Protocol Wrappers for Layered Network Packet Processing in Reconfigurable Hardware

A NETWORK PLATFORM CALLED THE FIELD-PROGRAMMABLE PORT EXTENDER (FPX) STREAMLINES AND SIMPLIFIES NETWORK TRANSMISSION PROCESSING DIRECTLY IN HARDWARE.

Florian Braun
University of Stuttgart

John Lockwood
Washington University in St. Louis

Marcel Waldvogel
IBM Zurich Research Laboratory

A library of layered protocol wrappers processes Internet packets in reconfigurable hardware. Collectively, the wrappers simplify and streamline the implementation of high-level networking functions by abstracting the operation of lower-level packet processing functions. The library synthesizes into field-programmable gate array (FPGA) logic and is utilized in a network platform called the field-programmable port extender (FPX). The library processes asynchronous transfer mode (ATM) cells, ATM adaptation layer 5 (AAL5) frames, Internet protocol (IP) messages, and user datagram protocol (UDP) packets directly in hardware.[1]

Applications can process data at several layers of the protocol stack using the library of wrappers discussed in this article. Layers are important for networks because they let applications abstract from above and below details of the network protocols. At the lowest layer, networks modify raw data passing between interfaces. At higher levels, the applications process variable length frames or IP packages. For example, an Internet router or firewall uses the IP, frame, and cell wrapper together with a circuit to perform routing lookups. At the user level, a network application may transmit directly or receive UDP messages by instantiating all wrappers, as shown in Figure 1.

Background

The FPX is a networking platform that processes packets in reprogrammable hardware. The platform allows modular hardware components to be dynamically loaded into an FPGA device over a network. The FPX is part of a larger set of networking, switching, routing, and active networking hardware and software components developed at Washington University in St. Louis. The modules described in this article are primarily targeted for the FPX, though we wrote the design in portable VHSIC hardware description language (VHDL) suited for use in any FPGA-based system.

Switch fabric

The central component of our research is the Washington University gigabit switch (WUGS).[2] This fully featured ATM switch can handle up to 20 Gbps of network traffic.

0272-1732/02/$17.00 © 2002 IEEE

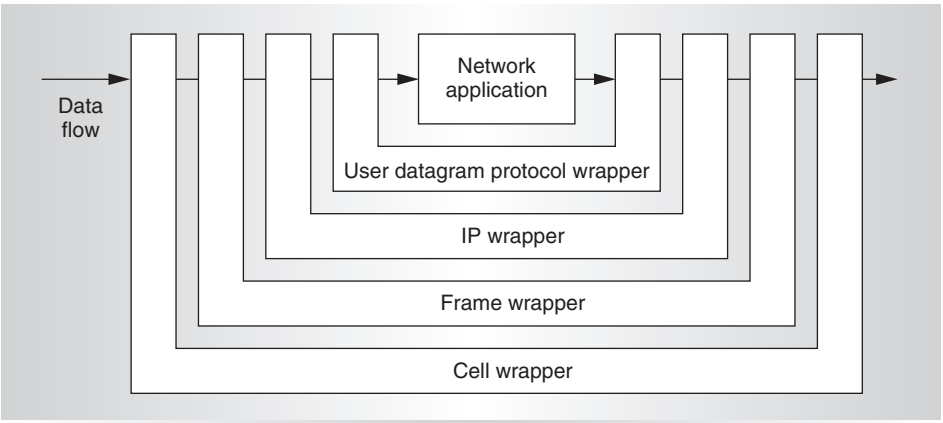

Figure 1. The wrapper concept.

The WUGS main circuit board has eight connectors used to attach line cards through the front of the unit. The WUGS enables hardware insertion between the line cards and the backplane in a daisy-chain fashion, providing configuration flexibility.

There are two extension cards for the WUGS so far, both providing programmable means for advanced cell and/or packet processing. The smart port card (SPC)[3] provides an Intel Pentium processor for implementing user-specified packet processing functions. The SPC attaches to the network via an advanced port interconnect[4] ATM network interface card. We use the SPC whenever the processing applied to packets is suited for software implementation. The FPX card[5,6] provides reprogrammable logic for user applications. Like the SPC, we can insert the FPX between the switch fabric mainboard and any line card, as shown in Figure 2. Figure 3 (next page) shows the major components on an FPX board.

The FPX contains two FPGAs: the network interface device (NID) and the reprogrammable application device (RAD). The NID interconnects the WUGS, the line card, and the RAD via an on-chip ATM switch core. The NID also provides logic to dynamically reprogram the RAD. Programming the RAD to hold user-defined modules enables network modules to be dynamically loaded into the system. The RAD connects to two static RAM (SRAM) and two synchronous dynamic RAM (SDRAM) components. The memory modules can cache cell data or hold large tables.

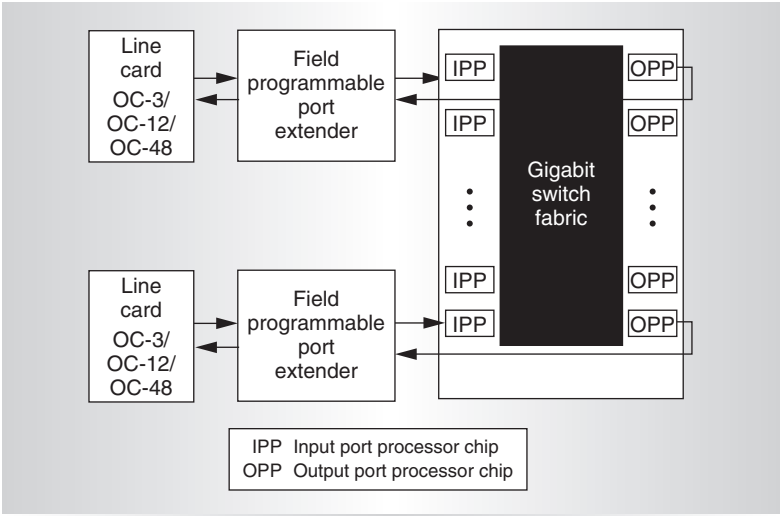

Figure 2. The WUGS configuration using the field-programmable port extender.

FPX modules

Our approach uses modules to implement user applications on the RAD. Modules are hardware components that contain well-defined interfaces. These interfaces communicate with the RAD and other infrastructure components. The basic data interface is a 32-bit wide Utopia. Utopia uses a single signal indicating the beginning of a new ATM cell, whereas the data is sent as a burst of 14 consecutive 32-bit words on the data bus, with the first two holding the ATM header. Internet packets enter the module using classic IP over ATM encapsulation and segmentation into ATM cells.[7] The data bus carries cell headers and payloads. The other signals in the module interface control congestion and con-

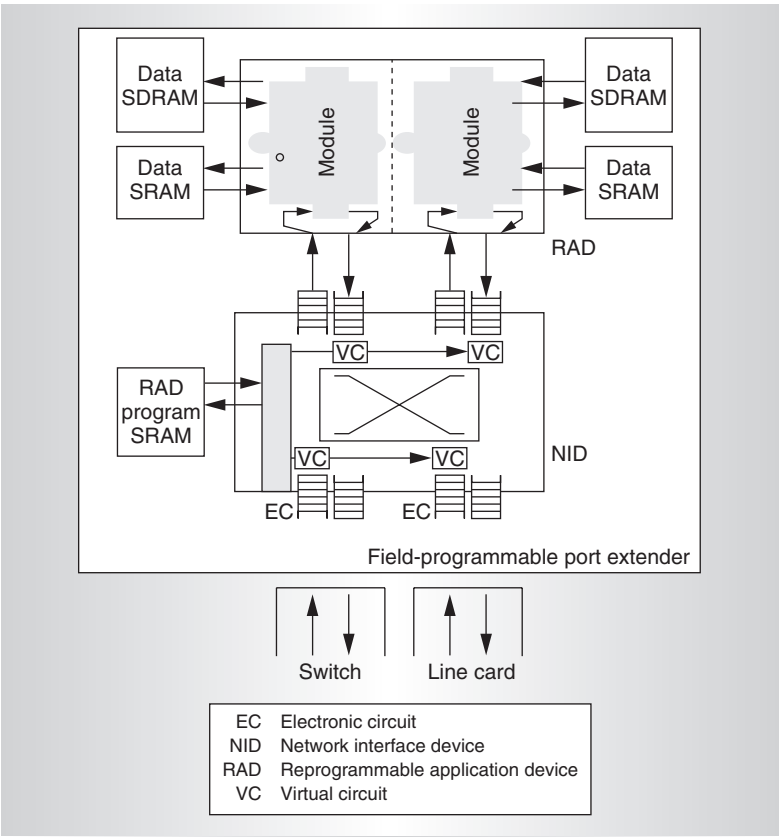

Figure 3. Components on a field-programmable port extender.

nect to memory controllers to access the off-chip memory.[8]

Usually, there are two application modules on the RAD. Typically, one handles data from the line card to the switch (ingress), and the other handles data from the switch to the line card (egress). Programmers can replace the modules at any time by partially reprogramming the RAD FPGA. (Documentation and source code to many of the FPX modules are available at http://www.arl.wustl.edu/arl/projects/fpx/).

Network wrapper concept

Components using the FPX let applications handle data on several protocol layers. Similar circuits implement IP over Ethernet[9] data in static systems. Instead of offloading protocol processing to a coprocessor,[10,11] FPX components let hardware implement all packet-processing functions.

Translation steps are necessary between layers. A classical approach creates components for each protocol translation. We combine these two translation units into one component that has four interfaces: two support the lower level protocol and two provide a higher-level interface. Some components connect to exchange additional information or to bypass the application. The latter occurs in the cell processor.

Protocol wrappers surround the user's application logic like the letter "U," as shown in Figure 1. Regarding the data stream, the application only connects to the translating component, which wraps up the application itself. Therefore, we refer to the surrounding components as wrappers.

To support higher levels of abstraction, we can nest the wrappers. Each has a well-defined interface for an outer and an inner protocol level and, therefore, fit together as shown in Figure 1. As a result, we get a modular design method to support applications for different protocols and levels of abstraction. Associating each wrapper with a specific protocol, we get a layer model comparable to the well-known Open Systems Interconnection (OSI) Basic Reference Model. This modularity lets application developers implement functions at several protocol layers in their designs. They can interface their logic to a wrapper with the level of abstraction appropriate for the specific application. User-level applications, for example, can completely ignore handling of complicated protocol issues, such as frame boundaries or checksums.

When referring to specific components, we refer to the single translation component as *processor*, and the combination of processors make up a *wrapper* for this protocol level. For instance, the IP processor performs the translation steps on IP packets only, whereas the IP wrapper also includes the frame processor to handle AAL5 frames.

Cell wrapper

The wrapper on the lowest level is the cell processor, as shown in Figure 4. The cell processor performs every necessary step on the cell level common to all FPX modules. It checks incoming ATM cells against their header error control (HEC) field, which is part of the 5-octet header. An 8-bit cyclic redundancy code (CRC) prevents corrupted cells from misrouting. If the check fails, the cell is dropped.

This level processes accepted cells according to their virtual circuit information. The

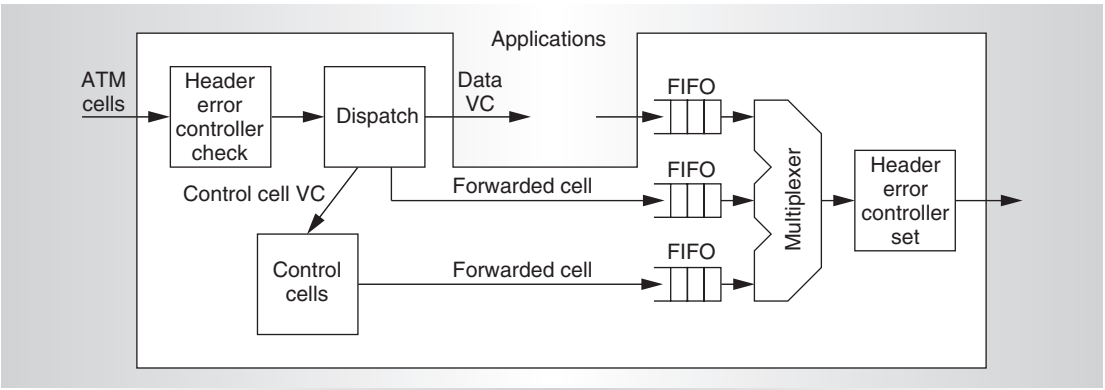

Figure 4. Field-programmable port extender cell wrapper.

cell processor distinguishes between three different flow types:

- The cell is on the data virtual circuit for this module. In this case, the cell is forwarded to the inner interface of the wrapper and thus to the application.
- The cell is on the control cell virtual circuit and tagged with the correct module identification. The cell processor itself processes control cells.
- None of the above—the cell is not destined for this module. These cells are forwarded around the inner layers of the module and bypass processing by the higher-level protocol processors.

The cell processor has three FIFO pipes to buffer cells from the three paths. A multiplexer combines the cells and forwards them to their final stop. A new HEC is computed before the cell leaves the cell processor.

Control cells modify the behavior of an FPX module. Control cells are ATM cells with a well-defined structure that provide a communication path between an external controller, for example, software, and the on-chip modules. A standard control cell format is used on the FPX. Control cells to the RAD contain a module identification field opcode to address the application module. All FPX modules understand a common subset of generic opcodes. For example, commands to change the virtual path and virtual channel identifier can always be specified to let a module dynamically change the flow processed by the hardware.

The design of the control cell handling functions inside the cell processor is very flexible. Application developers can easily extend the control cell functionality to fit their modules' needs. User applications typically support more control cell opcodes than the generic subset. The extensions are typically used to interact with remote software components. An important goal of our research was extendibility in the design of the cell processor. A control cell processing framework checks and generates CRCs, buffers common data structures, and implements a mechanism to share common information.

A master state machine waits for control cells destined for its module, and, after receiving them, stores opcodes, user data, and a sequence number. The master state machine also checks the control cell CRC. Every opcode has its own state machine; therefore, adding a new command does not interfere with implementation of existing commands. Every state machine polls the master state to check if the cell processor has received a control cell with a valid CRC; the state machine becomes active upon recognizing its control cell opcode. For any incoming control cell (request), the module should send a response cell if the command has been processed successfully. Since an independent state machine handles every opcode, generating its own individual response cells, a multiplexer merges the response cells at the output port and then sets the CRC.

Frame wrapper

To handle data with arbitrary lengths, several adaptation layers exist for ATM networks to organize data in frames and send it as multiple cells.[12] AAL5 is widely used for IP networks.[7] It allows efficient transmission of

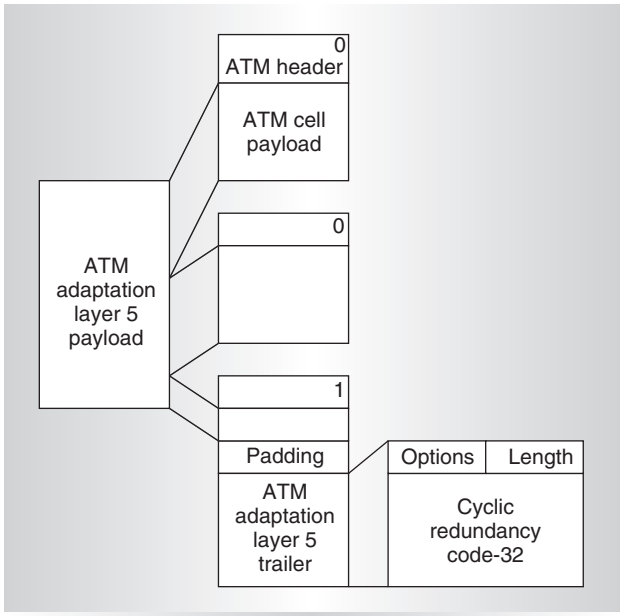

Figure 5. ATM adaptation layer 5 frame segmentation.

packets longer than a single ATM cell over ATM links. During encapsulation into AAL5, the higher layer packet receives padding and an 8-byte appended trailer (see Figure 5). The amount of padding fills the resulting frame to an even multiple of 48 bytes (the size of a single ATM cell). The trailer contains the original payload length (16 bits), a 16-bit wide field available to higher protocol layers, and a 32-bit CRC for integrity checks. A special bit set in the last ATM cell header enables decapsulation. The length field and this last cell bit let the decoder identify the payload start and end. The length and CRC fields identify lost, inserted, and corrupted cells in the stream.

The FPX's frame wrapper module handles AAL5 frame data. The module's interface enables application modules to transmit and receive variable length frames. The frame processor replaces the start of cell signal with three signals, namely start of frame, end of frame, and data enable (DataEn).

The start-of-frame signal indicates the transmission of a new frame. Note that HEC support is not available with this wrapper, because it is assumed that only valid ATM cells pass to this wrapper and that the cell processor will generate a valid HEC for outgoing cells.

DataEn indicates valid payload data. Acting as an enable signal for the data processing application, DataEn is completely independent from the cell structure. Applications can therefore easily resize frames or append data, facilitating new frame generation. The frame processor does not assert DataEn when padding is sent, because it is not part of the actual frame contents. The end-of-frame signal is asserted with the last valid payload word sent. This gives applications enough time to start appending data to a frame, if necessary.

After the end-of-frame signal, the frame wrapper sends two additional 32-bit words. These 8 octets represent the AAL5 trailer and include some additional information, which helps the wrapper recreate the length and CRC fields. It is essential that applications copy and forward these two additional words, even if they do not inspect or modify them. To achieve the desired modularity of our system, the frame processor is not (and in fact should not be) aware of the modifications carried out at higher processing layers. We therefore were unable to improve the efficiency of the CRC calculation by applying techniques we developed earlier for fast incremental CRC updates in IP over ATM networks.[13]

IP packet wrapper

IP processing is a critical feature of the wrappers. IP dictates packet formatting on the Internet. Subprotocols, such as UDP or transmission-control protocol (TCP), send connectionless datagrams or establish reliable connections by inserting their own payloads into IP packets. The Applied Research Lab at Washington University in St. Louis developed an IP processor to support IP-based applications. Our IP processor inherits the signaling interface from the frame processor, and adds a start-of-payload signal, to indicate the payload coming after the IP header, which can be of variable length. This wrapper serves three purposes:

- Checks the IP header integrity to verify the correctness of the header checksum. Corrupted packets are dropped.
- Decrements the time-to-live (TTL) field. With RFC 1812,[14] all IP processing entities must decrement this field. Once this field reaches zero, packet forwarding should stop. This prevents packets from looping in networks owing to misconfigured routers.

Table 1. Wrapper implementation results.

Wrapper	Space		Speed	Delay (short)		Delay (long)		Throughput (short)		Throughput (long)	
processor	Lookup tables	Relative (%)	(MHz)	In	Out	In	Out	Relative (%)	Gbps	Relative (%)	Gbps
Cell	781	3	125	4	6	4	6	100	3.5	100	3.5
Frame	1,251	5	116	21	22	10	31	84	2.7	93	3.0
IP	1,009	4	109	36	39*	24	197*	84	2.6	93	2.9
UDP	550	2	114	39	44*	27	202*	84	2.6	93	2.9

*Depending on packet size.

- Recomputes the length and the header checksum on outgoing IP packets.

An IP header usually has a length of 20 bytes, or 5 words, but can be longer in the rare case that it contains IP options. Before the IP processor can make a decision about a header's integrity, the entire header must pass through the wrapper. The IP processor computes and then compares the header checksum. On a failure, the IP packet is dropped; a signal is not propagated to the application. If an incoming packet's TTL field is already zero, the IP processor also drops the packet and instead returns an Internet control message protocol (ICMP) error packet. Otherwise, the TTL field is decremented. Buffering the outgoing IP packets allows the IP processor to determine the actual length by counting the words received from higher protocol layers. The IP processor sets the corresponding field in the header and recomputes the header checksum accordingly. Therefore, a whole packet must be buffered before it is sent.

Because the IP wrapper needs to buffer outgoing packets anyway, it provides a buffer update service to higher layers, freeing them from buffering the packet themselves. After an IP packet has been completely received and buffered, but before it is forwarded to the higher-layer wrapper, the IP processor can apply changes to the packet payload for fields—such as in a header—set when the packet originally streamed through the hardware. We have extended the interwrapper protocol for IP applications to support this update feature. Update commands are optional and the IP processor inserts them between the last payload word (when it asserts the end-of-frame signal) and the AAL5 trailer. Our approach uses an unused bit (bit 15) in the AAL5 length to indicate update words or the start of the trailer. We also use the length field to hold an error code to enable packet dropping before it is sent. Recall that the frame processor recomputes the word count of the payload by monitoring the DataEn signal. Therefore only the least significant two bits can ever be nonzero in a correct length field, indicating the number of bytes valid in the last word. Update words contain a 16-bit update field and a 15-bit update offset address. The update field replaces the 16-bit word at the offset address in the buffer.

UDP datagram wrapper

The UDP processor supports connectionless communication between user level applications using the UDP/IP protocol, sitting on top of IP. This wrapper computes and generates the UDP checksum and the length field in the header for outgoing datagrams. The wrapper checks the checksum on incoming datagrams as well, but the result is only available after the whole packet has passed through the wrapper. The UDP processor uses similar signals as the IP processor. It replaces the start-of-payload signal with the start-of-datagram signal. Applications can simply process datagrams or even generate new ones without interpreting or generating UDP headers.

Buffering the whole packet is necessary before the UDP wrapper can determine the correct checksum for outgoing datagrams. Instead of buffering again, it takes advantage of the IP processor's buffer update feature described previously; this saves memory and other on-chip resources.

Implementation results

We have synthesized wrappers to operate on the RAD FPGA on the FPX. The RAD is a Xilinx Virtex XCV1000E-7 and the FPX system clock is 100 MHz. Table 1 summarizes the results of our framework. The table's space

column gives the number of lookup tables used to implement each function and the relative fraction of the chip required to hold the logic. The speed column specifies the maximum frequency of each synthesized wrapper. The delay columns show delays in clock cycles of data passing through the wrappers and are split into delays before (in) and after (out) an embedded application.

To measure delays, we sent ATM cells back-to-back, containing UDP packets. We used UDP packets with only one word (short) and packets with 512 bytes of payload (long). The short datagrams fit into a single cell and therefore have the highest relative protocol overhead—representing the worst-case scenario. The longer datagrams represent a common size, giving an average delay. Note that the delays marked with an asterisk in the table depend on IP packet length, because the IP wrapper performs a store-and-forward operation.

Table 1's throughput columns show the theoretical relative and absolute maximum throughput in gigabits per second for each wrapper and for both the short and the long UDP packets.

Wrapper example applications

Researchers and graduate networking classes have used the layered protocol library to implement several applications, including encryption, compression, routing, and active processing functions with low overhead in reprogrammable hardware. For each of these applications, the protocol wrapper library processed UDP, IP, AAL5, and ATM headers, whereas, the remaining gates on the FPGA processed packet payloads.[15] Several application level payload-processing modules have been implemented. An implemented run length encoder shortened the length of payloads containing repeated bytes. The circuit replaces runs of repeating bytes with a single byte and a count indicating the length of that run. A corresponding run length decoder restored the content to the original value for use on the remote side of the connection. A simple encryption circuit was also implemented to scramble the data in each byte in the payload. For all of these circuits, throughput exceeded 2.4 Gbps for all packet sizes using the Virtex XCV1000E-7 FPGA on the FPX. The high throughput for large packets results from performing parallel computation on the FPGA using multiple instances of hardware components. The high throughput for small packets is a result of the protocol library's low overhead.

An IP lookup engine has been implemented on top of the IP wrapper to route IP packets.[16] The router runs at the 2.4 Gbps rate of the line card (Sonet OC-48), that is, it handles 6.25 million IP packets per second. The circuit, including the necessary wrappers, occupies only 17 percent of the chip space.

A pair of tunnel modules has been implemented to transport IPv6 packets through an IPv4 network.[17] These modules reside in FPX devices at remote ends of a network. The source module packages IPv6 packets into IPv4 packets then tunnels them into an IPv4 network. The destination module extracts the original IPv6 packets and transmits them on the remote network. Both modules accept control cells to configure the tunnels and specify the destination IPv4 address and IPv6 network address mask. The circuit is compatible with RFC 1933 and achieved a throughput of 2.5 gigabits per second.

In addition to these data intensive processing applications, an active processing module has been implemented on the FPX that included the protocol wrapper library and a soft-core processor called the constant (k) coded programmable state machine (KCPSM) from Xilinx.[18] This FPX KCPSM module implementation showed that processor program memory could be dynamically reprogrammed over the Internet via a single UDP datagram. Once a new program loads into the module, the processor swaps context and implements a new processing function on the payload of the subsequent data packets. The layered protocol library made processor cycles available exclusively to the application, sparing them from the overhead of processing protocol functions.

Although we created our current implementation for use in the FPX, the framework is very general and easily adaptable to other platforms. Developers of networking hardware components can use our framework. The entire IP processing framework only uses 14 percent of the RAD FPGA on the FPX, leaving sufficient space to implement user-

defined logic. Many students have implemented simpler functions as part of their coursework, indicating that the framework is easy to use. MICRO

Acknowledgment

This research was supported in part by NSF ANI-0096052 and Xilinx and was conducted while all of the authors were based at Washington University in St. Louis.

References

1. J.W. Lockwood, "An Open Platform for Development of Network Processing Modules in Reprogrammable Hardware," *Proc. Int'l Engineering Consortium DesignCon* (IEC DesignCon 01), 2001, p. WB-19; http://www.designcon.com/2001/ (current Dec. 2001).
2. T. Chaney et al., *Design of a Gigabit ATM Switch*, tech report WU-CS-96-07, Applied Research Laboratory, Washington Univ. in St. Louis, 1996.
3. W.N. Eatherton and T. Aramaki, *SPC Specification*, working note ARL-WN-98-01, Applied Research Laboratory, Washington Univ. in St. Louis, 1998; http://www.arl.wustl.edu/arl/TechRpts/wn/ps/spcspec.ps (current Dec. 2001).
4. Z. Dittia, G. Parulkar, and J. Cox Jr., "The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques," *Proc. IEEE Infocom*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 179-187.
5. J.W. Lockwood, J.S. Turner, and D.E. Taylor, "Field Programmable Port Extender (FPX) for Distributed Routing and Queuing," *Proc. ACM Int'l Symp. Field Programmable Gate Arrays* (FPGA), ACM Press, New York, 2000, pp. 137-144.
6. J.W. Lockwood et al., "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," *Proc. ACM Int'l Symp. Field-Programmable Gate Arrays* (FPGA), ACM Press, New York, 2001, pp. 87-93.
7. P. Newman et al., IETF RFC 1954, *Transmission of Flow Labeled IPv4 on ATM Data Links*, Internet Engineering Task Force, May 1996; www.ietf.org/rfc (current Dec. 2001).
8. D.E Taylor, J.W. Lockwood, and S. Dharmapurikar, *Generalized RAD Module Interface Specification on the Field Programmable Port Extender (FPX)*, tech. report WU-CS-TM-01-15, Computer Science Dept., Washington Univ. in St. Louis, 2001; http://www.arl.wustl.edu/~det3/fpx_module_interface.pdf (current Dec. 2001).
9. H. Fallside and M.J.S. Smith, "Internet Connected FPGAs," *Proc. 10th Int'l Conf. on Field Programmable Logic and Applications* (FPL), Springer Verlag, Heidelberg, Germany, 2000, pp. 48-57.
10. E.A. Arnould et al., "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers," *Proc. 3rd Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS-III), ACM Press, New York, 1989, pp. 205-216.
11. M. Zitterbart et al., "HeaRT: High Performance Routing Table Look Up," *Proc. IEEE High-Performance Computing Symp.* (HPCS), IEEE CS Press, Los Alamitos, Calif., 1997.
12. Recommendation I.363, *B-ISDN ATM Adaptational Layer AAL Specification*, Int'l Telecommunications Union, Geneva, 1991.
13. F. Braun and M. Waldvogel, "Fast Incremental CRC Updates for IP Over ATM Networks," *Proc. IEEE Workshop High Performance Switching and Routing*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 48-52.
14. *Requirements for IP version 4 routers*, IETF RFC 1812, Internet Eng. Task Force, June, 1995; www.ietf.org/rfc/rfc1812.txt (current Jan. 2002).
15. J.W. Lockwood, "Platform and Methodology for Teaching Design of Hardware Modules in Internet Routers and Firewalls," *Proc. Int'l Conf. Microelectronic System Education* (MSE), IEEE CS Press, Los Alamitos, Calif., 2001, pp. 56-57.
16. F. Braun, J. Lockwood, and M. Waldvogel, "Reconfigurable Router Modules Using Network Protocol Wrappers," *Proc. Field-Programmable Logic and Applications* (FPL), Springer Verlag, Heidelberg, Germany, 2001, pp 254-263.
17. J. Moscola, D. Lim, and A. Tetley, *IPv6 Tunneling Over an IPv4 Network*, Washington University. in St. Louis, 2001, http://www.arl.wustl.edu/~lockwood/class/cs535/project/tunnel/IPv4Tunnel.pdf (Current 15 Jan. 2002).

18. H. Fu and J.W. Lockwood, *The FPX KCPSM Module: An Embedded, Reconfigurable Processing Module for the Field Programmable Port Extender (FPX)*, tech. report WUCS-01-14, Computer Science Dept., Washington Univ. in St. Louis, 2001.

Florian Braun is working toward the Diplom (MS) from the University of Stuttgart, Germany. His research interests include high-speed networking, reprogrammable hardware, and mobile communications. Braun has a Vordiplom (BSc) in computer engineering from the University of Stuttgart, Germany. He worked on reconfigurable networking hardware from 2000 to 2001 as an exchange student at Washington University in St. Louis.

John Lockwood is an assistant professor at Washington University in St. Louis. His research interests include the design of reconfigurable networking hardware systems. Lockwood has a BS, MS, and PhD in electrical engineering from the University of Illinois at Urbana/Champaign. He is a member of the IEEE, the ACM, Tau Beta Pi, and Eta Kappa Nu.

Marcel Waldvogel is a research staff member at IBM Zurich Research Laboratory. His research interests include high-speed networking and data dissemination efficiency. Waldvogel has a PhD in electrical engineering and a Dipl.-Ing. in computer science from the Swiss Federal Institute of Technology (ETH), Zurich. He is a senior member of the IEEE and a member of the ACM.

Direct questions and comments about this article to John Lockwood, Applied Research Laboratory, Department of Computer Science, Washington University, Campus Box 1045, 1 Brookings Drive, St. Louis, MO 63130; lockwood@arl.wustl.edu.