A Reconfigurable and Programmable Gigabit Ethernet Network Interface Card

Jeffrey Shafer and Scott Rixner

Rice University
Houston, TX
E-mail: {shafer,rixner}@rice.edu

Abstract

RiceNIC is a reconfigurable and programmable Gigabit Ethernet network interface card (NIC). It is an open platform meant for research and education into network interface design. The NIC is implemented on a commercial FPGA prototyping board that includes two Xilinx FPGAs, a Gigabit Ethernet interface, a PCI interface, and both SRAM and DRAM memories. The Xilinx Virtex-II Pro FPGA on the board also includes two embedded PowerPC processors. RiceNIC provides significant computation and storage resources that are largely unutilized when performing the basic tasks of a network interface. The remaining processing and storage resources are available to customize the behavior of RiceNIC. This capability and flexibility makes RiceNIC a valuable platform for research and education into current and future network interface architectures.

1 Introduction

Networking has become an integral part of modern computer systems. While the network interface has traditionally been a simple device that forwards raw data between the network and the operating system, its role is changing. More sophisticated network interfaces are being developed every day that perform functions such as TCP offloading [3], iSCSI [2], encryption and firewalling [1], remote direct memory access [6], and so on. Different network interfaces perform these functions in different manners and there are typically no common interfaces to utilize them. The wide variety of services that are migrating to the network interface clearly motivates the need for directed research into the most effective services and abstractions that can be implemented by the network interface. Unfortunately, no suitable platform exists to successfully carry out such research.

This paper presents the design of RiceNIC, a reconfigurable and programmable Gigabit Ethernet network interface card (NIC) that provides a substrate for performing research and education activities in network interface design. RiceNIC is built on top of a commercially available Avnet Virtex-II Pro Development Kit. The Avnet board includes Xilinx Virtex-II Pro and Spartan-IIE FPGAs, flash memory, SRAM, SDRAM, a SODIMM DDR SDRAM slot, and a copper Gigabit Ethernet physical interface and connector. The Virtex-II Pro FPGA includes two embedded PowerPC 405 processors that can be operated at 300 MHz. Custom NIC firmware is provided along with RiceNIC device drivers for both Linux and FreeBSD.

Using a single PowerPC processor, the RiceNIC is able to saturate the Gigabit Ethernet network link with maximum-sized packets. This leaves significant resources—including 50% of the reconfigurable logic elements on the Virtex-II Pro FPGA, a spare PowerPC processor, and hundreds of megabytes of memory—available on the RiceNIC to use for advanced networking research. For example, the RiceNIC FPGA hardware and firmware was modified in a research program to allow multiple virtual machines running on the same system to concurrently control a single NIC, increasing network throughput by 2-4 times. In addition, simpler educational activities are also possible without hardware reconfiguration, as the NIC's firmware and drivers are easily modifiable. For example, without any additional hardware support, a network address translation firewall can be implemented on RiceNIC that achieves bandwidths within 3% of line rate while only using one PowerPC processor. Therefore, the RiceNIC is a capable platform for performing both research and educational activities investigating the performance and behavior of current and future network interfaces.

The RiceNIC design is available for public use. The Avnet development kit with the FPGA board can be purchased commercially, while the custom RiceNIC software—including device drivers, NIC firmware, and FPGA bitstream programming files—is available for free download. The FPGA hardware design can be modified

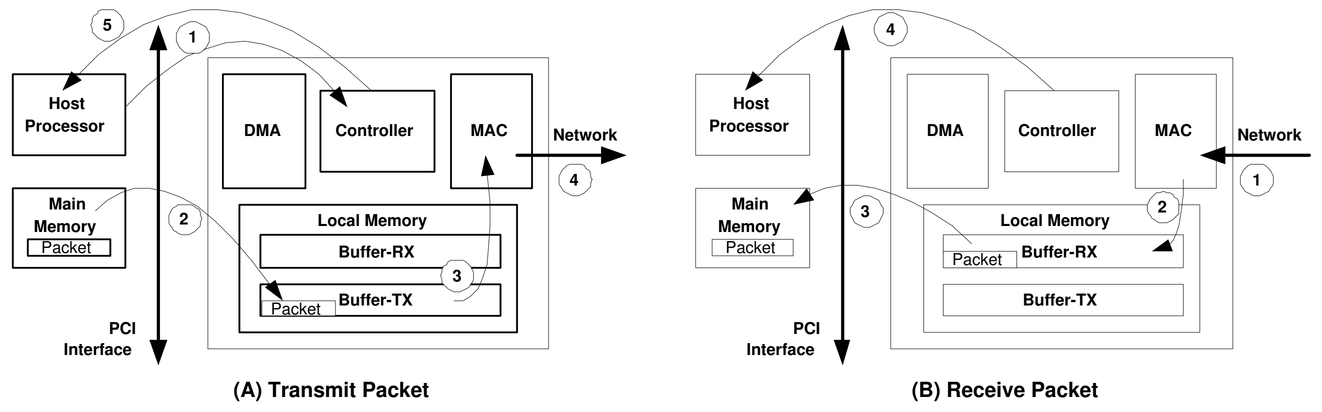

Figure 1. Data Flow to (a) Transmit Packets and (b) Receive Packets

if the developer independently obtains licenses for the embedded Xilinx PCI and MAC cores, which cannot be redistributed except in a compiled bitstream format.

In the remainder of this paper, Section 2 presents existing programmable and reconfigurable network devices used for research. Next, Section 3 introduces the Avnet FPGA development board used to construct the RiceNIC. Section 4 details the custom FPGA programming of the Avnet board. Then, Sections 5 and 6 present an evaluation of the RiceNIC. Finally, Section 7 concludes the paper.

2 Background

Many new system architectures have been proposed that integrate advanced network services onto the network interface card (NIC). However, commodity network interfaces are only designed to perform basic networking functions. Therefore, they provide just enough computational and memory resources to achieve the desired performance at minimal cost. This choice, however, limits the extent to which advanced network services can be implemented and evaluated on commodity NICs. Furthermore, even if the network interface does provide additional resources, they often do not provide the technical documentation, licenses, and tools that would be necessary to use them for research purposes. This section describes how commodity network interfaces work, discusses previous efforts to use them in research, and past attempts to develop more flexible alternatives. This motivates the need for RiceNIC, a network interface with considerable design flexibility, significant computational and memory resources, and open hardware and software specifications.

2.1 Network Interface Operation

The primary task of a network interface card is to enable the host system to transfer data between main memory and the network. Modern network interfaces all accomplish this task in a similar fashion, which will be described in this section. The RiceNIC is able to behave comparably, although it also provides additional flexibility and performance to augment or even completely change its behavior.

NICs connect to the host system via a local interconnect such as the Peripheral Component Interconnect (PCI) bus. A device driver running on the host system is responsible for communicating with the NIC over this interconnect. On the NIC itself, a small amount of memory, on the order of a few hundred kilobytes, is used as a transmit (TX) or receive (RX) circular buffer. These separate buffers are connected to a medium access controller (MAC) which implements the link level protocol. The MAC is attached to a physical interface (PHY) which performs the actual signal processing necessary to send the signal on the copper, wireless, or optical network.

The process employed by a generic NIC to transmit a packet over the network is shown in Figure 1, part (a). First, the host system creates a transmit descriptor containing the location and length in main memory of the packet to send. In step 1, the host system transfers this control descriptor to the NIC via programmed I/O. The NIC memory used in this transfer, referred to as *mailboxes* in the RiceNIC design, is a small region directly accessible by both the host system and the NIC. In step 2, the NIC examines the descriptor and initiates a DMA transfer to copy the data packet from main memory of the host system into the NIC transmit buffer. In step 3, the MAC starts reading the packet out of the buffer, and transmits it over the network in step 4. Finally, in step 5, the NIC notifies the host system, typically via an interrupt, that the packet has been transferred.

The process to receive a packet from the network is shown in Figure 1, part (b). The incoming packet is received from the network in step 1 and stored in the NIC receive buffer in step 2. The host system has previously allocated multiple receive buffers in main memory and trans-

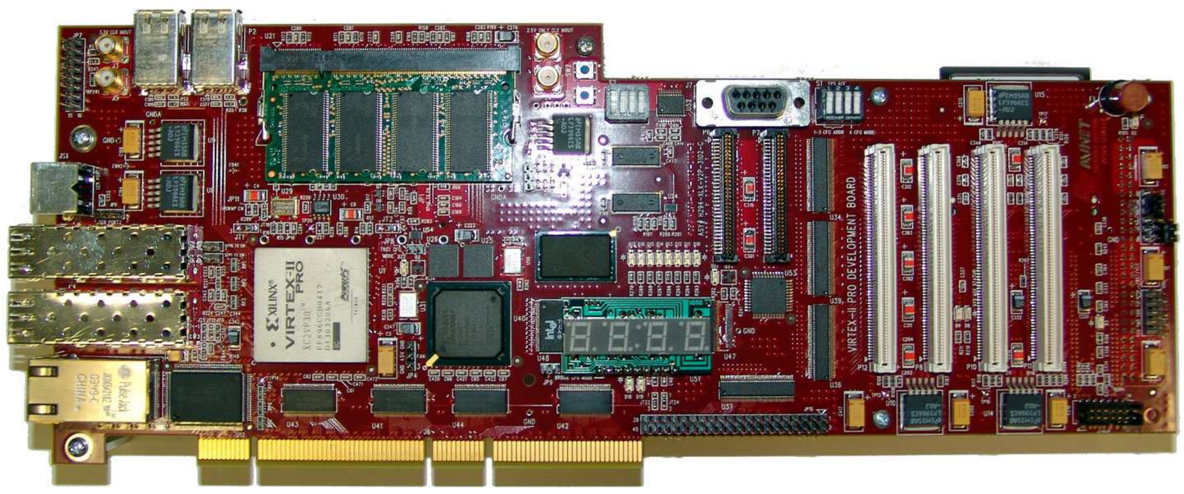

Figure 2. Avnet Virtex-II Pro Board

ferred their locations to the NIC. Thus, in step 3, the NIC transfers the received packet via DMA from the local buffer to the host main memory. Once the packet has been completely transferred, the NIC notifies the host system via an interrupt in step 4.

As these tasks are fixed, most commodity NICs are implemented with just enough computational and memory resources to achieve the desired performance at a minimal cost. This choice, however, makes them difficult to use for research purposes, since many active areas of research in network architecture involve performing additional network services on the NIC. Thus, researchers typically use more specialized network interfaces to gain performance or design flexibility.

2.2 Network Interfaces in Research

The complexity of the network subsystem makes it difficult to evaluate proposed modifications without an actual hardware implementation. The capability and flexibility of RiceNIC could provide significant benefits to projects of this type that would otherwise have to use older Ethernet-based programmable NICs or newer non-Ethernet programmable NICs.

The Tigon2 [4] is a programmable full-duplex gigabit Ethernet NIC that has been frequently used for network research. The NIC contains two in-order 88MHz single issue processors based on the MIPS instruction set. The processors share a 64-bit bus to 1 MB of 100 MHz SRAM. In addition, each processor has its own small scratch pad memory under program control. The NIC also has direct memory access (DMA) and medium access controller (MAC) hardware assist modules. The DMA module performs reads and writes over the 64-bit, 66 MHz PCI bus and the MAC module transmits and receives data over the network. Device drivers, firmware, and tools were publicly distributed for the Tigon, which was extensively used in research into message passing [14], firmware parallelization for 2-CPU NICs [9, 15], NIC data caching [10], and user-level network access [13].

In all cases, the authors did an excellent job of extracting as much performance from the hardware as possible. Some limitations in the Tigon2 NIC, however, were difficult to bypass. First, parallelization efficiency could be improved since the Tigon2 only implements a single semaphore for CPU synchronization. Second, the shared hardware units, such as the MAC and DMA, have no concept of concurrency and require external synchronization between the processors. The FPGA-based RiceNIC could be modified to add better synchronization between processors and allow parallel control of shared hardware units. Third, the memory resources of the Tigon2 NIC are extremely limited.

In addition to Ethernet NICs such as the Tigon, other academic research projects have used more capable specialized Myrinet/Infiniband NICs. For example, projects involving payload caching on routers and firewalls [20] and NIC-based intrusion detection [12] have used these specialized NICs. These projects are actually designed for, and best suited for, commodity Ethernet networks, not high performance supercomputing interconnects. Using the gigabit Ethernet RiceNIC produces a more convincing demonstration because, as an experimental platform, it is much closer to a final commercial product.

2.3 Reconfigurable Network Devices

In addition to pure software-programmable NICs, many FPGA-based NICs have also been developed [7, 8]. These devices are less capable than the RiceNIC, however, as the CiNIC only has a 10 Mb/s Ethernet link and 16 MB of SRAM, while the gigabit S-LINK NIC does not have any embedded processors on the Altera FPGA used. FPGA-based switches and routers have also been created [11], including the active “NetFPGA” design from Stanford [5, 17]. These boards and programming kits are freely available for academic and research use.

The second generation system, NetFPGA-v2, provides a four-port Gigabit Ethernet PHY, a Virtex-II Pro FPGA with two PowerPC processors, 4 MB of onboard memory, and a 32-bit/33MHz PCI bus interface [17]. This FPGA also provides high-speed serial links to connect several NetFPGA boards together. Although initially used as a classroom teaching tool, NetFPGA has recently been used for research into new network protocols and intrusion detection [17].

The NetFPGA and RiceNIC projects can be used in a complimentary fashion. NetFPGA is well suited for use as a router or switch due to its 4 network ports and slower PCI bus that limits communication to the host system. RiceNIC is better used as a NIC because it only has 1 network port and a faster (64-bit/66-MHz) PCI bus. Both systems could be used to build an entirely reconfigurable and programmable networking lab where all of the endpoints and routing fabric can be modified at will.

3 Hardware Platform

The RiceNIC was built on a commercial FPGA prototyping board, the Avnet Virtex-II Pro Development Board, which is shown in Figure 2. This board includes all of the components necessary for a Gigabit Ethernet network interface, as well as plentiful computation and storage resources. Therefore, the board is an excellent substrate on which to construct a high performance reconfigurable and programmable NIC. Furthermore, the commercial availability of this board obviates the need to custom-design a similar board.

The architecture and interconnection of the FPGAs, memories, and other devices on the Avnet board is shown in Figure 3. This board includes a Xilinx Virtex-II Pro 30 FPGA, which contains both reconfigurable logic and two embedded IBM PowerPC 405 processors that can run at 300 MHz. These in-order processors have 32 general purpose registers, separate 16 KB 2-way set associative instruction and data caches, and hardware multiply and divide units [19]. The Virtex FPGA is connected to a 10/100/1000 copper Ethernet PHY on the board, as well as an RS-232 serial port and high speed serial (Rocket I/O) links.

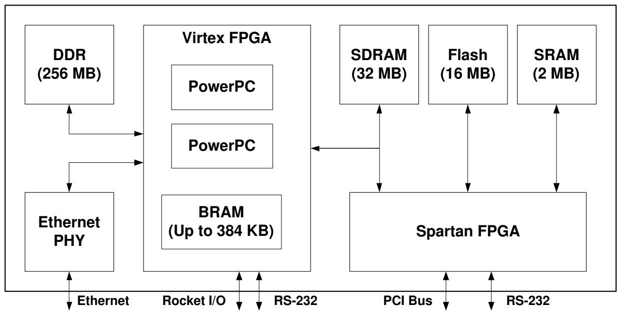

Figure 3. Avnet Board Architecture

The board also includes a Spartan-IIE 300 FPGA that is primarily used as the PCI controller. The card supports a 64-bit, 66MHz PCI interface which also powers the entire board. The Spartan FPGA is connected to that PCI interface and its own RS-232 serial port.

The Avnet board includes multiple on-board memories: 2 MB of SRAM, 16 MB of flash, 32 MB of SDRAM, and a SODIMM socket. A 256 MB DDR SDRAM module was added to replace the 128 MB module that is bundled with the development kit. In addition, a PROM and compact flash reader are provided for the purpose of programming the two FPGAs at power-on.

The architecture and interconnection of the FPGAs, memories, and other devices on the Avnet board is shown in Figure 3. While the development board does provide all of the necessary components for a Gigabit Ethernet network interface, it does impose a few architectural constraints on the RiceNIC. First, the PCI bus is connected only to the Spartan FPGA, which therefore must serve as the PCI controller. However, this allows the RiceNIC to operate in both 3.3V and 5V PCI slots, as the Spartan II supports both 3.3V and 5V I/O signaling, whereas the Virtex II Pro only supports 3.3V I/O signaling. Second, the Ethernet PHY and DDR SDRAM is only connected to the Virtex II Pro FPGA. This is not a serious constraint, as the Virtex FPGA is more capable and is a more reasonable location for the main functionality of the network interface. Finally, the data bus between the Spartan and Virtex FPGAs has limited bandwidth and is shared with the SDRAM. This means that the SDRAM cannot be used in a Gigabit network interface design as the full bandwidth of the bridge will be needed to transfer data between the PCI bus and the Virtex FPGA. Additional bandwidth between the FPGAs would be desirable to allow use of this memory. While the network interface must be designed around these constraints, there is still considerable flexibility to modify the FPGA configurations to tailor the network interface’s functionality and performance for specific application requirements.

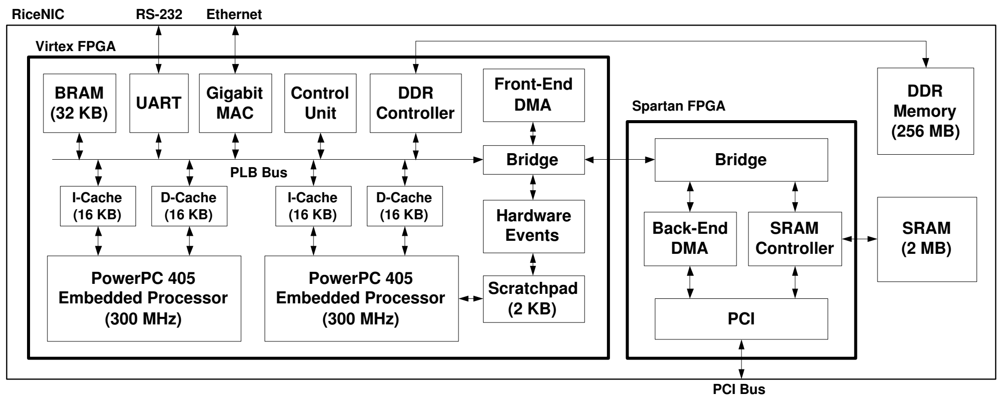

Figure 4. RiceNIC System Architecture

4 NIC FPGA Design

Figure 4 shows the overall organization of the RiceNIC. This architecture satisfies the constraints of the Avnet board while efficiently integrating the components of a Gigabit Ethernet network interface. The MAC unit, DMA unit, inter-FPGA bridge, hardware event management unit, and PCI interface were custom designed for RiceNIC. The remaining components were provided by Xilinx and used with little or no modification. Both the MAC unit and the PCI interface are built around low-level interfaces provided by Xilinx; however, those units are still mostly custom logic to integrate them into the rest of the system and to provide flexible software control over the hardware functionality.

The Xilinx Virtex-II Pro FPGA on the Avnet development board contains most of the NIC logic, including the PowerPC processors, on-chip memories, MAC controller, DMA unit front-end, and DDR memory controller. The smaller Spartan-IIE FPGA contains the PCI controller, the back-end DMA controller, and a SRAM memory controller. The SDRAM, although connected to a shared data bus between the Spartan and Virtex FPGAs, was not used because the entire bus bandwidth was needed to efficiently use the PCI interface.

To save development time, pre-built Xilinx cores were used for several of the hardware modules, including the PCI interface, DDR controller, and a low-level MAC. However, these cores can not be connected directly to form a working NIC. For example, although Xilinx provides a PCI core, it must be wrapped within a custom DMA unit to allow the PowerPC to initiate and manage high-performance burst transfers to/from the host system across the FPGA bridge. Similarly, although Xilinx provides a low-level MAC core, it must be outfitted with an advanced descriptor-based control system, data buffers, and a DMA unit to transfer packet data between NIC memory and the PHY. Finally, the DDR controller required modifications to function in this specific development board with its unique wiring.

The processors, memories, and hardware units are interconnected on the Virtex FPGA by a processor local bus (PLB). The PLB is a high performance memory-mapped 100 MHz 64-bit wide split-transaction bus that can provide a maximum theoretical bandwidth of 12.5 Gbits/sec in full-duplex mode. The PLB allows for burst transmissions of up to 128 bytes in a single operation, which is used by the DMA and MAC units to improve memory access efficiency.

Also attached to the PLB is a small memory-mapped control module used to route descriptors to or from the MAC and DMA hardware assist units. This module also provides a central location for hardware counters, event thresholds, and other low-level control functions. By attaching this central control unit to the PLB, either PowerPC processor can manipulate these important NIC functions. The control unit takes 18 PowerPC cycles to read and 12 cycles to write a 32-bit word, primarily due to bus arbitration delays.

Unlike any other commercially available network interface, RiceNIC provides a UART that is accessible over the PLB. This UART interfaces with a serial port on the Avnet board that can be connected to an RS-232 serial port on an external computer, allowing terminal access directly to the firmware on the network interface. This can be used both for the firmware to display status and debugging information to the terminal and to provide a command-line interface for querying the network interface as it operates. Such terminal access greatly facilitates debugging, development, and evaluation of network interface firmware.

The rest of this section explains how the components depicted in Figure 4 operate to form a functioning NIC.

4.1 Hardware Events

The RiceNIC provides an event-based architecture to notify the firmware running on the PowerPC when the hardware assist units have completed particular tasks. Existing assist units such as the MAC and DMA use this extensible event infrastructure, which can also support additional hardware units.

An event vector is stored in the scratchpad, which is connected to one of the PowerPC processors, as shown in Figure 4. The event vector is automatically updated by the hardware to indicate which hardware units have pending events. The hardware delivers nine different events to the firmware: four MAC events, four DMA events, and a mailbox event. The PowerPC also provides several timers that allow for periodic timer events.

Figure 5 shows one possible firmware event loop that utilizes the hardware event vector. In the figure, `hw_events`, a 32-bit unsigned integer (`uint32`), points to the location of the hardware event vector. The event vector is ordered such that the location within the vector indicates the event's priority. Higher priority events are stored in higher bit positions. A global software event vector, `sw_events` in the figure, can also be used to create software events. The software event bits should be chosen to not overlap with the preset hardware event locations, so that the software and hardware event registers can be merged with a simple bitwise or operation. Once they are combined, the processor can optionally mask off specific event bits to ignore them. Then, the PowerPC's `cntlzw` instruction (count leading zeros) can be used to determine the highest priority bit position in the event vector that is set, indicating an event is pending. The resulting event index can then be used as an offset into an array of function pointers to invoke the correct handler. To simplify the figure, the handler functions are assumed to be properly assigned to the `handlers` array. Note that if no bits are set, `cntlzw` returns 32, so the 32nd handler is the *null* event handler. Processing the event should clear it automatically by the associated hardware, so the next time through the event loop, lower priority events will be handled, unless a new higher priority event occurs.

Many other firmware organizations are possible, including polling the hardware units or other event loop architectures. However, using the hardware event vector can greatly improve system performance, as it is more efficient to poll a single location than to poll each hardware unit individually.

4.2 MAC Unit

The media access control (MAC) unit is responsible for transferring data between NIC memories attached to the PLB and the physical interconnect (PHY) module. RiceNIC extends the MAC capabilities to provide the PowerPC firmware more flexibility in controlling the MAC operation. The MAC unit in the RiceNIC combines the low-level Xilinx MAC core, which communicates directly with the PHY, with a custom wrapper that provides data buffers and control interfaces.

Figure 6 shows the MAC receive unit architecture. The

```
/* Array of event handler function pointers */
void (*handlers[32])(void);

/* Bit vectors of events */
volatile uint32 *hw_events = HW_EVENTS_ADDR;
extern uint32 sw_events;
uint32 event_vector;

/* Highest priority event */
uint32 event_index;

while (true) {
  /* Merge hardware & software event vectors */
  /* Must reload HW event vector every time  */
  event_vector = (*hw_events) | sw_events;

  /* Find first event bit that is set */
  event_index = cntlzw(event_vector);

  /* Invoke the corresponding event handler */
  handlers[event_index]();
}
```

Figure 5. Sample firmware event loop.

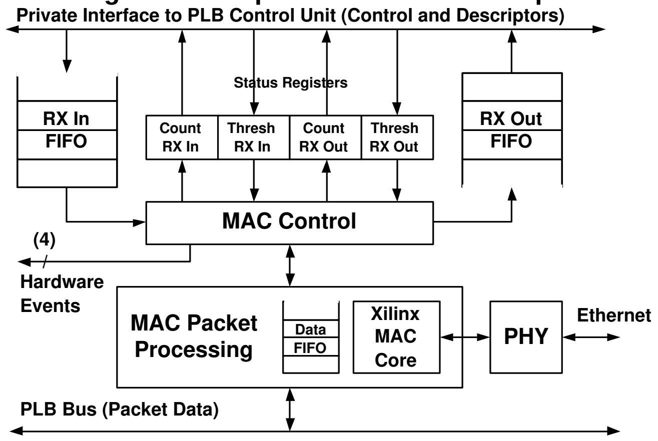

Figure 6. MAC Unit Receive Side Organization

transmit side contains an identical set of FIFOs and status registers. The NIC firmware running on the PowerPC manages the MAC through the use of 64-bit descriptors. These descriptors are transferred to the MAC unit, stored in FIFOs, and consumed by the MAC when it is able to process them. To receive data, the PowerPC must first allocate buffer space in the NIC memory to hold a sequence of packets. Then, it creates multiple receive descriptors containing the address of each buffer and transfers these to the MAC RX (receive) input FIFO via the PLB control unit. Up to 127 descriptors can be sent by the firmware and stored in the FIFO so that the MAC is always able to receive and properly store packets. Once a descriptor is received by the MAC, it operates independently of the firmware.

When a packet is later received from the PHY, the MAC packet processing module accepts it from the network and stores it in its temporary internal 16 KB data receive FIFO.

The hardware prevents the receive buffer from overflowing. Once the data receive FIFO contains more than 14 KB of data, all subsequent packets will be completely dropped until the FIFO has room for a new packet. Thus, standard Ethernet packets will either be received in their entirety, or not at all.

The receive FIFO only provides temporary storage for received packets. A received packet is automatically transferred over the PLB to the buffer location indicated by the next descriptor in the MAC RX input FIFO. If no descriptor is available, the packet remains in the MAC's internal receive FIFO until the PowerPC produces more receive descriptors. After the packet has been transferred to the specified location, a new descriptor is created by the hardware containing status information on the received packet, including the memory location at which it was stored. This descriptor is placed in the MAC RX output FIFO. The PowerPC should read this descriptor via the PLB control unit in order to determine that a packet has been received.

By using explicit descriptors under processor management, this architecture gives the firmware flexibility to manage the hardware functions of the MAC unit. Unlike conventional NICs, such as the Tigon [4], that use a contiguous circular receive buffer, the RiceNIC allows buffers to be placed arbitrarily in memory under firmware control. This allows the use of advanced algorithms, such as out of order processing of received packets, that would be difficult if a simple circular buffer was mandated by the hardware.

The firmware has two methods to monitor MAC progress and determine if a new RX input descriptor should be produced or an output descriptor consumed. The processor can poll two different count registers via the PLB control module to determine the number of descriptors stored in the RX input and output FIFOs. Or, the processor can use the hardware event vector. The MAC receive unit provides two separate hardware events to notify the firmware when more descriptors can be enqueued on the receive input side or when descriptors must be dequeued on the receive output side. Two additional count registers and events serve the same function for the transmit FIFOs.

These hardware events are triggered based on the number of descriptors in the corresponding FIFO. Each FIFO has an independent threshold that indicates when the event should be triggered, allowing the firmware to aggregate processing and increase efficiency. For example, if the MAC RX input FIFO threshold is set to 5, then that event will not be triggered until there are 5 available slots in the FIFO. When thresholds greater than 1 are used for the output FIFOs, the firmware must also periodically poll the count registers to ensure they will be serviced when packets are received or transmitted infrequently. The hardware events are automatically cleared when the output FIFO occupancy or input FIFOs vacancy drops below their thresholds.

The process of transmitting a packet is similar to the process described for receiving a packet. To send data, the PowerPC ensures that the entire packet is resident in memory, and then sends a descriptor to the MAC with the address (or a sequence of addresses if the packet is fragmented). The MAC unit autonomously transfers data from the appropriate memory on the PLB into its temporary 8 KB data transmit FIFO and then sends the data over the network in one continuous burst. These operations are pipelined for subsequent packets to make full use of the transmit FIFO.

An optional hardware checksum unit can assist by calculating the TCP checksum on both transmitted and received packets. For transmitted packets, the checksum is calculated when the packet is transferred from on-NIC memory to the MAC, and placed directly in the outgoing datastream at a user-specified location. This allows the NIC processor to modify the packet (or create new packets entirely!) and still gain the performance benefit of a hardware checksum module. For received packets, the checksum is calculated before the packet is stored in memory. It is delivered to the processor via the receive descriptor, allowing the processor to verify that the data was successfully received.

The MAC unit also implements a unique form of gather transmit. It can gather discontiguous regions from NIC memory for transmission as a single packet. This capability is very useful for NIC research, such as network stack offloading, where the NIC processor generates packets itself instead of merely forwarding data generated by the host system. In this scenario, the NIC does not need to waste time copying packet data to a contiguous buffer for transmission.

4.3 DMA Unit

All high performance NICs use direct memory access (DMA) to transfer data to the host system by reading and writing its memory directly. RiceNIC uses a custom-built DMA assist unit to transfer data between the NIC memories and the host memory via the PCI bus. The DMA unit can transfer data to/from any NIC memory unit with a controller on the Virtex that is connected to the PLB, which is all memories except for the scratchpad and SRAM.

As shown in Figure 4, the DMA unit is partitioned into two components. The front-end DMA engine contains the descriptor FIFOs and control registers, which are constructed in a similar fashion to the MAC unit. This unit is responsible for moving data between NIC memory and the bridge that connects the two FPGAs. The bridge transfers data between the front-end and back-end DMA controllers. The back-end DMA controller uses the Xilinx PCI core to move data between the Spartan FPGA and host memory. The DMA unit contains 2 KB read and write buffers that allow it to sink and source data for PCI bus transfers, which restricts the maximum sized DMA operation to 2 KB. The size of these buffers is physically limited by the available memory on the Spartan FPGA.

As with the MAC unit, the DMA unit is controlled by the firmware through the use of descriptors. The firmware writes 128-bit DMA descriptors to the DMA unit's input descriptor FIFOs. Each descriptor contains a NIC address, host address, length, and other flags. The DMA unit processes each descriptor and performs the data transfer asynchronously. Unlike the MAC unit, the DMA unit increments several counters when each transfer completes—there are no descriptor output FIFOs. The firmware uses these counters to track the progress of the DMA engine.

The first counter tracks the number of reads and writes completed since the NIC hardware was last initialized. These counts are maintained independently, but are read by the processor in a single 32-bit word PLB control unit transfer. The DMA write count uses the upper 16 bits and the DMA read count uses the lower 16 bits.

Two more counters track the number of reads and writes completed since the corresponding counter was last read. These counts are maintained independently and read by the processor at separate PLB control unit locations. Once the firmware has read either the read or write counter, it is reset to zero. This allows the firmware to easily determine how many DMA transfers have completed since it last checked.

In addition to the counters, the DMA unit also generates hardware events in a similar fashion as the MAC unit. These events indicate when the DMA unit is ready for another read or write descriptor and when a read or write transfer has been completed. Thresholds can also be used in a similar fashion to the MAC unit to aggregate firmware processing.

4.4 Mailboxes

To facilitate communication from the host to the network interface, RiceNIC provides special *mailbox* locations. These mailboxes are locations within the SRAM, shown in Figure 4, that can be written by the host using programmed I/O (PIO). They are used by the host to transfer control information to the network interface. Bulk data, in contrast, would be transferred using DMA by the NIC.

The low 512 KB of the SRAM memory is divided into 128 *contexts* of 4 KB each. These contexts may be used in any fashion by the device driver and the NIC firmware. However, within each context, the lowest 24 memory locations are mailboxes which trigger a hardware event. When any mailbox is written by PIO from the host system, the mailbox event is automatically generated by the hardware, notifying the firmware of a mailbox update. When the firmware chooses to process the mailbox event, it can efficiently determine which mailboxes have been written by decoding a hierarchy of bit vectors that are automatically generated by the hardware. These vectors are stored in the data scratchpad to allow low latency access. The first 128-bit vector in the hierarchy indicates which of the 128 potential contexts have updated mailbox events to process. The second array of 24-bit vectors in the hierarchy indicate which mailbox(es) in each context have been updated.

Once the specified mailbox has been identified, the firmware can read the appropriate SRAM location via the PLB and inter-FPGA bridge. After processing the information, the firmware can then clear that mailbox event by writing a mask of mailboxes per context to a special PLB control address. This allows multiple mailbox events to be cleared with a single store operation. Note that the mailboxes do not interfere with the normal operation of the 2 MB SRAM. The SRAM outside of the 128 contexts is always available as general-purpose storage accessible to both the PowerPC processors and the host. Furthermore, the firmware can choose to completely ignore the mailbox events and use the entire SRAM as general-purpose storage.

4.5 NIC Memory

RiceNIC includes a substantial amount of memory both immersed in the FPGA fabric and on external chips on the NIC card. The total memory provided is far in excess of the requirements of a basic NIC. Each memory on the RiceNIC has varying performance, capacity, and accessibility from the PowerPC, host system, and hardware assist units like the MAC and DMA.

Each processor has independent 16 KB instruction and data caches that can operate on any memory connected to the PLB. The caches have a single-cycle access latency. The firmware controls which memory regions are cached by manipulating a processor register mask.

A 2 KB word-addressable on-FPGA scratch-pad is directly connected to one PowerPC through an on-chip memory (OCM) interface, and is not accessible over the PLB. This high speed data bus is a private interface for a small amount of high-speed memory and is not connected to the data cache. 528 bytes of scratchpad are consumed by the hardware event vector and the mailbox bit-vectors. This region is read-only.

For general purpose storage, the Virtex contains two memories connected to the PLB: a 32 KB block RAM (BRAM) and a 256 MB DDR SODIMM. The BRAM is constructed from the same FPGA resources as the scratchpad, but has higher latency because it is accessed via the PLB. Note that the size of the scratchpad and BRAM can be altered by reprogramming the FPGA. The memory controller for the DDR SODIMM is on the Virtex FPGA. The current memory controller is preconfigured for a specific SODIMM module, but a modified DDR controller with different timing parameters could accept a larger SODIMM memory module. Together these memories are orders of

Memory Type	Controller Location	Storage Location	Capacity	Memory Burst	Latency	
					Read	Write
Data Cache	PowerPC	PowerPC	16 KB	N/A	1	1
OCM (Scratchpad)	Virtex	Virtex	2 KB	N/A	6	6
SRAM	Spartan	Off-Chip	2 MB	Not Supported	98	45
BRAM	Virtex	Virtex	32 KB	Disabled	27	21
				Enabled	7	5
DDR	Virtex	Off-Chip	256 MB	Disabled	51	30
				Enabled	10	12

Table 1. NIC Memory, Location, Capacity, and Access Latency (in PowerPC 300 MHz cycles)

magnitude larger than those found on a conventional network interface, enabling storage-intensive NIC research.

Finally, there is 2 MB of SRAM that is accessible to both PowerPC processors, via the PLB, and the host system, via the PCI bus through PIO. Therefore, the SRAM can be used to transfer data between the host and the NIC. As described in Section 4.4, the SRAM contains special mailbox locations. Furthermore, the firmware can trigger an interrupt of the host by writing to a specific SRAM address.

Table 1 shows the size, location, and latency of the various memories in the system as measured from the PowerPC processors. The DDR and BRAM memories can optionally be cached by the PowerPC. The cache uses 32-byte burst transfers between the cache and memory, greatly improving bus utilization and memory efficiency. Thus, the results are presented both with and without burst access averaged across 5 million accesses. In the disabled case, all accesses were performed to a single address, while in the enabled case accesses were swept across a region of memory in excess of the cache size. Because the custom bridge between the two FPGAs does not currently implement bursting, burst access mode is not available for the SRAM.

As the benchmarks show, the RiceNIC memories have widely differing performance and capabilities. The SRAM has very high latency due to its remote location across the FPGA bridge, but is very useful as shared memory between the host system and the NIC. The DDR module provides the most storage capacity, but at a higher access latency than the small on-chip BRAM and scratchpad. Given these differences, data should be placed in the appropriate memories based upon its size and the frequency with which it is accessed. Thus, the RiceNIC gives the programmer substantial flexibility to explore design tradeoffs regarding NIC memory capacity and price.

4.6 Hardware Utilization

Table 2 shows the FPGA utilization for the RiceNIC devices. The Virtex FPGA still has substantial resources available for future research and development even after implementing a functional NIC. For instance, less than 40% of the embedded BRAM has been consumed by the current design. In addition to the FPGA memory resources, over 50% of the reconfigurable logic elements are available for additional hardware assist modules to be added to the NIC. Additional modules can be easily connected to the main PLB for data transport and to the PLB control unit and hardware event register for efficient control.

FPGA	Component	Utilization	
Virtex	Slice Flip Flops	9,089 / 27,392	33%
	4-input LUTs (logic)	11,811 / 27,392	43%
	4-input LUTs (total)	13,126 / 27,392	47%
	Occupied Slices	9,164 / 13,696	66%
	BRAM	51 / 136	37%
Spartan	Slice Flip Flops	2361 / 6144	38%
	4-input LUTs (logic)	2504 / 6144	40%
	4-input LUTs (total)	4735 / 6144	77%
	Occupied Slices	3070 / 3072	99%
	BRAM	6 / 16	37%

Table 2. FPGA Device Utilization

The Spartan FPGA, in contrast, is essentially filled to capacity in the current design. Due to the design of the Avnet development board, shown in Figure 3, only the Spartan FPGA can contain the PCI core and SRAM memory controller. Fortunately, it is likely that most hardware modifications would be most effective on the larger and faster Virtex chip. If additional logic resources were required on the Spartan, the DMA data buffers stored there could be reduced from their current 2 KB size at a cost of reducing the burst length and efficiency of PCI transfers. Or, the SRAM controller and mailbox logic could be removed instead.

5 Performance Evaluation

To measure the RiceNIC performance, the card was tested in a modern Opteron server with dual Broadcom 5704C NICs. The RiceNIC installed in one system was directly connected to the onboard Broadcom NIC of a second similarly-configured system. On both machines, a Linux 2.6 kernel was used with the TCP stack configured to saturate the Broadcom NIC with minimal CPU utilization using

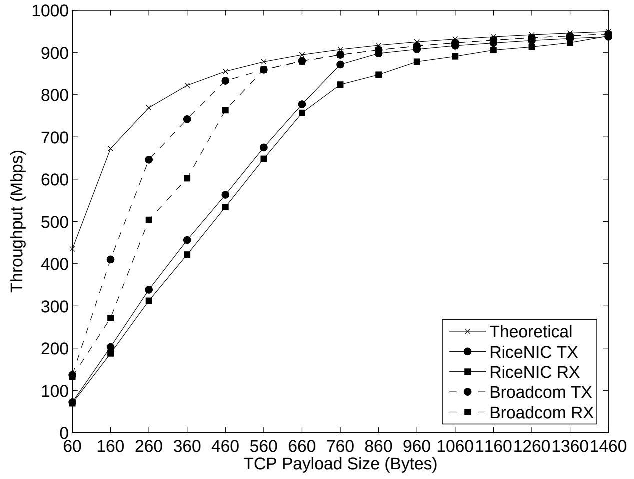

Figure 7. Network Throughput

a lightweight TCP streaming network benchmark. The host system was not the bottleneck in any test.

The RiceNIC was configured with custom packet processing firmware that only uses 1 of the 2 available PowerPC processors. This firmware is a baseline example that manages the hardware units and enables basic NIC functionality. Upon initialization, one PowerPC processor boots a custom loader contained within the Virtex programming. This loader allows the device driver to recognize the RiceNIC and download the appropriate firmware.

Figure 7 shows the throughput of the RiceNIC, the Broadcom NIC, and the theoretical Ethernet limit for a wide range of TCP packet sizes. The MTU of the transmitting interface was adjusted in each test to prevent the host operating system from merging small TCP packets into larger ones. For all results involving the Broadcom NIC, checksum offloading and scatter/gather I/O were enabled, but TCP segmentation offloading was disabled. For all RiceNIC results, all three features were disabled. Support for checksum offloading and scatter/gather require a new driver and updated firmware, currently in development.

For packet sizes larger than 960 bytes, the RiceNIC performance closely tracks the theoretical Ethernet limit. For smaller packets, the RiceNIC performance trails the theoretical limit, although the second unused PowerPC processor could be employed to increase the NIC performance. Note that the MTU was specifically lowered on the host system to transmit small packets. Otherwise, most operating systems would merge several small TCP packets into one large packet before delivering it to the NIC for transmission.

6 Case Studies

To demonstrate the flexibility of RiceNIC for networking research, this section presents two case studies. First, RiceNIC is modified to include network address translation (NAT) services. Obviously, NIC-based NAT services are not novel, but their implementation does serve to show how the RiceNIC firmware can be easily extended to implement additional services above and beyond basic networking. Second, RiceNIC is modified to support direct access from guest operating systems within a virtual machine monitor. This much larger modification requires changes to both the hardware and firmware. Both case studies show the flexibility of RiceNIC for advanced research projects.

6.1 Network Address Translation

In its most common configuration, NAT allows many systems on an internal network to share one address on an external network [16]. A NAT device examines each TCP packet in-flight and transparently rewrites its source or destination port and address to preserve the appearance that only 1 device is attached to the external network. The NAT functionality adds per-packet overhead to the NIC in several places. The headers of incoming and outgoing packets must be processed and a mapping table searched to determine the correct forwarding or dropping action for the NAT to perform. Finally, after updating the packet header based on the search results, the IP and TCP checksums must be partially recalculated, and the packet transferred to the internal or external network.

The RiceNIC firmware was modified to perform NAT services in which all internal nodes can initiate connections to the external network, but external nodes can only initiate connections to specific ports that are forwarded to internal servers. All traffic not belonging to either connection type is dropped. A server with one RiceNIC and one conventional NIC can then operate as a fully functioning NAT firewall. The RiceNIC is the interface to the external network and the conventional NIC is the interface to the internal network. IP forwarding is enabled on the host Linux system to forward all packets between the internal and external networks.

In this configuration, RiceNIC was able to sustain TCP stream throughput within 3% of the theoretical Ethernet limit for incoming and outgoing NAT traffic. The NAT processing code is running on the same PowerPC processor that performs all of the NIC management tasks, and that processor still has idle cycles remaining. In addition, the second PowerPC processor is completely idle and available for other tasks. While the implementation of a NAT module on a NIC is not a novel application, it shows how the RiceNIC firmware can easily be extended with new functionality and that substantial processing resources exist on the NIC for more advanced research.

6.2 Support for Virtualization

Virtual machine monitors (VMMs) allow multiple virtual machines running on the same physical machine to share hardware resources. To support networking, such VMMs must virtualize the machine's network interfaces by presenting each virtual machine with a software interface that is multiplexed onto the actual physical NIC. The overhead of this software-based network virtualization severely limits network performance.

The RiceNIC was used in research that eliminated the performance limits of software multiplexing by providing each virtual machine safe direct access to the network interface [18]. To accomplish this, network traffic multiplexing was performed directly on the RiceNIC, rather than in software on the host. This required both hardware and firmware modifications to RiceNIC. First, multiple contexts were added to the SRAM, as described in Section 4.4. A conventional network interface would only need a single context, but in order for the NIC to directly communicate with multiple guest operating systems in a virtual machine monitor, each guest needs its own set of mailbox resources. By making each context the same size as a physical page, 4 KB, the hypervisor can map each context into the address space of a single guest operating system. Second, the firmware was modified to independently communicate with the guest operating systems through these contexts. Third, the firmware was modified to perform network traffic multiplexing and demultiplexing. This requires about 8 MB of additional NIC memory, which easily fits within the 256 MB DDR SODIMM. Finally, the hypervisor was modified to communicate with the firmware to provide memory protection among the guest operating systems and ensure that they do not direct the RiceNIC to transfer data to or from physical memory that is not owned by that guest. Even with all of the firmware modifications, it was still not necessary to use the second PowerPC processor on the RiceNIC.

These modifications to the firmware and FPGA hardware of the RiceNIC resulted in significant networking performance improvements for virtual machine monitors. Both the system throughput with a single guest OS, and the system scaling as the number of guest OS' was increased, were improved by significant margins. (For final results, see [18]). These improvements would be difficult, if not impossible, to achieve with any other Ethernet network interface, and show the advantages of using the RiceNIC for research into future network interface architectures.

7 Conclusions

RiceNIC is a programmable and reconfigurable Gigabit Ethernet network interface. RiceNIC provides significant computation and storage resources that are largely unutilized when performing the basic tasks of a network interface. The remaining processing resources—including a spare PowerPC processor and over 15 thousand reconfigurable logic elements—and memory resources—including hundreds of megabytes of memory—are available to customize the behavior of the RiceNIC. This makes the RiceNIC an ideal platform for research into advanced networking architectures that require new services of the network interface.

RiceNIC is meant to be an open platform for such network interface research. The design is freely available for public use. The Avnet development board is commercially available and we provide the FPGA configuration and supporting software, including firmware and device drivers. Everything on the RiceNIC is easily modifiable and is oriented towards experimentation. The serial console makes the RiceNIC a friendly platform for research and education, as the NIC can easily display status information and the user can interactively control the NIC. The capabilities of the platform also provide the opportunity for other tools that are commonly only found on general-purpose systems. For example, a timer-based statistical profiler is also available for RiceNIC.

The case studies of the previous section highlight the usefulness of the RiceNIC. The NAT firewall was easily implemented by simply modifying the firmware. The support for direct network access from guest operating systems within virtual machine monitors was a much larger effort that encompassed both firmware and hardware changes. In both cases, RiceNIC provided more than enough computation and storage resources to provide the necessary performance.

The Avnet Virtex-II Pro Development board has proved to be an appropriate substrate for the implementation of RiceNIC. However, the board limits the NIC to use PCI and Gigabit Ethernet. In the future, we intend to investigate the design of a similar board with PCI Express and 10 Gigabit Ethernet capabilities. Much of the design is applicable to such a system, as long as the Xilinx PCI and MAC interfaces are upgraded appropriately. The MAC and DMA units could otherwise remain unchanged, unless Xilinx changes the user interface to their units. Furthermore, the PLB would need to be upgraded to a higher bandwidth bus to accommodate the ten fold increase in network traffic.

8 Acknowledgments

The authors thank Tinoosh Mohsenin for her initial design of a Gigabit NIC on an FPGA. This work is supported by gifts from Advanced Micro Devices and Xilinx, and grants from the Los Alamos Computer Science Institute and the National Science Foundation under grant No. CCF-0546140.

References

[1] 3COM secure copper NIC, 3CR990B.

[2] Adaptec iSCSI ASA-7211C gigabit ethernet adapter.

[3] T210-CX 10GbE protocol engine with TCP offload.

[4] Alteon Networks. *Tigon/PCI Ethernet Controller*, August 1997. Revision 1.04.

[5] M. Casado, G. Watson, and N. McKeown. Reconfigurable networking hardware: A classroom tool. *13th Symposium on High Performance Interconnects (HOTI'05)*, Aug 2005.

[6] D. Dalessandro and P. Wyckoff. A performance analysis of the ammasso RDMA enabled ethernet adapter and its iWARP API. In *Proceedings of RAIT Workshop*, Sep 2005.

[7] J. W. Dawson, D. Francis, S. Haas, and J. Schlereth. *High Level Design of Gigabit Ethernet S-LINK LSC*. Atlas DAQ, Oct 2001. Revision 1.3.

[8] J. Hatashita, J. Harris, H. Smith, and P. Nico. An evaluation architecture for a network coprocessor. In *Proceedings of the 2002 IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Nov 2002.

[9] H.-Y. Kim, V. S. Pai, and S. Rixner. Exploiting task-level concurrency in a programmable network interface. In *ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP)*. ACM Press, Jun 2003.

[10] H.-Y. Kim, S. Rixner, and V. Pai. Network interface data caching. *IEEE Transactions on Computers*, Nov 2005.

[11] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays*, Feb 2001.

[12] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda. Towards NIC-based intrusion detection. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2003.

[13] I. Pratt and K. Fraser. Arsenic: A user-accessible Gigabit Ethernet interface. In *IEEE INFOCOM*, Apr 2001.

[14] P. Shivam, P. Wyckoff, and D. Panda. EMP: Zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'2001)*, Nov 2001.

[15] P. Shivam, P. Wyckoff, and D. Panda. Can user-level protocols take advantage of multi-CPU NICs? In *Proceedings of the International Parallel and Distributed Processing Symposium*, Apr 2002.

[16] P. Srisuresh and K. Egevang. RFC 3022: Traditional IP network address translator. *Network Working Group*, Jan 2001.

[17] G. Watson, N. McKeown, and M. Casado. NetFPGA: A tool for network research and education. *2nd Workshop on Architecture Research using FPGA Platforms*, Feb 2006.

[18] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, , and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *International Symposium on High-Performance Computer Architecture (HPCA'07)*, 2007.

[19] Xilinx Inc. *Virtex-II Pro Platform FPGA: Complete Data Sheet*, Sep 2005. DS083 (v4.4).

[20] K. Yocum and J. Chase. Payload caching: High-speed data forwarding for network intermediaries. In *Proceedings of the 2001 USENIX Technical Conference*, Jun 2001.