Control and Configuration Software for a Reconfigurable Networking Hardware Platform

Todd S. Sproull, John W. Lockwood , David E. Taylor
Applied Research Laboratory
Washington University
Saint Louis, MO 63130
Web: http://www.arl.wustl.edu/arl/projects/fpx/

Abstract—**A suite of tools called NCHARGE (Networked Configurable Hardware Administrator for Reconfiguration and Governing via End-systems) has been developed to simplify the co-design of hardware and software components that process packets within a network of Field Programmable Gate Arrays (FPGAs). A key feature of NCHARGE is that it provides a high-performance packet interface to hardware and standard Application Programming Interface (API) between software and reprogrammable hardware modules. Using this API, multiple software processes can communicate to one or more hardware modules using standard TCP/IP sockets. NCHARGE also provides a Web-Based User Interface to simplify the configuration and control of an entire network switch that contains several software and hardware modules.**

I. INTRODUCTION

An experimental platform called the Field Programmable Port Extender (FPX) has been developed to enable the rapid deployment of hardware modules in networks [1] [2]. In this system, data are processed by reconfigurable hardware, while control and configuration of the network system is implemented in software. Control cells are used to communicate between the hardware and software.

A suite of tools has been developed to manage the network of reconfigurable hardware on the FPX called NCHARGE (Networked Configurable Hardware Administrator for Reconfiguration and Governing via End-systems). NCHARGE enables new hardware modules to be dynamically deployed onto the Field Programmable Gate Array (FPGA) logic of the FPX over a network. Once the hardware module is installed, NCHARGE controls the operation of each FPX by issuing customized control cells to control each hardware module.

NCHARGE provides a standard Application Programming Interface (API) for software applications that communicate with hardware. NCHARGE uses TCP/IP to communicate reliably with hosts on the Internet. It also implements a reliable protocol to transmit and receive messages between hardware and software, as shown in Figure 1. Software applications, in turn, connect to NCHARGE via TCP/IP to issue commands to the remote hardware modules. One application that uses this service is an Internet routing module that forwards packets
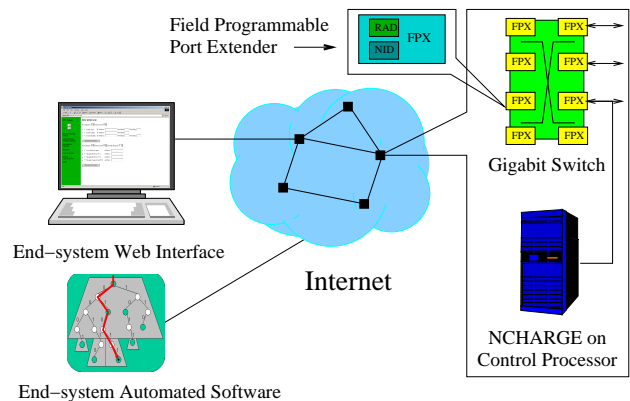
Fig. 1. Overview of system architecture: NCHARGE runs on the control processor of a gigabit switch equipped with FPX reconfigurable hardware modules and communicates to end-systems via the Internet.

in FPGA hardware and manages the data structures for a forwarding table in software.

NCHARGE also provides a web-based user interface so that remote users can perform remote operations on the FPX from distant end-systems. Through the web interface, users can reprogram the FPGAs on the FPX, update remote memory attached to the FPGAs, check the board's status, or issue custom commands to their specific hardware module. In a fraction of a second, NCHARGE is able to program the FPX to act as an IP router from a web browser. With a few additional button clicks the router can be reconfigured. If the need arises, a new module can be loaded onto the FPX to perform, encryption, run length encoding, compression, and other customized packet processing functions. The features of NCHARGE fully exploit the flexibility of the reconfigurable hardware.

II. REPROGRAMMABLE HARDWARE PLATFORM

Reprogrammable hardware has proven to be effective at accelerating functions within a network [3] [4]. The ability to easily reprogram hardware via the Internet allows networking features to be deployed without physical access to the hardware [5]. The (FPX) is a networking hardware platform that has been built to perform data

processing for cells and packets in a high-speed network switch [6].

The FPX provides an open platform for the rapid prototype and deployment of networking functions in hardware [7]. Components of the FPX include two FPGAs, five banks of memory, and two high-speed network interfaces [2]. Networking interfaces on the FPX were optimized to enable the simultaneous arrival and departure of data cells at SONET OC48 rates. This is the equivalent bandwidth of multiple channels of Gigabit Ethernet.

The FPX integrates with another open hardware platform, the Washington University Gigabit Switch (WUGS) [8]. By inserting FPX modules at each port of the switch, parallel FPX units can be used to simultaneously process packets on all ports of the network switch.

In a single rack mount cabinet, a system has been demonstrated that can hold 8 * 2 * 2 = 32 Xilinx Virtex FPGA devices connected via a 20 Gbps network switch. Hardware resources on the FPX have been designed in such a way as to promote ease of application development. Hardware modules use a standard interface to request access to memory and execute memory transactions. The interfaces abstract application developers from device specific timing constraints, insulating hardware modules from changes in memory device technology.

A. Network Interface Device (NID)

The FPX includes a statically-programmed FPGA that is called the Network Interface Device (NID). It controls how packet flows are routed to and from modules. The NID contains a per-flow switching module to selectively forward traffic between networking ports of the RAD, the switch, and an optical line card. The NID also provides mechanisms to dynamically load hardware modules over the network and into the router. The combination of these features allows data processing modules to be dynamically loaded and unloaded without affecting the processing of packets by the other modules in the system.

B. Reprogrammable Application Device (RAD)

The FPX also includes a dynamically-programmed FPGA that is called the Reprogrammable Application Device (RAD). The RAD enables application-specific functions to be implemented as dynamic hardware plug-in (DHP) modules [9]. A DHP consists of a region of FPGA gates and internal memory, bounded by the well-defined interface. A standard interface has been implemented so that modules can transmit and receive packet data and communication with off-chip memories [10].

Each hardware module has a unique identification number (ModuleID) that is used for application-specific communication. When a control cell arrives at the input of a hardware module, it compares the ModuleID field in the control cell to the ModuleID stored within the mod-ule to determine if the control cell should be processed. If the control cell is addressed to the hardware module, the module processes the cell, builds a response cell, and sends the response cell back to NCHARGE. If the control cell is not addressed to the hardware module, the module simply passes the cell to its cell output interface, as the cell may be addressed to a subsequent module in the system or may be a response cell from a previous module in the system.

C. Control Cell Processor (CCP)

The Control Cell Processors (CCPs) are entities implemented in FPGA hardware on the NID and RAD that process control cells sent by NCHARGE. Control cells addressed to a CCP cause the hardware to perform an operation and generate a return message. Several types of control cells are used to perform the memory transactions. State machines within the CCPs read and write the physical memory at addresses specified in control cells. The CCPs perform memory transactions consisting of any combination of reads and writes to either of the SRAM devices or SDRAM devices [11].

In order to perform the reliable transmission protocol with the software, the CCP formats a response cell containing the data written to and read from the addresses in the designated memory device. The CCP also computes checksums to ensure the integrity of the messages. If a control cell is dropped or corrupted, another will be retransmitted by NCHARGE to the CCP.

D. Reconfiguration Control (RC)

The Reconfiguration Control (RC) module prevents data and state loss when reconfiguring hardware modules in the RAD FPGA. In order to reconfigure a hardware module in the RAD FPGA, configuration data is sent to the NID where it is stored in an off-chip SRAM. When all configuration data has been received for the new module, the NID initiates a reconfiguration handshake with the RC in the RAD FPGA, designating the ModuleID of the module which is to be reconfigured.

III. SOFTWARE COMPONENTS

NCHARGE is the software component that controls reprogrammable hardware on a switch. Figure 2 shows the role of NCHARGE in conjunction with multiple FPX devices within a switch. The software provides connectivity between each FPX and multiple remote software processes via TCP sockets that listens on a well-defined port. Through this port, other software components are able to communicate to the FPX using its specified API. Because each FPX is controlled by an independent NCHARGE software process, distributed management of entire systems can be performed by collecting data from multiple NCHARGE elements [12].
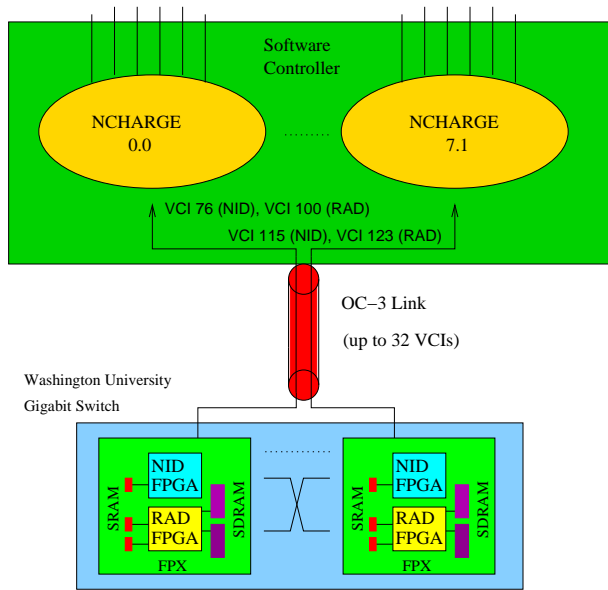
Fig. 2. Detail of the hardware and software components that comprise the FPX system. Each FPX is controlled by an NCHARGE software process. The contents of the memories on the FPX modules can be modified by remote processes via the software API to NCHARGE.

Fig. 3. Example of software applications communicating with NCHARGE

A. NCHARGE Features

NCHARGE provides an API for debugging, programming, and configuring an FPX. Specifically the API includes commands to: check the status of an FPX, configure routing on the NID, perform memory updates, and perform full and partial reprogramming of the RAD. NCHARGE also provides a mechanism for applications to define their own custom control interface.

Control cells are transmitted by NCHARGE and processed by CCPs on the RAD or the NID. To update and configure routes for traffic flows, NCHARGE writes commands to modify routing entries on the gigabit switch or on the NID. To check the status of the FPX, NCHARGE sends a control cell to the NID or RAD and waits for a response.

To modify the content of memories attached to the FPX, NCHARGE supports several commands to perform memory updates. Applications can read or write words to the SRAM and SDRAM memory modules. These memory updates are packed into control cells and sent to the FPX. The number of updates that fit into a single control cell depends on the number of consecutive updates at a particular location in memory and the size of the memory update. The size of the control word allows for up to eight 32-bit address/data pairs to be specified in a single message. For full-width, 36-bit memory words in SRAM, up to six words can be specified in a single control cell. For the 64-bit words in SDRAM, up to four words can be updated at once.
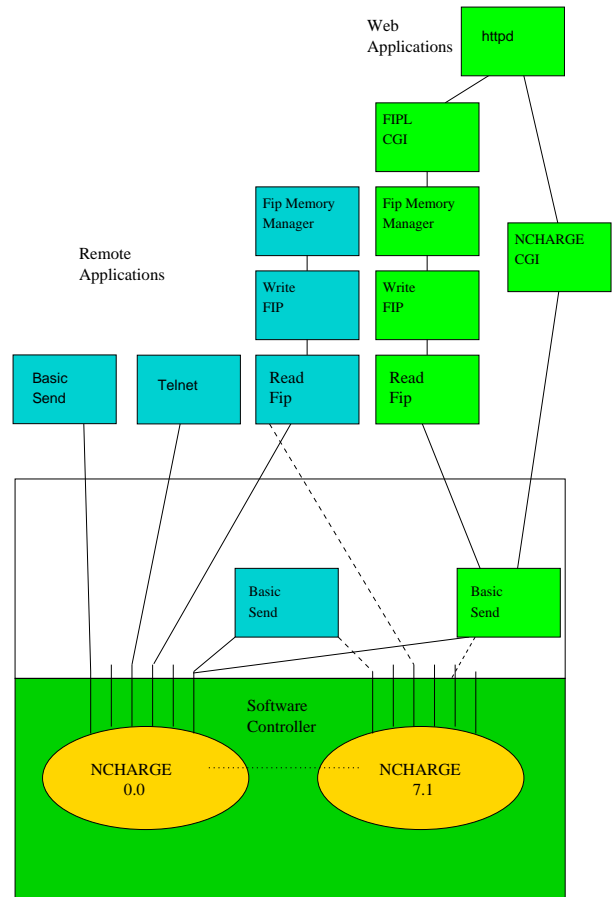
Sending designs to an FPX via control cells is done by

loading the NID memory with the design and then issuing another control cell for the NID to program the RAD. The content of memory can be a complete bitfile or a partial bitfile, as would be produced by a tool like PARBIT[13]. NCHARGE allows for a user to program a section of the RAD or the entire FPGA.

B. NCHARGE API

NCHARGE communicates with a wide range of software through the use of sockets. The API defines a set of text strings that comprise of NCHARGE commands. Once the command is sent to NCHARGE over its TCP socket the user will then receive a response control cell with any relevant return information as well as the indication of a successful transmission.

C. Web Communication

All of the functionality of NCHARGE is brought to the web to allow a simple straight forward interface. The web interface also provides switch level configuration and maintenance.

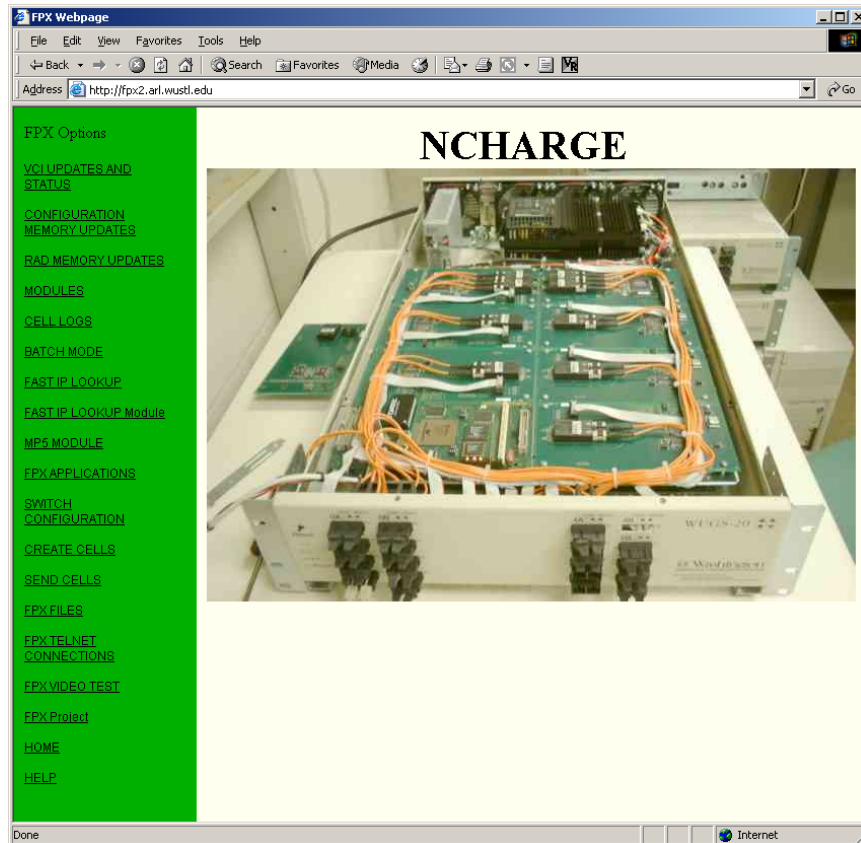A screen shot of the web page is shown in Figure 4.

Fig. 4. Screen shot from the FPX Web Interface

This is the main page of the NCHARGE web interface. From here users select which commands to issue to an FPX.

The mechanism for handling the web-based commands is through the use of CGI scripts. Forms are processed by scripts, which in turn send commands to the NCHARGE software. The communication between NCHARGE and the web interface is done using a TCP socket program called 'basic_send'. Basic_send is discussed in more detail in the next section.

D. Cell Logging

NCHARGE can log control cells sent from and received by the control software. Logging can be enabled, disabled through the web or command line interface. Futher, the contents of the logs can be viewed via the web. Each log is a text file that contains the content of the cell and time the cell was transmitted. Figure 5 shows how logging can be enabled for cells transmitted to and from the NID and/or the RAD.

NCHARGE and the web interface look for files in a particular directory, for all references made to loading files or read from files. A user may upload or edit files over the network by using the web interface to access the
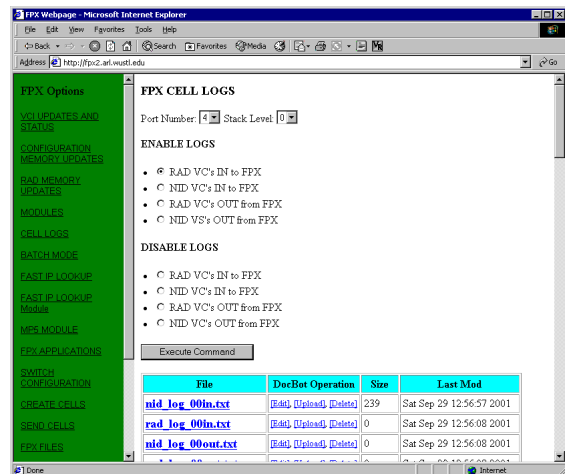

Fig. 5. Screen shot from of the FPX Web Interface showing the cell logging functions

documents. The user has the ability to modify, upload or delete any file in NCHARGE public directory.

E. Socket API

NCHARGE monitors a well-known TCP port, then accepts control connections via TCP/IP sockets. The software multiplexes control of the hardware among multiple software clients. One mechanism for issuing single commands is through a 'basic_send' application. This program accepts control commands and returns the result via a remote control cell. These commands are the same as those that would be entered manually at a console or through a telnet session. Figure 3 illustrates how different applications can compose layered modules to communicate with NCHARGE through sockets.

F. VCI Updates and Status

NCHARGE allows user to issue a read status command to the NID to display the current state of the NID FPGA. Fields are set to display the current state of the FPX and a few special purpose fields such as the RAD Programming byte count and the TYPE LINK. The RAD Programming Byte Count lists the number of bytes that the NID has programmed on the RAD. The TYPE LINK specifies the type of link connected to the FPX, which includes OC3, OC48, GLink, and Gigabit Ethernet.

G. Read and Write Memory

The SRAM and SDRAM memory on the RAD can be updated by issuing memory reads and writes. NCHARGE includes a web-based interface to issue these updates, which is shown in Figure 6 .The user specifies the type of memory operation (Read or Write) the memory width (32, 36, or 64bits), the number of consecutive updates, (depends on the width, max of 8 for 32 bit, 2 for 36 bit, and 4 for 64 bit values), the memory bank (0 or 1), the Module number (default is 0 for the SRAM and SDRAM) as well as a starting address. The 32 and 36 bit memory operations are for SRAM while the 64 bit memory operations are for SDRAM. By default, data is specified by the API in hex. Data can also be specified as strings of ASCII as shown in Figure 6. A read string option is also available to let users read strings out of text.

The time required to reprogram the RAD over the network is governed by: (1) the time to send the bitfile over the network, (2) the time to transfer the configuration data into the FPGA.

H. Configuration Memory Updates

When programming the RAD with modules the user first must load the compiled FPGA bitfile onto the reconfigurable memory attached to the NID, as shown in Figure 7. Once it is loaded a separate command is issued to program the RAD, or a portion of the RAD. The Load RAD Memory command reads a file from the FPX switch controller and places it in the NID's SRAM. Once loaded in the NID, a user may issue a partial or full reprogram of
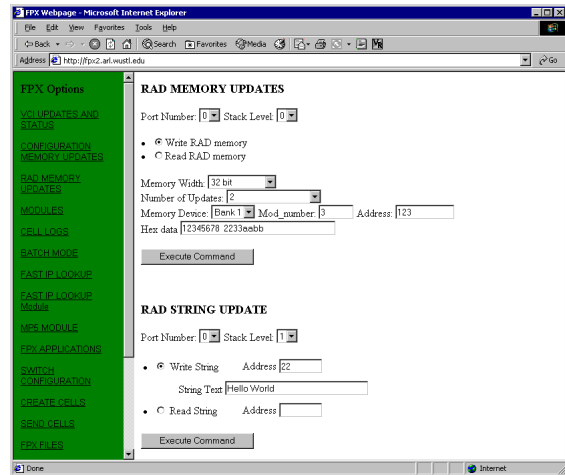

Fig. 6. Screen shot from of the FPX Web Interface showing memory updates and string updates

the RAD using the NID's contents. When issuing the partial or full reprogram of the NID a user may also specify the offset in memory where the RAD to stored the bitfile.

Loading a RAD design is done with the command Load RAD Memory radio button as shown in Figure 8, where filename is the bitfile to send to the FPX. Partial and Full Programming are issued from the web page as well, here word_offset is the location in the NID memory to start reading data from and byte count indicates the number of bytes that should be read to program the RAD. To perform both the loading of a bitfile and the programming of the RAD, the Complete Configuration option was added. This allows a user to simply click here and load their hardware module. This performs the load command and the Full Programming command.

Using a Pentium III 500MHz PC running NetBSD, benchmarks were performed to determine the actual time it takes to issue a Complete Configuration. For an 800KByte bitfile, a complete configuration takes 4.5 seconds. The limited rate of reconfiguration is caused by the use of the Stop and Wait protocol to issue program cells. By removing the Stop and Wait and launching cells at it as fast as possible that time drops to 2.8 seconds. Removing the CRC check brings the complete configuration time down to 1.9 seconds.

I. Batch Mode

NCHARGE can be used to issue a group of commands to be executed consecutively. The user specifies the commands in a text file using the command line format and enters the filename on the web page as shown in Figure 9. NCHARGE then executes the sequence of commands by sending and receiving control cells to and from the NID.
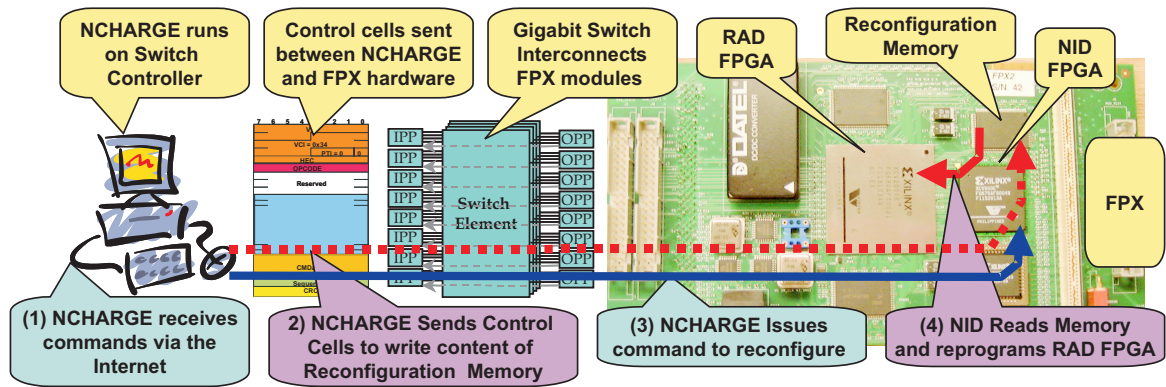
Fig. 7. Diagram of control cells being sent from NCHARGE over a gigabit switch to program an FPX
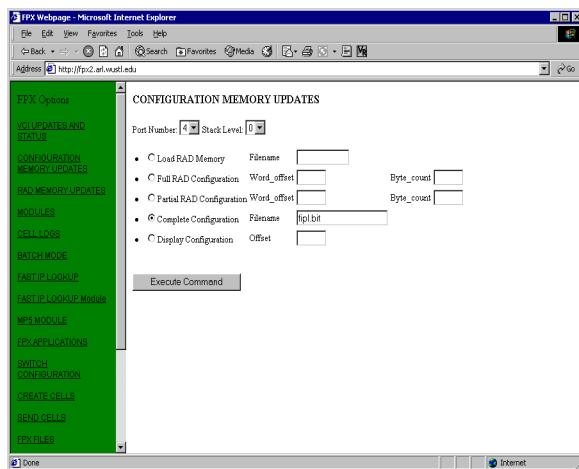

Fig. 8. Screen shot from of the FPX Web Interface that allows a user to load NID configuration and program the RAD
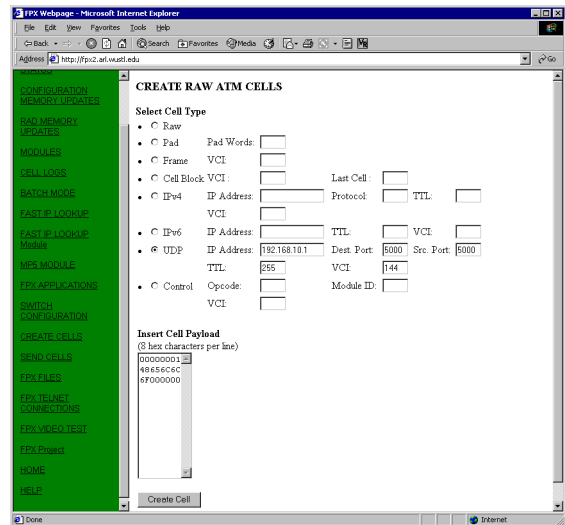

Fig. 10. Screen shot from of the FPX Web Interface showing the Create and Cells page
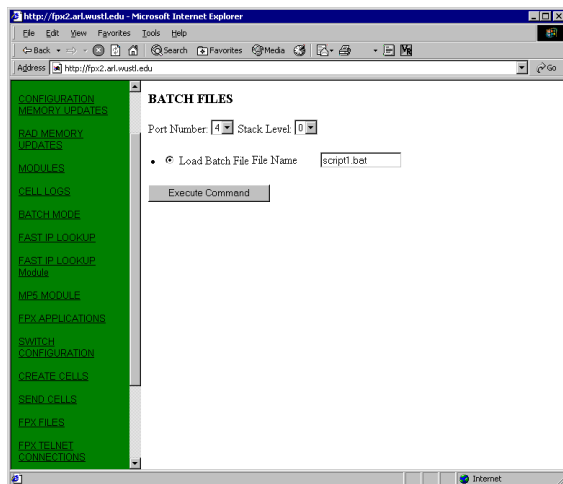

Fig. 9. Screen shot from of the FPX Web Interface that allows a user to issue a set of commands to the FPX

J. NCHRARGE Library calls

The NCHARGE tools include a C application programming interface library that allows communication with the FPX without explicit use of sockets. Functions defined in this API can be used to perform most types of FPX operations.

K. Test Cell Generation

NCHARGE includes software to send and receive raw ATM cells to the FPX. This feature is handled through the web interface to allow a user to specify a payload and the type format. Cells maybe formatted as AAL0/AAL5 Frames, IP packets, or UDP datagrams. Cells are sent, NCHARGE waits for a response on a specified VCI and reports a timeout if no cells were returned to the switch controller.
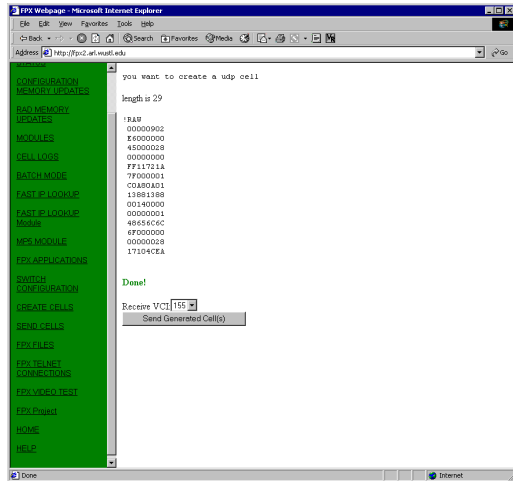
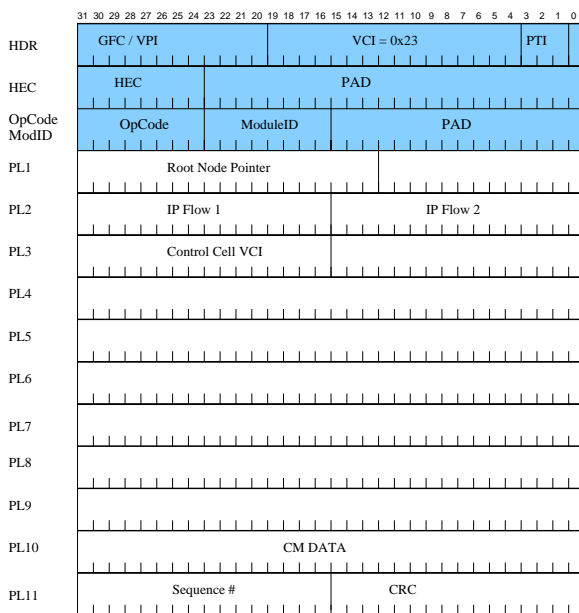Fig. 11. Second Page after creating cells and waiting to send cells

Fig. 12. Layout of a software plug-in module inside a control cell

L. Extendable Modules

NCHARGE allows unique control cells to be defined for customized communication between hardware and software. These custom control cells give a user complete control over the payload (12 words) of a cell. Figure 12 shows a sample control cell that has been customized to communicate parameters for the Fast IP Lookup Module.

M. Example: Fast IP Lookup Memory Manager

The Fast Internet Protocol Lookup (FIPL) engine, developed at Washington University in St. Louis, is a high-performance, solution to perform IP address lookups. It performs a longest prefix match over 32 bit destination ad-

dresses. This module is used to implement Internet Routing on the FPX. [14]

The Fast IP Lookup (FIPL) memory manager is a stand alone application that accepts commands to add, delete, and update routes. The program maintains a trie data structure in memory on the processor and then pushes updates to the SRAM on the FPX.

The Fast IP Lookup (FIPL) application is loaded as a module on the FPX. A few other software components interconnect the FIPL memory manager with remote hosts and the NCHARGE software. The first piece of software is 'write_fip', this accepts FIPL memory manager commands on a specified TCP port from the httpd web server that processed the commands. It then forwards the commands to the FIPL memory manager. These commands are of the form 'Add route $A_1.A_2.A_3.A_4$/netmask nexthop', 'Delete Route $A_1.A_2.A_3.A_4$/netmask'. FIPL memory manager processes these route updates and outputs memory update commands suited for NCHARGE. These memory updates are then piped to another application which reads in multiple memory updates and issues them to NCHARGE over a TCP socket. NCHARGE then packs the memory updates into as few cells as possible and sends them to the FPX attached to the switch. The overall flow of data with FIPL and NCHARGE is shown in Figure 13.

The format of control cells sent between hardware and software is broken down into fields, as shown in Figure 14. These fields are defined in an XML style format. The first field is the module name. This is a simple string identifier that displays to the user which modules are currently installed on NCHARGE. This field identifies the module name as well as version number. The second field enumerates the input opcodes. In the example shown, the user is specifying several different fields, called the 'Root Node Pointer', 'IP Flows 1 and 2', as well as the 'Control Cell VCI'.

Inside the input opcodes field, the first line declares the Root Node Pointer variable. This variable will be used as a parameter for opcode 0x10. The next XML field defines the output opcodes. These fields define the return cells sent from FPX as decoded by NCHARGE. The output code lists the opcode, the number of variables to read, the output syntax, as well as the specified variables. Here the Root Node Pointer has one variable to read for its response.

The next XML field is the fields section. This area declares all variables used by the custom control software as well as its location in the control cell. The parameters here are the variable name, start word in the data, most significant bit of the control cell, stop word in the data, and the least significant bit.

The final field is the help section. This is a string field that gives the user the keyboard arguments associ-
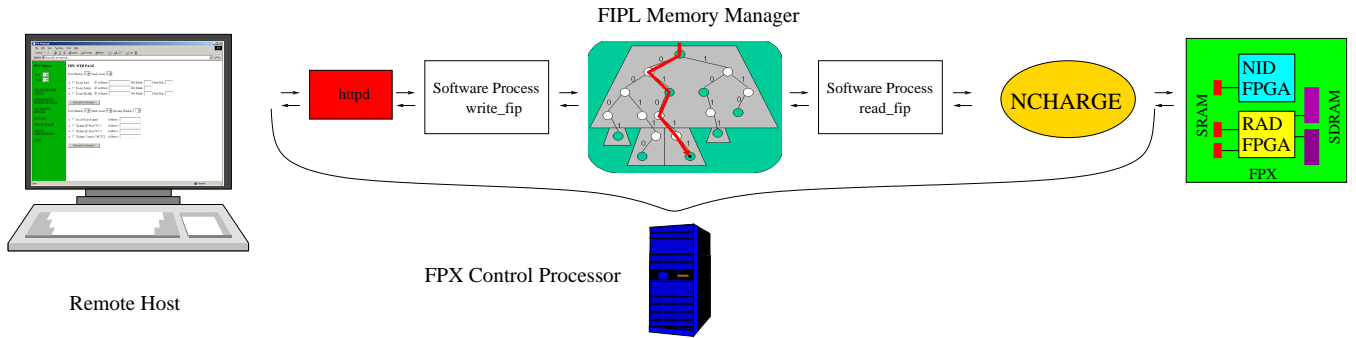
Fig. 13. Example of data flow from Web Interface to FPX

ated with each command. This is useful for the command line mode of the control software to provide a summary of the plug-in modules features. This field is not necessary in a web implementation of a module.

To use the customized control cells, a module is first loaded into NCHARGE. From the web page, the module to load is selected as well as the ID used to reference the module. After loading the module the probe command can be issued to verify the module was loaded and list other models loaded into NCHARGE. To use a module, commands can be issued from the generic model interface on the web or through a custom web page for their specific module. When a user wishes to finish using the module they may unload it by issuing the Unload Module Command specifying the module id.

IV. PERFORMANCE

For applications like FIPL, the rate at which memory can be updated is a major factor in the overall performance of the application. In order to gain insight into the performance of NCHARGE, several experiments were conducted to measure the memory throughput as a function of word width and number of updates packed into a transaction. Figure 15 shows how the memory throughput increases for larger numbers of updates for each of the three types of memory operations supported on the FPX (32-bit, 36-bit, and 64-bit). The discontinuity of the performance is due to the number of address/data memory update pairs that fit into a single message. For 32-bit data operations, the number of updates is limited to eight. Thus, when a transaction includes a 9th memory update, two messages must be transmitted from software to hardware. Similar jumps are visible for the 36-bit and 64-bit updates, occurring at six and four updates per message, respectively.

Figure 16 shows how the memory throughput of NCHARGE is affected by the processor performance, the error control protocol, and the application type. Two processors were used to evaluate the performance: an AMD Athlon running at 1100 MHz and an Intel Pentium Pro running at 200 MHz, both machines communicated with the WUGS via a Gigabit link interface card. The performance was evaluated both with and without the use of the Stop and Wait error control protocol. The Stop and Wait protocol is not needed if the network connection between hardware and software is error-free (i.e., cells are never dropped or corrupted). Lastly, the performance was measured for both local and remote applications. For local applications, a procedure to send control messages was run directly on the local FPX control processor, while for remote applications, all transactions were conducted though an additional TCP/IP socket.

The best performance of 200,000 updates per second, was obtained for local applications transmitting messages without stop and wait on the fast processor . Disabling the Stop and Wait protocol on the fast processor increases performance significantly because the throughput is limited only by the rate at which the processor can issue commands. The performance is degraded by approximately 25% for applications that use the TCP/IP socket interface. The performance is degraded further when NCHARGE runs over an unreliable link using the Stop and Wait protocol. For the slow processor, the performance does not vary as much with the changes in protocols because the software is limited by the performance of the processor.

V. RELATED WORK

There are other systems that perform reconfigurable computing on FPGAs. SPLASH 2 [15] was an early implementation of reconfigurable computing that led to the commercial developement of the Wildforce, Wildstar, and Wildfire [16] boards from AnnapolisMicro. The Wildstar contains a Virtex FPGA and is attached to the PCI bus of a host machine. The control of the Wildstart products is via the Wild Application Programming Interface (Wild API). The device also allows for 'Internet Reconfiguration'. This is made possible through the use of the JBits Interface software [17].

Some of the applications developed for the FPX have been previously implemented on an FPGA. Specifically,

```
<module>
Fast IP Lookup Example Module 1.0
</module>
<input_opcodes>
0x10,R,1,Root_Node_Pntr,
0x12,I,1,IP_Flow_1,
0x14,F,1,IP_Flow_2,
0x16,B,2,IP_Flow_1,IP_Flow_2,
0x18,C,1,Control_Cell_VCI,
</input_opcodes>
<output_opcodes>
0x11,1,Root node pntr is,Root_Node_Pntr,
0x13,1,IP Flow 1 updated to ,IP_Flow_1,
0x15,1,IP Flow 2 updated to ,IP_Flow_2,
0x17,2,2 IP Flows updated to ,IP_Flow_1,IP_Flow_2,
0x19,1,New Control Cell VCI is ,Control_Cell_VCI,
</output_opcodes>
<fields>
Root_Node_Pntr,x,1,31,1,13,
IP_Flow_1,x,2,31,2,16,
IP_Flow_2,x,2,15,2,0,
Control_Cell_VCI,x,3,31,3,16,
</fields>
<help>
R Root Node Pointer address update: R address (hex)
I Update IP Flow 1: I address (hex)
F Update IP Flow 2: F address (hex)
B Update IP Flows 1 and 2: b address1 address2 (hex)
C Update Control Cell VCI: C VCI (hex)
</help>
```

Fig. 14. Fast IP Lookup Software Plug-in Module

Fig. 15. Memory throughput as a function of 32-bit, 36-bit, and 64-bit memory sizes and number of updates
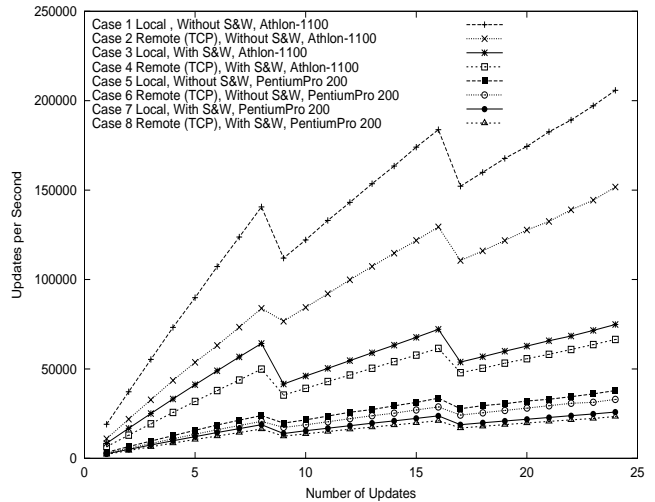

Fig. 16. Memory throughput as a function of processor (AMD Athlon 1100 vs. Pentium Pro 200), error control mechanism (Stop and Wait vs. none), and application type (local application vs. remote TCP socket)

circuits which provide IP routing have been implemented. The previous work, however, does not scale, update, or route as fast as FIPL [18]. An API for reconfigurable computing has also been developed to control and communicate with an arbitrary device. NCHARGE sets itself apart from these solutions in that it has the ability to program an FPGA over the Internet, perform partial programming of an FPGA, and can deliver customized control interfaces to software applications.

VI. CONCLUSION

A suite of tools called NCHARGE has been developed to manage the reconfigurable hardware within an Internet router or firewall. NCHARGE provides an efficient mechanism to monitor the status of remote hardware, configure network traffic flows, reprogram hardware, and perform updates to memory. The rate at which NCHARGE can update remote memory has been measured to be 200,000 updates per second. NCHARGE is able to load a bitfile to the NID program memory in 1.8 seconds. The time to program the RAD from the NID is 16 milliseconds. A standard API has been defined in NCHARGE to communicate between software applications and hardware modules. NCHARGE supports an XML-like language to define an API and control message format for customized hardware modules. This feature was used to implement the communication interface between the hardware and software components of the Fast IP Lookup (FIPL) algorithm. Lastly, a web-based interface has been implemented to provide an intuitive interface to the hardware of the Field Programmable Port Extender.
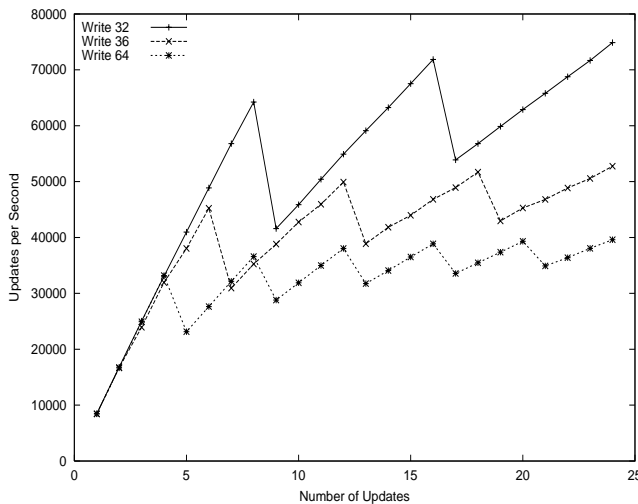
<div align="center">REFERENCES</div>

[1] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, Monterey, CA, USA, Feb. 2001, pp. 87–93.

[2] John W. Lockwood, Jon S. Turner, and David E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, Monterey, CA, USA, Feb. 2000, pp. 137–144.

[3] Toshiaki Miyazaki, Kazuhiro Shirakawa, Masaru Katayama, Takahiro Murooka, and Atsushi Takahara, "A transmutable telecom system," in *Proceedings of Field-Programmable Logic and Applications*, Tallinn, Estonia, Aug. 1998, pp. 366–375.

[4] J. W. Lockwood, S. M. Kang, S. G. Bishop, H. Duan, and A. Hossain, "Development of the iPOINT testbed for optoelectronic asynchronous transfer mode networking," in *International Conference on Information Infrastructure*, Beijing, China, Apr. 1996, pp. 509–513.

[5] Hamish Fallside and Michael J. S. Smith, "Internet connected FPL," in *Proceedings of Field-Programmable Logic and Applications*, Villach, Austria, Aug. 2000, pp. 48–57.

[6] Sumi Choi, John Dehart, Ralph Keller, John Lockwood, Jonathan Turner, and Tilman Wolf, "Design of a flexible open platform for high performance active networks," in *Allerton Conference*, Champaign, IL, 1999.

[7] John W Lockwood, "An open platform for development of network processing modules in reprogrammable hardware," in *IEC DesignCon'01*, Santa Clara, CA, Jan. 2001, pp. WB–19.

[8] Jon S. Turner, Tom Chaney, Andy Fingerhut, and Margaret Flucke, "Design of a Gigabit ATM switch," in *INFOCOM'97*, 1997.

[9] David E. Taylor, Jon S. Turner, and John W. Lockwood, "Dynamic hardware plugins (DHP): Exploiting reconfigurable hardware for high-performance programmable routers," in *IEEE OPENARCH 2001: 4th IEEE Conference on Open Architectures and Network Programming*, Anchorage, AK, Apr. 2001.

[10] David E. Taylor, John W. Lockwood, and Sarang Dharmapurikar, "Generalized RAD Module Interface Specification on the Field Programmable Port Extender (FPX)," `http://www.arl.wustl.edu/arl/projects/fpx/references`, Jan. 2001.

[11] David E. Taylor, John W. Lockwood, and Naji Naufel, "Rad module infrastructure of the field programmable port extender (fpx)," `http://www.arl.wustl.edu/arl/projects-/fpx/references/`, Jan. 2001.

[12] James M. Anderson, Mohammad Ilyas, and Sam Hsu, "Distributed network management in an internet environment," in *Globecom'97*, Pheonix, AZ, Nov. 1997, vol. 1, pp. 180–184.

[13] Edson Horta and John W. Lockwood, "PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs)," Tech. Rep. WUCS-01-13, Washington University in Saint Louis, Department of Computer Science, July 6, 2001.

[14] David E. Taylor, John W. Lockwood, Todd Sproull, and David B. Parlour, "Scalable IP Lookup for Programmable Routers," `http://www.cs.wustl.edu/cs/techreports-/2001/wucs-01-33.pdf`, Oct. 2001.

[15] Duncan A. Buell, Jeffrey M. Arnold, and Walter J. Kleinfelder, "Splash 2: Fpgas in a custom computing machine," IEEE Computer Society Press, 1996.

[16] Bradley K. Fross, Dennis M. Hawver, and James B. Peterson, "Wildfire heterogeneous adaptive parallel processing systems," `http://www.annapmicro.com`.

[17] Steven A. Guccione, Delon Levi, and Prasanna Sundararajin, "Jbits: A java-based interface for reconfigurable computing," in *2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, 1999.

[18] Jason Hess, David Lee, Scott Harper, Mark Jones, and Peter Athanas, "Implementation and evaluation of a prototype reconfigurable router," in *IEEE Symposium on FPGAs for Custom Computing Machines*, Kenneth L. Pocek and Jeffrey Arnold, Eds., Los Alamitos, CA, 1999, pp. 44–50, IEEE Computer Society Press.