An Extensible, System-On-Programmable-Chip, Content-Aware Internet Firewall[⋆]

John W. Lockwood, Christopher Neely, Christopher Zuver,
James Moscola, Sarang Dharmapurikar, and David Lim

Applied Research Laboratory
Washington University in Saint Louis
1 Brookings Drive, Campus Box 1045
Saint Louis, MO 63130 USA
{lockwood, cen1, cz2, jmm5, sarang, dlim}@arl.wustl.edu
http://www.arl.wustl.edu/arl/projects/fpx/

Abstract. An extensible firewall has been implemented that performs packet filtering, content scanning, and per-flow queuing of Internet packets at Gigabit/second rates. The firewall uses layered protocol wrappers to parse the content of Internet data. Packet payloads are scanned for keywords using parallel regular expression matching circuits. Packet headers are compared to rules specified in Ternary Content Addressable Memories (TCAMs). Per-flow queuing is performed to mitigate the effect of Denial of Service attacks. All packet processing operations were implemented with reconfigurable hardware and fit within a single Xilinx Virtex XCV2000E Field Programmable Gate Array (FPGA). The single-chip firewall has been used to filter Internet SPAM and to guard against several types of network intrusion. Additional features were implemented in extensible hardware modules deployed using run-time reconfiguration.

1 Introduction

Recently, demand for Internet security has significantly increased. Internet connected hosts are now frequently attacked by malicious machines located around the world. Hosts can be protected from remote machines by filtering traffic through a firewall. By actively dropping harmful packets and rate-limiting unwanted traffic flows, the harm caused by attacks can be reduced.

While some types of attacks can be thwarted solely by examination of packet headers, other types of attacks—such as network intrusion, Internet worm propagation, and SPAM proliferation—require that firewalls process entire packet payloads [1]. Few existing firewalls have the capability to scan entire packet payloads. Of those that do, most are software-based and cannot process packets at the high-speed rates used by modern networks. Hardware-accelerated firewalls are needed to process entire packet payloads at high speeds.

[⋆] This research was supported in part by a grant from Global Velocity, the National Science Foundation (ANI-0096052), and a gift from Xilinx

Application Specific Integrated Circuits (ASICs) have been used in firewalls to implement some packet filtering functions. ASICs allow firewalls to achieve high throughput by processing packets in deep pipelines and parallel circuits. But ASICs can only protect networks from threats known to the designer when the chip was fabricated. Once an ASIC is fabricated, its function is static and cannot be altered. The ability to protect networks against both the present and future threats is the real challenge in building modern firewalls [2].

2 System-On-Programmable-Chip (SOPC) Firewall

A System-On-Programmable-Chip (SOPC) Internet firewall has been implemented that protects high-speed networks from present and future threats. In order to protect networks against current threats, the SOPC firewall parses Internet protocol headers, scans packet payloads, filters traffic, and performs per-flow queuing. In order to protect against future threats, the SOPC is highly extensible. This paper details the implementation of that firewall in a single Xilinx Virtex XCV2000E FPGA.

The top-level architecture of the SOPC firewall is shown in Figure 1. When a packet first enters the SOPC firewall, it is processed by a set of *layered protocol wrappers*. These wrappers segment and reassemble frames; verify and compute checksums; and read and write the headers of the Internet packet. Once the packet has been parsed by the wrappers, a *payload scanner* searches the entire content of the packet for keywords and regular expressions. Next, a *Ternary Content Addressable Memory (TCAM) filter* classifies packets based on a set of reconfigurable rules. The packet then passes through one or more *extensible modules*, where additional packet processing or packet filtering is implemented. Next, the data and flow ID are passed to a *queue manager*, which schedules the packets for transmission from a *flow buffer*. Finally, the packet is transmitted to a switch or to a Gigabit Ethernet or SONET line card. Details of these components are given in the sections that follow.

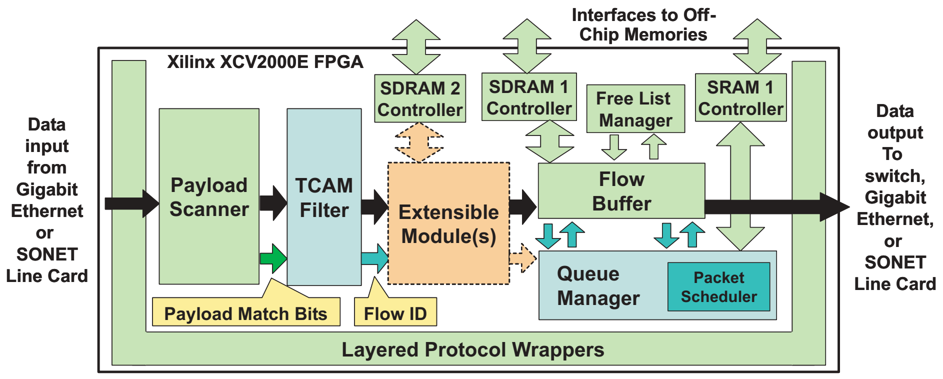

Fig. 1. Block Diagram of System-On-Chip Firewall

2.1 Protocol Processing

To process packets, a set of layered protocol wrappers was implemented to parse protocols at multiple layers [3]. At the lowest layer, data is segmented and reassembled from short cells into complete frames. At the network layer of the protocol stack, the fields of the Internet Protocol (IP) packets are verified and computed. At the transport layer, the port numbers and packet lengths are used to extract application-level data.

2.2 Payload Processing with Regular Expressions

Many types of Internet traffic cannot be classified by header inspection [4]. For example, junk email (SPAM) typically contains perfectly valid network and transport-layer data. In order to determine that a message is SPAM, a filtering device must be able to classify packets based on the content rather than just the values that appear in the packet headers.

Dynamically reconfigurable hardware performs content classification functions effectively. Unlike an ASIC, new finite automata can be programmed into into hardware to scan for specific content. In order to scan packet payloads, a regular expression matching circuit was implemented. Regular expressions provide a shorthand means to specify the value of a string, a substring (specified by '**( )**'), alterative values (separated with '**|**'), any one character (specified by '**.**'), zero or one characters (specified by '**?**'), or zero or more characters (specified by '$*$'). For example, to match all eight case variations of the phrase "make money fast" and to allow any number of characters to reside between keywords, the expression: "**(M|m)ake.*(M|m)oney.*(F|f)ast**" would be specified.

The architecture of a payload scanner with four parallel content scanners searching for eight Regular Expressions, *RE[1]-RE[8]*, is illustrated in Figure 2. A match is detected when a sequence of incoming data causes a state machine to reach an accepting state. In order to differentiate between multiple groups of regular expressions, the scanner instantiates a sequence of parallel machines that each search for different expressions. In order to maximize throughput, multiple sets of parallel content matching circuits are instantiated. When data arrives, a

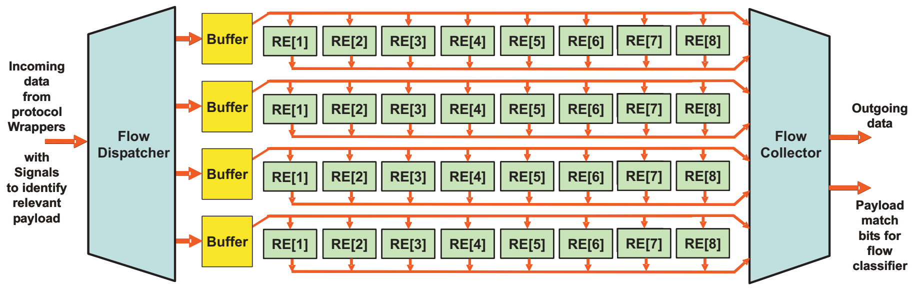
Incoming data from protocol Wrappers with Signals to identify relevant payload
Flow Dispatcher
Buffer
RE[1]
RE[2]
RE[3]
RE[4]
RE[5]
RE[6]
RE[7]
RE[8]
Flow Collector
Outgoing data
Payload match bits for flow classifier

Fig. 2. Regular Expression Payload Scanner

flow dispatcher sends it to an available buffer which then streams data though the sequence of regular expression search engines. Finally, a *flow collector* combines the traffic into a single outgoing stream of data. The final result of the circuit is a set of *payload match bits* that identify which regular expressions were present in each data flow. To implement other high-speed payload scanning circuits, an automated design process was created to generate circuits from a specification of regular expressions [5].

2.3 Rule Processing

A diagram of the rule matching circuit used on the SOPC firewall is shown in Figure 3. An on-chip, TCAM is used to classify packets as belonging to a specific flow. As an Internet Protocol (IP) packet arrives, *bits in the IP header*, which include the *source address, destination address, source port, destination port*, and *protocol* as well as the *payload match bits* are simultaneously compared to the *CAM value* fields in all rows of the TCAM table. Once all of the values have been compared, a *CAM mask* is applied to select which bits of each row must match and which bits can be ignored. If all of the values match in all of the bit locations that are unmasked, then that row of the TCAM is considered to be a match. The *flow identifier* associated with the rule in the highest-priority matching TCAM is then assigned to the flow. Rules can be added or modfied by sending control packets to the firewall to change values in the *CAM mask*, *CAM value*, and *flow ID* fields. Large CAMs can be implemented on the FPGA by using block RAMs [6].

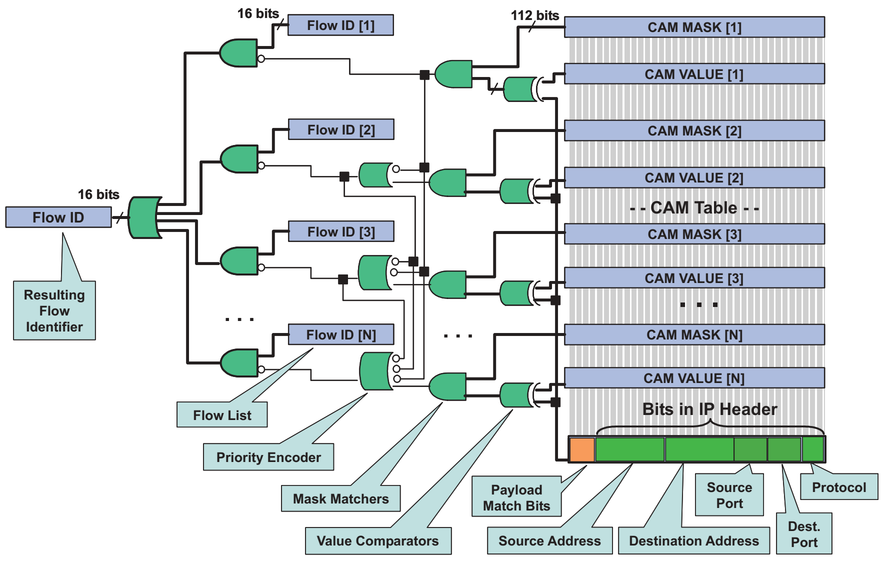

Fig. 3. Header Processing Module

2.4 Flow Buffering

To provide different levels of Quality of Service (QoS) for different traffic passing through the network, the SOPC firewall implements both class-based and per-flow queuing. Class-based queuing allows certain types of traffic to receive better service than other traffic. Per-flow queuing ensures that no single traffic flow consumes all of the network's available bandwidth [7].

To support multiple classes of service, traffic flows are organized by the firewall into four priority classes. Within each class, multiple linked lists of packets are maintained. Management of queues and tracking of free memory space is performed by the FPGA hardware in constant-time using linked-list data structures.

A diagram of the flow buffer and queue manager is shown in Figure 4. The queue manager includes circuits to enqueue traffic flows, dequeue traffic flows, and to schedule flows for transmission. Within the scheduler, four separate queues of flows are maintained (one for each class).

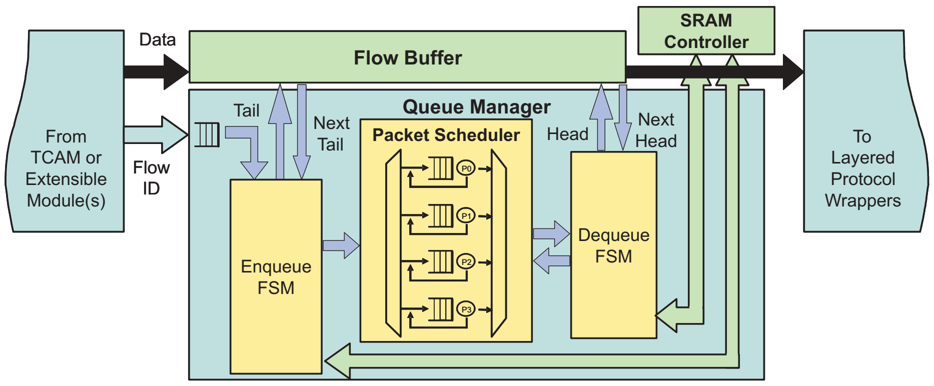

Fig. 4. Queue manager for organizing traffic flows

When a packet arrives, the packet's data is delivered to the *flow buffer* and the packet's flow identifier is passed to the *enqueue FSM* in the *Queue Manager*. Using the *flow ID*, the *enqueue FSM* reads SRAM to retrieve the flow's state. Each entry of the flow's state table contains a pointer to the *head* and *tail* of a linked list of stored packets in SDRAM as well as counters to track the number of packets read and written to that flow.

Meanwhile, the flow buffer stores the packet into memory at the location specified by the enqueue FSM's tail pointer. The flow buffer includes a controller to store the packet in Synchronous Dynamic Random Access Memory (SDRAM) [8]. After writing a packet to the next available memory location, the value of the tail pointer is passed to the queue manager to identify the next free memory location.

Within each class of traffic, the queue manager performs round-robin queuing of individual traffic flows. When the first packet of a flow arrives that has no packets already buffered, the flow identifier is inserted into a scheduling queue for that packet's class of service and the flow state table is updated. When another packet of a flow that is already scheduled arrives, the packet is simply appended to the linked list and the packet write count is incremented.

To transmit packets, the *dequeue FSM* reads a flow ID from the scheduler. The scheduler dequeues the flow from the next available flow in the highest priority class of traffic. The dequeue FSM then reads the flow state to obtain a pointer to the data in the flow buffer. The flow identifier is then removed from the head of the scheduler's queue and re-enters at the tail of the same queue if that flow has additional packets to transmit.

2.5 Extensible Features and Design Tools

Custom functionality is programmed into the SOPC firewall to implement specific functions demanded for a particular network application or service. Extensible modules that perform data encryption, enhanced packet filtering functions, and other types of packet scheduling have been implemented. One such module blocks patterns of TCP Synchronize/Acknowledge (SYN/ACK) packets characteristic of a DoS attacks. Other extensible functions that have been implemented as modules that fit into the SOPC firewall are listed in Table 1 [9] [10]. The set of features in the firewall is only limited by the size of the FPGA.

Table 1. SOPC Firewall Modules

Virus blocking	Content filtering
Denial of Service Protection	AES and 3DES Decryption
Bitmap image filtering	Network Address Translation (NAT)
Internet route lookup	IP Version 6 (IPV6) tunneling
Resource Reservation (RSVP)	Domain Name Service (DNS) caching

To facilitate development of extensible modules, an integration server was created that allows modules to be automatically configured using an application developer's information submitted via the web [11]. The tool transforms time-consuming and mistake-prone task of individually wiring ports on newly created modules into the relatively simple task of selecting where a module fits within the system interface. Developers upload their extensible modules to the integration server, which registers them in a database. The server parses the uploaded VHDL files and prompts the user to supply the port mapping to one or more of the standard, modular interfaces on the SOPC firewall. Then a bitfile is generated that contains the new module integrated into the baseline firewall, which is then returned to the user for testing. The server can also be used to create a bitfile for a SOPC firewall with any subset of previously uploaded extensible modules.

The integration server tracks and displays the estimated device utilization for each module to aid the selection process.

To further ease the development of extensible modules, a networked test server was developed to allow new extensible modules to be tested over the Internet [12]. After developing an extensible module on the integration server, a developer submits the resulting placed and routed design via a web interface to the test server. The test server: (1) remotely programs the FPGA platform, (2) generates test traffic in the form of network packets, and (3) displays the resulting packets. The traffic generated by the test server is processed by the firewall and returned to the test server. The developer then compares the contents of the resulting packets to expected values to verify correctness. Automation simplifies the process of migrating designs from concept to implementation.

3 Results

The results after place and route for the synthesized SOPC Firewall on the Xilinx Virtex XCV2000E are listed in Table 2. The core logic occupied 43% of the logic and 39% of the block RAMs. Placement of the SOPC circuitry was constrained using Synplicity's Amplify tool to lock the location of modules into specific regions of the FPGA. A view of the placed and routed Xilinx Virtex XCV2000E is shown in Figure 5. Note that the center region of the chip was left available for insertion of extensible modules.

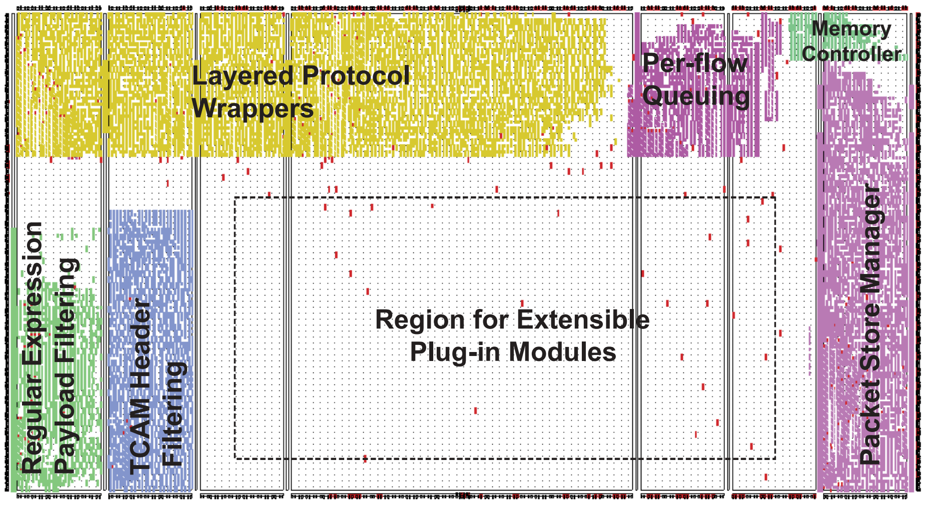
Layered Protocol Wrappers
Per-flow Queuing
Memory Controller
Regular Expression Payload Filtering
TCAM Header Filtering
Region for Extensible Plug-in Modules
Packet Store Manager

Fig. 5. FPGA Layout of the SOPC Firewall

Table 2. SOPC Firewall Implementation Statistics

Resource	Virtex XCV2000E Device Utilization	Utilization Percentage
Logic Slices	8342 out of 19200	43%
BlockRAMs	63 out of 160	39%
External IOBs	286 out of 512	55%

3.1 Throughput

The components of the SOPC firewall that implement the protocol wrappers, CAM filter, flow buffer, and queue manager were synthesized and operated at 62.5 MHz. Each of these components process 32 bits of data in every cycle, thus giving the SOPC firewall a throughput of 32*62.5MHz = 2 Gigabits/second. The regular expression scanning circuit for the SPAM pipeline synthesized at 37 MHz. Given that each pipeline processes 8 bits of data per cycle, the throughput of the SPAM filter is 8*37MHz=296 Megabits/second per pipeline. By running 8 SPAM pipelines in parallel, the payload matching circuit achieves a throughput of 8*8*37MHz=2.368 Gigabits/second.

3.2 Testing the SOPC Firewall on the FPX Platform

The Field Programmable Port Extender (FPX) platform was used to evaluate the performance of the SOPC firewall with real Internet traffic. The FPX is an open hardware platform that supports partial run-time reconfiguration [13] [14]. As with other Internet-based reconfigurable systems, the FPX allows some or all of the hardware to be dynamically reprogrammed over the network [15] [16]. On the FPX, however, all reconfiguration operations are performed in hardware and the network interfaces run at multi-gigabit/second rates [17].

The core components of the SOPC firewall include the layered protocol wrappers, TCAM packet filters, the payload processing circuit, and the per-flow packet buffer with the SRAM and SDRAM controllers. All of these components were synthesized into the Virtex XCV2000E device that implements the Reprogrammable Application Device (RAD) on the FPX. The resulting bitfile was uploaded into the RAD and in-system testing was performed.

Using the features described above, a SPAM filter was implemented on the SOPC firewall to detect and filter unwanted junk email, commonly referred to as SPAM. By scanning the payloads of packets passing through our network, potentially unwanted messages were filtered before they reached their targeted endpoint. To implement our SPAM filter, we observed that within our network, emails could be classified into one of eight general categories. Within these categories, we identified several regular expressions that were commonly present or absent in SPAM-infested email. Our listed included the terms: “MAKE MONEY FAST”, “Limited Time Offer”, as well as 32 other terms. These terms were programmed into hardware as regular expressions in a way that allowed case-insensitive matching where needed. Meanwhile, the TCAM was programmed with rules to assign SPAM-infested packets to put SPAM-infested email in lower-priority traffic flows [10].

Actual network traffic was sent to the hardware over the network from remote hosts. The SOPC firewall filtered traffic passing between a local area network and a wide area network, as shown in Figure 6. Malicious packets were dropped, the SPAM was rate-limited, and all other flows received a fair share of bandwidth.

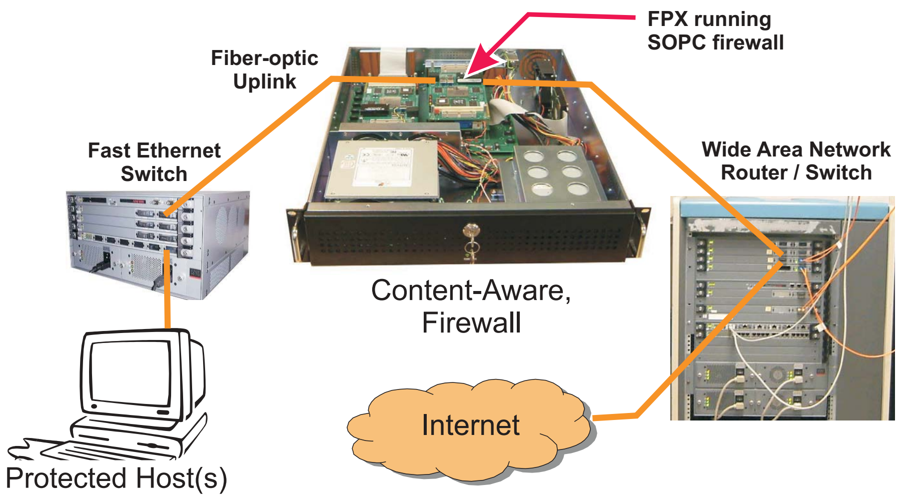

Fig. 6. Firewall Configuration

4 Conclusions

An extensible firewall has been implemented as a reconfigurable System-On-Programmable-Chip (SOPC). In addition to the standard features implemented by other Internet firewalls, the SOPC firewall performs payload scanning and per-flow queuing. The circuit was implemented on a Xilinx XCV-2000E FPGA. The resulting bitfile was tested on the Field Programmable Port Extender (FPX) network platform.

By using parallel hardware and deeply pipelined circuits, the SOPC firewall can process protocol headers with TCAMS and search the entire payload using regular expression matching at rates over 2 Gigabits/second. Attacks are mitigated by sorting each traffic flow into its own queue and scheduling traffic from all flows. These features allow the firewall to filter SPAM and protect networks from intrusion. A region of gates in the FPGA was left available to be used for extensible plugins. By using reprogrammable this hardware, new modules can be added and the firewall can protect a network against future threats.

References

1. R. Franklin, D. Carver, and B. L. Hutchings, "Assisting network intrusion detection with reconfigurable hardware," in *FCCM*, (Napa, CA), Apr. 2002.
2. J. W. Lockwood, "Evolvable internet hardware platforms," in *The Third NASA/DoD Workshop on Evolvable Hardware (EH'2001)*, pp. 271–279, July 2001.
3. F. Braun, J. Lockwood, and M. Waldvogel, "Reconfigurable router modules using network protocol wrappers," in *Field-Programmable Logic and Applications (FPL)*, (Belfast, Northern Ireland), pp. 254–263, Aug. 2001.
4. Y. Cho, S. Nahab, and W. H. Mangione-Smith, "Specialized hardware for deep network packet filtering," in *Field Programmable Logic and Applications (FPL)*, (Montpellier, France), Sept. 2002.
5. J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an Internet firewall," in *FCCM*, (Napa, CA), Apr. 2003.
6. J.-L. Brelet, "Using block RAM for high performance read/write CAMs." Xilinx XAPP204, May 2002.
7. 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.
8. S. Dharmapurikar and J. Lockwood, "Synthesizable design of a multi-module memory controller." Washington University, Department of Computer Science, Technical Report WUCS-01-26, Oct. 2001.
9. "Acceleration of Algorithms in Hardware." `http://www.arl.wustl.edu/~lockwood/class/cs535/`, Sept. 2001.
10. "Reconfigurable System-On-Chip Design." `http://www.arl.wustl.edu/~lockwood/class/cs536/`, Dec. 2002.
11. D. Lim, C. E. Neely, C. K. Zuver, and J. W. Lockwood, "Internet-based tool for system-on-chip integration," in *International Conference on Microelectronic Systems Education (MSE)*, (Anaheim, CA), June 2003.
12. C. E. Neely, C. K. Zuver, and J. W. Lockwood, "Internet-based tool for system-on-chip project testing and grading," in *International Conference on Microelectronic Systems Education (MSE)*, (Anaheim, CA), June 2003.
13. E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *Design Automation Conference (DAC)*, (New Orleans, LA), June 2002.
14. T. Sproull, J. W. Lockwood, and D. E. Taylor, "Control and configuration software for a reconfigurable networking hardware platform," in *IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM)*, (Napa, CA), Apr. 2002.
15. S. McMillan and S. Guccione, "Partial run-time reconfiguration using JRTR," in *Field-Programmable Logic and Applications (FPL)*, (Villach, Austria), pp. 352–360, Aug. 2000.
16. H. Fallside and M. J. S. Smith, "Internet connected FPL," in *Field-Programmable Logic and Applications (FPL)*, (Villach, Austria), pp. 48–57, Aug. 2000.
17. J. W. Lockwood, J. S. Turner, and D. 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), pp. 137–144, Feb. 2000.