Spinach: A Liberty-based Simulator for Programmable Network Interface Architectures[*]

Paul Willmann, Michael Brogioli, and Vijay S. Pai
Electrical and Computer Engineering
Rice University
Houston, TX 77005
{willmann, brogioli, vijaypai}@rice.edu

ABSTRACT

This paper presents Spinach, a new simulator toolset specifically designed to target programmable network interface architectures. Spinach models both system components that are common to all programmable environments (e.g., ALUs, control and data paths, registers, instruction processing) and components that are specific to the embedded systems and network interface environments (e.g., software-controlled scratchpad memory, hardware assists for DMA and medium access control).

Spinach is built on the Liberty Simulation Environment (LSE) and exploits LSE's modularity to support easy reconfiguration of programmable network interface cards (NICs) and embedded systems, enabling wide design space exploration with little or no code variation. For example, the same underlying C code is used whether supporting a uniprocessor Gigabit network interface, a multiprocessor Gigabit interface, or a multiprocessor 10 Gigabit interface with a highly heterogeneous memory system. The only difference is in a small number of lines of high-level scripting code used to configure the various modules into a simulation model.

Spinach is validated by modeling the Tigon-2 programmable Ethernet controller by Alteon Websystems running actual Ethernet processing firmware and by comparing the reported results to actual hardware benchmarks. Spinach is then used to obtain new insights about the performance of Gigabit and 10 Gigabit network interfaces.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*; B.4.4 [**Input/Output and Data Communications**]: Performance Analysis and Design Aids—*Simulation*

General Terms

Measurement, Performance, Design

Keywords

Programmable Network Interfaces, Simulation, Embedded Systems

1. INTRODUCTION

A computer system's network interface translates between the formats understood on the external network and the local interconnect. The most common are network interface cards (NICs) that support the Ethernet protocol externally and the PCI bus internally. Because the standard Ethernet protocol is well-defined, a NIC has a minimal set of required features, allowing most previous NICs to be built as ASICs. However, recent trends toward offloading networking services from the host CPU have motivated the use of programmable processors on NICs, as programmable processors provide increased flexibility, easier debugging and modification to support evolving protocols, and potentially lower design costs. Examples of tasks supported directly on programmable NICs include interrupt reduction, TCP protocol processing, caching of frequently requested content, TCP/IP checksum offloading, iSCSI protocol implementations, or message-passing [2, 5, 6, 8, 13, 15].

As NICs start to take on more tasks traditionally performed by the host CPU, the design and evaluation of programmable NICs will become a more important component of system architecture. Just as architectural simulation has become the dominant mode of evaluation for general-purpose programmable processors, it can also serve well for the field of programmable NICs. However, existing architectural simulators are not well-suited to programmable NIC designs. First, NICs with programmable processors also incorporate nonprogrammable units such as direct memory access (DMA) and medium access control (MAC) units that asynchronously interact with the host I/O interconnect, the external Ethernet, and the local NIC memory system. Second, NICs are by their very nature I/O intensive, and the workload consists not only of the firmware to implement those tasks, but also the I/O interactions of the NIC with the host and the Ethernet.

This paper presents Spinach, a new simulation infrastructure for programmable NIC architectures. Spinach models both system components that are common to all programmable environments (e.g., ALUs, control and data paths, registers, instruction processing) and components that are specific to the embedded system and NIC environments (e.g., software-controlled scratchpad memory, hardware assists for DMA and medium access control). Ease of configurability is fundamental to Spinach; this modularity and configurability allows the same modules of C source code to be used whether de-

[*]This work is supported in part by a donation from Advanced Micro Devices and by the National Science Foundation under Grant Nos. ACI-03050691, ANI-0216467, CCR-0209174, and CCR-0238187.

signing a uniprocessor Gigabit network interface, a multiprocessor Gigabit network interface, or a multiprocessor 10 Gigabit network interface with a highly heterogeneous memory system. Spinach is validated through modeling of and comparison with the Tigon-2 programmable Ethernet controller by Alteon Websystems, running actual Ethernet processing firmware. The Spinach Tigon-2 model relies only on high-level architectural parameters determined from publicly available documentation, exploratory benchmarks that reveal processor details, and visual inspection of the hardware's components, such as memory. For all configurations evaluated, the resultant Spinach model is accurate within 8.9% of real hardware performance and within 4.5% on average.

Spinach is an execution-driven simulator, but its I/O interactions are processed in a trace-driven fashion using a special-purpose harness that mimics the behavior of the host and Ethernet. This harness exploits the fact that Ethernet frames are processed in-order to avoid any intricate interactions with the actual execution of the firmware.

Spinach is comprised of 19 modules for the Liberty Simulation Environment (LSE), a system that enables extensive reuse of software building block modules and easy reconfiguration of designs [18]. Existing LSE-based simulators and modules, however, rely on an underlying emulator to maintain architectural state and use high level modules for system timing only. Such an emulator backend can introduce correctness issues in dealing with the asynchronous state interactions of the non-programmable NIC units and harness and can also complicate reconfigurability by requiring additional low-level state to be incorporated into the emulator for each functional change in the NIC design. In contrast to standard LSE modules, Spinach modules maintain architectural state internally. This removes the burden of maintaining and understanding complex state interactions and module source code beyond what the modules themselves exchange, and also allows Spinach to provide a one-to-one mapping of real system hardware to software modules.

This paper uses Spinach to obtain or confirm several key insights about programmable NIC design. Spinach confirms the results of Kim et al. indicating the benefits of multiprocessing in a NIC [7] and use the results presented there as a method of verifying the Spinach model of the Tigon-2 NIC architecture. Spinach then varies the memory bandwidth of a multiprocessor NIC to verify Kim's speculation that bandwidth is indeed a performance limiter for multiprocessor NICs. Finally, this paper shows Spinach modeling a 10 Gigabit network interface and discusses how Spinach facilitated the architectural design decisions for that system.

The remainder of this paper proceeds as follows. Section 2 gives background information on the operation of NICs and the Liberty Simulation Environment (LSE). Section 3 describes the new Spinach modules. Section 4 describes how Spinach is validated against an actual programmable NIC, and Section 5 describes studies that use Spinach to explore other NIC architectures. Section 6 discusses additional types of modeling performed using Spinach. Section 7 describes related work, and Section 8 concludes the paper and discusses future directions.

2. BACKGROUND

This section describes Spinach's background, focusing on the programmable network interface systems Spinach targets and the Liberty Simulation Environment on which Spinach is based.

2.1 Programmable Network Interfaces

Typical network interfaces have a PCI hardware interface to the host system and a wired, wireless, or fiber-optic link to an Ethernet network. The basic tasks of an Ethernet network interface card (NIC) are the same regardless of whether the NIC is programmable or not; the only difference is whether the flow of Ethernet frame processing through various states is controlled by a programmable CPU or an ASIC. Even programmable NICs typically have ASIC units to assist with performance-sensitive tasks such as direct memory access (DMA) interactions with the host or the actual transmission of data on the network.

The network interface is responsible for sending and receiving data by transferring data between the host memory and the network. The key difference between these two tasks is that a send is initiated by the host, whereas a receive arrives unsolicitedly from the network. Since the received data must ultimately be deposited in the host memory, the receive process actually consists of two portions: one in which the host first informs the NIC of pre-allocated memory available for later receives, and one in which the NIC actually transfers received data into the host memory. Although NICs vary somewhat in their particular implementation, the basic steps for sending and receiving are as described below.

To inform the NIC that a host memory buffer either has been prepared for sending as an Ethernet frame or has been pre-allocated for a later receive, the device driver in the host operating system writes the address and length of the host memory buffer in a special structure in host memory called a *buffer descriptor*. The device driver then informs the NIC of a new buffer descriptor through a programmed I/O operation. The NIC initiates a DMA read of the buffer descriptor from host memory. For sends, the NIC then uses the buffer descriptor to initiate a DMA read of the actual frame specified by the address and length. The result of that DMA is stored in the NIC's memory, and the NIC uses its MAC transmit unit to send the data on the network when allowed by Ethernet medium access policies.

Data from the Ethernet network is first received by the NIC's MAC receive unit, and the NIC then creates a buffer descriptor for the data. The NIC then writes the actual data and the buffer descriptor to host memory through DMA.

After any of the send, receive pre-allocation, or data receive sequences, the NIC may interrupt the host CPU to inform it that a send was completed or data was received. Because of the high rate of packet arrivals and completions, most Gigabit Ethernet NICs do not interrupt the host CPU until some threshold has been crossed, such as a certain number of frames sent or received or a timeout.

Ethernet frames may vary in size from 64 bytes to 1518 bytes. The above protocol processing steps do not depend on the size of the frame; however, the DMA and MAC units must each touch every bit of every frame, so all data must be touched at least twice in the NIC memory. This may lead to a memory bandwidth bottleneck at higher throughput rates, as suggested by previous work [7]. Insufficient bandwidth limits not only the DMA and MAC units, but may also slow down the firmware if the NIC processor contends with these units for memory bandwidth.

2.2 Liberty Simulation Environment

Spinach is built using the Liberty Simulation Environment (LSE), a system for creating simulators by composing hierarchical models comprised of individual code modules [18]. The programming model for Liberty consists of modules written in C and composed into higher-level modules using a language for specifying structural connections between modules. These structural connections are called ports, and they consist of data, enable, and acknowledge signals. Modules react to signals on their input ports; all invocation of modules takes place as a response to port activity rather than function calls (as in simulators that use functional interfaces between components) or through the scheduling of an event (as in explicit discrete event-driven simulators). As in most simulators, the modules can take ordinary values as their parameters. Unlike most other simulators, however, the modules can also take user-written functions as parameters; these functions are allowed to pre-process the input data and set the behavior of the module accordingly. Thus, key advantages of the LSE include the straightforward retargetability and reconfiguration of modules into simulators of various systems, as well as the easy integration of various simulators with each other. The latter is particularly straightforward since even a top-level simulator is itself an LSE module.

Vachharajani et al. have previously described the model of computation within the LSE, compared it to other simulation alternatives, and used the LSE to study microarchitectures [18]. Although not discussed in their paper, a major feature of the Liberty Simulation Environment is the *emulator* interface, in which a fast functional simulator maintains the architectural state of the system and the modules are only used for timing.

3. SPINACH MODULES

This section shows how to construct and configure Spinach modules to simulate the various structures in a programmable network interface. Section 3.1 explains how Spinach modules differ from standard LSE modules and motivates Spinach's specific contributions. Section 3.2 covers the modules common to all programmable systems, such as processors and local memories. Section 3.3 details the modules that are unique to this environment, such as those that manage host and Ethernet interactions, as well as some of the specialized memory modules specific to embedded architectures.

3.1 Comparison to Standard LSE Modules

Unlike standard Liberty modules, Spinach does not use LSE's emulator interface described in Section 2. Spinach is intended for architectures that include multiprocessing, self-modifying code, and memory interactions that are asynchronous to instruction execution via the nonprogrammable assists and harness. In such systems, the timing of instruction execution and state modifications may have an impact on the actual instruction sequences that are computed [9]. More importantly, by not maintaining system state in an underlying emulator, significantly less software engineering effort is required to migrate to multiprocessor based systems and beyond. For example, modifying an emulator-based system to model a dual processor design instead of a uniprocessor requires emulator modifications to support additional fetch engines, register files, access to shared memory regions, scratchpads, and other architectural state. This problem is only exacerbated when adding custom instruction sets or migrating to four- or eight-processor architectures required in 10 Gigabit NIC configurations. Spinach, however, encapsulates all architectural state into the modules themselves; modifying this state requires interactions over module ports. For instance, migrating from a uniprocessor NIC to an eight-way multiprocessor NIC simply required creating seven more processor and cache module instances and connecting them to the on-chip memory module via their port connections. No additional simulator code was required, nor was any additional system state needed. Hence, Spinach encourages exploration of radically different architectures by freeing the computer architect from understanding simulation implementation details that may affect correctness.

While LSE allows the user to build arbitrarily flexible simulators, Spinach allows the user to create and use modules that have a one-to-one mapping to corresponding hardware structures. This enables the user to accurately and easily verify concrete architectural changes to the system via simple high level module connections, effectively providing composable and heterogeneous multiprocessor systems with asynchronous actions.

3.2 General-Purpose Modules

Tables 1 and 2 present the Spinach modules that are generally applicable for programmable processors and their memory systems. A key benefit of Spinach in this context is that processor and memory modules acquired from other LSE users could be easily integrated in place of these modules; this paper uses new modules because the LSE still has few users and existing modules tend to use the emulator interface described in Section 2.

The processor modules are used to maintain the state of the processor pipeline. These include an ISA-specific fetch unit, an ISA-specific decode unit, and a configurable number of generic ALUs for the execute stage. Processor units such as the register file and pipeline latches are implemented through memory and pipe modules, described below.

Memory state modules maintain memory state internally, and other modules access this state through references to the address input ports with accompanying data and type specifiers. Addresses and data are arbitrarily wide, and the user may configure the number of ports. All writes in a given cycle are processed before reads, enabling internal forwarding. The same type of module is used for all memories in the system, including register files, scratchpads, main memory, and memory-mapped registers. The state information for the memory is maintained in the memory module itself; when coupled hierarchically with simple pipelined delay elements, the hierarchical module then manages both timing and state.

For the main memory, the burst timing is maintained through the memory controller module. The memory controller is configured by the user to detect contiguous memory bursts of desired length and enforce latency penalties between bursts. Furthermore, the memory controller segments the memory address space into separate regions; these are used in Spinach to separate accesses to memory-mapped registers from main-memory references.

Spinach provides one controller each for SRAM and DRAM. The SRAM memory controller has burst detection, parameterizable fixed initial latency, and enforces in-order completion and issue. The DRAM controller is similar except that the initial latency is variable according to parameterizable row activation latencies; the current model also enforces in-order completion and issue. Both controllers model burst-mode memories with a higher-latency initial access and higher bandwidth for later sequential accesses. The burst-mode parameters (initial latency, sequential access time, and burst/row-length) of each segmented physical memory region the controller manipulates are user-configurable. This allows Spinach to accurately model the bandwidth and latency of an SRAM-based NIC; future work will expand these memory controllers to support out-of-order memory completion via split transaction buses and will examine higher-bandwidth DRAM NIC architectures.

Instances of configurable memory arbiters handle all bus arbitration. The memory arbiter uses an algorithmically parameterized policy to choose which memory reference to issue next from a variety of different sources (CPUs or other hardware units). The memory arbiter is also used to handle arbitration for the separate DMA assist units that may simultaneously request host attention.

Spinach also includes five general-purpose utility modules that act as glue logic between the components units: an N-to-1 multiplexer, a 1-to-N demultiplexer, a *function* that determines its per-cycle output based on its inputs and a user-specified algorithmic parameter, *tee* that fans out one input port to several different outputs, and a *pipe* to provide a configurable pipelined delay between its input and output ports.

Module Name	Description	Inputs	Outputs
ALU	General-purpose math unit. Supports integer and floating-point operation.	Two data inputs, one op-code	One output, one flags port for condition codes.
MIPS Fetch Unit	Integrated program counter register, instruction register, and simulation controller for start and stop conditions. Accesses memory hierarchy for instructions. Specific to MIPS ISA.	Input PC, Instruction in, Halt Detect	Next PC (sent to pipeline), Instruction Out, Load PC (sent to memory to fetch next instruction.)
MIPS Decode Unit	Decodes a MIPS instruction into individual control signals for the pipeline that are generally ISA-independent.	MIPS instruction	Control Signals (Register Write, Instruction Format, Immediate Field, Memory Read, Memory Write, etc.)

Table 1: Description of general-purpose processor modules

3.3 Special-Purpose Modules

Table 3 lists the special-purpose modules in Spinach. The DMA and MAC assists are fully autonomous modules that act independently of the processor modules and control the transfer of data between the NIC memory and its external media. The external media for the DMA and MAC assists are the PCI bus and the physical layer of the network, respectively. The DMA assist and the transmit MAC assist poll a memory-mapped register that tells them to initiate their actions (using dedicated ports not shown in Figure 2), while the receive MAC assist initiates its actions when it sees input data from the physical network. Each of these units then has a queue in the NIC's memory that consists of structures indicating the memory addresses and lengths of the data of interest. In the case of the DMA assist, the structure includes the host-side address and the NIC-side address; the data is transferred between the two according to whether it is a DMA read or write. The queues for the MAC unit give the NIC memory addresses from which to transmit or into which to receive network data. Since both DMA and MAC assists must read and write from the NIC memory, they share the portions of their code that performs those tasks. A more aggressive approach for future versions of Spinach may be to unify those assists into a common module and then have the actual access to the external medium (whether PCI or Ethernet) controlled through a user-configurable function parameter.

The scratchpad filter module is a unit that combines filtering and arbitration to separate shared-memory references from private-memory references based on a configurable address map. Arbitration is included to allow a single port into the scratchpad memory being shared by the instruction fetch and data memory stages of the processor pipeline.

4. EXPERIMENTAL VALIDATION

Spinach is validated by composing a simulator that models the hardware and software of an existing programmable Gigabit Ethernet controller, the Tigon-2 by Alteon Websystems [1], and comparing the reported results against actual hardware benchmarks. The follwing sections describe the model and the experimental results.

4.1 Architectural Simulation Model for a Tigon-2 NIC

Figure 1 depicts the key components of the Tigon-2 architecture. The Tigon-2 includes two 88 MHz MIPS R4000-based CPUs, a private on-chip memory (called a *scratchpad*) for each processor, a shared off-chip SRAM, small hardware-controlled instruction and data caches, and nonprogrammable units to perform direct memory access (DMA) transfers to and from host memory and to send

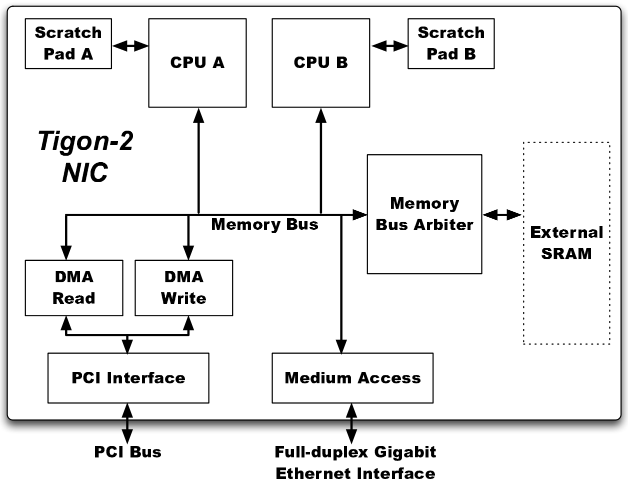


Figure 1: Block diagram of a programmable network interface based on the Alteon Tigon-2 programmable Gigabit Ethernet controller

or receive data on the network controlled by Ethernet medium access control (MAC). The scratchpad is of particular interest, as it is a fast software-managed memory that stores frequently used code and data. All firmware is initially loaded into the shared SRAM; any code that the firmware wishes to execute from the scratchpad must first be written into the scratchpad by the firmware itself, an example of self-modifying code. Each processor also includes small hardware-controlled caches that only cache contents from main memory (not the scratchpad). The instruction cache is a single 128-byte line for prefetching firmware code that is not found in the scratchpad. The data cache is a single 8-byte line with write-through and a no-write-allocate policy; this cache line size is chosen simply because the memory bus is 8-bytes wide and the cache allows the full memory width of memory to be used even though most instructions only operate on 4 bytes. The fact that the caches are only a single line indicate that they are intended primarily as spatial-locality prefetch buffers. Both caches and the scratchpad have single-cycle access.

Figure 2 shows how the Tigon-2 model is configured using the Spinach modules. Note that the 19 basic Spinach modules described in Section 3 are composed hierarchically into higher-level modules such as processor cores. Main memory, memory-mapped

Module Name	Description	Inputs	Outputs
Memory State	Maintains modifiable state on an array of bytes (endianness is user-configurable). Supports concurrent access through multiple ports, each one of which can either perform a read or write in a single memory cycle. Reads are not performed until all writes complete, enabling internal forwarding. Can be configured to "listen" to actions on specific addresses and generate output on special ports. Latency is modeled with separate modules.	Multiple channels of references (address, data, write-enable, and datatype)	Multiple channels of status, read-data (only valid on read), snooped port action.
Memory Controller	SRAM/DRAM memory controller with configurable support for variable-latency memories - may control multiple memory modules (each with different latencies) segmented by address. Ensures in-order retirement of memory state and in-order arrival of loaded data. 'Ack' on input ports signals reference acceptance.	Address, write-enable, data-to-write, datatype (size, alignment)	Read-data, latency until operation commits, and control signals to memory modules.
Memory Arbiter	Configurable memory channel arbiter. User may specify, with a function parameter, an arbitration policy according to which channel was last acknowledged (successfully won arbitration to higher level.) Stackable. Default arbitration policy is round-robin.	Multiple channels of address, write-enable, datatype, and data-to-write	Address, write-enable, datatype, and data-to-write of selected operation. Also, passed-back read-data and latency of operation.
Cache Controller	Configurable cache controller and cache modules for both instruction and data storage. Supports caches of arbitrary line lengths, associativity, and number of sets. Supports variable latency operations, write-back and write-thru policies, LRU and random replacement, prioritization of critical references on a cache miss, and address snooping for line invalidation in MP systems.	Requested reference (address, write-enable, data, datatype), cache data, fill data, and snooped address	Fill reference (address, write-enable, data, datatype) and cache reference (address, write-enable, data, and datatype.)

Table 2: Description of general-purpose memory modules

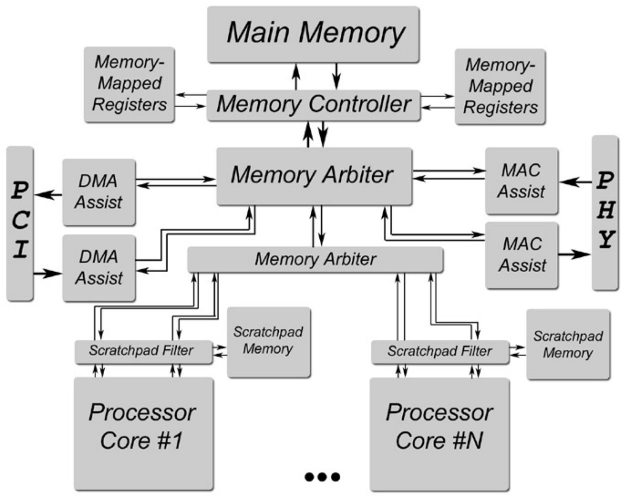


Figure 2: Mapping the Tigon architecture to Spinach modules

registers, and the scratchpads are all instantiations of the memory state module. Memory references are acknowledged using the acknowledgment signal on the LSE port, and acknowledgments are passed through the various levels of arbiters and filters back to the originator of the reference, such as the processor cores or assists. Thus, every level only needs to carry state about whether or not the reference was acknowledged by the next higher level, facilitating the simulation of this system's heterogeneous memory hierarchy.

This model results only from visual inspection, benchmarks, and research of publicly available documentation. The simulation modules are generic and include no low-level implementation details, such as an RTL description. Inspection of the Tigon-2 yields information about the specific SRAM it uses. Benchmarking the computation and memory system provides further insight into its microarchitectural charachteristics. Combining this information with the Alteon documentation yields an accurate, detailed architectural model that maps directly to Spinach modules.

Alteon Websystems released the firmware source code for their network interface [2]. This study specifically uses firmware based on Alteon's version 12.4.13, which was the last one distributed. This firmware version uses only one processor for all Ethernet data processing, using the second processor only for a timer loop. The firmware is self-modifying in that it explicitly copies code segments into the scratchpad memory and then executes those code segments. All versions of the Tigon firmware in this paper include minor changes to speed up initialization and account for the lack

Module Name	Description	Inputs	Outputs
DMA Assist	Configurable for unidirectional or bidirectional operation; this assist transfers data between the external interface bus (typically PCI) and the NIC memory system. It is configured based on parameters at fixed locations in the memory-mapped register region accessible to the DMA assist. Events detected by polling dedicated ports to memory-mapped registers.	Read-data from accessible memories and bus data In (data from PCI.)	Memory reference groups (address, data, write-enable, and datatype) for main data memory, memory mapped registers, and optional dedicated descriptor-only memory. Bus Data Out (data to PCI.)
MAC Assist	Configurable for unidirectional or bidirectional operation; this assist transfers data between the external physical layer and the NIC memory system. It is also configured based on parameters at fixed locations in the memory-mapped register region accessible to the MAC assist.	Physical Data In, NIC read data	Memory reference groups (address, data, write-enable, and datatype) for main data memory, memory mapped registers, and optional dedicated descriptor-only memory. Physical Data Out.
Scratchpad Filter	Specialized arbiter and filter used to steer memory references to a private memory unit and memory mapped registers. Also enforces higher-numbered channels must complete arbitration to higher levels before the lower-number channels are permitted to attempt arbitration; this ensures ordering of pipelined memory references which occur simultaneously but are time-shifted.	Multiple channels of references (address, data, write-enable, and datatype) and one channel each of returned load data from private memory, main memory, and memory-mapped registers	Passed-through channels of memory references (address, data, write-enable, and datatype) to private memory, memory-mapped registers, and main memory hierarchy.

Table 3: Description of special-purpose modules

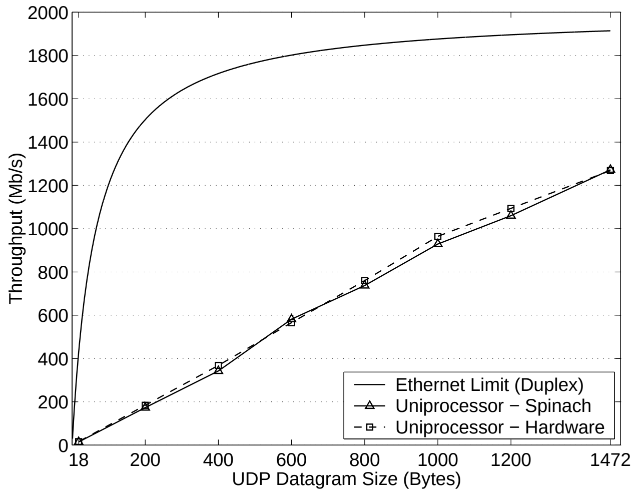
of a host device driver in the system under study, as the device driver normally initializes certain memory regions in the network interface according to the host system's configuration. The uniprocessor firmware in this paper runs all code (including the timer) on the single processor; the multiprocessor firmware uses the parallelization strategy of Kim et al. [7]. All of these changes still result in valid Ethernet firmware for the actual NIC, as the Ethernet frame-processing steps remain unchanged.

The network interface under test is studied using a set of Spinach modules that form an evaluation harness. The evaluation harness mimics the host and Ethernet interactions of the network interface, playing back traces of buffer descriptors based on actual sends, receive pre-allocations, and data receives of UDP traffic with packet sizes ranging from minimum-sized to maximum-sized (18-byte and 1472-byte payloads). The harness plays back the three traces as fast as possible while maintaining the ratio of actions of each the three types at any point in the simulation. In a real system, sends and receives would actually be correlated (e.g., TCP data transmissions and acknowledgments), but this behavior is not directly exploited by the Ethernet layer and is not captured by this harness. A more complex harness, however, could capture these behaviors by using TCP connection identifiers to determine which frames have been processed by the simulated NIC. The evaluation harness includes an Ethernet timing model and also includes a simple PCI model that adds a configurable amount of overhead to DMA transfers to and from host memory; the overhead is currently set to 30% based on the results reported by Kim et al. [6]. Because the simulated NIC has very little state that must warm up over the course of the run (e.g., no branch predictor, caches with only a few lines), convergence to a given throughput level is achieved within as few as 100 Ethernet frames.

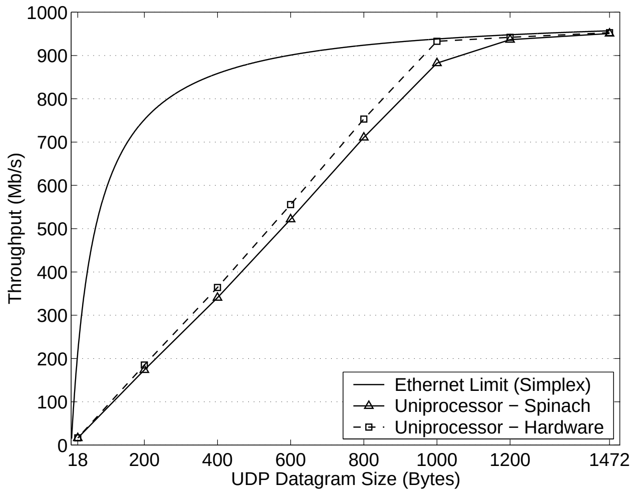
4.2 Validation Results

Figure 3 shows the UDP payload data bandwidth achieved in megabits per second on a Tigon-2 NIC operating with only one of its processors. Each graph compares Spinach modeling the Tigon-2 (labeled Uniprocessor - Spinach) against the performance of a 3Com Tigon-2 NIC (labeled Uniprocessor - Hardware); both the simulation system and the hardware system run the same version of the Tigon firmware that has been modified to operate on only one processor. The theoretical maximum UDP payload bandwidth on a 1 Gbit/sec physical link is provided as a reference; receive-only traffic has a theoretical maximum representative of its simplex nature.

Figure 3(a) presents total UDP throughput for bidirectional traffic, while Figure 3(b) shows total UDP throughput under a receive-only workload. For both the Spinach and 3Com tests, the bidirectional bandwidth reported is the maximum sum bandwidth measured under varying mixes of transmit-to-receive ratios. Typically this ratio is approximately 1:2 for both the Spinach model and 3Com NIC. This reflects the processing requirements for the two types of packets and the prioritization of the receive handler over the send handler, which ensures higher receive throughput [2]. Send packets require at minimum two data transfers from the host (one each for the header and payload) while receive packets require only one data transfer to the host. Each completed transfer requires processing by the NIC. Receive-only workloads are used as an additional reference point for verifying Spinach's model of the Tigon-2. For each uniprocessor case tested in both the bidirectional and receive-only scenarios, the performance of the Spinach system is within 8.8% of the 3Com NIC's performance and within 4.9% on average. As with the 3Com NIC, the Spinach model saturates only 66% of theoretical peak bandwidth when utilizing one processor.



(a) Bidirectional traffic



(b) Receive-only traffic

Figure 3: Spinach modeling uniprocessor Tigon

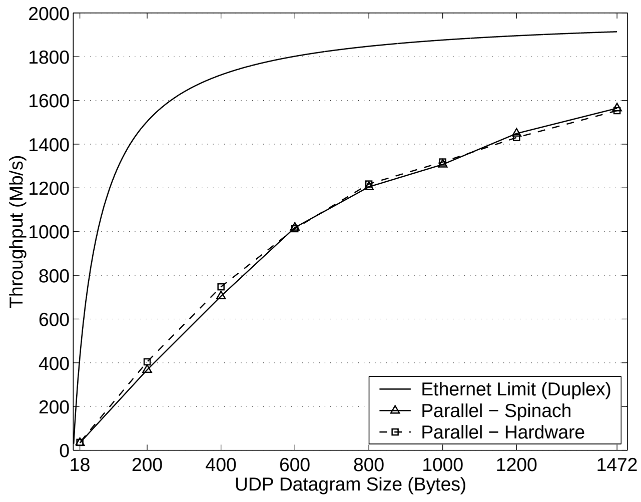


Figure 4: Spinach modeling Tigon with parallel firmware

Figure 4 shows the UDP payload bandwidth in megabits per second on a Tigon-2 NIC running Kim's parallelized firmware on both processors. The figure compares the 3Com Tigon-2 NIC (labeled Parallel - HW) against the Spinach model of the Tigon-2 architecture (labeled Parallel - Spinach). Traffic is full-duplex; in contrast to the uniprocessor case, traffic is mostly balanced on both processors since the firmware parallelization strategy puts most send-related tasks on one processor and most receive-related tasks on the other, avoiding resource contention among the processors . For each frame size tested, the multiprocessor Spinach model is within 8.9% of the benchmarks measured using the 3Com NIC. On average, the Spinach multiprocessor Tigon-2 model is within 3.2% of the real hardware. For maximum-sized UDP frames of 1472 bytes, both the Spinach model and 3Com NIC saturate 81% of the theoretical UDP peak throughput – an increase of 23% over the uniprocessor case. Creating a two-processor Tigon-2 Spinach model required less than 100 additional lines of high-level templated code in a script file.

5. FURTHER INVESTIGATIONS WITH SPINACH

Although validating Spinach against actual hardware is a useful exercise that shows the efficacy of modeling a network interface with this toolset, the true value of any simulator arises from its ability to model systems that have not yet been implemented. Such systems may be incremental modifications to existing architectures or radical changes to support vastly different goals. This section discusses using Spinach first to model a Tigon-2-based system with greatly increased memory bandwidth and second to model a novel 10 Gigabit per second Ethernet controller architecture.

5.1 Exploring the Tigon Memory Bottleneck

Kim et al. hypothesized that because firmware processing requirements are invariant of frame-size, limitations such as external SRAM bandwidth prevents the Tigon-2 architecture from achieving 100% of theoretical Ethernet peak throughput at maximum-sized UDP frames [7]. Spinach allows the verification of this hypothesis, since it is possible to increase the memory bandwidth while holding the processor frequency constant at 88 MHz by simply changing one parameter in one configuration file. Figure 5 shows the UDP payload bandwidth in megabits per second of two Tigon-2 Spinach models, comparing the base multiprocessor Tigon-2 Spinach model that uses 100 MHz SRAM (labeled "Parallel - Spinach (Base - 100 MHz SRAM)") and the same model modified to use SRAM operating at 250 MHz (labeled "Parallel - Spinach (250 MHz SRAM)"). As predicted by Kim's hypothesis, Spinach shows that the Tigon-2 architecture can fully utilize a 1 Gbit/sec full-duplex link when given adequate memory bandwidth. Interestingly, increased bandwidth also improves throughput for minimum-sized Ethernet frames, since the processors' memory accesses suf-

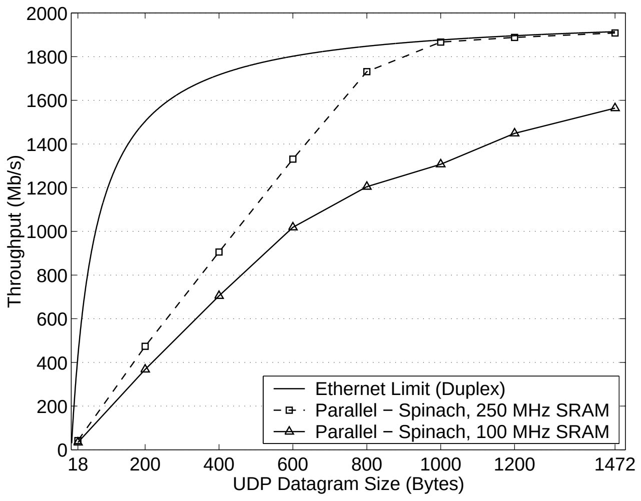


Figure 5: Spinach modeling Tigon with 250 MHz SRAM

fer fewer additional latencies caused by contention with DMA and MAC assist traffic. Spinach enables the investigation of this non-intuitive performance increase without perturbing the system as a runtime hardware profiler might.

5.2 Supporting 10 Gigabit Ethernet

As local area link speeds increase, a key target for modern network architectures is 10 Gigabit Ethernet. The greater speed of 10 Gigabit Ethernet obviously requires more processing power than Gigabit Ethernet; however, it is a challenge to incorporate the requisite processing power and memory bandwidth within the power constraints of the network interface environment. An exploration of the computation power and memory hierarchy requirements of 10 Gigabit Ethernet suggests that a NIC design with 8 MIPS R4000-based cores, a banked on-chip SRAM for frame metadata, high-bandwidth off-chip graphics DRAM for frame data, and separate instruction caches for each core can provide the needed level of throughput. However, such a system also requires firmware that can exploit a far greater level of parallelism than the systems described in Section 4.

Due to Spinach's inherent scalability, expanding beyond the Tigon-2 based multiprocessor design described in Section 2 required no additonal simulator source code or toolset modifications. Rather, to increase the number of processors, memory banks, caches and cache controllers, the number of module instances was simply increased and their respective input and output ports were connected to the on-chip memory interface module via high-level templated code.

Since no real hardware exists on which to develop the custom parallel firmware that runs on the 8-processor 10 Gigabit NIC, Spinach functions as a software development tool as well. Extensive debug information is maintained by all module instances, providing information such as instruction traces, memory reference traces, and timing. Spinach also includes non-intrusive source code profiling tools for the MIPS code that runs on the MIPS processor module used in these Spinach-based NIC simulators. The resulting firmware is based on a distributed task-queue model, with all task-processing functions written in a re-entrant style to enable simultaneous execution on multiple cores.

Figure 6 shows how the desired 10 Gigabit Ethernet controller architecture can be mapped to Spinach modules. Note that these

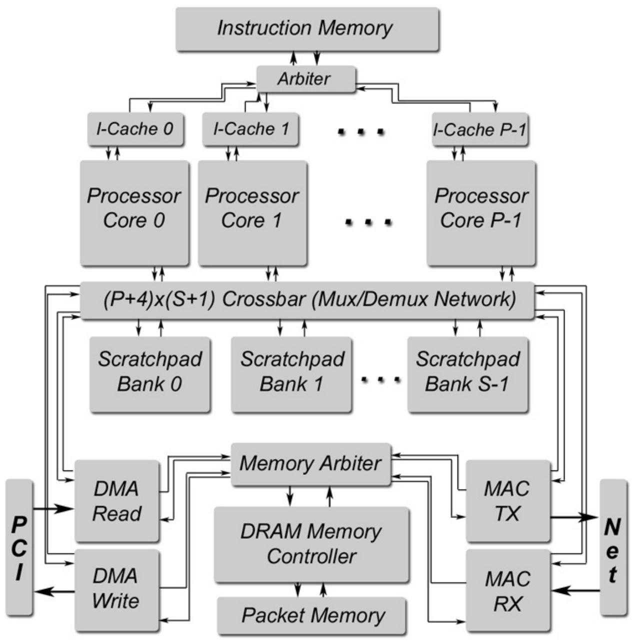


Figure 6: A scalable 10 Gigabit NIC architecture implemented in Spinach

are predominantly the same modules as used in the Tigon-2 model, simply connected in different ways. An implementation with 8 processors running at 200 MHz each, an on-chip memory network with 4 banks, instruction caches of 2 kB each, and GDDR SDRAM operating at 500 MHz fulfills the computational and memory bandwidth requirements needed to saturate the link; all elements are simulated with the base Spinach modules presented in this paper. Preliminary experimental results verify that this architecture saturates 98.6% of the theoretical peak full-duplex bandwidth, while initial scalability results show that operating with just 6 processor cores active shows 97.6% throughput.

6. DISCUSSION

In addition to simulating NICs, Spinach is designed to be arbitrarily flexible in simulating other types of embedded systems. Spinach includes user-configurable cache controllers and cache modules, as well as arbitration modules for the memory hierarchy, and variable-latency memory modules that can be used to simulate on-chip SRAM as well as off-chip SRAM and off-chip DRAM of various sizes. Using a different ISA would simply require a new ALU module definition and a new decode stage for the processor pipeline. Similarly, Spinach's flexibility also enables configuration of VLIW systems. Since the processor pipeline definition contains only high level module listings and no actual module source code, a DSP-style VLIW pipeline could be configured by replicating the width of the pipeline.

Virtually any special-purpose hardware that the user can envision in the system can be incorporated into a Liberty Simulation Environment module and plugged into the Spinach simulator of choice with relative ease. For example, recent work has added a reconfigurable functional unit into the execute stage of the NIC's processor to run application-specific instructions selected by the compiler. This reconfigurable unit is written as a Liberty module and follows a design similar to that of Ye et al. [20], with up to nine input source registers and a single output register. The compiler then folds basic blocks or other appropriate instruction sequences into a single reconfigurable fabric instruction when possible. Although the actual reconfigurable fabric is not a base module of Spinach, a user created an LSE module using C code to simulate the additional hardware needed and then added this module to the high-level module listings of a Spinach NIC simulator. The writer of the reconfigurable module required no intimate knowledge of the processor's internal workings; rather, the module only views processor state through the values on its input ports at the start of every clock cycle.

7. RELATED WORK

Sections 1 and 2 describe the background for Spinach. This section describes other related work in detail.

Crowley et al. have investigated the performance characteristics of programmable network interfaces and their impact on processor design, but the workloads evaluated have been computation-bound kernels (MD5, encryption, and IP-forwarding) rather than I/O-bound Ethernet firmware [3]. Consequently, that work used conventional processor simulators without the need to include support for special-purpose features such as DMA or MAC assist logic. In contrast, Spinach enables investigation of I/O interactions and their impact on bottlenecks such as memory bandwidth, which has been highlighted as a problem for networking performance by several recent works [7, 12, 16].

Efforts to simulate communication intermediaries in multiprocessor systems have substantial similarities with Spinach. For example, Heinrich et al. used simulation to determine the performance overhead of the MAGIC programmable communication controller in the Stanford FLASH multiprocessor through comparison to a system that could process communication protocol operations with no latency [4]. They found that the overheads of a realizable programmable controller were either small or could be hidden effectively for many full-scale multiprocessor applications. Kranz et al. studied the performance impact of integrating message-passing and shared-memory communication models in the MIT Alewife machine [10]. Although their system also included newly designed processors and networks, this integration was achieved largely through their network coprocessor, titled the Communication and Memory Management Unit (CMMU). Mukherjee et al. found substantial performance improvements for fine-grained communication by adding cacheability and coherence to the device state of a network interface that connects the nodes of a tightly coupled multiprocessor system to its coherent interconnect [14]. Each of these works coupled the simulator for a network controller to a larger simulator designed specifically for the multiprocessor system under study. In contrast, Spinach is a set of standalone modules based on the Liberty Simulation Environment, enabling this simulator to be easily retargeted and integrated with arbitrary Liberty-based system models.

8. CONCLUSIONS

Spinach provides the architecture community with a simple set of tools to evaluate an increasingly important class of systems. Built with only 19 base modules, Spinach allows for the easy reconfiguration of the systems under test and the integration of various architectural advances. While monolithic simulators are typically difficult to adapt and extend, the structural and composable nature of Spinach makes it highly flexible, requiring only high-level templated code to migrate from a uniprocessor Gigabit network interface to a multiprocessor Gigabit network interface to a multiprocessor 10 Gigabit network interface with a highly heterogeneous memory system. This paper then uses Spinach to confirm or obtain key insights on the performance of programmable NICs, allowing for study of systems that might otherwise be intractable because of their interaction with operating-system and network resources.

Future plans include the integration of Spinach with the Orion network models built using the LSE [19], power models for embedded systems such as Avalanche or the ARMulator extensions of Šimunić et al.[11, 17], and full-system simulators that support operating-system-intensive applications and enable the use of actual I/O rather than a trace-based harness.

Acknowledgments

The authors thank Hyong-youb Kim for assistance with the various Tigon firmware versions used in this paper, Scott Rixner for comments on the paper, and Neil Vachharajani for support with the Liberty backend.

9. REFERENCES

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

[2] Alteon WebSystems. *Gigabit Ethernet/PCI Network Interface Card: Host/NIC Software Interface Definition*, July 1999. Revision 12.4.13.

[3] P. Crowley, M. Fiuczynski, J.-L. Baer, and B. Bershad. Characterizing Processor Architectures for Programmable Network Interfaces. In *Proceedings of the 14th International Conference on Supercomputing*, pages 54–65, May 2000.

[4] M. Heinrich, J. Kuskin, D. Ofelt, J. Heinlein, J. Baxter, J. P. Singh, R. Simoni, K. Gharachorloo, D. Nakahira, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Performance Impact of Flexibility in the Stanford FLASH Multiprocessor. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 274–285, October 1994.

[5] Y. Hoskote, B. A. Bloechel, G. E. Dermer, V. Erraguntla, D. Finan, J. Howard, D. Klowden, S. G. Narendra, G. Ruhl, J. W. Tschanz, S. Vangal, V. Veeramachaneni, H. Wilson, J. Xu, and N. Borkar. A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS. *IEEE Journal of Solid-State Circuits*, 38(11):1866–1875, November 2003.

[6] H. Kim, V. S. Pai, and S. Rixner. Improving Web Server Throughput with Network Interface Data Caching. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 239–250, October 2002.

[7] H. Kim, V. S. Pai, and S. Rixner. Exploiting Task-Level Concurrency in a Programmable Network Interface. In *Proceedings of the 2003 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 2003.

[8] K. Kleinpaste, P. Steenkiste, and B. Zill. Software Support for Outboard Buffering and Checksumming. In *Proceedings of the ACM SIGCOMM '95 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 87–98, August 1995.

[9] E. J. Koldinger, S. J. Eggers, and H. M. Levy. On the Validity of Trace-Drive Simulation for Multiprocessors. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 244–253, May 1991.

[10] D. Kranz, K. Johnson, A. Agarwal, J. Kubiatowicz, and B.-H. Lim. Integrating message-passing and shared-memory: Early experience. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 54–63, May 1993.

[11] Y. Li and J. Henkel. A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems. In *Proceedings of the 35th Conference on Design Automation (DAC 1998)*, pages 188–193, June 1998.

[12] Y.-D. Lin, Y.-N. Lin, S.-C. Yang, and Y.-S. Lin. DiffServ over Network Processors: Implementation and Evaluation. In *Hot Interconnects X*, August 2002.

[13] K. Z. Meth and J. Satran. Design of the iSCSI Protocol. In *Proceedings of the 20th IEEE Conference on Mass Storage Systems and Technologies*, April 2003.

[14] S. S. Mukherjee, B. Falsafi, M. D. Hill, and D. A. Wood. Coherent Network Interfaces for Fine-Grain Communication. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 247–258, May 1996.

[15] P. Shivam, P. Wyckoff, and D. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing (SC2001)*, November 2001.

[16] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a Robust Software-Based Router Using Network Processors. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 216–229, October 2001.

[17] T. Šimunić, L. Benini, and G. De Micheli. Cycle-Accurate Simulation of Energy Consumption in Embedded Systems. In *Proceedings of the 36th Conference on Design Automation (DAC 1999)*, pages 867–872, June 1999.

[18] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural exploration with Liberty. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pages 271–282, November 2002.

[19] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pages 294–305, November 2002.

[20] Z. A. Ye, A. Moshovos, S. Hauck, and P. ithviraj Banerjee. Chimaera: A high–performance architecture with a tightly–coupled reconfigurable functional unit. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 225–235, June 2000.