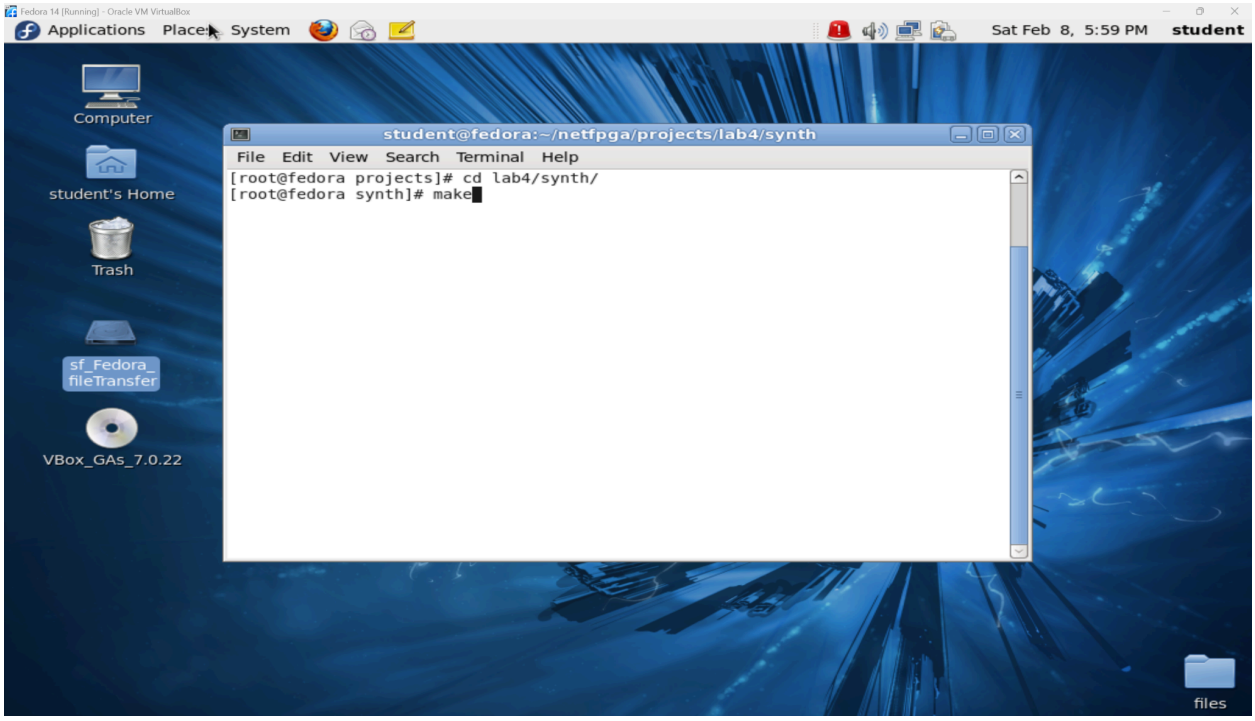
4. Copy the reference_router project into your project folder with ‘cp’ command



5. Change directory into a synthesis folder into your project with ‘cd’ command



6. Compile the design by typing ‘make’

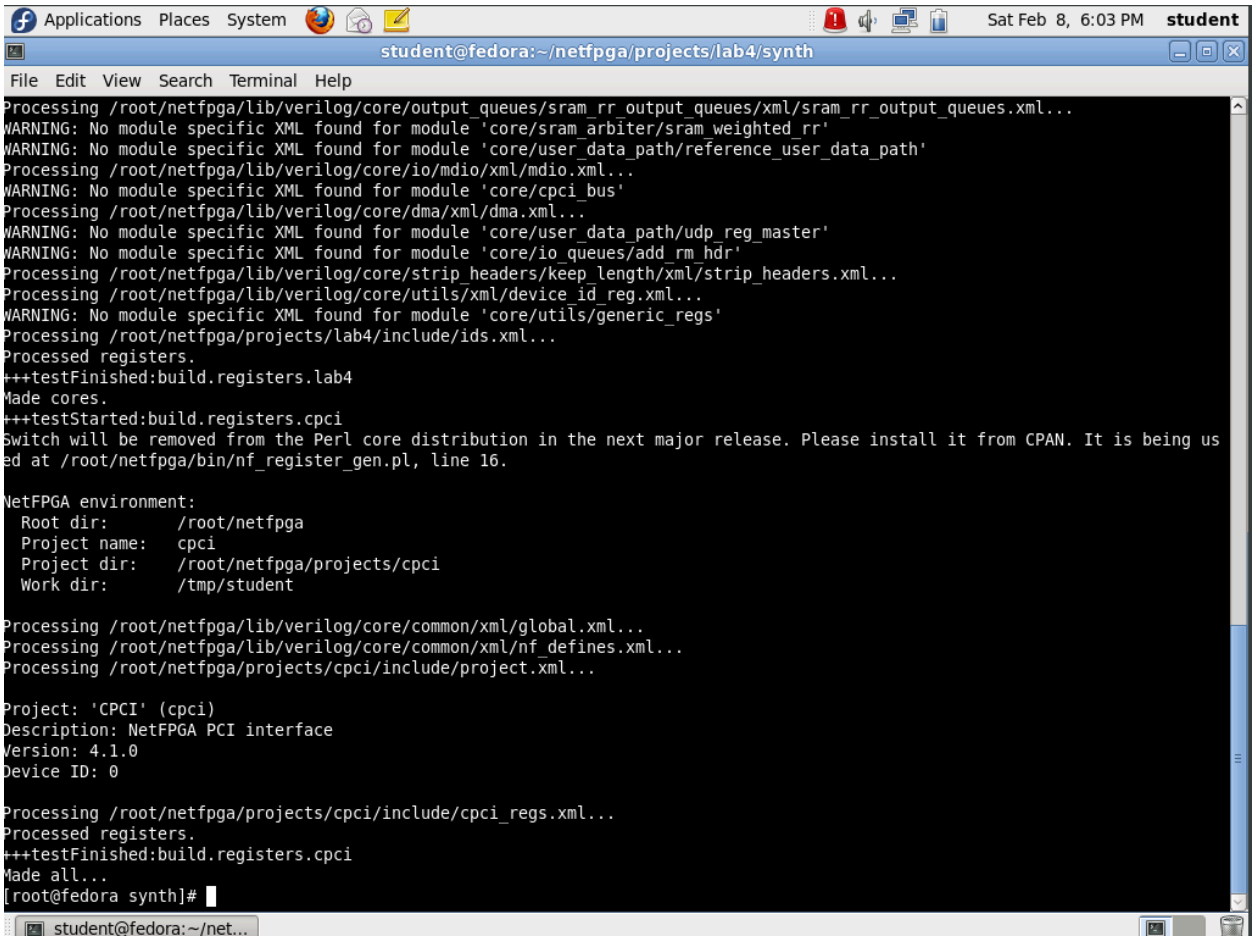


7. This should start the compilation and generation of a bit-file for your reference_router project

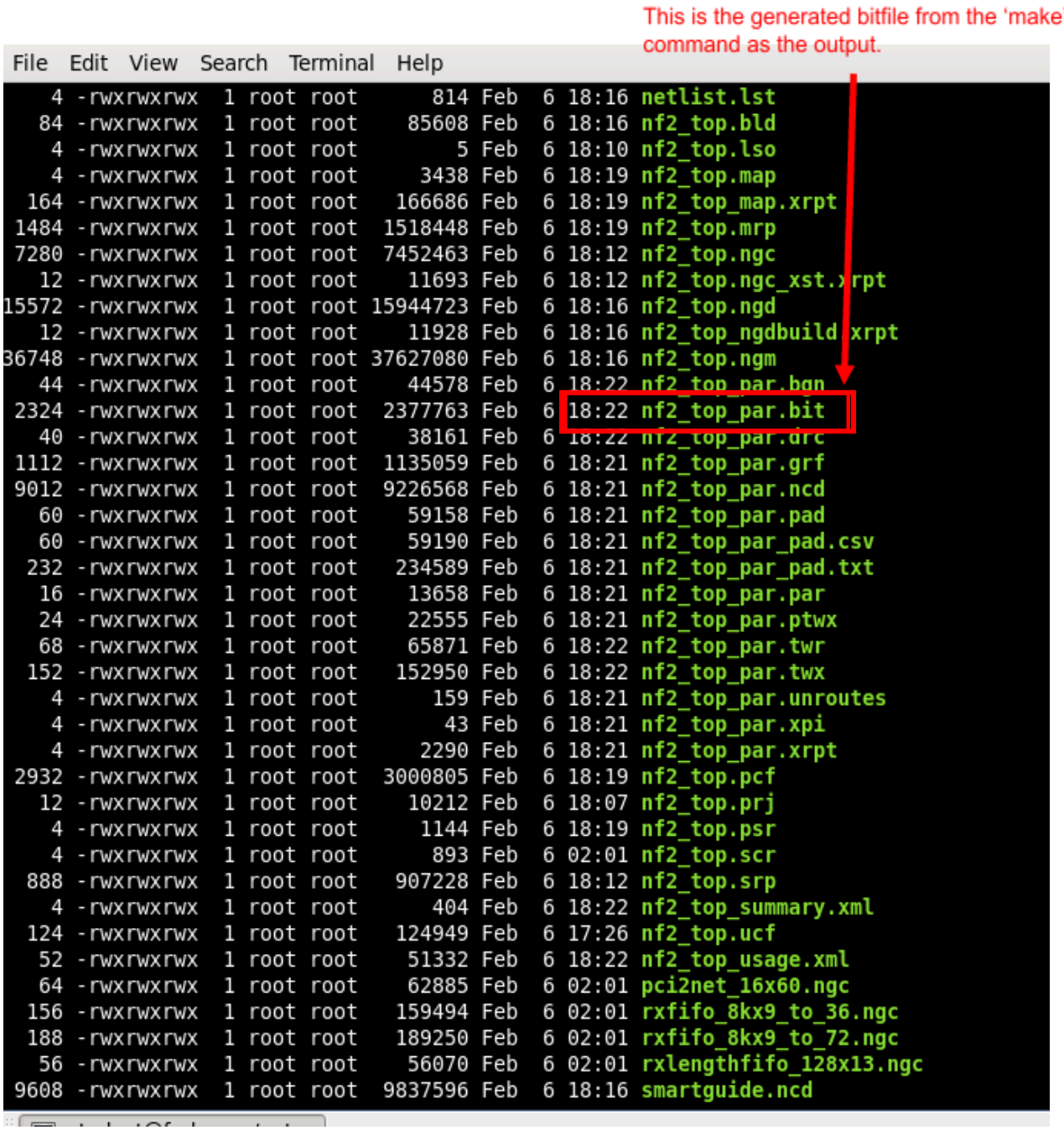
   The command ‘make’ was running and we found that there were many errors from the imported schematics from the previous lab-3. The debugging was a tedious process but was handled with efficiency.

Part-2: Compile and generate a design bitfile for NetFPGA

1. Converting schematics to verilog code as it was demonstrated in the following video.
2. After extraction of ‘ids_hw’ from lab-3_mini_ids_src.zip
3. Copied all converted verilog and ip core files into ids_hw/src.
4. This is how the make command ran and we got a few errors which have been depicted further below.



From the figure below, you can clearly see that we were able to successfully generate the bitfile. The mentioned bitfile that was generated was ***‘nf2_top_par.bit”***



When we used the 'ls' command we were able to clearly see our generated bitfile in the lab4/synth/folder. See the snippet below for details.

```
File  Edit  View  Search  Terminal  Help
[root@fedora synth]# ls
async_fifo_256x72_to_36.ngc              nf2_top.map            nf2_top_par.twr          srl_cam_32x32.ngc
async_fifo_512x36_progfull_500.ngc       nf2_top_map.xrpt       nf2_top_par.twx          srl_cam_64x32.ngc
async_fifo_512x36_to_72_progfull_500.ngc nf2_top.mrp            nf2_top_par.unroutes     srl_cam_unencoded_32x32.ngc
bram_cam_32x32.ngc                       nf2_top.ngc            nf2_top_par.xpi          syncfifo_512x32.ngc
bram_cam_64x32.ngc                       nf2_top.ngc_xst.xrpt   nf2_top_par.xrpt         syncfifo_512x36_fallthrough.ngc
bram_cam_unencoded_32x32.ngc             nf2_top.ngd            nf2_top.pcf              syncfifo_512x36.ngc
cam_16x32.ngc                            nf2_top_ngdbuild.xrpt  nf2_top.prj              syncfifo_512x72.ngc
cdq_rx_fifo_512x36.ngc                   nf2_top.ngm            nf2_top.psr              synth.txt
cdq_rx_fifo_512x36_to_72.ngc             nf2_top_par.bgn        nf2_top.scr              transcript
cdq_tx_fifo_256x72_to_36.ngc             nf2_top_par.bit        nf2_top.srp              tri_mode_eth_mac.ngc
cdq_tx_fifo_512x36.ngc                   nf2_top_par.drc        nf2_top_summary.xml      txfifo_1024x36_to_9.ngc
dualportmemory.ngc                       nf2_top_par.grf        nf2_top.ucf              txfifo_512x72_to_9.ngc
hdr_fifo.ngc                             nf2_top_par.ncd        nf2_top_usage.xml        xlnx_auto_0.ise
Makefile                                 nf2_top_par.pad        pci2net_16x60.ngc        xlnx_auto_0_xdb
net2pci_16x32.ngc                        nf2_top_par_pad.csv    rxfifo_8kx9_to_36.ngc    xst
netlist.lst                              nf2_top_par_pad.txt    rxfifo_8kx9_to_72.ngc
nf2_top.bld                              nf2_top_par.par        rxlengthfifo_128x13.ngc
nf2_top.lso                              nf2_top_par.ptwx       smartguide.ncd
[root@fedora synth]#
```

We also took screenshots of a few errors we encountered while running the ‘make’ command.

```
344 *                    Design Hierarchy Analysis                         *.
345 =========================================================================.
346 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
347 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
348 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
349 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
350 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
351 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
352 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
353 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
354 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
355 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
356 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
357 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
358 ERROR:HDLCompilers:209 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Count operand of multiple concatenation operator is negative.
359 ERROR:HDLCompilers:185 - "/root/netfpga/lib/verilog/core/output_queues/sram_rr_output_queues/src/oq_regs_generic_reg_grp.
    v" line 402 Illegal right hand side of continuous assign.
```

```
346 =========================================================================.
347 ERROR:HDLCompilers:88 - "../src/fallthrough_small_fifo_v2.v" line 41 Parameter 'PROG_FULL_THRESHOLD' does not exist in mo
    dule 'small_fifo'.
348 ERROR:HDLCompilers:88 - "../src/fallthrough_small_fifo_v2.v" line 41 Parameter 'PROG_FULL_THRESHOLD' does not exist in mo
    dule 'small_fifo'.
349 ERROR:HDLCompilers:88 - "../src/fallthrough_small_fifo_v2.v" line 41 Parameter 'PROG_FULL_THRESHOLD' does not exist in mo
    dule 'small_fifo'.
350 -->~.
351 .
```

```
10982 .
10983 Checking expanded design ....
10984 ERROR:NgdBuild:604 - logical block.
10985    'nf2_core/user_data_path/ids/drop_fifo/XLXI_34' with type 'dualportmemory'.
10986    could not be resolved. A pin name misspelling can cause this, a missing edif.
10987    or ngc file, or the misspelling of a type name. Symbol 'dualportmemory' is.
10988    not supported in target 'virtex2p'..
10989 WARNING:NgdBuild:452 - logical net 'nf2_core/core_256kb_0_reg_addr(100)' has no.
10990    driver.
10991 WARNING:NgdBuild:452 - logical net 'nf2_core/core_256kb_0_reg_addr(101)' has no.
```

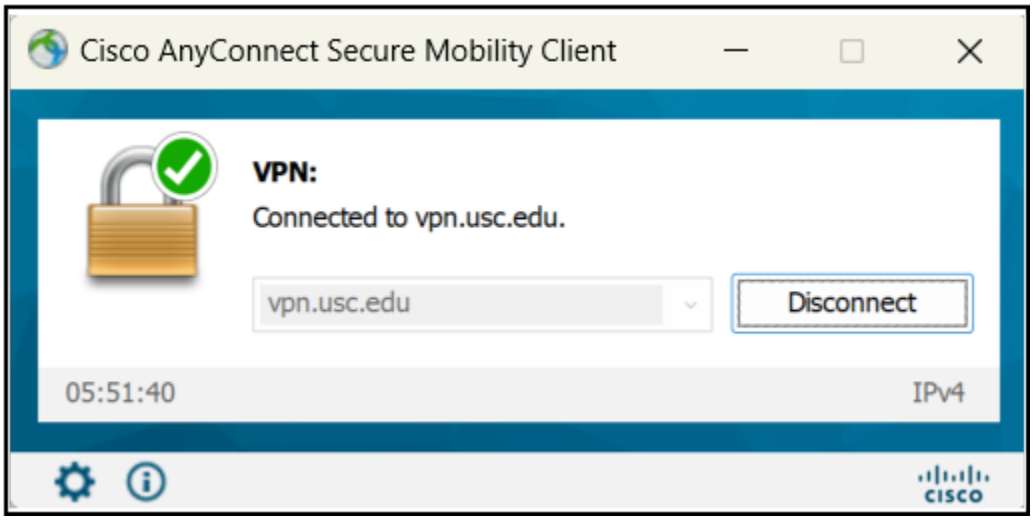
We also are uploading the bitfile generation summary & timing report as a proof of work it took us to debug each of the errors while running the make command.

```
2181
2182  Timing summary:
2183  ---------------
2184
2185  Timing errors: 0  Score: 0
2186
2187  Constraints cover 835729 paths, 4 nets, and 103291 connections
2188
2189  Design statistics:
2190     Minimum period:  11.622ns (Maximum frequency:  86.044MHz)
2191     Maximum path delay from/to any node:  1.850ns
2192     Maximum net delay:  0.603ns
2193     Minimum input required time before clock:  2.865ns
2194     Minimum output required time after clock:  4.689ns
2195
2196
2197  Analysis completed Thu Feb  6 18:22:11 2025
2198  --------------------------------------------------------------------------------
2199
2200  Generating Report ...
2201
2202  Number of warnings: 0
2203  Number of info messages: 3
2204  Total time: 27 secs
2205  +++testFinished:build.trce.nf2_top_par.twr
2206  ================================================
2207  === Generate bit stream file from ncd
2208  ================================================
2209  +++testStarted:build.bitgen.nf2_top_par.bit
2210  bitgen -intstyle ise -g Binary:No -w nf2_top_par.ncd
2211  WARNING:PhysDesignRules:1060 - Dangling pins on
2212     block:<nf2_core/cpu_queues[0].cpu_dma_queue_i/cpu_dma_rx_queue/cpu_fifos64.rx
2213     _fifo/BU2/U0/grf.rf/mem/gbm.gbmg.gbmga.ngecc.bmg/blk_mem_generator/valid.cstr
2214     /ramloop[0].ram.r/v2.ram/dp18x36.ram.A>:<RAMB16_RAMB16A>.  The block is
2215     configured to use input parity pins. There are dangling output parity pins.
2216  WARNING:PhysDesignRules:1060 - Dangling pins on
2217     block:<nf2_core/cpu_queues[1].cpu_dma_queue_i/cpu_dma_rx_queue/cpu_fifos64.rx
2218     _fifo/BU2/U0/grf.rf/mem/gbm.gbmg.gbmga.ngecc.bmg/blk_mem_generator/valid.cstr
2219     /ramloop[0].ram.r/v2.ram/dp18x36.ram.A>:<RAMB16_RAMB16A>.  The block is
2220     configured to use input parity pins. There are dangling output parity pins.
2221  WARNING:PhysDesignRules:1060 - Dangling pins on
```

```
2793    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2794    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2795  WARNING:PhysDesignRules:812 - Dangling pin <DOB22> on
2796    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2797    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2798  WARNING:PhysDesignRules:812 - Dangling pin <DOB23> on
2799    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2800    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2801  WARNING:PhysDesignRules:812 - Dangling pin <DOB24> on
2802    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2803    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2804  WARNING:PhysDesignRules:812 - Dangling pin <DOB25> on
2805    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2806    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2807  WARNING:PhysDesignRules:812 - Dangling pin <DOB26> on
2808    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2809    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2810  WARNING:PhysDesignRules:812 - Dangling pin <DOB27> on
2811    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2812    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2813  WARNING:PhysDesignRules:812 - Dangling pin <DOB28> on
2814    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2815    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2816  WARNING:PhysDesignRules:812 - Dangling pin <DOB29> on
2817    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2818    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2819  WARNING:PhysDesignRules:812 - Dangling pin <DOB30> on
2820    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2821    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2822  WARNING:PhysDesignRules:812 - Dangling pin <DOB31> on
2823    block:<nf2_core/user_data_path/output_queues/oq_regs/oq_reg_instances/oq_addr
2824    _hi_reg/ram/Mram_ram.B>:<RAMB16_RAMB16B>.
2825  +++testFinished:build.bitgen.nf2_top_par.bit
2826  Made all...
2827  rm nf2_top.ncd
```

Part-3: Set Up VPN to USC

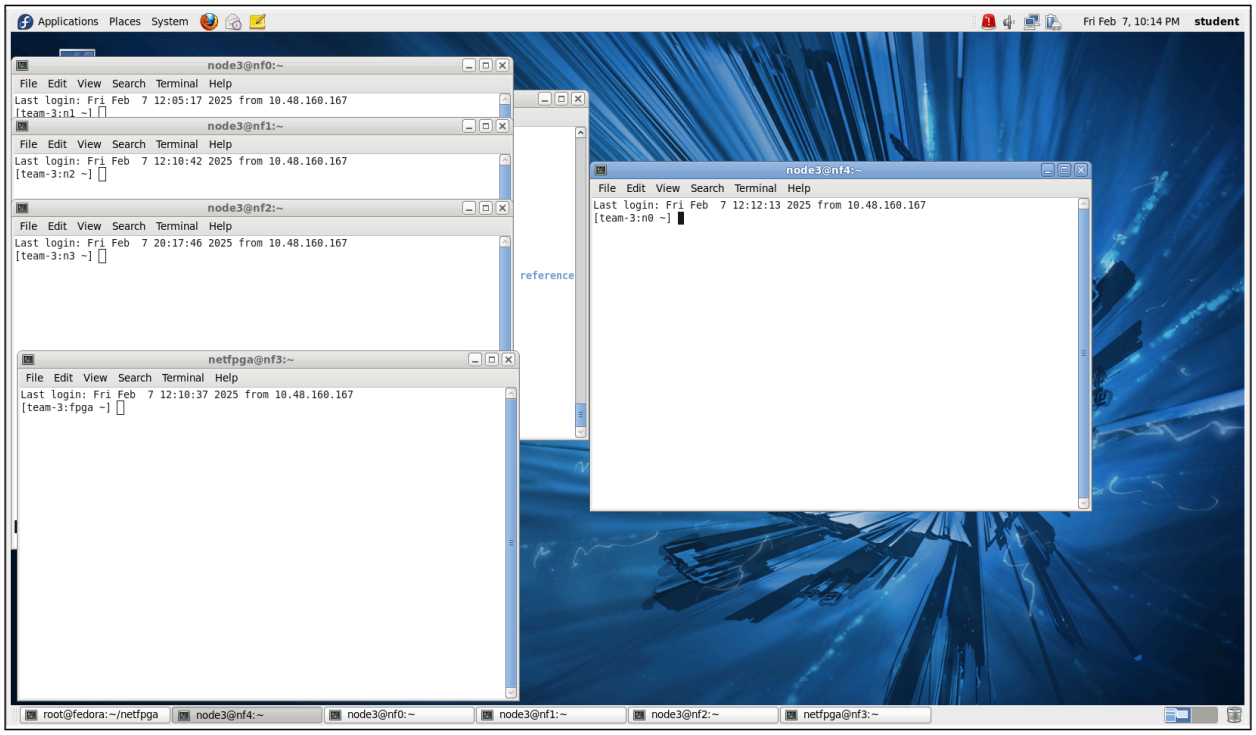
We successfully set up the VPN at *vpn.usc.edu*



Part-4: NetFPGA Environment

1. *Openssh* & *nano* were installed using:
   sudo apt-get install openssh-client nano
2. Ssh client was edited using nano editor:
   sudo nano /etc/ssh/ssh_config
3. Following text was added:
   HostkeyAlgorithms ssh-dss,ssh-rsa
   KexAlgorithms +diffie-hellman-group1-sha1
   StrictHostKeyChecking=accept-new
4. After creating the executable file and connecting to vpn.
   openterm <TEAM #> <TEAM_PASSWORD>
   [NOTE: We are Team#4 but the Team number assigned to us is Team#3]

As depicted in the following snippet, we were able to execute openterm and launch the NetFPGA node and 4 nodes that are connected to NetFPGA.



In order to better understand the openterm:
We have added a short explanation for this openterm.

```
#!/bin/bash
if [ "$#" -le 1 ]; then
      echo "$0 <TEAM #> <PASSWD>"
```

```
    exit 0
fi
```

`if [ "$#" -le 1 ]; then ... fi` → Ensures the script receives **at least two Arguments**:

- `$1` **(TEAM #)** → A team number.
- `$2` **(PASSWD)** → A password for SSH authentication.

**

```
ND=$(($1 % 5))          # Compute remainder when TEAM # is divided by 5
LL=$(($1 + 1))          # Increment TEAM #
CM=$LL"q;d"             # Unused variable (concatenates TEAM + "q;d")
PW=$2                   # Stores the SSH password
CL=$(( $(($1 / 5)) * 5 ))  # Rounds TEAM # down to the nearest multiple of 5
```

`ND=$(($1 % 5))` → Computes a **node index (0-4)** based on the team number.
`LL=$(($1 + 1))` → Adds **1 to the team number** (possibly used for reference).
`CM=$LL"q;d"` → Seems unused; it creates a string with **team number + "q;d"**.
`PW=$2` → Stores the **SSH password**.
`CL=$(( $(($1 / 5)) * 5 ))` → Computes the **nearest multiple of 5** for team number.

**

```
M0=$(($ND + $CL))
M1=$(( $(( $(($ND + 1)) % 5)) + $CL))
M2=$(( $(( $(($ND + 2)) % 5)) + $CL))
M3=$(( $(( $(($ND + 3)) % 5)) + $CL))
M4=$(( $(( $(($ND + 4)) % 5)) + $CL))
```

* This calculates **five machine numbers** (`M0 - M4`) that the script will SSH into.
* Each `M#` represents a **different machine ID** in a cyclic manner.
* It ensures the machines are assigned within **groups of five** (`CL` helps keep machines within the same batch).

**

gnome-terminal -- bash -c "sshpass -p $PW ssh -o StrictHostKeyChecking=no netfpga@nf$M0.usc.edu" &

gnome-terminal -- bash -c "sshpass -p $PW ssh -o StrictHostKeyChecking=no node$ND@nf$M1.usc.edu" &

gnome-terminal -- bash -c "sshpass -p $PW ssh -o StrictHostKeyChecking=no node$ND@nf$M2.usc.edu" &

gnome-terminal -- bash -c "sshpass -p $PW ssh -o StrictHostKeyChecking=no node$ND@nf$M3.usc.edu" &

gnome-terminal -- bash -c "sshpass -p $PW ssh -o StrictHostKeyChecking=no node$ND@nf$M4.usc.edu" &

Opens five separate GNOME terminal windows, each executing an SSH session.

`sshpass -p $PW ssh ...` → **Uses the given password (`$PW`) for SSH authentication.**

`-o StrictHostKeyChecking=no` → **Disables SSH's host key verification prompt.**

Targets machines:

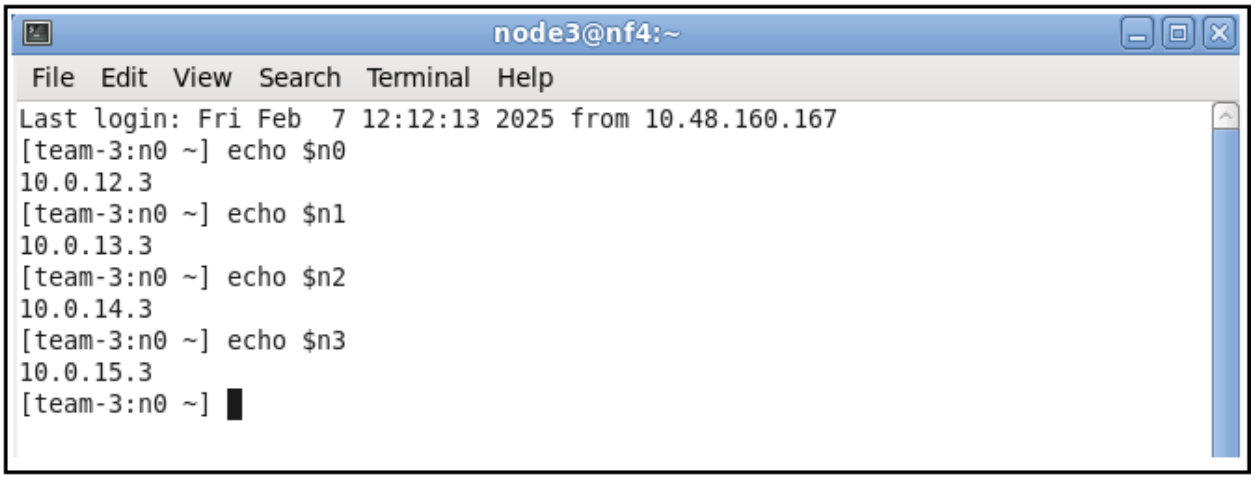
- **First SSH login is for `netfpga@nf$M0.usc.edu`.**
- **The next four logins are for `node$ND@nfM1.usc.edu`, `node$ND@nfM2.usc.edu`, etc.**

Each command ends with `&`, which runs it in the background so that all SSH sessions open simultaneously.

**
For TEAM # = 3, the script:

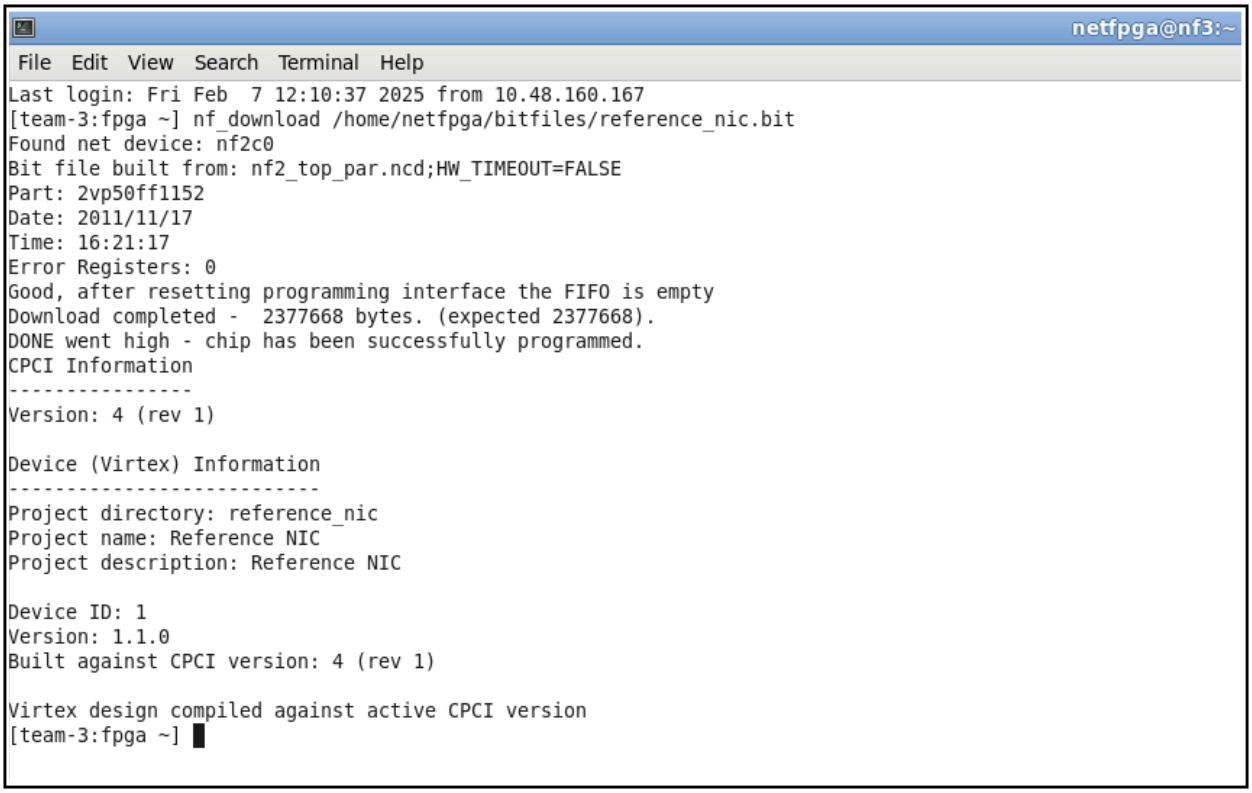
✅ **Computes machine IDs: `3, 4, 0, 1, 2`.**
✅ **Uses password authentication with `sshpass`.**
✅ **Opens five GNOME terminal windows, each running an SSH session.**
✅ **Logs into a mix of `netfpga` and `node3` user accounts on different machines.**

We were also able to find the IP addresses of each node using “echo”.

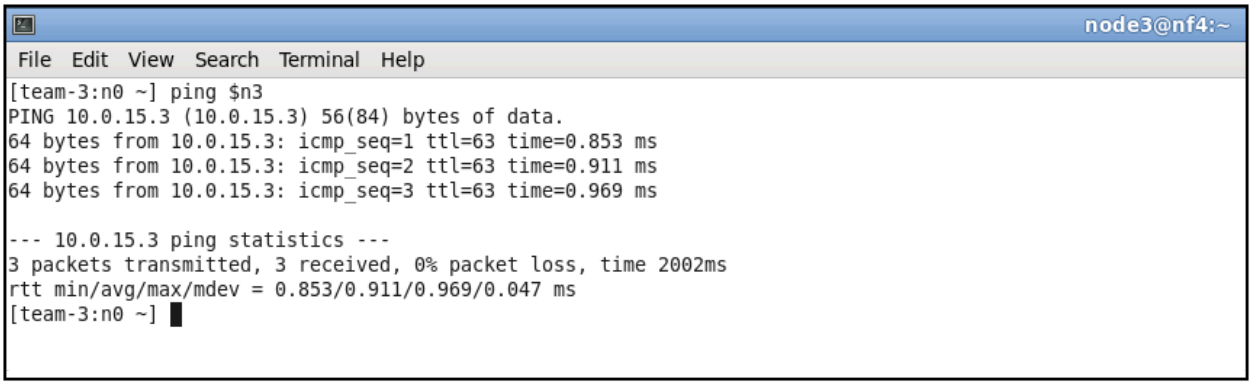


Part-5: NetFPGA-based Linux Kernel IP Router

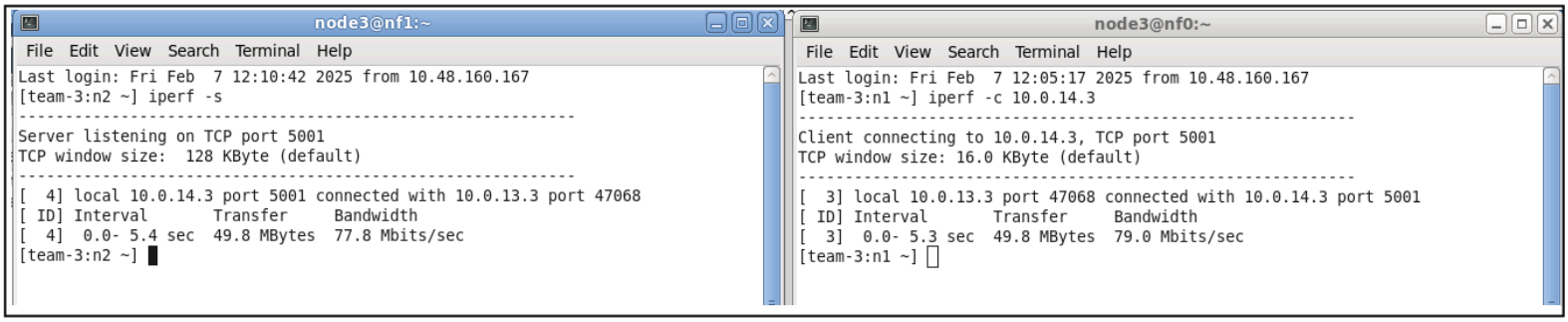
On the FPGA node, we loaded the bitstream of the reference_nic design.



We were also able to ping all nodes from each node. For instance, we were able to ping n3 from n0.



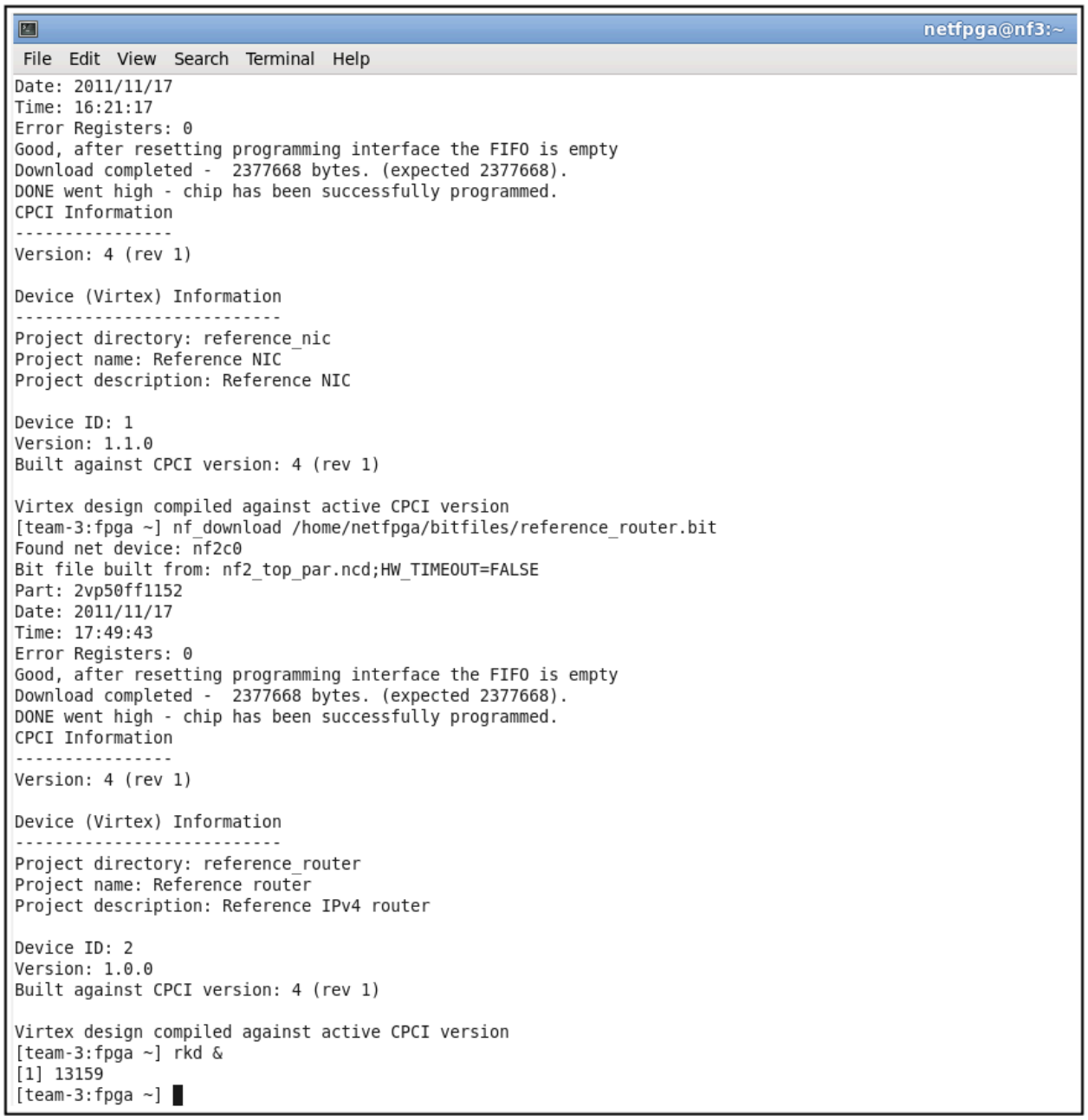
We then executed iperf commands to test the connectivity between nodes. In the following snippet, n2 is the server and n1 is the client. From the snippet, we note that the bandwidth is about 77.8 Mbits/sec.



Part-6: NetFPGA Hardware IP Router

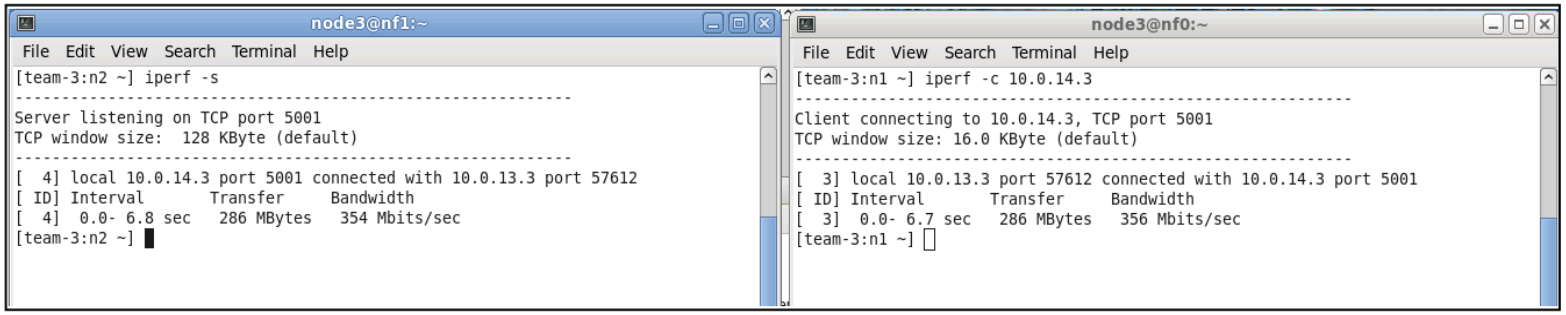
1. This is the part where the NetFPGA was tested as a hardware IP router.
2. Load the reference router into the NetFPGA. What does the reference router do? How should this help?

ANS: The reference router extracts all the information from the Linux Kernel to the routing table of the NetFPGA. This is done to accelerate the routing process.



We then loaded the reference_router bitstream onto the fpga and ran “rkd &” to load all of the needed information from the Linux kernel into the routing table of the NetFPGA in order to accelerate the routing process.

3. Now re-run the iperf tests. Has the bandwidth changed? If so, why?
   ANS: When we re-run the commands identify the acceleration. The measured bandwidth of the server was 354 MBits/sec. In comparison, this is way higher than the execution of 'iperf' when reference nic was loaded 77.8 MBits/sec

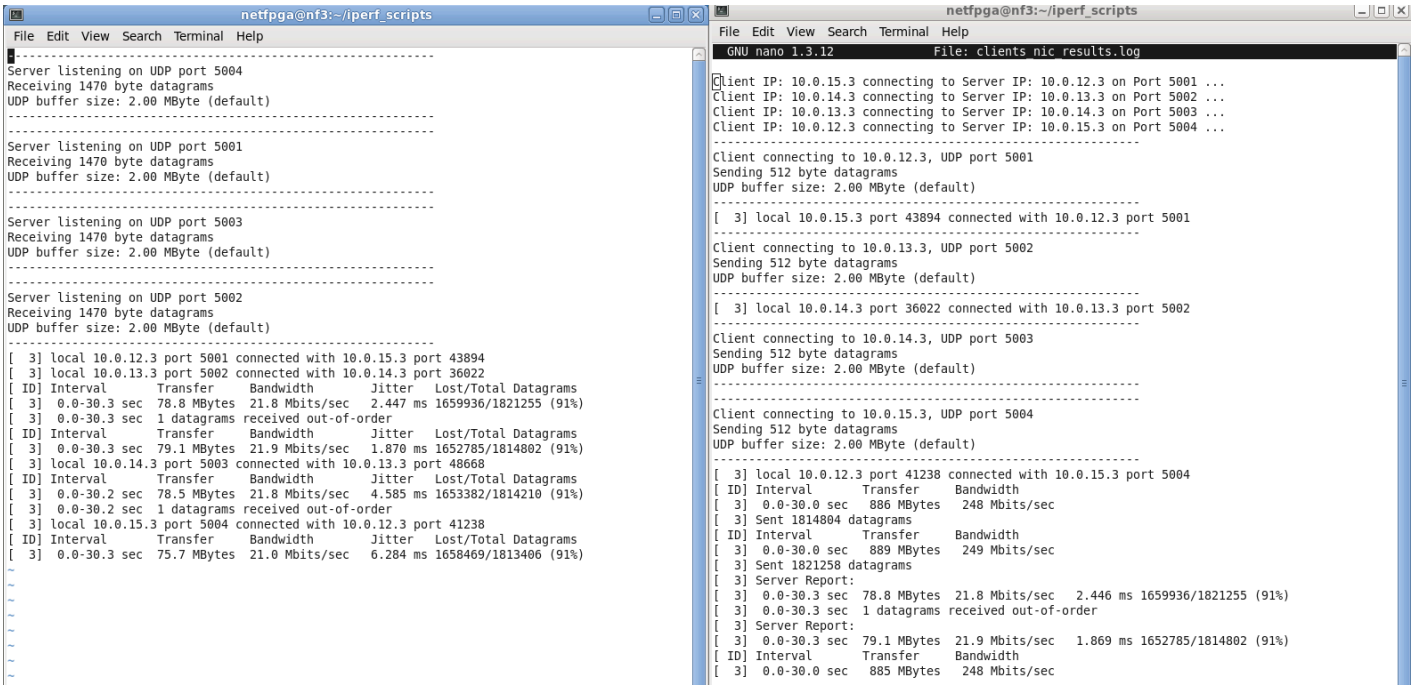


Now that we have the reference_router loaded, we re-run the 'iperf' commands to see the acceleration. In the snippet above, the bandwidth according to the server is 354 Mbits/sec. This is significantly higher than the execution of 'iperf' when reference_nic was loaded (77.8 Mbits/sec).

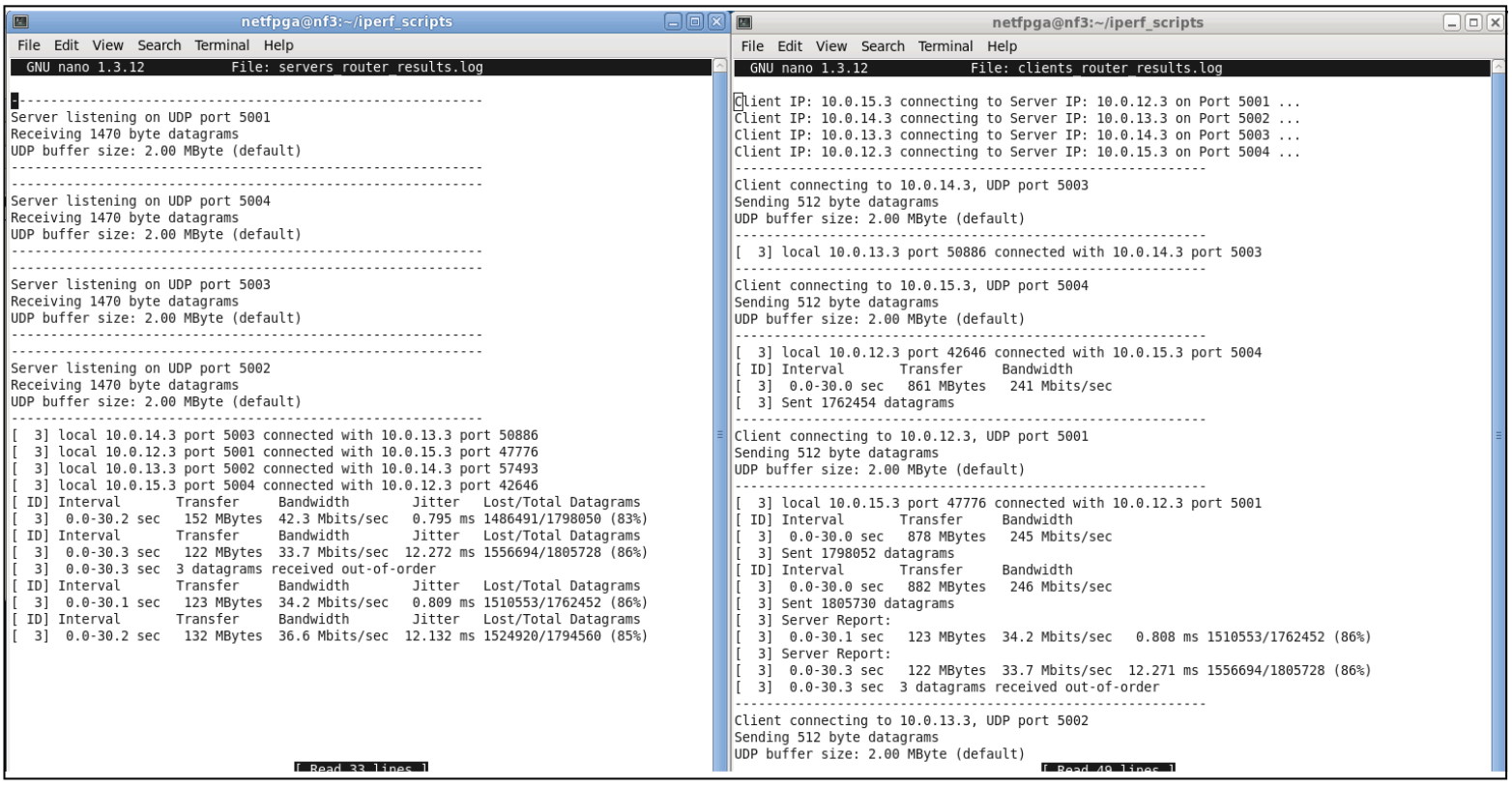
We then created two shell scripts on the fpga node, "udpclients_launch.sh" and "udpservers_launch.sh". The server script iterates through a loop that SSHes to each node (n0-n3) to launch a UDP server on each of them. The client script iterates through a loop that SShes to each node and launches a client on each node. Each client that is launched will then connect to a server that is on a different node (i.e. client n3 → server n0, client n2 → server n1, etc.). Additionally, the client script specifies a bandwidth of 1G, a packet size of 512 bytes, and a test duration. Both scripts write the results into a separate log file, one for the results from the client's side and one from the server's side.

Using the reference_nic bitstream, here are the results after running the two scripts.

4. 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 the small packets stress the system?
   ANS: From the snippet below, we can see that 91% of the datagrams are lost. Additionally, the total bandwidth is 86.5 Mbits/sec (adding up the bandwidth of each client node). Small packets stress the system due to increased packet processing. This limits the throughput and results in packet loss as the system cannot keep up with the incoming data rate. Additionally, small packets might lead to more frequent use of network buffers, leading to packet loss if buffer overflow occurs,



5. Load the reference router.bit file again and start the 'rkd' daemon. Run the same iperf. 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?
   ANS: We then loaded the reference_router.bit file and ran the scripts again. Below are the output log files. We can see that there is less packet loss, but it is still bad with 83% packet loss. Additionally, the total bandwidth is 146.8 Mbps by adding all the bandwidths from the server output log. Hence, it is not fair to say that the NetFPGA can route IP traffic bi-directionally at line-speed for a total of 4Gbps of cross-wise bandwidth.



Team GitHub: EE533-TEAM3/EE533_LAB_PROJECTS/ Commits & Changes