Reconfigurable Networking Hardware: A Classroom Tool

Martin Casado
Department of Computer Science
Stanford University
Stanford, CA 94305-9030
casado@cs.stanford.edu

Greg Watson
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-9030
gwatson@stanford.edu

Nick McKeown
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-9030
nickm@stanford.edu

***Abstract*—We present an educational platform for teaching the design, debugging and deployment of real networking equipment in the operational Internet. The emphasis of our work is on teaching and, therefore, on providing an environment that is flexible, robust, low cost and easy to use. The platform is built around 'NetFPGAs' – custom boards containing eight ethernet ports and two FPGAs. NetFPGA boards, when used with VNS (Virtual Network System - another tool we have developed), can be integrated into dynamically configurable network topologies reachable from the Internet. VNS enables a user-space process running on any remote computer to function as a system controller for the NetFPGA boards. NetFPGA and VNS are used at Stanford in a graduate level networking course to teach router implementation in hardware and software.**

I. Introduction

Most college-level digital design classes still use the microprocessor as a canonical design, yet few students go on to design microprocessors after they graduate. Instead, an increasing number of our graduates go on to design networking equipment: complex hardware-software systems that must interoperate with existing systems and infrastructure. In general, graduating students are ill-prepared for this task, and have little experience with this scale and complexity of project work. And so the goal of our work is to give students hands-on experience designing, debugging and deploying fully-functional networking equipment and deploying it into the Internet. In this paper we describe a low-cost and configurable platform which enables students, using industry standard design tools, to develop networking hardware.

The platform we have created is comprised of two core technologies, NetFPGA and VNS [7] which, when used cooperatively, provide the infrastructure to implement network hardware on rich and varied networks. NetFPGAs are custom boards with eight Ethernet ports and two on-board programmable FPGAs (Field Programmable Gate Arrays). A canonical design exercise would be for a student to design an Ethernet switch, or Internet router, by writing Verilog and downloading their design to the NetFPGA board. NetFPGA boards can be programmed and debugged over the network and therefore allow students to build sophisticated hardware, deploy and test it from the other side of the country, without ever having to see the physical board.

VNS (the virtual network system) virtualizes network topologies at the link layer by mapping between VLANs. With VNS a small network of commodity PCs running Linux can emulate thousands of different isolated virtual networks that are reachable from the internet and connect to real operating systems running real services. Any networking device can be included in a virtual topology emulated by VNS, including NetFPGA nodes. This allows us to provide an environment using VNS and NetFPGA where students program their NetFPGAs, and then insert them into a variety of different topologies. Their NetFPGA board processes traffic that comes from anywhere in the Internet, and their design must interoperate correctly with other nodes in the same topology. VNS supports multiple network topologies at the same time; so while one student is testing their NetFPGA design in their own private toplogy, another set of students might be testing how their designs interoperate on a different topology.

Typically, a student implements the main datapath of the system in hardware. For example, in an Ethernet switch, they write Verilog to implement the header processing, address lookup and switching in hardware. But the control software, such as a spanning tree algorithm, or command line interface, needs to run on a microprocessor, yet NetFPGA doesn't have an embedded CPU. VNS enables the control software to run as a user-space program running anywhere on the Internet (i.e. the NetFPGA has a remote virtual cpu). This allows the implementation of sophisticated, integrated hardware and software devices. For example students could build IP routers with hardware forwarding paths and complex protocol processing done in software, using a familiar development and debugging environment.

This work was funded by the NSF, under grant 02-082 and grant EIA-0305729

NetFPGA and VNS are used in "CS344: Build an Internet Router" - a graduate level class at Stanford. In this class, students work in teams of two (one proficient in hardware the other in software) to develop fully-functional, integrated hardware/software IP routers that process live Internet traffic. The routers support integrated TCP/IP stacks, an OSPF-like routing protocol and must interoperate with the routers developed by the other student teams.

Both NetFPGA and VNS are designed for use by other schools. Our goal is to make both tools available at little or no cost - and is funded to do so by the National Science Foundation. Initially, both tools are available for remote use. Several hundred students at different schools have already used VNS remotely. NetFPGA is being prepared for remote use starting in Fall of 2005. We provide pre-packaged assignments (and solutions), remote access to tools, tool support, classroom support, and will even grade assignments. In short, we plan to remove any barriers to the adoption of these tools widely in the networking curriculum.

This paper is organized as follows. The first two sections describe NetFPGA and VNS. Section 4 discusses how virtual topologies are created and managed. Section 5 discusses system transparency and debug support. We then present our experiences with NetFPGA and VNS in the classroom in section 6. Finally we end with an overview of our efforts to redesign NetFPGA and our conclusions.

II. The NetFPGA Platform

NetFPGA is a complete system for teaching the design of Ethernet-based network hardware such as routers or firewalls. The emphasis is on teaching and therefore on providing an environment that is flexible, robust, low cost and easy to use.

The NetFPGA system comprises some custom hardware, off-the-shelf design tools, and some additional scripts and libraries.

The current NetFPGA board measures six inches by nine inches and contains three Altera EP20K400 APEX devices, an eight-port Ethernet controller, three 1MByte SRAMs and ancillary logic. There is no on-board CPU. The architecture of the board is shown in figure 1. The Control-FPGA (CFPGA) is pre-programmed and does two things: it manages the 8-port Ethernet controller, and it provides a simple packet protocol for the two User-FPGAs. Each User-FPGA (UFPGA) has one Mbyte of private SRAM, as well as some on-chip SRAM. Each UFPGA is connected to the CFPGA via a common bus, and to each other via a private bus. The bandwidths of the buses are such that they are not the bottleneck in the system - the throughput limitation will be either the user designs or the Ethernets themselves. The only external connections are the eight 10Mb/s Ethernet ports and a small connector that provides power and a single reset line. Consequently, all communication with the board is via the Ethernets.

The operation of the board is as follows. The CFPGA buffers incoming packets in a local SRAM, and asserts a signal to the UFPGAs indicating that a packet is available from the relevant Ethernet port. Some time later one of the UFPGAs requests the packet which is then transferred to the UFPGA. The UFPGA(s) contain the student's design, be it a switch, router, firewall, XML analyzer, whatever. So the UFPGAs process the packet and may need to then send one or more packets. The UFPGAs pass their packets to the CFPGA, specifying the desired egress Ethernet port. The CFPGA accepts the packet and then forwards it to the Ethernet controller.

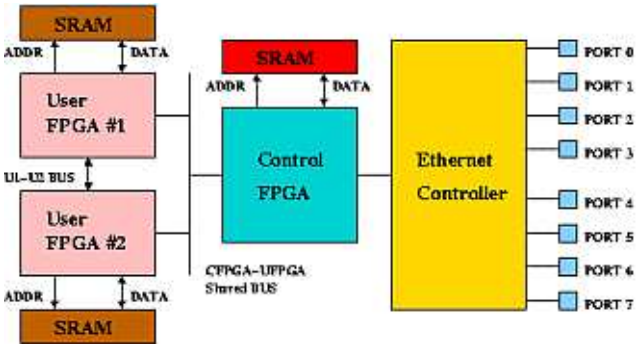

Fig. 1. NetFPGA Board Architecture

The design resources available to the student are the two UFPGAs, each with 1Mbyte of SRAM. Stanford students have implemented several 'obvious' designs include Ethernet switches and Internet Routers. Less obvious extensions to these have included IPSec packet encryption, flow-based rate control, and switches employing flow-based weighted fair queueing.

Most commercial networking products also include one or more CPUs to handle the more complex aspects of the device such as route update messages in an IP router. However the NetFPGA board has no CPU. Instead it has a mechanism that allows the 'CPU' to exist elsewhere. The NetFPGA board recognizes specially encoded packets called control packets that enable a process running on an external computer to perform low level register reads and writes on the student's design. So for example a student can run a routing protocol process on their laptop, and have that process control the actual hardware that is in their NetFPGA board.

We do not specifically advocate the CPU-less approach; indeed our latest version of NetFPGA has a CPU so that we can explore other architectures. On reflection, the use of these encoded packets to perform register accesses has been positive in that it uses the same basic transport mechanism (packets). On the negative side, the performance is poor (hundreds of accesses per second), and the student's software must deal with the fact that packets may be lost in transit.

In practice, given that we have a rack of NetFPGA boards, we need a way to be able to manage the flow of packets between the NetFPGA boards and the campus internet. This flow is handled by the VNS system described in the next section, using VLAN tags [9] to provide isolation between NetFPGA boards.

The next part of the NetFPGA system is the design flow. For this we use commercially available tools (which are usually very cheap or free to academic institutions). We use Synopsys' VCS and FPGA Compiler for logic simulation and synthesis respectively. Place and route of the design is performed by Altera's Quartus tool.

Since the external behavior of a NetFPGA system is completely defined by the sequence of ingress and egress packets, then verification tests consist of lists of packets. Each test specifies the sequence of ingress packets at each port, as well as

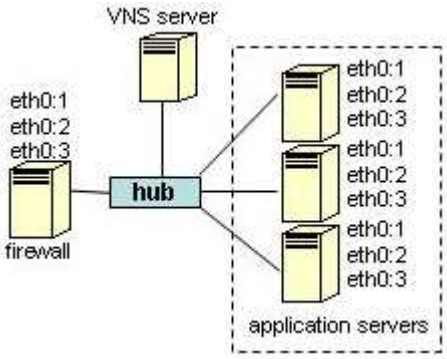

Fig. 2. A Typical setup of VNS. The application servers and firewalls are configured with multiple virtual interfaces each configured on a unique VLAN.

the sequence of expected egress packets from each port. The students use a Perl program to specify these packets. The reason for this is that the Perl program generates a test environment that works in simulation (with VCS) and also on the real hardware, thus obviating the need for separate hardware tests. A second script is used to verify that the observed sequence of egress packets matches the expected sequence. This script works for both simulation and the actual hardware tests.

Once the students have debugged and synthesized their design, they need to download it to the hardware. This is accomplished using the third part of the system: a Java-based web interface.

This interface enables students to:

- acquire a board from the rack of boards,
- download their design to the board,
- send packets to the board,
- capture all egress packets from their board.

The NetFPGA tools use the pcap library format [1] for packets, and so the students can use widely available network packet analysis tools such as Ethereal[3] for debugging their systems.

III. The Virtual Network System

NetFPGA provides the foundation on which students implement networking devices in hardware. In order to provide the students with a realistic, dynamic environment for development and testing on the operational Internet, we've developed a complementary technology, the Virtual Network System (VNS) [7]. In short, VNS virtualizes the network allowing the simulation of many complex and configurable network topologies on which NetFPGA boards can participate. Each of the simulated topologies is reachable from the Internet and thus carries live Internet traffic. In this section we present VNS and discuss how it interoperates with NetFPGA to integrate student projects into the Internet.

A typical setup of VNS is shown in Figure 2. The firewall and the application servers are standard PCs running Linux. Each PC is configured with one or more virtual interface each of which is on a unique VLAN (and therefore tagged with a unique VLAN id). The core of the system is the VNS server which accesses all of the traffic on the network by placing its interface in promiscuous mode. The server can determine the source of any packet by its VLAN ID and the destination by inspecting the destination address of the Ethernet header.

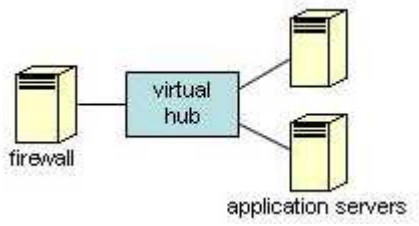

Fig. 3. A VNS virtual topology consisting of a hub between three commodity PCs. Each PC interface is a virtual interface with a unique VLAN identifier. The VNS server provides the mapping between the VLAN ids so that each packet that is sent by an interface on the topology is received by the other two.

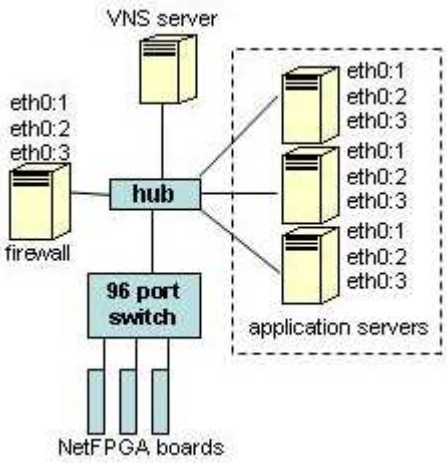

Fig. 4. VNS system with connected NetFPGA boards. The application servers and firewalls are configured with multiple virtual interfaces each configured on a unique VLAN. Packets from all NetFPGA ports are also tagged by the switch with unique VLAN IDs.

Because it can access all traffic on the network, the VNS server is able to to emulate arbitrary network topologies by mapping between VLANs. Figure 3 shows an example of a virtual topology emulated by the VNS server using the setup shown in figure 2. In this case, the virtual topology consists of the firewall connected to a hub with two connected application servers. For each packet the VNS server intercepts from one of the interfaces on the virtual topology (identified by their VLAN IDs) it will generate two more packets with VLAN IDs of the two other interfaces, thus emulating a network hub. Note that the original packet is never intercepted (taken off the wire) by the VNS server which can only passively listen, however because it is on a VLAN unique to the sending interface, the other listening interfaces are unable to process it. A virtual topology is therefore simply a mapping, performed by the VNS server between separate VLANs. Each of the connected PCs can be configured with many (hundreds) of virtual interfaces and each of these interfaces my participate on a different topology in complete isolation.

A. Integrating with NetFPGA

VNS allows us the ability to integrate one or more NetFPGA boards into multiple complex topologies. Doing so does

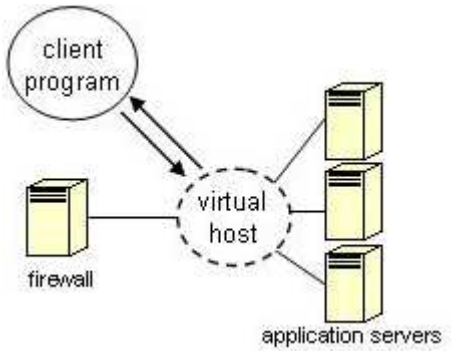

Fig. 5. A VNS virtual topology with a virtual host interposed between the firewall and two application servers. The VNS server tunnels all packets to the virtual host over a TCP connection to a client program executing in user space on a PC. The client program is therefore able to function as an entity on the network.

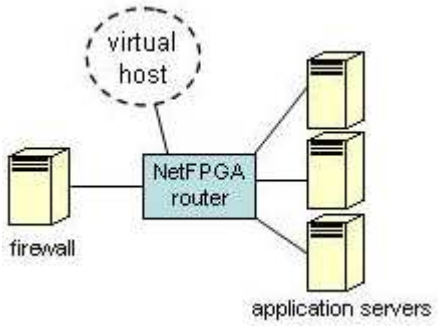

Fig. 6. VNS allows NetFGPA boards to participate in virtual topologies. A virtual host connected to port 0 of the NetFPGA board can function as an offboard control unit. The client program can control the NetFPGA by sending special packets which can read and write to registers.

not require any modifications to either NetFPGA nor VNS. As shown in figure 4, we connect NetFPGA boards to the network through a switch which marks each port with a unique VLAN identifier. Using the VLAN identifier, the VNS server can determine which port a packet originates from and thus can be configured to allow one or more NetFPGA boards to participate on the topology like any other networking device. Our experimental setup uses a single, 96 port switch, to connect 8 NetFPGA boards, the VNS server, a firewall and three unix servers running various services including http and ftp. Using this setup, VNS can be dynamically configured to include the NetFPGA boards in arbitrary topologies connected to commodity machines. Figure 6 shows a logical topology with a NetFPGA board interposed between the firewall and three application servers. The NetFPGA board can then be programmed to operate as an intermediary networking device such as an Ethernet hub, a learning switch or an IP router.

B. Providing a User Space 'CPU'

In addition to mapping between VLANs, the VNS server is able to interpose one or more virtual hosts in the network as shown in figure 5. Each virtual host is emulated at the link layer by the VNS server. The goal of supporting virtual hosts in the VNS server is to provide a mechanism in which user-level client programs running on standard operating systems can participate on the virtual topology. When the VNS server intercepts a packet destined to an interface on a virtual host, instead of mapping that packet to another VLAN ID and placing it back on the wire, the server strips of the VLAN header and sends the packet over a TCP connection to the associated client program. The client therefore will receive all packets seen by any of the virtual host's interfaces on the topology and can send packets to the server to be injected back into the network specifying, which of its 'interfaces' to send the packet out on. The server will then ensure that the sent packet is tagged with the correct VLANs for all connecting interfaces. The user level program is in essence, functioning as an entity on the network.

The ability of the VNS server to support integration of user space processes into the virtual network topology can be used to add a virtual 'cpu' to the NetFPGA cards. The NetFPGA can be controlled via control packets sent to port zero which read and write to all accessible registers. It is possible then, to create a topology in which a virtual host is connected directly to port zero of the NetFPGA board, as shown in figure 6. The VNS server will handle moving packets between the user process associated with the virtual host and port zero of the NetFPGA board. In this configuration the virtual host can effectively act as a 'cpu' for the NetFPGA. For example, assuming the configuration in figure 6, the NetFPGA board can be programmed to function as a 3 port Internet router. However, the more complex functions such as OSPF support can be implemented within the virtual host.

The model of allowing a user-space program to function as an offboard cpu has a number of advantages. Standard PC operating systems typically provide much richer debugging and development environments than those for embedded processors. There is no need to cross compile nor download code to the board and standard debuggers, such as gdb, can be used to monitor and debug the program during runtime. Furthermore, because the main focus of our work is to provide an educational platform we believe supporting a familiar development and runtime environment, such as a Unix, greatly eases introduction to an otherwise complex system.

C. Creating Topologies

Creating a virtual topology in VNS requires writing a topology description file in XML and copying it to a directory monitored by the VNS server. Topology description files contain the connectivity of the hosts in the topology, the IP addresses each interface and the VLAN identifiers assigned to the NetFPGA ports that will be participating on the topology. Given a description file an installation script will create a topology by performing the following actions:

- Create a virtual interface on each participating PC and assign the interface the given IP addresses and VLAN ids
- Set up routes on each of the PCs to forward packets
- Copy the topology file to the directory monitored by the server

Once the server determines that a topology file has been added or has changed it will perform the VLAN mappings specified by the new file. Because the topology files can be updated at runtime, the system supports dynamic topologies which may

include link failures, changes in connectivity or changes in link properties.

IV. System Transparency

Typical use of NetFPGA does not require the developer to have hands-on access to the hardware. The developer may in fact develop from anywhere on the Internet. A significant challenge to supporting remote development is to provide the same level of debugging transparency into the system as would be available if the developer had physical access to the board and the connected network. To provide transparency in the system via two mechanisms, we provide a remote logic probe and generation of pcap compatible trace files.

A traditional hardware development system would provide the capability to capture signals in real time via a logic analyzer. Indeed Altera provides such a capability with their embedded logic analyzer. However this capability requires a computer to be connected to each board which is prohibitively expensive for NetFPGA.

Instead we provide a logic analyzer that runs on one of the UFPGAs. In practice this is not much of a restriction because we found that most student designs only use a single FPGA. This logic analyzer, while not as sophisticated as commercial Logic Analyzers, has proven reasonably flexible. The user can specify a sophisticated four-stage trigger and the local SRAM can capture up to 256K samples of 32 signals.

Of course the students use the logic analyzer via the web: they can launch the analyzer and then start a simulation, and see when the analyzer triggers. The analyzer then automatically provides the student with the trace file in Value Change Dump format, and so the student can use commercial or free waveform viewers to examine the trace dump.

In addition to the logic probe the system provides debug support by logging all packets processed by the user space program operating as a NetFPGA cpu. VNS client programs communicate with the server using a simple libarary. This library logs all packets that are exchanged between it and the server in pcap[1] trace format. The packet traces are saved client side and can be used with tcpdump[2] or Ethereal[3] to inspect all traffic processed by the client.

V. Teaching Networking Hardware

NetFPGA and VNS are integral to an upper level projects course here at Stanford in which students learn to design and develop Internet routers. During the course, two-student teams design fully functional, integrated hardware/software routers that process live Internet traffic. Each team consists of a student proficient in hardware and a student proficient in software. At the end of the course, all student routers are tested on a large shared topology in which they must interoperate to build their forwarding tables and route traffic correctly despite multiple, periodic link failures. The students design their own interoperability tests, and use VNS to implement on arbitrary private topologies of their choosing that connect to the Internet.

The students are presented with the verilog code for a two port, unintelligent layer two switch. Their first job is to extend this design into a three port learning switch – their switch must automatically learn where nodes are located based on the source address of incoming packets. The next step is to provide control packet support so that an external software process can perform register accesses on their design. The third step is to implement ARP functionality - the students' router must identify ARP packets and be able to pass them to their CPU process. Then the main stage is the implementation of the actual IP routing protocol - the students must add longest-prefix matching capability, as well as the necessary route tables to their design.

In parallel with the hardware portion of the project, the student responsible for the software develops complex functionality in C which operates as the router CPU through VNS as discussed previously. The software must support the following functionality.

- an integrated TCP stack which exports a bsd/style socket interface
- a command line interface for managing the router (this runs on top of the student's socket interface and TCP stack)
- malformed packet processing, such as handling fragments or packets with bad headers
- ICMP support including generation of time exceeded messages, port and host unreachable messages and echo replies
- a simplified link state routing protocol (based on OSPFv2) which must interoperate with the other student routers
- setting up and managing the hardware forwarding table and ARP cache

Designing an integrated software and hardware router requires students to face complex design issues such as the performance flexibility tradeoff of moving functionality between hardware and software. These issues become particularly important in the final portion of the project in which the students must implement advanced functionality of their own design into their routers. Including an open-ended design component to the project allows students to explore areas of interest to them while tackling an integrated hardware/software design problem without the guidance of detailed specifications. The only requirement to this portion of the project is that the implemented features have both a hardware and software component that operate cooperatively. We have been delighted by the sophistication and inventiveness of the students in coming up with advanced functionality. During the pilot offering of the course during the Spring quarter of 2004, student projects included MAC level encryption, flow rate limiting, VPN support, NAT and a web programmable hardware firewall. In our latest class (spring '05) one team used their router to implement a man-in-the-middle security attack on SSH connections, and used it to demonstrate how to capture passwords. A more detailed description of the course is presented in [8].

Ultimately, it is our goal to host remote courses at other institutions and Universities. The full router project can be done remotely without requiring the students to have direct access with the test network or the NetFPGA boards. All course materials are publicly available online at the course website which is included in the conclusion of this paper. We provide all tools support and grading. Funding is provided by NSF and comes at no cost to the participating institution.

VI. Redesigning NetFPGA

After using NetFPGA for three years we have feedback from our users, and we are now in the process of developing new versions of hardware (and software). The issues we are addressing next are:

- The biggest problem with the current system is that the hardware is awkward to distribute to others. The NetFPGA boards use a proprietary backplane that distributes power and reset. The new board will use standard 3.3V PCI and will fit into a personal computer. Indeed we are designing the system to have up to five boards per computer, with one board as the control board and the remaining four for use by the students. Thus the entire NetFPGA system can be delivered 'shrink-wrapped' for installation on a standard x86-based personal computer.
- The networks are too slow. NetFPGA uses eight 10Mbit/s ports. While suitable for teaching they are too slow for interesting research projects. Consequently our new board will provide four Ethernet ports each operating at 10, 100 or 1000 Mbps. This will require us to upgrade the VNS server and infrastructure to use 1.0Gbps links also; fortunately they are now very affordable. Already two researchers have expressed interest in using the board to explore new transport protocols - their protocols require custom network interfaces which, of course, is the main attribute of NetFPGA.
- The new board also has two 3.0Gbps duplex serial links which can be used to interconnect multiple boards to form a larger system.
- The mechanism used to provide low-level register access to the hardware is awkward for students and slow, as mentioned earlier. Currently, the students' control unit running within VNS must send ethernet frames conforming to a particular format over an unreliable network to issue register reads and writes. The new board will address this by using the PCI bus to map the board's registers directly into memory thus providing a much faster access mechanism. The cost of this is added complexity to the system as we will need a way to provide students with restricted access to their board.
- While the basic functionality of the boards is still provided by a CFPGA (as in the first iteration) the new board will have an on-chip CPU (PowerPC). It is our hope that the students can use these in future projects. Note that the presence of a CPU does not mean that we are moving away from the remote CPU approach that we have used so effectively. Rather the presence of an onboard CPU provides new avenues to explore. For example it might provide the opportunity for some simple per-packet computations required by emerging variants of TCP. The cost of using the onboard CPU is a significantly more complicated tool flow and thus might be limited to more advanced projects.
- The current tool flow is outdated, slow and only runs on non-MSWindows operating systems. The new board will use a single tool for synthesis and place-and-route and is supported on MSWindows as well as Linux and other versions of UNIX. We also intend to support several commercial Verilog simulators, including VCS and ModelSim.
- The FPGA technology is now dated. We will be using newer FPGA devices (Virtex II Pro 30) and SRAMs with greater speed and capacity. This will provide the resources for more interesting student projects.
- Logic Analyzer support will be continued. The new board has two independent SRAMs, with one that can be dedicated for use by the Analyzer. However we have observed that students rarely use the Analyzer. We believe there are two reasons for this: first, the simulation environment is sufficient to identify almost all bugs. Second, it requires some work: deciding which signals to capture, hooking those signals to the analyzer block, and then setting up trigger definitions.

VII. Related Work

The FPX project [4] [5] at Washington University provides FPGA hardware that can be attached to a single OC-48 ATM port. As such it is not aimed at providing a complete networking system, but rather on providing programmable hardware assist for functions that are too complex to be done in software (such as on-the-fly JPEG encoding at Gbit/s rates).

Also at WU is the Open Network Laboratory project [6] This is a collection of open-source, extensible routers built using their FPX technology. They plan to have four of these configurable routers interconnected a programmable switch. This is primarily a research project, providing a high-speed flexible environment for use in router research.

VIII. Conclusions

University networking classes often contain projects, but these projects generally operate at the socket-layer and above. To operate below a socket might involve dabbling with operating system structures, dedicated hardware, kernel hacking, and security nightmares as students generate corrupt packets and inject them into the network. As a result, few networking classes are able to give hands-on experience at a lower level - where, arguably, the real networking infrastructure exists. Our goal is to give students hands-on experience with the Internet infrastructure. We don't expect our approach to replace traditional system design classes anytime soon, but we do hope to make it possible to include meaningful, low-level networking projects in a standard curriculum.

To this end, we produced two tools: NetFPGA and VNS, each of which aids in the development of networking hardware on configurable network topologies. Our approach allows users to build network devices out of custom boards that can be programmed and tested over the Internet. We provide a remotely accessible logic probe to aid in development and testing. In addition, the environment supports the emulation of multiple, complex network topologies that can integrate with commodity operating systems and arbitrary networking hardware. These topologies are reachable from the Internet and therefore allow testing of network devices on real traffic in real time. In addition, the system provides a method for adding a cpu to the network device as a user space process thus simplifying prototyping, developing and testing complex functionality.

The focus of our work has been to provide a robust, low-cost environment for teaching the implementation of networking hardware in relation to Internet infrastructure. We are using both NetFPGA and VNS in a graduate level projects course in computer networks in which students develop sophisticated routers in hardware and software. All use of NetFPGA and VNS can be done remotely. We are thus able to support remote courses in network hardware design and it is our goal to support such courses at other universities. If you are interested in learning more about NetFPGA, VNS or the course we offer, please visit the following websites or contact us directly by e-mail.

- **NetFPGA Website:** http://klamath.stanford.edu/NetFPGA/
- **NetFPGA 2 Website:** http://klamath.stanford.edu/nf2/
- **VNS Website:** http://yuba.stanford.edu/vns/
- **Course Website:** http://yuba.stanford.edu/cs344_public/

IX. Acknowledgements

This work was funded by the NSF, under grant 02-082 and grant EIA-0305729. Special thanks to Alan Swithenbank and Jim Weaver for their time spent on board design. We also acknowledge the tremendous support provided by Altera, Xilinx and Synopsys.

References

[1] libpcap packet capture library. http://www.tcpdump.org.
[2] tcpdump network sniffer. http://www.tcpdump.org.
[3] The ethereal network analyzer. http://www.ethereal.com.
[4] John W. Lockwood, An Open Platform for Development of Network Processing Modules in Reprogrammable Hardware, *IEC DesignCon 2001*, Santa Clara, CA, Jan. 2001, Paper WB-19.
[5] the FPX project page. http://www.arl.wustl.edu/arl/projects/fpx/
[6] Open Network Laboratory project http://www.arl.wustl.edu/projects/onl/
[7] Martin Casado, Nick McKeown. The Virtual Network System *ACM SIGCSE Bulletin*, Volume 37, Pages 76 - 80, 2005
[8] Martin Casado, Gregory Watson, Nick McKeown. Teaching Networking Hardware To appear in *ACM ITiCSE*, 2005
[9] IEEE 802.1Q VLAN Specification http://www.ieee802.org/1/pages/802.1Q.html