*First publ as: Paper / Hot Interconnects 9, Stanford, CA, USA, 2001, pp. 93-97*

# Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware

Florian Braun John Lockwood Marcel Waldvogel
Applied Research Laboratory*
Washington University in St. Louis

## Abstract

*A library of layered protocol wrappers has been developed that process Internet packets in reconfigurable hardware. These wrappers can be used with a reprogrammable network platform called the Field Programmable Port Extender (FPX) to rapidly prototype hardware circuits for processing Internet packets. We present a framework to streamline and simplify the development of networking applications that process ATM cells, AAL5 frames, Internet Protocol (IP) packets and UDP datagrams directly in hardware.*

## 1. Introduction

In recent years, Field Programmable Gate Arrays (FPGAs) have become sufficiently capable to implement complex networking applications directly in hardware. By using hardware that can be reprogrammed, network equipment can dynamically load new functionality. Such a feature allows, for example, firewalls to add new filters that can run at line speed. The Field Programmable Port Extender has been implemented as a flexible platform for the processing of network data in hardware. The library of wrappers discussed in this paper allows applications to be developed that process data at several layers of the protocol stack. Layers are important for networks because they allow applications to be implemented at a level where the details of a protocol layer can be abstracted from the layers above and below. At the lowest layer, networks modify raw data that passes between interfaces. At higher levels, the applications process variable length frames or Internet Protocol packages. An Internet router or firewall, for example, use 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 User Datagram Protocol messages by instantiating all wrappers discussed in section 3 and as shown in Figure 1.

*This research was supported in part by NSF ANI-0096052 and Xilinx Corp.

## 2. Background

In the Applied Research Lab at Washington University in St. Louis, a set of hardware and software components for research in the field of networking, switching, routing and active networking have been developed. The Field Programmable port extender (FPX) has been developed to enable modular hardware components to be implemented in reprogrammable logic. The modules described in this document are primarily targeted for the FPX, though the design is written in portable VHDL and could be used in any FPGA-based system.

### 2.1. Switch Fabric

The central component of this research environment is the Washington University Gigabit Switch (WUGS) [5]. The WUGS is a fully featured 8-port ATM switch, which is capable of handling up to 20 Gbps of network traffic. Each port is connected through a line card to the switch. The WUGS allows hardware to be inserted between the line cards and the backplane.

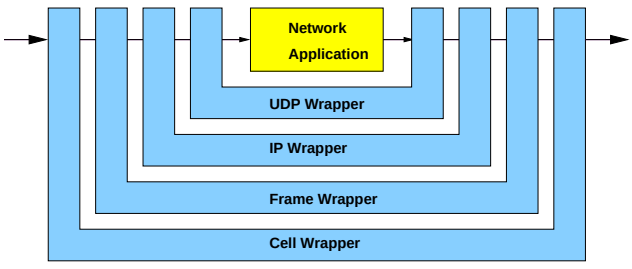


**Figure 1. Wrapper concept**

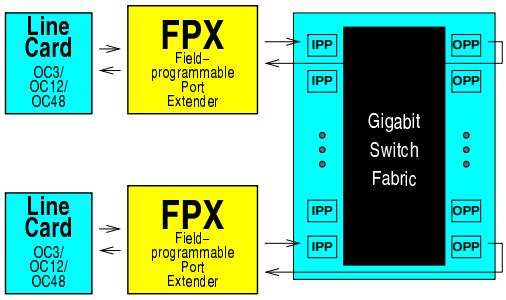


**Figure 2. WUGS configuration using the Field Programmable Port Extender**

### 2.2. Field Programmable Port Extender

The Field Programmable Port Extender (FPX) [8, 7] provides reprogrammable logic for user applications. It provides interfaces to both the WUGS and a line-card, so it can be inserted into a switch as illustrated in Figure 2.

Figure 3 illustrates 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. It also provides the logic to dynamically reprogram the RAD. The RAD can be programmed to hold user-defined modules. This feature enables user-defined network modules to be dynamically loaded into the system. The RAD is also connected to two SRAM and two SDRAM components. The memory modules can be used to cache cell data or hold large tables.

### 2.3. FPX Modules

User applications are implemented on the RAD as modules. Modules are hardware components with a well-defined interface which communicate with the RAD and other infrastructure components. The basic data interface is a 32-bit wide Utopia interface. Internet packets enter the module using classical IP over ATM encapsulation and segmentation into ATM cells [10]. The data bus carries header and payload of the cells. The other signals in the module interface are used for congestion control and to connect to memory controllers to access the off-chip memory. The complete module interface is documented in [12].

Usually, two application modules are present 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). As with the Transmutable Telecom System [9], modules can be replaced by reprogramming the FPGA in the system at any time. In the case of the FPX, this

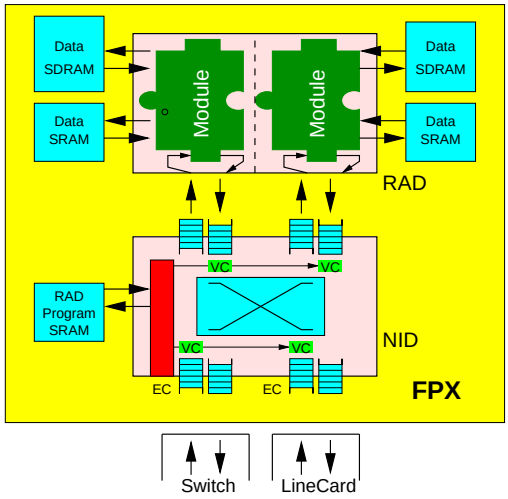


**Figure 3. Components on an FPX board**

functionality occurs via partial reprogramming of the RAD FPGA.

## 3. Network Wrapper Concept

Components have been developed for the FPX that allow applications to handle data on several protocol layers. Similar circuits have been implemented in static systems to implement IP over Ethernet [6]. Unlike systems that offload protocol processing to a co-processor [1], this library allows all packet processing functions to be implemented in hardware.

Translation steps are necessary between layers. A classical approach creates components for each protocol translation. In our approach, we combine these two translation units into one component, which has four interfaces as a consequence: two to support the lower level protocol and two to provide a higher level interface, respectively. Furthermore, some components are connected to each other. This is useful for exchanging additional information or to bypass the application. The latter is done in the cell processor (section 3.1).

When an application module is embedded into a protocol wrapper, the new entity surrounds the user's logic like the letter U (Figure 1). Regarding the data stream, the application only connects to the translating component, which wraps up the application itself. Therefore we will refer to the surrounding components as *wrappers*.

To support higher levels of abstraction, the wrappers can be nested. Since each has a well defined interface for an outer and an inner protocol level, they fit together as shown in Figure 1. As a result, we get a very modular design method to support applications for different protocols and

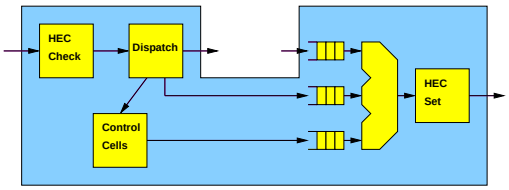


**Figure 4. FPX Cell wrapper**

levels of abstraction. Associating each wrapper with a specific protocol, we get a layer model comparable to the well known OSI/ISO networking reference model. This modularity gives application developers freedom to implement functions at several protocol layers in their designs. They can interface their logic to a wrapper with the level of abstraction appropriate for their specific application. User-level applications, for example, can completely ignore handling of complicated protocol issues, like frame boundaries or checksums.

### 3.1. The Cell Wrapper

The wrapper on the lowest level is the cell processor (Figure 4). It performs cell level operations that are common to all FPX modules. First, incoming ATM cells are checked against their Header Error Control (HEC) field, which is part of the 5 octet header. An 8 bit CRC is used to prevent errored cells from being misrouting. If the check fails, the cell is dropped.

Cell that are accepted are next processed by their virtual channel information. The cell processor distinguishes between three different flows:

1. The cell is on the data VC for this module. In this case, the cell will be forwarded to the inner interface of the wrapper and thus to the application.

2. The cell is on the control cell VC and is tagged with the correct module ID. Control cells are processed by the cell processor itself. This mechanism is covered later in this section.

3. None of the above, i.e. this cell is not destined for this module. These cells are bypassed and take a shortcut to the output of the cell processor.

The cell processor provides three FIFOs to buffer cells from either of the three paths. A multiplexer combines them and forwards the cells to their last stop. Just before they leave the cell processor, a new HEC is computed.

The behavior of an FPX module can be modified via control cells. Control cells are ATM cells with a well-defined structure and provide a communication path between an external controller (e.g. software) and the on-chip modules.

A standard control cell format has been developed to transmit information between software [11] that controls the FPX and hardware modules. Control cells to the RAD contain a module ID field to address the application module. Some standard opcodes are understood by all FPX modules. Commands to change the VPI/VCI registers, for example, allow a module to dynamically change the flow which will be processed.

The control cell handling function inside the cell processor is designed to be very flexible, thus making it easy for application developers to extend its functionality to fit the needs of their modules. User applications typically support more control cell opcodes than the standard codes. This feature is usually used to configure the module or to interact with software components, so extendibility was an important goal in the design of the cell processor. The control cell processing framework performs CRC check and generation functions, buffering of common data structures, and implements a mechanism to share common information.

A master state machine waits for control cells destined for this module and then stores opcodes, user data, the CM field and sequence number. At the same time it also checks the control cell CRC. Every opcode has its own state machine. So adding a new command does not interfere with existing ones. Every state machine polls the master state, if a control cell with a valid CRC has been read and becomes active on its opcode. For any incoming control cell (request), a response cell should be sent, if the command has been processed successfully. Because there is a state machine for every opcode, which generates its own response cell, a multiplexer takes care of forwarding the correct one to the output port. Finally, every new control cell gets a valid CRC before it is forwarded to the cell multiplexer.

### 3.2. The Frame Wrapper

To handle data with arbitrary length, data is organized in frames, which are sent as multiple cells. Several adaption layers have been specified ([3, 4]), which differ in the property of being connection-oriented or connectionless, in the ability to multiplex several protocols over one virtual channel and to reorder cells during transmission.

ATM Adaption Layer 5 (AAL5) is widely used for IP networks [10]). In AAL5 datagrams or frames of arbitrary length are put into protocol data units (PDU).

The frame wrapper module for the FPX handles AAL5 frame data. Its interface is designed to provide application modules with the ability to transmit and receive variable length frames. The frame processor replaces the Start-of-Cell signal with three signals, namely Start-of-Frame (SOF), End-of-Frame (EOF) and Data-Enable (DataEn).

As the name indicates, SOF indicates the transmission of a new frame. Note that the Header-Error-Control (HEC)

is not available with this wrapper. It is assumed that only valid cells are passed to this wrapper and that valid HECs are generated from outgoing cells.

The Data-Enable signal indicates valid payload data. It is an enable signal for the data processing application. It is completely independent from the cell structure. Applications can therefore resize frames or append data easily. They can also generate new frames. The Data-Enable signal is not asserted when padding is sent, since it is not considered a part of the frame. The End-of-Frame signal is asserted with the last valid payload word being sent. Applications thus have enough time to start appending data to a frame, if necessary.

After the EOF signal, two more words are sent. These 8 octets represent the AAL5 trailer, including some additional information for this wrapper, that is used to recreate the length and the CRC field. It is essential that applications copy and forward the two additional words.

### 3.3. The IP Packet Wrapper

The IP processor was developed to support Internet Protocol based applications. It inherits the signalling interface from the frame processor and adds a Start-of-Payload (SOP) signal, to indicate the payload after the IP header, which can be of variable length. This wrapper serves three primary functions:

1. It checks the IP header integrity to verify the correctness of the header checksum. Corrupt packets are dropped.

2. It decrements the Time To Live (TTL) field. As of RFC 1812 [2] all IP processing entities are required to decrement this field. Once this field reaches zero, the packet should not be forwarded. This prevents packets from looping in networks due to mis-configured routers.

3. It recomputes the length and the header checksum on outgoing IP packets.

An IP header usually has the length of 20 bytes, or 5 words.[1] The whole header must be processed by a checksum circuit before any decision about its integrity can be made. The IP Processor computes and then compares the header checksum. On a failure, the IP-packet is dropped by not propagating any signal to the application. If the Time-To-Live field of an incoming packet is already zero, the packet is also dropped and an ICMP packet is sent instead. Otherwise the TTL field is decremented by one. Outgoing IP packets are buffered so that the actual length can be determined. The corresponding field in the header and the header checksum are set accordingly. Therefore a whole packet has to be buffered, before it can be sent out.

To save and share resources with other wrappers, the IP wrapper understands a protocol to update the contents of bytes earlier in the packet. The IP processor can apply changes to the packet payload for fields, such as a header, that were set when the packet originally streamed through the hardware. Update commands are optional and are inserted between the last payload word (EOF signal asserted hi) and the AAL5 trailer. An unused bit (15) in the AAL5 length field is used to indicate update words or the start of the trailer. The length field is also used to hold an error code, so that packets can be dropped before they are sent out. Update words contain a 16 bit update field and a 15 bit update offset address. The 16 bit word at the offset address in the buffer is replaced by the update field.

[1] This applies to the vast majority of IP packets that do not contain any IP options.

### 3.4. The UDP Datagram Wrapper

The UDP processor is a wrapper that supports connectionless communication between user level applications using the UDP/IP protocol. This wrapper computes and generates the UDP checksum and the length field in the header for outgoing datagrams. Incoming datagrams are also checked for the checksum, 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 replacing the SOP signal with the Start-of-Datagram (SOD) signal. Applications can simply process datagrams or even generate new ones without the need to interpret or generate UDP headers.

To determine the correct checksum for outgoing datagrams, the whole packet is buffered. Since the IP processor already buffers a full IP packet, it would have been a waste of on-chip resources to implement an additional buffer just for UDP. Instead, the UDP processor informs the IP processor about necessary updates in the packet and leaves the buffering to that wrapper.

## 4. Implementation Results

Wrappers have been synthesized to operate on the RAD FPGA on the FPX. The system clock on the FPX is 100 MHz and the RAD is a Xilinx Virtex XCV1000E-7. Table 1 summarizes the results of our framework. The first 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 second column specifies the maximum frequency of each synthesized wrapper. The third and the fourth columns give information about delays in clock cycles of data passing through the wrappers and are split into delays before (in) and after (out) an embedded application.

| Wrapper/Module | Space | | Speed | Delay(short) | | Delay(long) | | Throughput(short) | | Throughput(long) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | LUTs | rel | MHz | in | out | in | out | rel. | Gbps | rel. | Gbps |
| Cell Processor | 781 | 3% | 125 | 4 | 6 | 4 | 6 | 100% | 3.5 | 100% | 3.5 |
| Frame Processor | 1251 | 5% | 116 | 21 | 22 | 10 | 31 | 84% | 2.7 | 93% | 3.0 |
| IP Processor | 1009 | 4% | 109 | 36 | 39* | 24 | 197* | 84% | 2.6 | 93% | 2.9 |
| UDP Processor | 550 | 2% | 114 | 39 | 44* | 27 | 202* | 84% | 2.6 | 93% | 2.9 |

* see text

**Table 1. Implementation results of the wrappers**

The delays have been measured by sending ATM cells back to back, containing UDP packets. UDP packets with only one word (short) and packets with 512 bytes of payload (long) have been sent. The short datagrams fit in a single cell and therefore have the highest protocol overhead, representing the worst case scenario. The longer datagrams represent a common size, giving an average delay. Note that the delays marked with a (*) are dependent on the IP packet length, because the IP wrapper buffers an entire packet before forwarding it.

The last two columns show the relative and the theoretical maximum throughput in gigabits per second for each wrapper and for both the short and the long UDP packets.

## 5. Conclusions

We have presented a framework for IP packet processing applications in hardware. Although our current implementation was created for use in the Field Programmable Port Extender, the framework is very general and can easily be adapted to other platforms. A library of Layered Protocol wrappers has been implemented. Each handles a particular protocol level. By using an entity that surrounds an application module (a U-shape wrapper), the related logic to convert to and from a protocol are linked, increasing the flexibility and reducing the number of cross-dependencies. The common interface between layers simplifies development of hardware at all levels of the protocol stack.

The framework is useful for developers of networking hardware components. The complete IP processing framework only utilizes 14% of the RAD FPGA on the FPX, leaving sufficient space to implement user-defined logic.

## References


[1] E. A. Arnould, F. J. Bitz, E. C. Cooper, H. T. Kung, R. D. Sansom, and P. A. Steenkiste. The design of Nectar: A network backplane for heterogeneous multicomputers. In *Proceedings of the Third International conference on Architectural support for Programming Languages and Operating systems (ASPLOS-III)*, pages 205–216, Apr. 1989. Also available as Technical Report CMU-CS-89-101, School of Computer Science, Carnegie Mellon University, Pittsburgh.

[2] F. Baker. Requirements for IP version 4 routers. Internet RFC 1812, June 1995.

[3] B-ISDN ATM adaptional layer AAL Functional Description. CCITT: Recommendation I.362, 1991.

[4] B-ISDN ATM adaptional layer AAL Specification. CCITT: Recommendation I.363, 1991.

[5] T. Chaney, J. A. Fingerhut, M. Flucke, and J. S. Turner. Design of a gigabit ATM switch. Technical Report WU-CS-96-07, Washington University in St. Louis, 1996.

[6] H. Fallside and M. J. S. Smith. Internet connected FPGAs. In *Proceedings of Field-Programmable Logic and Applications (FPL)*, pages 48–57, Villach, Austria, Aug. 2000.

[7] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *Proceedings of FPGA 2001*, Monterey, CA, USA, Feb. 2001.

[8] J. W. Lockwood, J. S. Turner, and D. E. Taylor. Field programmable port extender (FPX) for distributed routing and queuing. In *Proceedings of FPGA 2000*, pages 137–144, Monterey, CA, USA, Feb. 2000.

[9] T. Miyazaki, K. Shirakawa, M. Katayama, T. Murooka, and A. Takahara. A transmutable telecom system. In *Proceedings of Field-Programmable Logic and Applications*, pages 366–375, Tallinn, Estonia, Aug. 1998.

[10] P. Newman et al. Transmission of flow labelled IPv4 on ATM data links. Internet RFC 1954, May 1996.

[11] T. Sproull, J. W. Lockwood, and D. E. Taylor. Control and configuration software for a reconfigurable networking hardware platform. In *submitted to Globecom 2001*, San Antonio, TX, USA, Nov. 2001.

[12] D. E. Taylor, J. W. Lockwood, and S. Dharmapurikar. Generalized RAD module interface specification on the field programmable port extender (FPX). http://www.arl.wustl.edu/arl/projects/fpx/references, Jan. 2001.