Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)

John W. Lockwood[*] , Naji Naufel, Jon S. Turner, David E. Taylor
Department of Computer Science
Applied Research Lab
Washington University
1 Brookings Drive
Saint Louis, MO 63130
http://www.arl.wustl.edu/arl/projects/fpx/

ABSTRACT

A prototype platform has been developed that allows processing of packets at the edge of a multi-gigabit-per-second network switch. This system, the Field Programmable Port Extender (FPX), enables packet processing functions to be implemented as modular components in reprogrammable hardware. All logic on the on the FPX is implemented in two Field Programmable Gate Arrays (FPGAs). Packet processing functions in the system are implemented as dynamically-loadable modules.

Core functionality of the FPX is implemented on an FPGA called the Networking Interface Device (NID). The NID contains the logic to transmit and receive packets over a network, dynamically reprogram hardware modules, and route individual traffic flows. A full, non-blocking, switch is implemented on the NID to route packets between the networking interfaces and the modular components. Modular components of the FPX are implemented on a second FPGA called the Reprogrammable Application Device (RAD). Modules are loaded onto the RAD via reconfiguration and/or partial partial reconfiguration of the FPGA.

Through the combination of the NID and the RAD, the FPX can individually reconfigure the packet processing functionality for one set of traffic flows, while the rest of the system continues to operate. The platform simplifies the development and deployment of new hardware-accelerated packet processing circuits. The modular nature of the system allows an active router to migrate functionality from softare plugins to hardware modules.

[*]This research is supported by NSF: ANI-0096052 and Xilinx Corp.

Keywords

FPGA, reconfiguration, hardware, modularity, network, routing, packet, Internet, IP, ATM, processing

Categories and Subject Descriptors

B.4.3 [**Hardware**]: Input/Output and Data Communications—*Interconnections (Subsystems)*; C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Architecture and Design*

1. BACKGROUND

Internet routers and firewalls need high performance to keep pace with the the growing demands for bandwidth. At the same time, however, these devices need flexibility so that they can be reprogrammed to implement new features and functionality. Most of the functionality of a router occurs at the ingress and egress ports of a switch. Routers become bottlenecked at these locations when they are required to perform extensive packet processing operations.

Hardware-based routers and firewalls provide high throughput by including optimized packet-processing pipelines and parallel computation circuits. By using Application-Specific Integrated Circuits (ASICs), traditional routers are able to implement the performance-critical features at line speed. The static nature of an ASIC circuit, however, limits the functionality of these performance-critical features to only a fixed set of the system's functionality.

Software-based based routers and firewalls excel in their ability to implement reprogrammable features. New features can be added or removed in an active router by loading software new software modules. The sequential nature of the microprocessor that executes that code, however, limits the throughput of the system. Routers that solely use software to process packets typically archive throughputs that are several orders of magnitude slower than their hardware-based counterparts.

Routers and firewalls that utilize FPGAs can implement a desirable balance between performance and flexibility [1]. They share the performance advantage of ASICs in that customized pipelines can be implemented and that parallel logic functions can be performed over the area of the device They also share the flexibility found in software systems to be reconfigured [2].

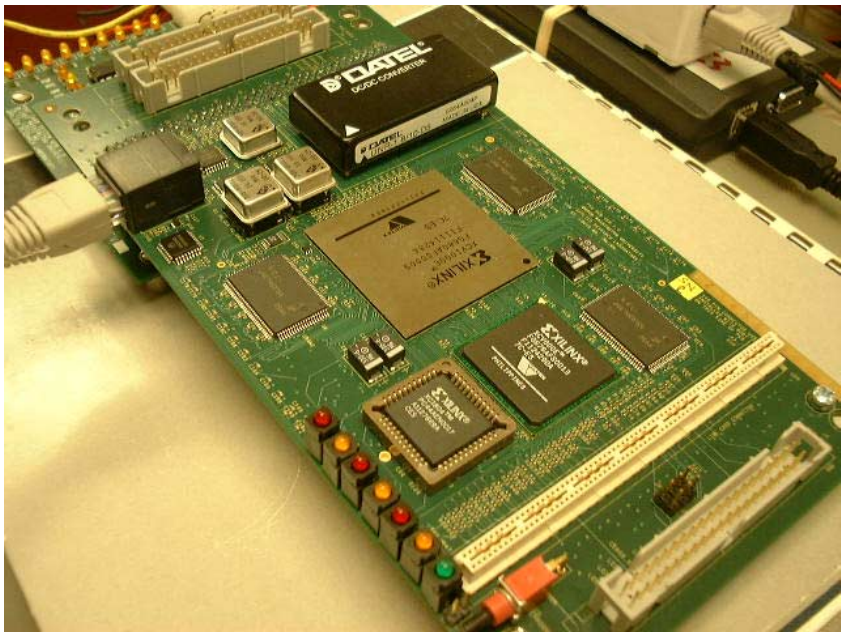

Figure 1: FPX Module

2. THE FPX SYSTEM

The Field-programmable Port Extender enable the rapid prototype and deployment of hardware components for modern routers and firewalls [3]. The system is intended to allow researchers or hardware developers to quickly prototype new functionality in hardware, then download that functionality into one or more nodes in a network. The architecture of the FPX makes it well suited to implement applications like IP routing, per-flow queuing [4], and flow control algorithms [5] in hardware.

Components of the FPX include two FPGAs, five banks of memory, and two high-speed network interfaces. 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.

A photograph of the FPX module is shown in Figure 1. The FPX circuit itself is implemented as a 12-layer printed circuit board with the dimensions 20 cm × 10.5 cm. The FPX has two Xilinx Virtex FPGAs: a XCV600Efg676 and a XCV1000Efg680. Future versions of the FPX will utilize a larger, pin-compatible XCV2000Efg680 to provide additional logic resources for user-defined functionality.

SRAM components on the FPX are mounted above and below the RAD. SDRAM memories are mounted on the back of the FPX in sockets. Reconfiguration data for the FPGAs are stored both in non-volatile Flash memory for the NID and SRAM memory for the RAD.

The FPX integrates with another open hardware platform called the Washington University Gigabit Switch (WUGS) [6]. Figure 2 shows an FPX mounted in one port of a WUGS. By inserting one or more FPX modules at each port of the switch, parallel FPX units are used to simultaneously process packets as they pass through the network on all ports of the switch.

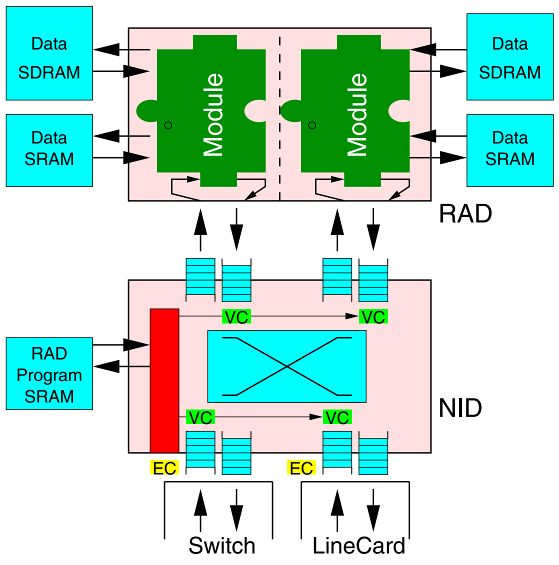

Figure 3: NID and RAD Configuration

2.1 Logical Configuration

The FPX implements all logic using two FPGA devices: the Network Interface Device (NID) and the Reprogrammable Application Device (RAD). The interconnection of the RAD and NID to the network and memory components is shown in Figure 3.

The RAD contains the modules that implement the module-specific functionality. Each module on the RAD connects to one Static Random Access Memory (SRAM) and to one, wide Synchronous Dynamic RAM (SDRAM). In total, the modules implemented on the RAD have full control over four independent banks of memory. The SRAM is typically used for applications that need to implement table

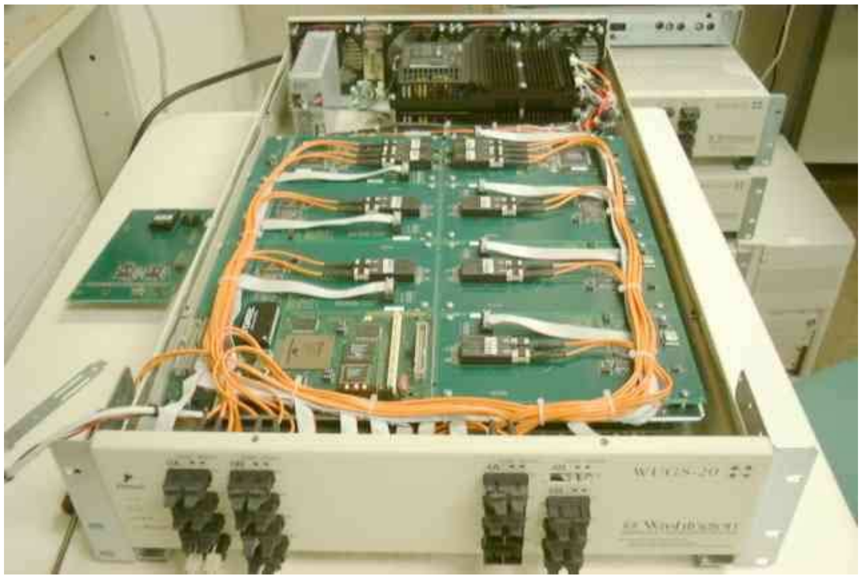

Figure 2: Photo of FPX mounted in WUGS Switch

lookup operations (like the Fast IP lookup algorithm), while the SDRAM interface is typically used for applications like packet queuing that transfer bursts of data and can tolerate a higher memory latency.

The RAD communicates with the NID using a Utopia-like interface. Packets on this interface are segmented into a sequence of fixed-size cells that are formatted as IP over ATM. Each interface has a small amount of buffering and implements flow control. A Start of Cell (SOC) signal is asserted at the input of a module to indicate the arrival of data. The Transmit Cell Available (TCA) signal is asserted back towards an incoming data source to indicate downstream congestion.

3. NETWORK INTERFACE DEVICE

The Network Interface Device (NID) on the FPX controls how packet flows are routed to and from modules. It also provides mechanisms to dynamically load hardware modules over the network and into the router. The combination of these features allows these modules to be dynamically loaded and unloaded without affecting the switching of other traffic flows or the processing of packets by the other modules in the system.

As shown in Figure 4, The NID has several components, all of which are implemented in FPGA hardware. It contains a four-port switch to transfer data between ports; *Virtual Circuit lookup tables* (VC) on each port in order to selectively route flows; a *Control Cell Processor* (CP), which is used to process control cells that are transmitted and received over the network; logic to reprogram the FPGA hardware on the RAD; and synchronous and asynchronous interfaces to the four network ports that surround the NID.

3.1 Per Flow Routing

The NID routes flows among the modules on the RAD and the network interfaces to the switch and line card using a four-port switch. Each traffic flow that arrives on any incoming port can be forwarded to any destination port.

Each of the NID's four interfaces provide a small amount

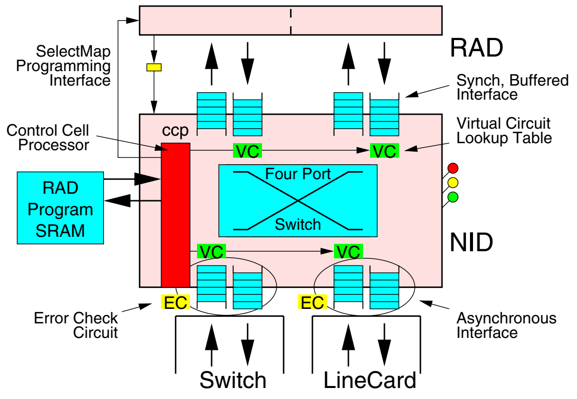

Figure 4: Network Interface Device Configuration

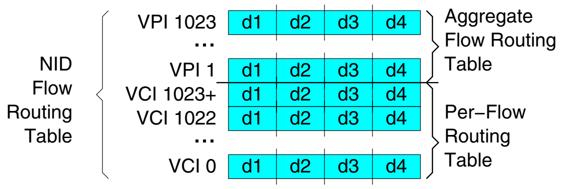

Figure 5: Per-flow routing on the NID

of buffering for short-term congestion. Buffers on the NID are implemented using on-chip memory. When packets contend for transmission to the same destination port, the NID performs arbitration. For longer term congestion, the NID avoids data loss by sending a back-pressure signal to the previous module or network interface along the that network flow's path. The design of the four-port switch and scheduling algorithm used to arbitrate among flows is based on the design of the iPOINT switch [7] [8].

IP Packets are routed through the FPX and switch based on assignment of cell headers that transport that packet.

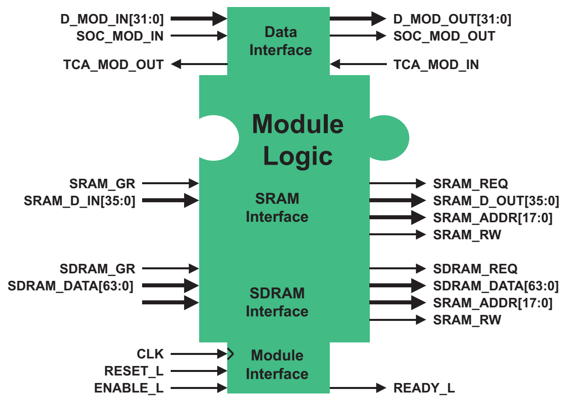

Figure 7: Modular Component of FPX

The NID supports forwarding for both aggregate traffic flows and individual traffic flows. The NID's *Virtual Circuit lookup table (VC)* maps these flows into next-hop destinations at each of the four ports. As shown in Figure 5, the NID's flow routing table contains entries for each of the four ports in the switch that identify the destination port ($d_1 \ldots d_4$) of each flow. The table has sufficient entries to support 1024 virtual paths for aggregated traffic and another 1024 virtual circuits for individual flows.

Examples that illustrate the NID's switching functionality are shown in Figure 6. By default, cells are simply passed between the line card interface and the switch. To implement egress flow processing (i.e., process packets as they exit the router), the NID routes a flow from the switch, to a RAD module, then out to the line card. Likewise, to implement ingress cell processing, the NID routes a virtual circuit from the line card, to a RAD module, then out to the switch. Full RAD processing occurs when data is processed in both directions by both modules on the RAD. Loopback and partial loopback testing can be programmed on the NID to debug experimental modules. Modules can implement selective packet forwarding by reassignment of the headers that transport each packet.

3.2 Control functions

The NID implements a *Control Cell Processor (CCP)* in hardware to manage the operation of the FPX and to communicate over the network. On the ingress interface from the switch, the CCP listens and responds to commands that are sent on a specific virtual circuit. The NID processes commands that include: (1) modification of per-flow routing entries; (2) reading and writing of hardware status registers, (3) reading and writing of configuraton memory, and (4) commands that cause the logic on the RAD to be reprogrammed. After executing each command, the NID returns a response in a control cell.

3.3 FPX Reprogrammability

In order to reprogram the RAD over the network, the NID implements a reliable protocol to fill the contents of the on-board RAM with configuration data that is sent over the network. As each cell arrives, the NID uses the data and the sequence number in the cell to write data into the *RAD Program SRAM*. Once the last cell has been correctly received, and the FPX holds an image of the reconfiguration

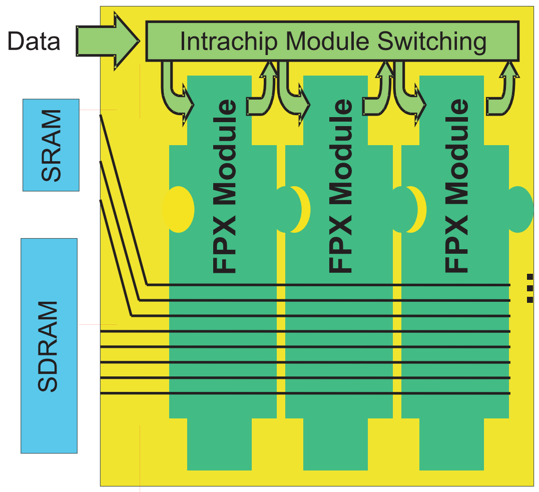

Figure 8: Larger Configuration of FPX Modules

bytestream that is needed to reprogram the RAD. At that time, another control cell can be sent to NID to initiate the reprogramming of RAD using the contents of the *RAD Program SRAM*.

The FPX supports partial reprogramming the RAD by allowing configuration streams to contain commands that only program a portion of the logic on the RAD. Rather than issue a command to reinitialize the device, the NID just writes the frames of reconfiguration data to the RAD's reprogramming port. This feature enables the other module on the RAD to continue processing packets during the partial reconfiguration. Similar techniques have been implemented in other systems using software-based controllers [9] [10].

3.4 Modular Interface

Application-specific functionality is implemented on the RAD as modules. A modular interface has been developed that provides a standard interface to access packet content and to interface with off-chip memory.

Hardware plugin modules on the RAD consist of a region of FPGA gates and internal memory, bounded by a well-defined interface to the network and external memory. Currently, those regions are defined as one half of an FPGA and a fixed set of I/O pins.

The modular interface of an FPX component is shown in Figure 7. Data arrives at and departs from a module over a 32-bit wide, Utopia-like interface. Data passes through modules as complete ATM cells. Larger IP datagrams pass through the interface in multiple cells.

The module provides two interfaces to off-chip memory. The SRAM interface supports transfer of 36-bit wide data to and from off-chip SRAM. The Synchronous Dynamic RAM (SDRAM) interface provides a 64-bit wide interface to off-chip memory. In the implementation of the IP lookup module, the off-chip SRAM is used to store the data structures of the fast IP Lookup algorithm [3].

3.5 Larger Configurations

As the capacity of FPGAs increases, it is possible to integrate more modules together onto the same FPGA. In

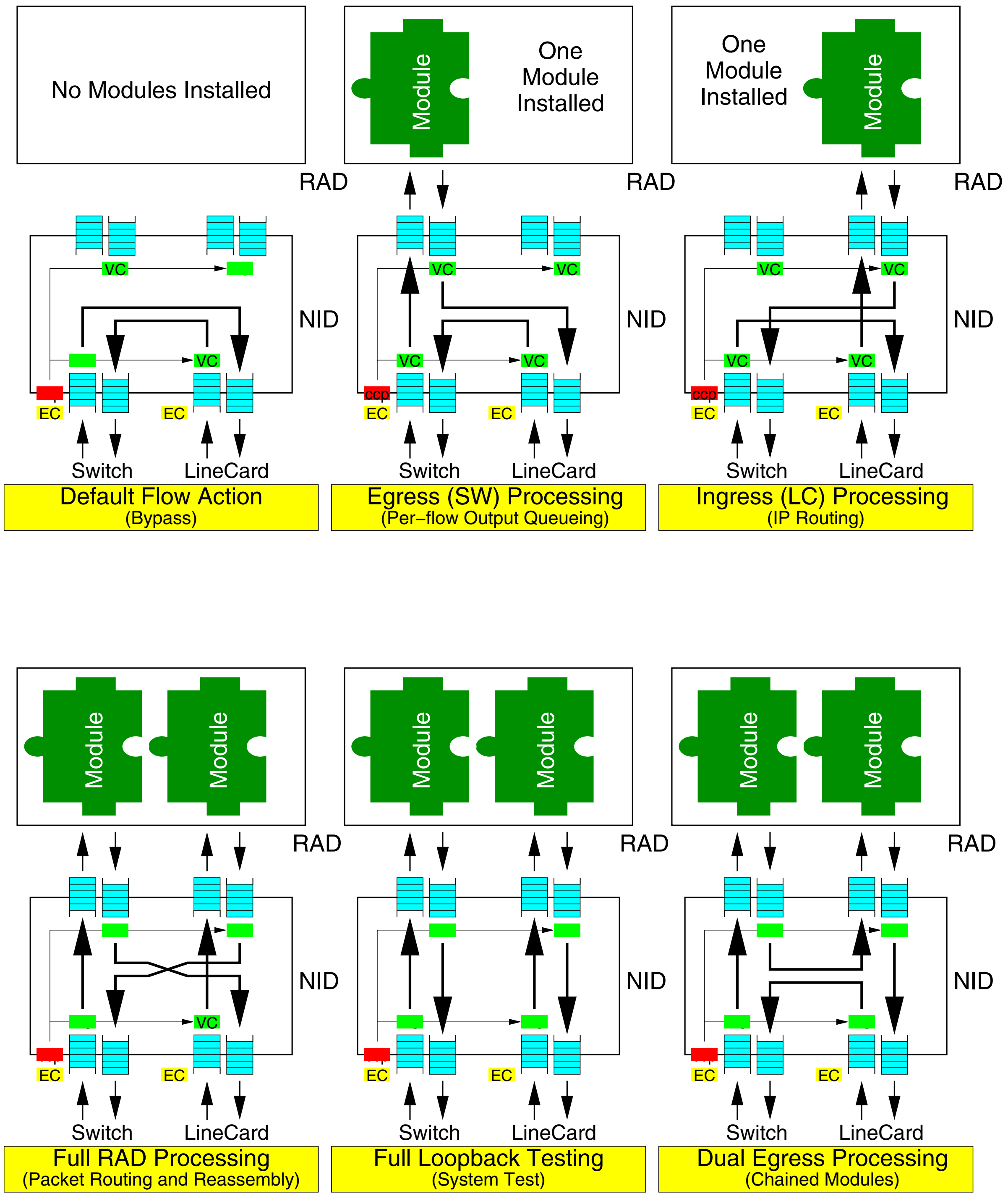

Figure 6: NID Switching Functionality

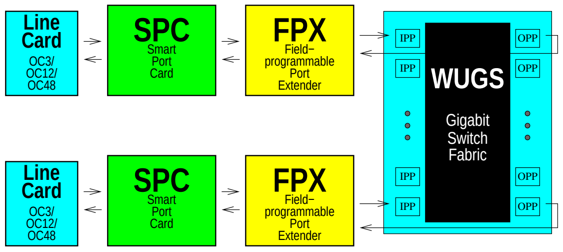

Figure 9: FPX/SPC Physical Configuration

the current system, each module occupies one-half of an XCV1000E. In future systems, a larger number of modules can fit into a higher-capacity FPGA.

The interface that is used by hardware modules in the existing system is designed to remain unchanged in a larger configuration. Applications retain the same interfaces and timing relationships to the data bus, memory buses, and control lines. Hardware modules need only be resynthesized so that they fit into a fixed region of the new device.

A configuration of larger configuration of FPX modules is shown in Figure 8. Physically, each module occupies a fixed region of the FPGA device. Intra-chip routing interconnects the data paths between the modules. Common signal lines internconnect the modules with SRAM and SDRAM.

3.6 Combined Hardware/Software Modules

Complex packet processing algorithms can require both hardware and software components [14] [15].Through another research project, an active processing node called the Smart Port Card (SPC) has been built that contains a routing chip called the APIC and an embedded Pentium processor that runs the NetBSD kernel [16] [17]. When combined with the FPX, interesting applications can be implemented which perform active processing in both hardware and software. The physical configuration of the combined system is shown in Figure 9. Physically, the FPX and SPC stack between the ports of the WUGS switch.

The logical configuration of the combined system is shown in Figure 10. Packets enter and exit the combined system at the interfaces to the switch fabric and line card. Data flows can be selectively forwarded between the network interfaces, the hardware modules on the FPX, and the software modules on the SPC by configuration of virtual circuits and virtual paths on the NID and APIC.

As an example of an application that can utilize both hardware and software modules to process packets, consider the implementation of an Internet router in the combined system. Packet flows would enter the line card and be forwarded by the APIC and NID to a hardware module on the FPX. This hardware module, in turn, performs a fast IP lookup on the packet, then forwards standard packets on a pre-established virtual circuit that leads through the NID and switch fabric to the appropriate egress port [3]. For packets that require non-standard processing, such as those with IP options, the hardware module can forward the packet to a software module on the SPC for further processing. By using both hardware and software, the combined system can provide high-throughput for the majority of the packets and full processing functionality for exceptions.

3.7 Networking Testbed

The FPX provides an open-platform environment that can be used for the rapid prototype of a broad range of hardware-based networking components. Through a National Science Foundation grant, FPX hardware can be made available to researchers at Universities interested in developing networking hardware components [11].

The implementation of a sample FPX module, including the VHDL source code and I/O pin mappings on the RAD, can be downloaded from the web [12]. Other modules have been developed for the FPX to perform IP routing, packet buffering, and packet content modification. Details about the FPX are available on the project website [13].

4. CONCLUSION

The Field Programmable Port Extender enables customized packet processing functions to be implemented as modules which can be dynamically loaded into hardware over a network. The Networking Interface Device implements the core functionality of the FPX. It allows the FPX to individually reconfigure the packet processing functionality for a set of traffic flows, while not disrupting the packet processing functions for others. The modular design of the FPX has made the system of interest for active networking systems, as it allows customized applications to achieve the higher performance via hardware acceleration.

5. ACKNOWLEDGMENTS

The authors would like to thank Dave Parlour of Xilinx for his support of this project and for his research on the mechanisms to implement interfaces between on-chip modules.

6. REFERENCES

[1] S. Hauck, “The roles of FPGAs in reprogrammable systems,” *Proceedings of the IEEE*, vol. 86, pp. 615–638, Apr. 1998.

[2] W. Marcus, I. Hadzic, A. McAuley, and J. Smith, “Protocol boosters: Applying programmability to network infrastructures,” *IEEE Communications Magazine*, vol. 36, no. 10, pp. 79–83, 1998.

[3] J. W. Lockwood, J. S. Turner, and D. E. Taylor, “Field programmable port extender (FPX) for distributed routing and queuing,” in *FPGA’2000*, (Monterey, CA), pp. 137–144, Feb. 2000.

[4] H. Duan, J. W. Lockwood, S. M. Kang, and J. Will, “High-performance OC-12/OC-48 queue design prototype for input-buffered ATM switches,” in *INFOCOM’97*, (Kobe, Japan), pp. 20–28, Apr. 1997.

[5] M. Bossardt, J. W. Lockwood, S. M. Kang, and S.-Y. Park, “Available bit rate architecture and simulation for an input-buffered and per-vc queued ATM switch,” in *GLOBECOM’98*, (Sydney, Australia), pp. 1817–1822, Nov. 1998.

[6] J. S. Turner, T. Chaney, A. Fingerhut, and M. Flucke, “Design of a Gigabit ATM switch,” in *INFOCOM’97*, 1997.

[7] J. W. Lockwood, H. Duan, J. J. Morikuni, S. M. Kang, S. Akkineni, and R. H. Campbell, “Scalable optoelectronic ATM networks: The iPOINT fully functional testbed,” *IEEE Journal of Lightwave Technology*, pp. 1093–1103, June 1995.

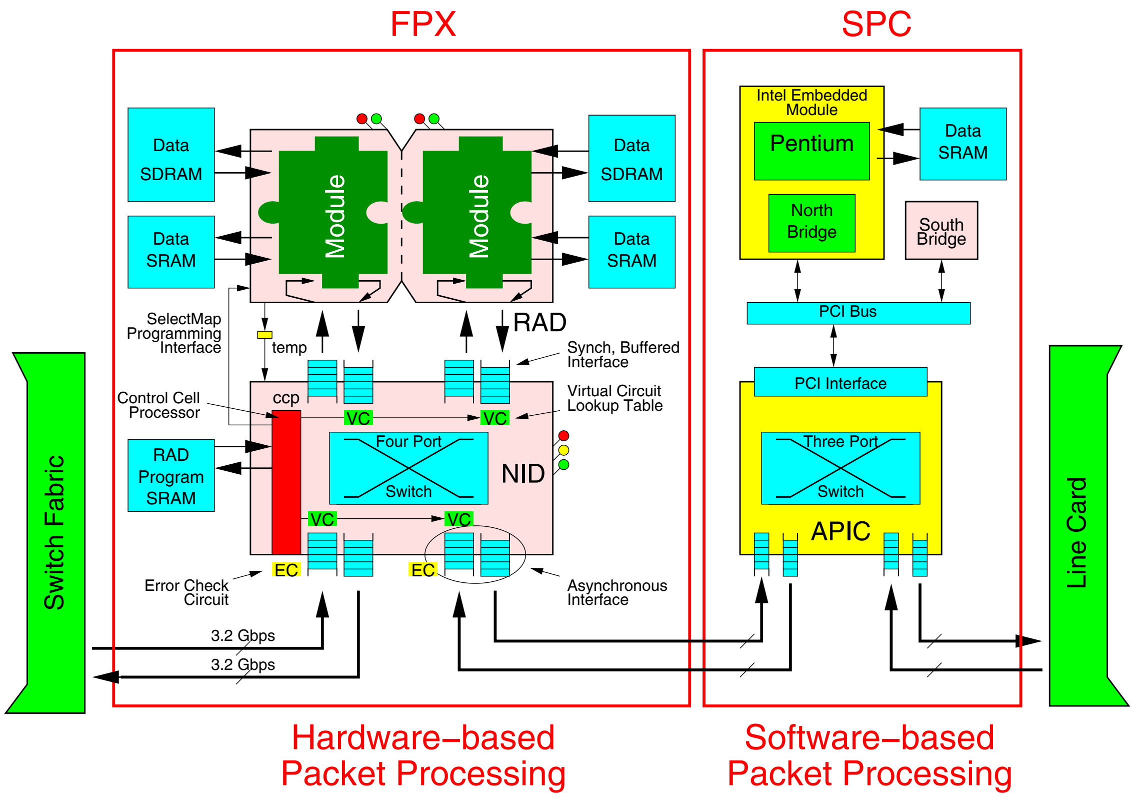

Figure 10: Combined FPX/SPC Logical configuration

[8] "Illinois Pulsar-based Optical Interconnect (iPOINT) Homepage." `http://ipoint.vlsi.uiuc.edu`, Sept. 1999.

[9] W. Westfeldt, "Internet reconfigurable logic for creating web-enabled devices." Xilinx Xcell, Q1 1999.

[10] S. Kelem, "Virtex configuration architecture advanced user's guide." Xilinx XAPP151, Sept. 1999.

[11] J. S. Turner, "Gigabit Technology Distribution Program." `http://www.arl.wustl.edu/gigabitkits/kits.html`, Aug. 1999.

[12] J. Lockwood and D. Lim, "Hello World: A simple application for the field programmable port extender (FPX)," tech. rep., WUCS-00-12, Washington University, Department of Computer Science, July 11, 2000.

[13] "Field Programmable Port Extender Homepage." `http://www.arl.wustl.edu/projects/fpx/`, Aug. 2000.

[14] S. Choi, J. Dehart, R. Keller, J. Lockwood, J. Turner, and T. Wolf, "Design of a flexible open platform for high performance active networks," in *Allerton Conference*, (Champaign, IL), 1999.

[15] D. S. Alexander, M. W. Hicks, P. Kakkar, A. D. Keromytis, M. Shaw, J. T. Moore, C. A. Gunter, J. Trevor, S. M. Nettles, and J. M. Smith in *The 1998 ACM SIGPLAN Workshop on ML / International Conference on Functional Programming (ICFP)*, 1998.

[16] J. D. DeHart, W. D. Richard, E. W. Spitznagel, and D. E. Taylor, "The smart port card: An embedded Unix processor architecture for network management and active networking," *IEEE/ACM Transactions on Networking*, Submitted July 2000.

[17] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner, "Design issues for high performance active routers," *IEEE Network*, vol. 13, Jan. 1999.