NetFPGA Logic Analyzer

Andrew Goodney, Shailesh Narayan, Mengchen Wang, Peigen Sun, Vivek Bhandwalkar, Young H. Cho
Department of Computer Science
University of Southern California, Los Angeles, CA

Abstract

The NetFPGA network interface card provides a platform for deploying reconfigurable hardware in the network. Researchers are able to build and simulate a design within an integrated design environment; however debugging a design on the NetFPGA itself is non-trivial. Furthermore, manually creating signals and input data for a simulation that accurately represents a real network data flow is difficult. In this paper we present a NetFPGA Logic Analyzer with a triggering mechanism that captures the control signals and data path of the NetFPGA at the full 125M samples per second for the allotted duration. The triggering mechanism is a programmable pattern matcher module with mask that can be modified while the system is on-line. The trace data is copied to the host CPU where it is converted into a Verilog test benchmark file. The testbench then can be integrated to the rest of the NetFPGA file simulation for the purpose of precise logic debugging and validation.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—simulation; C.0 [**General**]: *System Architecture;*

General Terms

Design, Experimentation, Measurement, Performance,

Keywords

NetFPGA, Modular Design, Debug, DETER, etc.

1. Introduction

The NetFPGA reconfigurable network interface card has become a popular platform for research and teaching in network systems. A researcher or students in a network systems course may design several NetFPGA modules over the course of a semester. Module designers will undoubtedly simulate their designs within the Xilinx ISE tool, however doing so has several drawbacks.

To properly test a design with a simulation, the data driving the simulation (called a testbench) must accurately recreate the signals and timing seen inside the NetFPGA. Manually creating an accurate testbench with correct timing is a difficult if not impossible task. Given this, NetFPGA module designers usually get by with small, synthetic test cases that exercise the important functions of their design. Once the design passes these minimal tests, a bitfile is generated and the design is tested *in situ* on the NetFPGA.

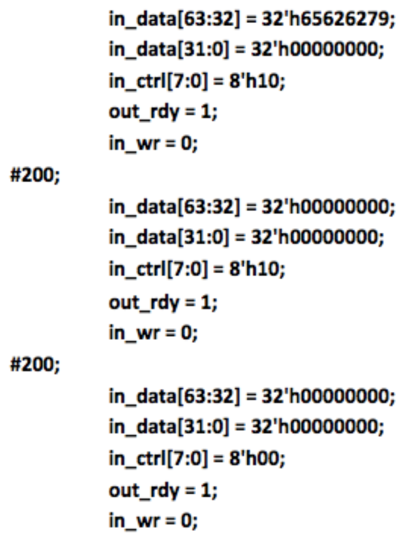
If the design works as expected, this design methodology will work. However, for large, complex modules it may not be possible to manually generate a testbench that properly exercises all features and edge conditions.

NetFPGA module designers need a way to simulate their design using a testbench that generates accurate, real-world data and signal timings. Therefore, we seek a way to capture this data from the NetFPGA itself and import it into the Xilinx ISE tool for simulation.

The Xilinx FPGA at the heart of the NetFPGA includes an industry standard JTAG interface. This interface can be used to program the device as well as debug a running device. Using the JTAG interface to debug a device also requires a PC and software from Xilinx, along with additional domain expertise. Also, it is not clear if real-time capture of data and signals could be accomplished using JTAG debugging (JTAG debugging requires stopping the device while clocking out data and signals in a serial fashion).

The reconfigurable nature and modular design of the NetFPGA provides a solution. We have designed a NetFPGA logic analyzer module. The module may be inserted into the reference router, reference NIC, or at the input or output of any new module under test. The logic analyzer captures the data and control signals at the full 125MHz internal clock of the NetFPGA. The data is written into a memory and subsequently transferred over the PCI bus to the host PC.

We designed three trigger mechanisms into the logic analyzer module: start capturing whenever there is data coming on the bus, capturing only when the valid bit is set and capturing after a particular pattern is encountered. Finally, we developed software that will automatically convert the collected traces into Verilog suitable for use as a testbench.



```
        in_data[63:32] = 32'h65626279;
        in_data[31:0] = 32'h00000000;
        in_ctrl[7:0] = 8'h10;
        out_rdy = 1;
        in_wr = 0;
#200;
        in_data[63:32] = 32'h00000000;
        in_data[31:0] = 32'h00000000;
        in_ctrl[7:0] = 8'h10;
        out_rdy = 1;
        in_wr = 0;
#200;
        in_data[63:32] = 32'h00000000;
        in_data[31:0] = 32'h00000000;
        in_ctrl[7:0] = 8'h00;
        out_rdy = 1;
        in_wr = 0;
```

Figure 1: Verilog generated from captured data

Our NetFGPA logic analyzer is a powerful new tool that greatly simplifies NetFPGA testbench generation. Researchers and module designers can now simulate their designs with a testbench that generates data and signal timings that accurately reflect the conditions present inside the NetFPGA user data path.

Additionally, by converting captured data to industry standard Verilog format, researchers may share traces with others. Modules tested using an accurate simulation will be more likely to operate correctly once deployed on a NetFGPA, reducing the amount of initial debugging.

We believe the ability to share NetFPGA testbenches will enhance the collaborative process in situations where access to a physical NetFPGA is a scarce resource.

2. Related Work

The NetFPGA [1] is a PCI network interface card comprising four gigabit Ethernet interfaces and a Xilinx FPGA. Researchers exploring reconfigurable network hardware have developed various modules ranging from switches [7], and routers [8], to programmable packet processing [9].

Built-In-Self-Test [4] methodologies are used in FPGAs to detect manufacturing defects and thus are not usually available for correctness testing or debugging purposes.

More similar to our work, Grahm et al [5][6] have discussed how to modify the high-level designs of FPGA circuits such that embedded logic analyzers can be hooked to the design. This work focuses on how to instrument and efficiently recompile FGPA designs for in-circuit debugging on the FPGA and is limited by how many output pins may be made available to the logic analyzer.

In this work we take advantage of the ability to transfer data from the NetFPGA to the host PC to capture in real-time the circuit conditions occurring during operation.

3. Design

In the NetFPGA user data path there are three modules: Input Arbiter, Output Port Lookup and Output Queues. Four types of signals are used for inter module communication: control, data, in_write, out_ready. The control message is 8 bits long, which indicates the type of content in data message. The data path is 64 bits wide. The in_write and out_ready are 1 bit respectively, which show whether the first module is ready to send and whether the next module is ready to receive.

Therefore, the total number of binary signals in the complete data path is 74 bits.

To integrate a NetFPGA module into the user data path, a designer inserts it between the Input Arbiter and Output Port Lookup. The module receives the 74 data and control signals, and is expected to output signals in the same format. These signals are clocked at 125Mhz, regardless of if there is packet data present or not.

To test such a module, the designer must somehow generate the levels and timing for these 74 data bits. Without access to real signals, the designer must create the signals manually, usually resulting in a short testbench that may not properly test the module under design. The real data flow of the signals in the NetFPGA user datapath is assuredly different from the manually generated data.

The basic design of the logic analyzer module is to capture the data that flows from one module to another. The module is usually inserted first in the pipeline at the output of the Input Arbiter. The logic analyzer module captures all 74 bits in the data path and directly forwards these bits to the next module. At the same time, the module writes the bits into a dual-ported memory. The memory is 74 bits wide and consists of 64K entries. Signals are captured at the 125MHz internal clock rate of the NetFPGA.

The logic analyzer module also includes hardware to control the point at which the module begins capturing data. When using the first method, capture begins as soon as the NetFPGA hardware clock is running. The second method captures only when there is valid packet data present on the data path. And third, an 8-byte pattern may be set as the trigger condition. Only once this 8-byte patter is present on the data path does the module begin capture.

The three triggering options are included so that designers may fine-tune when and what data is captured to better fit their simulation scenarios.

Finally we have designed software to copy the captured data over the PCI bus to the host PC. The captured data is then converted into a Verilog file that can be incorporated into a testbench. See figure 1 for an example of the testbench Verilog generated by our system.

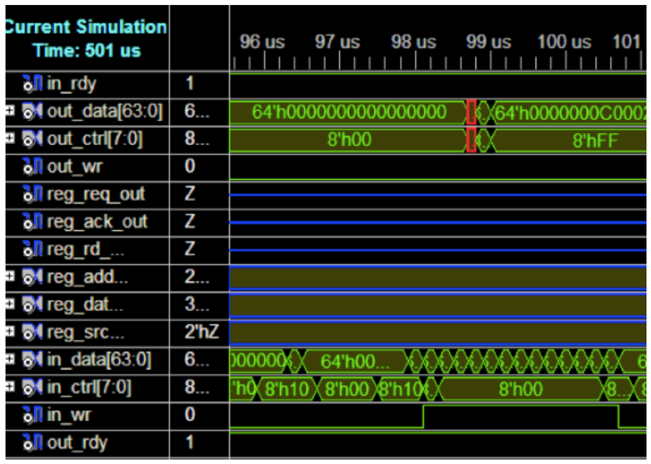


Figure 2: Screen-shot of a simulation driven using captured data

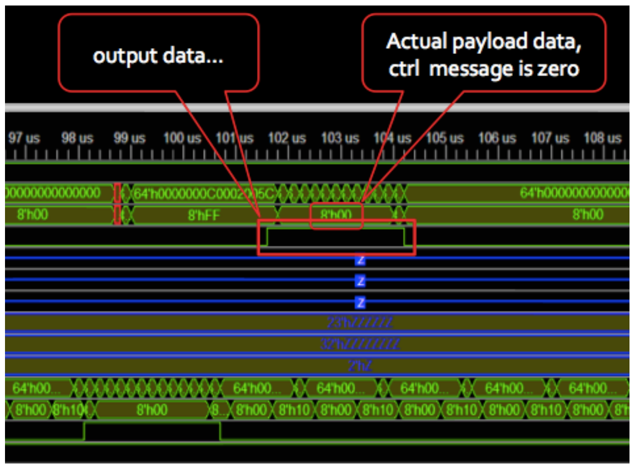


Figure 3: Close-up on the simulation trace showing output data capture

4. Experimental Verification

To verify the correct operation of the logic analyzer module we inserted the module into the user data path in front of a deep packet inspection module. The captured data was converted into a Verilog testbench and the testbench was used to simulate the same DPI module inside Xilinx ISE.

Figures 2 and three are screen captures from the Xilinx ISE simulation time line. Input signals are at the bottom of the figure, while output signals are at the top of the figure.

The trace data was collected under a light network load, thus the NetFPGA user data path is idle for the majority of the simulation. This is indicated by the **in_wr** signal, which is high for only a few microseconds about 98 microseconds into the simulation.

Figure 3 is a close up showing the proper capture of the **out_wr** signal. When **out_wr** is high, the control word is zero, indicating payload packet data is present on the 64 data bits. During this period the figure shows the **out_data[63:0]** signal is changing state, corresponding to the captured data.

5. Conclusion

In this paper we have presented the design of a logic analyzer module for NetFPGA. The logic analyzer module captures all of the signaling and data lines in the NetFPGA user data path at the full 125MHz clock rate. Captured traces are exported to the host CPU and converted into Verilog. The Verilog files can then be used to construct a simulation testbench that will exercise a module under design in a manner that accurately reflects the data and signaling conditions present in a physical NetFPGA.

We used our testbench to capture data from a NetFPGA under light network load. This data was then used to simulate and debug a deep-packet inspection engine.

6. References

[1] Glen Gibb, John W. Lockwood, Jad Naous, Paul Hartke, and Nick McKeown. "NetFPGA: An open platform for teaching how to build gigabit-rate network switches and routers," IEEE Transactions on Education, 2008.

[2] G. A. Covington, G. Gibb, J. Lockwood, and N. McKeown, "A packet generator on the NetFPGA platform," FCCM'09: Proceedings of the 17th Annual IEEE symposium on field-programmable custom computing machines, 2009.

[3] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. "NetFPGA: Reusable Router Architecture for Experimental Research," in SIGCOMM PRESTO Workshop, 2008

[4] C. Stroud, P. Chen, S. Konala, and M. Abramovici, "Evaluation of FPGA Resources for Built-In Self-Test of Programmable Logic Blocks," in Proc. ACM/SIGDA International Symp. on FPGAs, pp.107-113, 1996.

[5] P. Graham, B. Nelson, and B. Hutchings, "Instrumenting bitstreams for debugging fpga circuits," in *Symposium on Field-Programmable Custom Computing Machines*, April - May 2001.

[6] Paul Graham, "Logical Hardware Debuggers For FPGA-Based Systems," Ph.D. thesis, Brigham Young University, December 2001.

[7] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. *Implementing an OpenFlow switch on the NetFPGA platform*. In ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pages 1–9, New York, NY, USA, 2008. ACM.

[8] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. *NetFPGA: reusable router architecture for experimental research*. In PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, pages 1–7, New York, NY, USA, 2008. ACM.

[9] M. Labrecque, J. G. Steffan, G. Salmon, M. Ghobadi, and Y. Ganjali. NetThreads: Programming NetFPGA with Threaded Software, 2009.