E533 Laboratory #5

Debugging and Testing Custom NetFPGA Design

Instructor: Young Cho - youngcho@isi.edu

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 4 and generate the bitfile Download the generated bitfile 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.

I 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.

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.

In the lab4_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).

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.

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
```

ABCDEFG might be a 7 character string of your choice.

```
>idsreg matches
```

At this point, you should see count = 0x00000000

Start an iperf server on each node. Start threee 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 FPGA node, run idsreg matches, and you should see the count of packets dropped.

Continue with the above process until everything works as it is suppose to.

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 complete the process of going through your design in your YouTube demo video.

Please include the following in your report:

- Schematics
- Generated Verilog
- GitHub records