

# A Stream-based Reconfigurable Router Prototype

David C. Lee, Scott J. Harper, Peter M. Athanas, and Scott F. Midkiff

Bradley Department of Electrical and Computer Engineering  
Virginia Polytechnic Institute and State University  
Blacksburg, Virginia 24061, USA

Phone: +1 (540) 231-2295 Fax: +1 (540) 231-3362 Email: dlee@vt.edu

## ABSTRACT

*Flexible and fast routers are generally considered a contradiction in terms since maintaining high-throughput requires fast but fixed-configuration application specific integrated circuits (ASICs) while flexibility requires slower but flexible configuration of general-purpose processors. A configurable stream-based reconfigurable router is a fast, flexible solution that includes the best features of both hardware and software processing through the efficient use of field-programmable gate array (FPGA) technology, Active VHDL, and hardware/software partitioning and scheduling algorithms. This paper discusses the architecture of the stream-based reconfigurable router and presents the design, implementation, and evaluation of a prototype reconfigurable router.*

## 1. INTRODUCTION

One issue to consider when designing a network system is choosing a technology that is both flexible enough to support the need for the ever new, ever evolving protocols that reside at the network and link layers but fast enough to support the high-throughput requirements of a core network. New protocols for integrated services, resource reservation, network security, and IPv6 place new demands on network routers. Attempts to bridge these two disparate goals of flexible and fast routing is performed at the router operating system level, typically under the moniker of active networks or open signaling. New technologies deployed within the router operating system have the potential of offering increased flexibility in the router but may not increase its performance. In fact, it is possible that new software technologies may decrease the performance of the router.

Thus, it is not clear if these software technologies, while holding significant potential, will be practical, especially considering the current problems with maintaining port performance. Keshav and Sharma [1] note that the reduction of port cost is currently a tradeoff between application specific integrated circuits (ASICs) and general-purpose processors. In this dichotomy, the only solution that provides reconfigurability is the use of a general-purpose processor. Clearly, some other solution must be found to provide on-demand scheduling of hardware resources to assist in the development of flexible network software technologies.

One potential solution is to use reconfigurable hardware, such as field-programmable gate arrays (FPGAs) or other reconfigurable devices, in a stream-based processing mode.

With suitable computational units, stream processing has the potential to allow hardware-level configurable processing of packets at line speeds. The application of hardware/software co-design techniques is used to partition and schedule protocol code for execution on both a general-purpose processor and stream-based hardware. The use of hardware/software co-design techniques has the potential shield protocol designers from knowing the details of hardware design. The long-term goal of this research is to develop a reconfigurable router, which uniquely integrates the best features of work being conducted in active software with run-time reconfigurable hardware.

This paper discusses a prototype stream-based reconfigurable router. Traditional router architectures and evolving router operating systems are presented in Sections 2 and 3. Sections 4 and 5 discuss the model for the reconfigurable router and stream-based processing. The design and implementation of the prototype stream-based reconfigurable router is presented in Section 6, with results presented in Section 7.

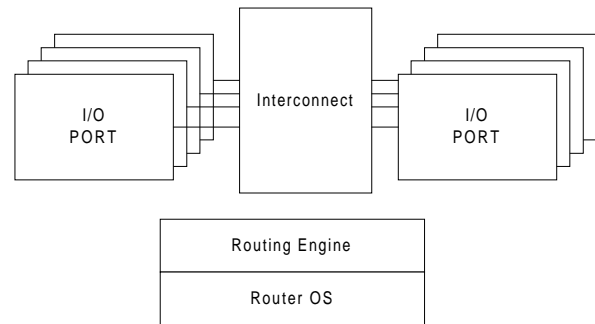


Figure 1. Architecture of a typical router.

## 2. ROUTER ARCHITECTURE

The basic architecture of any router is a set of input and output ports that are connected via an interconnection architecture, as shown in Figure 1. The interconnection architecture, which has a strong impact on performance, can deliver packets from port to port or deliver pointers to packets from port to port. For better performance, each port may have one or more processors. Further complicating the matter, a separate routing engine is used to process routing protocols and to load forwarding tables, which are then used by forwarding engines to determine where the packets should go. Forwarding engines can be located within the input/output port devices or as separate port devices.

It has been shown that the time spent to process a packet on an IP switch should not exceed 0.27 ms [2], which means that maintaining high-throughput is clearly a problem. As line speeds continue to rise and the upper bound on processing time continues to fall, some solution is required to provide reconfigurability at the hardware layer. A means of providing this low-level reconfigurability is through configurable computing technology. Contemporary FPGAs, which serve as the flexible fabric in configurable computing platforms, are already being used to provide field-upgrades of firmware in the IBM 8265 ATM switch [3] and network-based firmware upgrades in NTT research switches [4]. FPGAs provide an intermediate operating point between the relative slowness, flexible configuration, and low cost of a general-purpose processor and of the high-performance, fixed configuration, and high cost of an ASIC. Further, one would expect a future reconfigurable router to be composed of custom FPGAs, custom ASICs, and custom general-purpose processors to obtain the optimal combination of performance, flexibility, and cost.

### 3. ROUTER OPERATING SYSTEMS

There is much interest in developing the next generation of open router operating systems and related services. The research and development can generally be categorized into two areas, open signaling and active networks. In open signaling [5], a set of application programming interfaces (APIs) are defined so that users can access, in a standard way, information stored within a router, similar to the various Telephony API (TAPI) interfaces used in telecommunication networks. Active networks [6, 7] consist of nodes in which computational operations, or protocol implementations, can be “injected” at run-time. Not only does an active network provide uniform (software) signaling interfaces, it provides platform-independent mobile code. Thus, code written for router vendor A’s hardware can, by itself, migrate and run on router vendor B’s hardware. One can view open signaling as a part of the larger goal of an active network.

### 4. RECONFIGURABLE ROUTER ARCHITECTURAL MODEL

Figure 2 shows the general model of the reconfigurable router. The router operating system (OS) provides the management and control of the router and can be based on open signaling, active networking, or some other paradigm. For a reconfigurable router to work, a standard API must be defined. The router OS provides the resource management, route protocol processing, etc. It would also have the capability to receive, authenticate, and install any new protocols as required.

The protocol installation process could work as follows. Once the protocol code is loaded into the operating system, the code is partitioned and scheduled for execution on various hardware units, based on algorithms developed for hardware/software co-design. The hardware/software co-design algorithms are typically used to improve computational performance but can be adopted to support the reconfigurable router. The system partitions the functionality

such that some portion runs on a general-purpose processor and the rest runs on reconfigurable hardware.

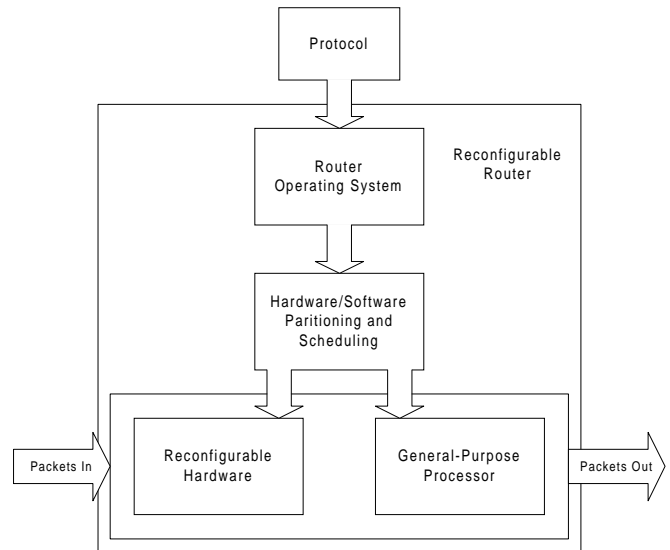


Figure 2. Ideal reconfigurable router architecture.

One approach [8] is to take a programming language, such as ANSI C, and convert the language statements into a behavioral graph. The graph is analyzed, bound to library elements, partitioned spatially, and scheduled for hardware execution where appropriate. Some types of program statements are inappropriate for hardware execution and are mapped to instructions for software execution. Thus, the analyzed graph results in a partitioning of the system into software and hardware components.

The scheduling of the hardware components may result in direct synthesis of the necessary hardware. The use of an intermediate synthesis to VHDL step and then a synthesis of the necessary hardware may help divide the complexity of the system into manageable portions. In the proposed scheme, a special extension of VHDL, called Active VHDL, would be used. Active VHDL is a type of open signaling mechanism for hardware. The authors are also investigating the use of Xilinx’s JBits [9] and JHDL [10] to both improve the synthesis time and aid in the partitioning and scheduling process.

There are two ways to program the router to support new protocols. Figure 2 illustrates the traditional model in which the router is programmed through a separate, out-of-band process. This work investigates the use of an in-band, or stream, programming process in which the protocol program is sent along with the data, somewhat similar in concept to an active network capsule [7]. The stream programming technique does not preclude operating system authentication and access control checks. Synthesis issues were not investigated in this work. Issues such as code execution safety, access control and authentication, and resource control are addressed by the router operating system and active network research.

### 5. STREAM-BASED RUN-TIME RECONFIGURABLE ROUTER

A run-time reconfigurable configurable computing machine (RTR/CCM) is the term for a computing device that allows hardware to be dynamically configured within the flow of normal computation. This provides the performance benefits of hardware while maintaining the flexibility of software. CCMs are characterized by having both programmable datapath operators and programmable interconnection resources. Stream-based computing is based on the use of self-guiding streams of programming information and user data to perform a computational task [11]. The stream programming information configures a CCM for the computation task on the user data that follows, as shown in Figure 3.

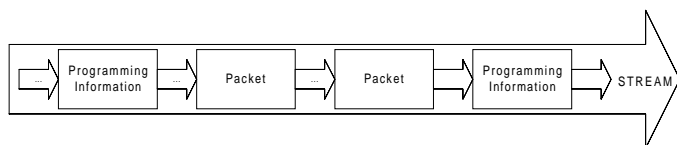


Figure 3. Stream programming format.

Within a reconfigurable router, the stream programming header can be viewed as a protocol and the data as the packets. The power of the stream-based paradigm allows the information stream also to be viewed as a network flow with switching and other features defined by the programming header. Thus, both a standard protocol and a flow-based programming model are supported by the reconfigurable router. The stream-based programming concept can be extended so that the packet stream arriving at the router is stream-based; in other words, the creation of a stream-based active network.

This is similar in concept to how Protocol Boosters can work in an FPGA [12]. Boosters are composable elements of protocols that may be strung together to form more complex protocols. This approach seeks to provide a more integrated approach to existing adaptive computing and active networking paradigms, or to provide a more flexible hardware composition paradigm to which boosters can be transparently adapted.

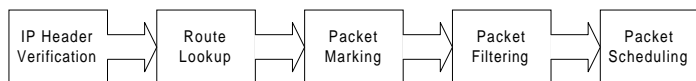


Figure 4. Example modular design.

Because the stream processor is similar to a fine-grained, superpipelined processor, bits are produced on each clock pulse. If the clock speed is equivalent to the line speed, then stream processing can operate at line speed. This also simplifies design by allowing packets that are to be discarded to be dropped at the end of the pipeline; otherwise, a mechanism must be developed to discard the packet while it is entering and being processed by the pipeline. Because of the pipeline design, a stream-based processor relies on a modular and configurable set of functional units. These

stream modules can be strung together to provide the necessary functionality, as shown in Figure 4.

The example consists of modules to perform required IP header checks, routing of the packet, examination of the packet stream with differential packet marking as needed, packet filtering and scheduling as necessary. If the marking algorithm requires modification, additional stream programming information can be sent to modify the packet marking processing module. Thus, a packet is received and verified, passed on to the route lookup module, passed on for marking, and so forth.

In a fully configurable stream processor, one can add, remove, and modify modules and interconnects on the fly. For example, if the packet marking process was no longer needed, it could be removed on demand. Modules can be developed independently, greatly simplifying the implementation of a design. The cost of modularity, however, is an increase in the number of gates required to implement a particular function and the fact that some computational resources available in the module may not be used by all functions. However, because of the significant increase in FPGA resources, this is not expected to be a concern.

Because a module later in the stream may need results of calculations from a module earlier in the stream, packets may be preceded by a small amount of header information that contains output calculations from various modules. This information can follow the packet stream on a separate data bus. Because the algorithm is known in advance, the amount of header information is also known and the stream interface controllers (SICs) within each module can ignore, pass on, or use the information as needed.

### 6. PROTOTYPE DESIGN AND IMPLEMENTATION

To investigate the possible architectures for a stream-based reconfigurable router, a simple demonstration prototype was built. This prototype performed simple routing using variable length IPv4 packets [13] with random source and destination addresses. The general design is shown in Figure 5 and consists of three stream modules: a packet source, the actual filter, and a packet sink. Each module was implemented in a Xilinx 4028EX FPGA on a GigaOps SPECTRUM G900 computing platform [14].

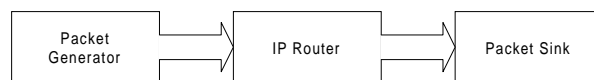


Figure 5. Prototype modular IP router design.

The packet source generated IPv4 packets with valid headers that used four possible source and destination addresses and variable time-to-live (TTL) values. Options and packet checksums were not supported. Checksum generation and verification can be easily added as separate modules. The source also generated packets of variable length from 21 to 1,024 bytes. The pseudo-random addresses and packet lengths were generated using linear feedback shift registers. The router verified header fields, updated the TTL

as necessary, and performed a route lookup. The packet sink captures the packets and the nexthop routing information to disk on the local host.

Each stream module, such as the router of Figure 6, has a uniform input and output (I/O) interface. The SIC determines if the data on the bus is programming data or computational (packet) data. Streams enter the module and leave the module on a data bus. Because the module may need to obtain data from external sources, such as obtaining routing information from Content-Addressable Memory, additional input data paths should be available – this is effectively the same as having two streams per module. Due to switching restrictions and a 16-bit maximum bus size restriction of the GigaOps platform, this capability was not implemented in the prototype. The bus limitation meant that the data bus, on which the stream programming and user information is sent, was restricted to 8 bits. A separate 4-bit wide control data bus was used to transfer intermediate computational information such as nexthop addresses and output ports. The final 4 bits was used to for stream control signals.

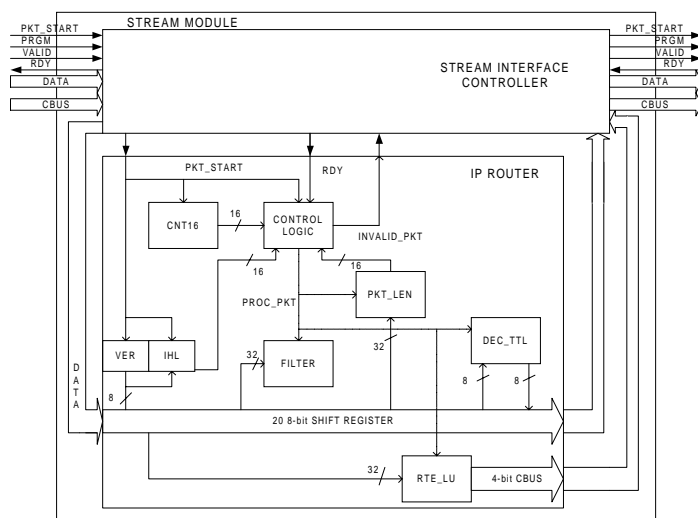


Figure 6. Router module block schematic.

Four control signals, *Prgm*, *Valid*, *Rdy*, and *Pkt\_Start* are used to ensure that the stream information is sent and processed correctly. *Prgm* indicates that the stream data is programming information, *Valid* indicates if the stream data is valid packet or computational data. Because stream processing operates like a pipeline, the *Valid* signal is also used to indicate when the data in the pipeline is no longer useful. There are many reasons why this may occur, such as if a header checksum failed. *Rdy* is used to signal a stoppage in the pipeline. Multiplexed stream control signals are not used to ensure that the stream interface controller within each stream module is as simple and as fast as possible. *Pkt\_start* simply indicates where a packet starts in the data stream. The GigaOps board provides global clocking and system initialization signals.

Each prototype router module was developed and verified in VHDL. The SIC is a simple state machine that alternatively programs the module or feeds the module data. The packet generator and IP router consist of a number of simple digital devices, including registers, counters, comparators, adders, pseudo-random number generators, and look-up tables. A number of state machines were also used to generate control signals. The block schematic of the router stream module is shown in Figure 6.

A 20-octet shift register is used to pipeline the packet data as it streams through the system. The SIC passes the *Pkt\_start* signal to indicate the start of each packet, at which time the router performs a header check and stores the IP header length for future use. A counter is used to track the number of bytes processed by the shift register and is input into the control logic. When the header is completely loaded in the shift register, the router performs a route lookup, a source-based address filter for net 10 addresses, decrements the TTL and stores the new value, and performs a number of header verification checks. As the packet leaves the module, the route lookup unit multiplexes nexthop and output port information onto the control data bus. The SIC may halt the router by dropping the *Rdy* line and uses the router's *INVALID\_PKT* line to generate the *Valid* signal for the next module.

## 7. RESULTS

The VHDL design was verified by simulation and synthesized into Xilinx 4028EX FPGAs [15], which provide 1024 Configurable Logic Blocks (CLBs). The design was also verified in hardware using the GigaOps board. The utilization and maximum delays for the Packet Generator and IP Router module are in Table 1. Statistics for the Packet Sink module are not reported as that module is implemented using a logic analyzer. The router and generator, each with SIC logic, take up only a fraction of a FPGA. Note that the synthesized device only supports four routes and that external or on-chip CAMs or other devices will be required to support large routing tables. Nor was Internet Control Message Protocol (ICMP) [16] functionality implemented. Regardless, the control logic portion of a router is fairly small and other functionality, such as resource reservation, have the potential to be easily added in a small amount of hardware.

Maximum path delay is determined by tracing the longest route between various component types in the final placed design. Maximum pin delay is the maximum delay between any two pins in the final placed and routed design. Using the maximum path delay of 40 ns, the maximum clock speed for the device is 25 Mhz, which means that the maximum throughput for the device is 200 Mbps as an octet is processed on each clock pulse. This performance was verified using the hardware.

TABLE 1. PROTOTYPE PLACEMENT AND ROUTING STATISTICS

	CLB Utilization	Maximum Pin Delay	Maximum Path Delay
Generator	15% (149 CLBs)	20 ns	40 ns
IP Router	18% (179 CLBs)	18 ns	40 ns

The prototype's throughput and resource consumption can be easily improved by the use of FPGA-specific design of components such as adders and comparators. Further, the use of custom FPGA-board designs allows the use of wider data buses, which significantly improves throughput. Parallel hardware is easily built by replicating the set of stream modules on other FPGAs.

Note that the prototype merely performs route and other processing on the headers and, potentially, the data in a packet. Hardware to perform the actual switching or forwarding is also required and may reduce actual throughput.

## 8. CONCLUSIONS AND FUTURE WORK

Stream-based computation has been shown to be useful in a soft radio testbed [17]. The results presented here show that stream-based computation has significant potential for use in reconfigurable routers as well. A router with basic functionality will utilize an insignificant amount of CLBs on future FPGAs and careful design and use of parallel processing techniques will help increase clock rates. Clearly, many functions traditionally implemented in software have the potential to be implemented in hardware for a significant performance boost.

A stream-based processing model in network devices is a natural extension that simplifies design by allowing digital logic devices to be used and has the potential to increase throughput by applying superpipelining and superscalar techniques. The prototype is a useful model for experimentation as it is built using uniform stream modules. Because of their potential run-time reconfigurability, the stream modules could be upgraded to support advanced packet filtering operations or completely changed to other functionality. Thus, the basic router elegantly demonstrates that stream modules can provide the basis for a fast and flexible reconfigurable router.

More efficient routers that fully support IP routing must be built and studied before the benefits of reconfigurable routers can be fully understood and utilized. Integrating adaptive computing technologies with software techniques of active networks and open signaling has the potential to revolutionize how network devices, and the network itself, are viewed. In this model of a reconfigurable router, protocol developers can achieve hardware-level performance without knowledge of hardware design. This enables high-performance active networking as well as "low in the stack" active networking. Everything, except the physical interfaces, now has the potential to be dynamically updated. Terabit routing, network security, firewalls, distributed computing, and embedded network devices are only some of the areas that may benefit from this approach.

Additional research, however, must be performed to fully understand the interactions between the different technologies as well as the technologies themselves. Different router architectures and hardware configuration techniques must be explored. Design tools must also be created to support the

new technologies. As part of this effort, the authors are studying how to integrate adaptive computing technologies with active network technologies, through Active VHDL, JBits, and JHDL.

## REFERENCES

- [1] S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Communications Magazine*, Vol.36, No.5, pp. 144-51, May 1998.
- [2] S. Lin and N. McKeown, "A Simulation Study of IP Switching," *Computer Communications Review*, pp. 15-24, 1997.
- [3] IBM Corp., "IBM 8265 ATM Switch Overview," *White Paper*, September 1997.
- [4] N. Yamanaka, E. Oki, H. Hasegawa, and T.M. Chen, "Active-ATM: User-Programmable Flexible ATM Network Architecture," *Workshop on Active Networking and Programmable Networks (at International Conference on Communications)*, June 11, 1998.
- [5] "OPENSIG Fall'97 Workshop," October 6-7 1997, New York, NY, <http://comet.ctr.columbia.edu/opensig/activities/fall.97.html>.
- [6] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden, "A Survey of Active Network Research," *IEEE Communications*, January 1997, pp. 80-86.
- [7] D.L. Tennenhouse and D.J. Wetherall, "Towards an Active Network Architecture," *Computer Communication Review*, Vol. 26, No. 2, pp. 5-18, April 1996.
- [8] J. Peterson, R. O'Connor, and P. Athanas, "Scheduling and Partitioning ANSI-C Programs onto Multi-FPGA CCM Architectures," *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, California, pp. 178-87, April 1996.
- [9] Xilinx, Inc., "Xilinx: Silicon Xpresso - Java-based Design Environment for Programmable Logic Design," *Web Document*, December 9, 1998 (December 21, 1998), <http://www.xilinx.com/products/software/sx/sxpresso.html>.
- [10] P. Bellows and B. Hutchings, "JHDL-An HDL for Reconfigurable Systems," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, California, pp. 175-184, April 15-17, 1998.
- [11] R. Bittner and P. Athanas, "Wormhole Run-time Reconfigurable," *ACM/SIGDA Int. Symposium of FPGAs*, Monterey, CA, pp. 79-85, February 1997.
- [12] I. Hadzic, W. Marcus, and J. Smith, "On-the-fly Programmable Hardware for Networks," *GLOBECOM '98*, Sydney, Australia, November 1998, To appear.
- [13] J. Postel, "Internet Protocol," *RFC 791*, September 1981.
- [14] "SPECTRUM Reconfigurable Computing Platform Documentation," Release 2.5, Giga Operations Corporation, Berkeley, CA, 1997.
- [15] Xilinx, "Xilinx Data Book," 1998, <http://www.xilinx.com/partinfo/databook.htm#FPGA>.
- [16] J. Postel, "Internet Control Message Protocol," *RFC 792*, September 1981.
- [17] S. Swan, S. Harper, and P. Athanas, "A Stream-Based Configurable Computing Radio Testbeds," *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, California, April 1998.