Field Programmable Port Extender (FPX) for Distributed Routing and Queuing

John W. Lockwood *
Jon S. Turner
David E. Taylor

Department of Computer Science
Applied Research Lab
Washington University
1 Brookings Drive
Saint Louis, MO 63130

Phone: (314) 935-4460
Web: http://www.arl.wustl.edu/arl

1. ABSTRACT

Field Programmable Gate Arrays (FPGAs) are being used to provide fast Internet Protocol (IP) packet routing and advanced queuing in a highly scalable network switch. A new module, called the Field-programmable Port Extender (FPX), is being built to augment the Washington University Gigabit Switch (WUGS) with reprogrammable logic.

FPX modules reside at the edge of the WUGS switching fabric. Physically, the module is inserted between an optical line card and the WUGS gigabit switch backplane. The hardware used for this project allows ports of the switch populated with an FPX to operate at rates up to 2.4 Gigabits/second. The aggregate throughput of the system scales with the number of switch ports.

Logic on the FPX module is implemented with two FPGA devices. The first device is used to interface between the switch and the line card, while the second is used to prototype new networking functions and protocols. The logic on the second FPGA can be reprogrammed dynamically via control cells sent over the network.

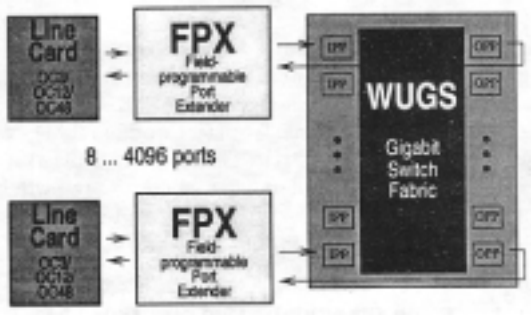

Figure 1: FPX between Line Card and WUGS

The flexibility of the FPX has made the card of interest for several networking applications. This year, fifty FPX hardware modules will be fabricated and distributed to researchers at eight universities around the country who are interested in experimenting with reprogrammable networks and per-flow queuing mechanisms. The FPX hardware will first be used to implement fast IP lookup algorithms and distributed input queueing.

2. BACKGROUND

The growth of the Internet has affected the design of network switches in two major ways. First, the complexity of packet processing algorithms have increased in order to provide better network utilization. Second, the throughput of the switches have increased in order to provide greater bandwidth. Past experiences provide valuable insight to the role of FPGAs for future high-speed networks.

*This research is supported by NSF: ANI-98-18964

FPGA '2000 Monterey CA USA

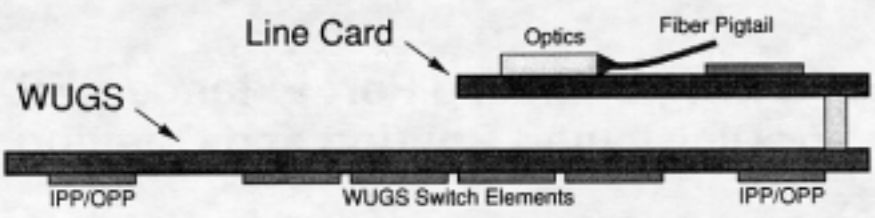

Figure 2: Original configuration of WUGS backplane with line card (side view)

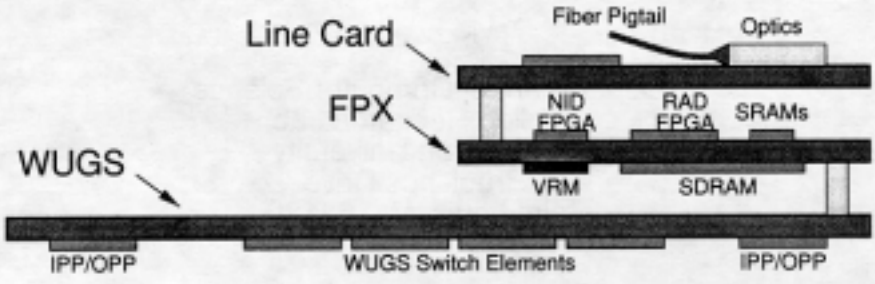

Figure 3: New configuration of WUGS backplane, FPX, and line card (side view)

2.1 FPGA Queuing Module

Field Programmable Gate Arrays have proven to be highly effective for some aspects of networking hardware. In the development of the iPOINT testbed, a complete Asynchrounous Transfer Mode (ATM) switch was built using FPGAs [1]. This system utilized a Xilinx 4013 FPGA to implement a single-stage switch [2] and multiple Xilinx 4005 FPGAs to implement queuing modules at each of the inputs of the switch [3].

Simple queuing mechanisms, such as First-In-First-Out (FIFO), would typically prevent this type of an input queuing system from achieving full network utilization [4]. To overcome this performance barrier, the Illinois Input Queue (iiQueue) was implemented to improve the performance of distributed input queue by sorted packets according to their flow, destination, and priority. A prototype of this system was implemented using Xilinx XC4013 FPGAs [5].

The benefit of using reprogrammable logic in the iiQueue was that complex algorithms for queuing data could be tuned by simply reprogramming the FPGA logic. Further, the reprogrammable logic enabled additional packet processing features to be integrated into the same FPGA devices that controlled the queuing circuits.

2.2 WUGS Scalable Switch

A critical feature for a switch involves it's ability to scale with the number of ports. The Benes structure enables switching elements to be combined efficiently to implement a multi-stage switch. The gate-count complexity of this type of switch grows only as $O(N \log N)$ with the number of switch ports [6].

Through a separate research program, ASICs were fabricated to implement the functionality of the Benes switching element. These chips were used to build complete network switches. An eight-port implementation of the WUGS switch has been built that provides 20 Gigabits/second of throughout. Over this past year, fifty of these these kits were built and distributed to researchers at over thirty univerities around the world [7].

A multi-stage implementation of this switch is currently being developed that provides 160 Gbps of aggregate throughput. In a larger configuration, the switching components of the system could be arranged to provide as many as 4096 ports for an aggregate throughput of 9.8 Terabits/second.

2.3 Combined System

By joining the flexible FPGA-based queuing circuitry of the iiQueue with the highly scalable switching fabric of the WUGS, a combined network router can be built that can provide flexibile packet processing and scale to provide high throughput.

By conforming to standard Utopia interfaces between the WUGS backplane and the switch, the FPX is designed to require no modifications to existing switch hardware. In the original configuration, a line card directly attaches to the WUGS backplane, as shown in Figure 2. In the new configuration, an FPX module is inserted between the line card and the WUGS backplane, as shown in Figure 3.

As with the iiQueue, the use of FPGAs at the switch input allows the system to implement flexible packet processing. It was a goal of the FPX design to provide a reprogrammable module optmized that was optimized for both flexiblity and performance.

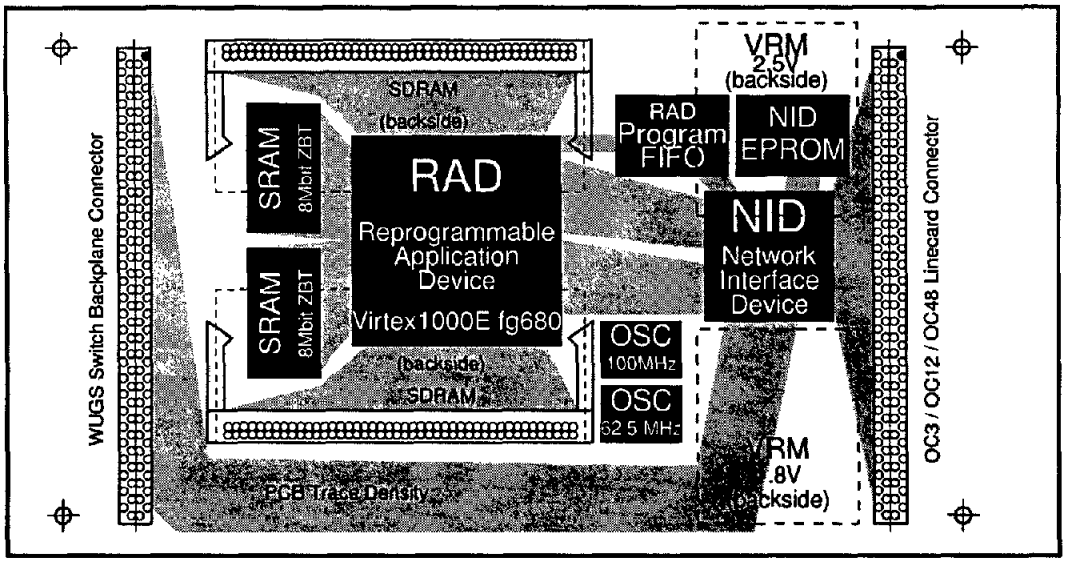

Figure 4: FPX Printed Circuit Board

3. THE FPX MODULE

The logic for the FPX is implemented using two FPGA devices, the Network Interface Device (NID) and the Reprogrammable Application Device (RAD). These components are mounted on the 20 cm × 10.5 cm printed circuit board shown in Figure 4.

The RAD is implemented with a Virtex 1000Efg680 FPGA, while the NID is implemented with a Virtex 400bg560 FPGA. The NID FPGA contains sufficient logic to interface to the switch and line card, while the larger RAD FPGA was chosen to provide sufficient logic density in order to implement complex IP routing and queuing functions.

3.1 FPX Memory Configuration

To enable high-speed, bi-directional cell queuing and cell buffering operations, the FPX module provides four independent banks of memory. All memory devices operate at a frequency of 100 MHz.

Two banks of 32-bit wide SRAM are provided to maintain pointers and data structures in a low-latency control memory. This memory utilizes Zero Bus Turnaround (ZBT) SRAMs to provide full-throughput access to memory.

Two banks of 64-bit wide Synchronous Dynamic Random Access Memory (SDRAM) are provided to buffer data from the network. At Gigabit/second rates, several Megabytes of memory can be required to buffer a large burst of data.

3.2 FPX Memory Analysis

The total number of memory operations that can be used to process a packet depend on the Line Card rate and the length of the packet. The faster the link, the fewer the number of cycles: Fifty-six bytes is the smallest packet that would be processed by the WUGS switch-/router. This size is slightly larger than the 53-byte size of an Asynchrounous Transfer Mode (ATM) cell.

For line cards that operate at OC3 rates (155 Mbits/second), each memory provides $53 * 8/155M = 273$ operations per cell. The four parallel banks of memory can, therefore, perform a total of $273 * 4 = 1092$ memory operations within the time period of a cell slot. At OC12 (622 Mbits/second), the same hardware can implement 273 operations. At OC48 (2.4 Gbits/second), the FPX provides 68 memory operations per slot period. Of these operations, $56/8 = 7$ writes to the SDRAM memory are used to enqueue a cell, and $56/8 = 7$ reads from the SDRAM are used to dequeue a cell. All remaining memory operations can be used to implement the routing and buffer management functions.

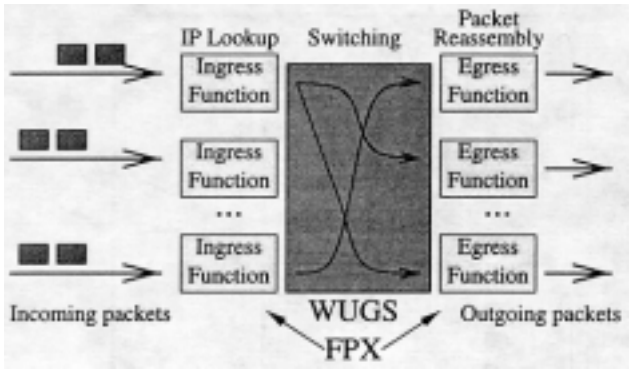

Figure 5: Configuration of FPX as IP Router

4. IP ROUTING

Given the exponential growth of the Internet, the need exists for fast Internet Protocol (IP) lookup engines to eliminate bottlenecks in high performance routers. The FPX provides an excellent platform for implementing IP routing functions at the port of a WUGS.

A router configuration using a WUGS equipped with FPX modules is shown in Figure 5. Each FPX hardware module implements two functions: one at the ingress (input) of the switch and the other at the egress (output) of the switch. The ingress functionality of the FPX performs the IP lookup function. The result of the lookup is used by the WUGS switch to deliver the packet to the appropriate egress port. The egress function of the FPX re-assembles segments of packets before they are transmitted on the outgoing link.

4.1 IP Lookup Functions

One algorithm capable of performing IP lookups at link speed in the FPX is the Tree Bitmap algorithm [8]. The Tree Bitmap algorithm is a hardware based algorithm that employs multibit trie data structures to perform IP lookups with efficient use of memory.

An IP lookup consists of finding the longest matching prefix stored in the routing table for the given 32-bit IPv4 destination address and retrieving the associated next hop for the IP packet. As shown in Figure 6, the unicast IP address is compared to the stored prefixes starting with the left-most bit. Once the longest matching prefix is found, the packet is forwarded to the specified next hop by modifying the packet header with the stored next hop information.

To efficiently perform this lookup function in hardware, the Tree Bitmap algorithm starts by storing the prefixes in a binary trie as shown in Figure 7. The shaded nodes denote a stored prefix. A search is conducted by using the IP address bits to traverse the trie. To speed up this searching process, subtrees of the binary trie are combined into single nodes, reducing the number of memory accesses needed to perform a lookup. The depth of the subtrees combined to form a single node is called the

Prefix	Next Hop
*	4
10*	7
01*	2
110*	9
1011*	1
0001*	0
01011*	5
00110*	3

32-bit IP Address
0101 1011 ... 0001

Next Hop
5

Figure 6: IP prefix lookup table of next hops. Next hops for IP packets are found using the longest matching prefix in the table for the unicast destination address of the IP packet.

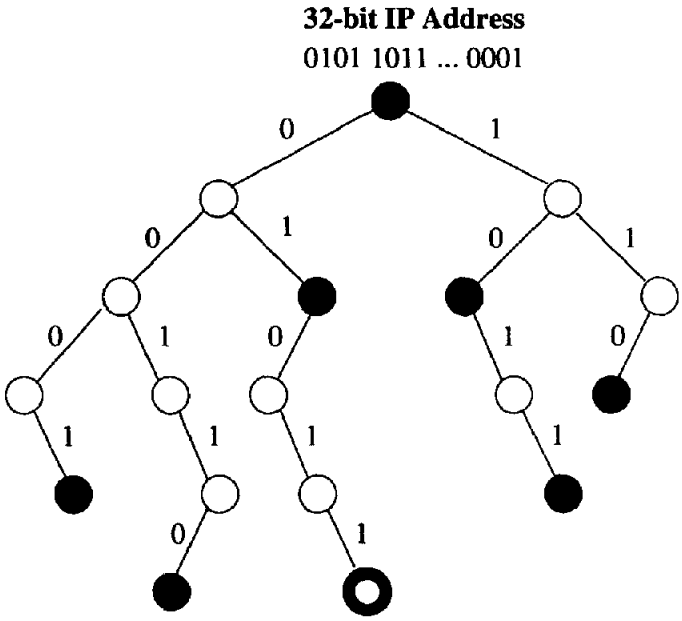

Figure 7: IP lookup table represented as a binary trie. Stored prefixes are denoted by shaded nodes. Next hops are found by traversing the trie.

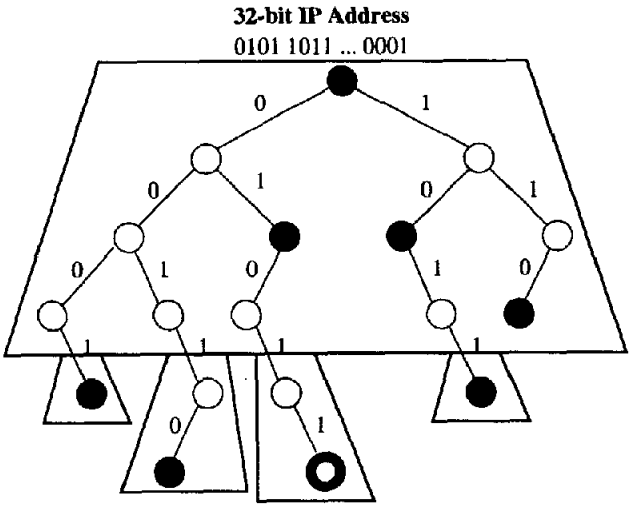

Figure 8: IP lookup table represented as a multibit trie. A stride, 4-bits, of the unicast destination address of the IP packet are compared at once, speeding up the lookup process.

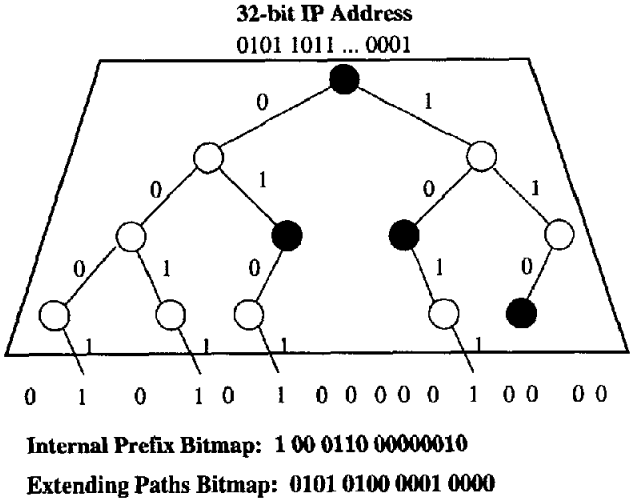

Figure 9: Bitmap coding of a multibit trie node. The internal bitmap represents the stored prefixes in the node while the extending paths bitmap represents the child nodes of the current node.

stride. An example of a multibit trie using 4-bit strides is shown in Figure 8. The Tree Bitmap algorithm codes information associated with each subtree as a node containing two bitmaps. The *internal bitmap* identifies the nodes of the original binary tree corresponding to the prefixes. the *external bitmap* identifies the "exit points" of each subtree that correspond to actual edges. An example is shown in Figure 9.

By requiring that all child nodes of a single parent be stored contiguously in memory, the address of a child node can be calculated using a pointer to the array of child nodes and an index into that array computed from the external bitmap. The 4-bit stride example is shown as a Tree Bitmap data structure in Figure 10. When there are no valid extending paths, the longest matching prefix is used to fetch the next hop information which is stored in a separate table.

4.2 FPX Design Constraints

The FPX architecture places several constraints on the implementation of the Tree Bitmap algorithm. Within the context of the FPX Module, the Tree Bitmap algorithm will simply be referred to as the Fast IP Lookup Engine (FIPLE). To allow for packet reassembly and other active processing functions requiring memory resources, the FIPLE has access to one of the 8 Mbit ZBT (Zero Bus Turnaround) SRAMs which provide a 32-bit data path with 2-clock cycle latency. Since this memory is "off-chip" both the address and data lines must be latched at the pads of the FPGA, providing for a total latency to memory of 4 clock cycles. The 32-bit data path width and 4 clock cycle latency of the memory suggest an optimal stride length of 4 bits. With a 4-bit stride, external bitmaps are 16-bits long occupying a half-word of memory, leaving the second 2 bytes free for

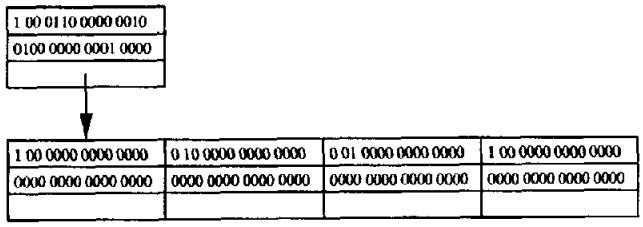

Figure 10: IP lookup table represented as a Tree Bitmap. Child nodes are stored contiguously so that a single pointer and an index may be used to locate any child node in the the data structure.

a memory address pointer to the child node array. With both the external bitmap and child node array pointer stored in a single memory word, the address of the next child node may be resolved in a single clock cycle. The remaining three clock cycles of memory latency may be used to examine the internal bitmap of the node data structure, compute the longest matching prefix and the address of the corresponding next hop entry in the table. By pipelining the fetching of child nodes, a worst case lookup using 4-bit strides would require examining 8-nodes to find the longest matching prefix. Combined with the cycles needed to fetch the next hop information, a single lookup would require (8*4)+6 = 38 clock cycles.

Since the FPGAs and SRAMs run on synchronous 100-MHz clocks, all single cycle calculations must be completed in less than 10ns. As expected, the critical path in the FIPLE design is the next node address calculation. In order to resolve this address, the number of ones in the extending path bitmap to the left of an index supplied by the stride of the IP address must be summed. This sum must be added to the 16-bit child node array address pointer. Finally, the result must be shifted to form the 18-bit physical address of the next node to be fetched. Current FPGA technology performs 16-bit adds in 5 ns or less. However, the lookup table used to generate a 16-bit mask for the summing of ones in a bitmap requires approximately 7ns. Obviously, all of the calculations necessary to resolve the physical address cannot be performed one clock cycle and more extensive pipelining of the algorithm is necessary. On chip memory may also be used as a cache for the first two levels of nodes to minimize memory latencies and decrease the total time per lookup.

4.3 Critical Path

If the critical paths can be eliminated to achieve a 10ns clock period, this IP lookup engine is capable of performing a lookup every 380ns. Minimum length IPv4 packets fit in one ATM cell. An FPX programmed with the Tree Bitmap algorithm could perform lookups within one cell time for links operating at or below OC12 rates. Increasing the link speed to OC48 would result in approximately one lookup every two cell times, given nominal delay between cells. Overlapping the lookup of two IP packets would fill the pipeline gaps and enable full system throughput, even with minimum-length packets.

4.4 Other IP Routing Algorithms

In addition to the Tree Bitmap algorithm being designed at Washington University, another design team is currently working on adapting the work presented in [9] onto the FPX. Instead of linearly increasing the prefix length searched through, this algorithm uses binary search over the prefix lengths, with all the prefixes of a given length stored in a hash table.

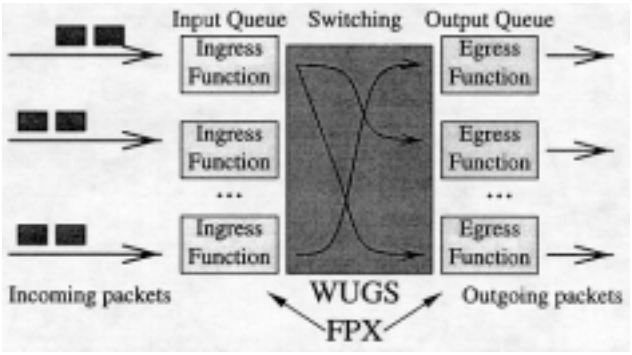

Figure 11: Configuration of FPX for Distributed Input/Output Queuing

5. DISTRIBUTED QUEUEING

The FPX also enables research and experimentation on a broad class of algorithms that provide distributed queuing. The logical configuration of the WUGS with multiple FPX modules is shown in Figure 11. By utilizing one pair of SRAM/SDRAM for input queuing and the other for output queuing, the FPX can implement Combined Input Output Queueing (CIOQ). Each queue represents one half of an FPX module (the FPX module processes both the ingress and egress data).

5.1 Input Cell Scheduling

Complex queuing circuits are required in order to fully utilize the throughput of an input buffered switch. Cells must be selected so that no more than one cell is read from an ingress module and no more than one cell is written to an egress module. At any given time, an ingress module may be buffering cells destined for any permutation of output ports. The optimal selection of these cells from the input of a switch is an active topic of research.

For systems utilizing simple First-In-First-Out (FIFO) memories, cells can only be transmitted from the head of each queue. Cells behind the head of the queue are blocked—regardless of whether or not their destination port is available. For unicast traffic with Poisson arrivals, Head-of-Line (HOL) blocking limits the throughput of an input-buffered FIFO switch to 58% [4].

Queuing by destination port can eliminate HOL blocking. It allows the scheduling algorithm to evaluate all possible flows from the ingress module to the egress module. Each ingress module maintains an N-queue virtual output queue structure that sorts by destination port.

Optimal performance can be found by solving Maximum Weighted Matching (MWM) of a bipartite graph. In the MWM graph, one node partition is assigned to the input ports and the other to the output ports. Edges between inputs and outputs are assigned an edge weight representing the availability of cells from the input to the output. While the solution is optimal, the $O(N^3 \log N)$

running time makes it impractical to implement [10].

5.2 Reducing Circuit Complexity

Several approaches have been taken to reduce the running time of the selection algorithm. In the Matrix Unit Cell Scheduler (MUCS) [11], a scheduler reads each port's destination vector to determine which cells await delivery to which output port. The algorithm iterates on the matrix to select a maximum set of cells that can be transmitted. By first selecting ports that have the least flexibility, the aggregate throughput of the switch nearly equals that of the output buffered switch.

Other heuristic algorithms exist to optimize the transmission of cells from an input port. For the Longest Port First (LPF) method, the scheduler first finds the input port and output port pairs that can transmit maximum number of packets, i.e. a set of maximum size matches. Based on this set, the scheduler then selects the transmission matches that can maximize total weight. The LPF algorithm can achieve 100% utilization with a reduced complexity of $O(N^{2.5})$. By avoiding comparison operations in the critical path of the algorithm, it also has an advantage in hardware implementation [12]. Fast, input-buffered scheduling techniques are the basis for the Tiny Tera switch [13].

6. RESEARCH TESTBED

The reprogrammable logic in the FPX module enabled it to serve as a common hardware platform for networking research. There are currently several research projects planned that will utilize the FPX to implement new networking features in hardware. While the features are different, they can all be implemented using the same FPX hardware at the edges of the WUGS. Some of the projects outside of Washington University that plan to utilize the FPX are listed below.

Site	PI	Project
Georgia Tech	Karsten Schwan	Packet scheduling
Rice	Edward Knightly	Traffic shaping
Wisconsin	Parmesh Ramanathan	QoS for Internet traffic
UCS/ISI	Jon Touch	Active network congestion control
UCSB	P M Melliar-Smith	Many-to-many multicast
Oakland	Ronald Srodawa	Video queuing for distance learning

6.1 FPX Reprogrammability

As a research tool for prototyping new algorithms in hardware, the FPX has been designed to simplify reprogramming steps. To enable the system to remain fully operational even while it is being reprogrammed, the system has been designed to operate throughout the reprogramming cycle.

When the system first boots, an EPROM is used to fetch a static configuration for the NID. By default the NID is programmed to pass-through data between the line card and the switch.

The FPX programs the RAD by reading configuration data over the network. Once the NID is programmed, it waits for control cells on VCI=34 to arrive that contain the configuration data for the RAD. As each sequential cell arrives, the contents of the cell are buffered in the FIFO. When the final byte of RAD configuration is loaded, the contents of the FIFO are used to program the RAD.

After configuration, the NID continues to listen for control cells. The switch controller (a software entity running elsewhere in the network), can dynamically reprogram the RAD by sending a new bitstream over the network. Again, after the last cell arrives, a final confirmation cell can be issued to reprogram the RAD with the new configuration.

The FPX module is wired to enable partial reprogramming the RAD. By using the NID to control the RAD's JTAG pins, it is possible to reprogram only a portion of the logic on the RAD. Rather than download the entire chip configuration, the NID transfers only those parts of the design that have changed. This feature enables the RAD (not just the NID) to continue processing packets through a design modfication. It is hoped that improved software support for partial reprogrammability will be forthcoming in a way that allows open access between the tools and the devices [14] [15].

6.2 Web-based Toolkit

To enable sharing of designs, a web-site has been created which holds configuration files for the RAD. Complete bitfiles that implement the IP routing function for the RAD will be made available on the project website, along with synthesizable cores and VHDL source code for commonly-used network functions. The web-site for this project is:
`http://www.arl.wustl.edu/arl/projects/fpx`.

7. CONCLUSIONS

Field Programmable Gate Arrays have been used to implement a key component in the FPX/WUGS router. They provide the IP packet forwarding and queueing functions at the edge of the highly scalable switch.

The FPX provides a common hardware platform for research, developement, and experimentation on networking algorithms in hardware. By using reprogrammable logic, designs can be implemented without traditional design cycle delays.

This spring, fifty FPX modules will be fabricated and distributed to several research universities throughout the country. The hardware designs developed for this project will be maintained on the project website.

The performance of the FPX module enables it to per-

form IP routing and per-flow queuing at a rate of 2.4 Gigabits/second. When used in an eight-port WUGS-20 switch, the system provides an aggregate throughput of 20 Gigabits/second. The distributed topology of the router and the scalable nature of the switch enable much larger configurations. A router with 4096 ports would provide an aggregate throughput of 9.8 Terabits/second.

8. REFERENCES

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

[2] 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.

[3] H. Duan, J. W. Lockwood, and S. M. Kang, "FPGA prototype queueing module for high performance ATM switching," in *Proceedings of the Seventh Annual IEEE International ASIC Conference*, (Rochester, NY), pp. 429–432, Sept. 1994.

[4] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input vs. output queueing in space division packet switching," *IEEE Transactions on Communications*, vol. Com-35, pp. 1347–1356, Dec. 1987.

[5] 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.

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

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

[8] W. N. Eatherton, "Hardware-Based Internet Protocol Prefix Lookups." thesis, Washington University in St. Louis, 1998.

[9] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing table lookups," in *Proceedings of ACM SIGCOMM '97*, pp. 25–36, September 1997.

[10] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *INFOCOM'96*, Mar. 1996.

[11] H. Duan, J. W. Lockwood, and S. M. Kang, "Matrix unit cell scheduler (MUCS) for input-buffered switches," *IEEE Communication Letters*, vol. 2, pp. 20–23, Jan. 1998.

[12] N. McKeown and A. Mekkittikul, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," in *INFOCOM'98*, (San Francisco), Apr. 1998.

[13] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M. Horowitz, "The Tiny Tera: A packet switch core," in *IEEE Micro*, pp. 26–33, Jan. 1997.

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

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