

# A Case Study of a System Level Approach to Exploration of Queuing Management Schemes for Input Queue Packet Switches

Chen He  
Motorola, Embedded Memory Center  
chen.he@motorola.com  
Also with University of Texas at Austin

Marcello Lajolo  
NEC USA, C&C Research Labs  
Princeton, NJ  
lajolo@nec-lab.com

Margarida Jacome  
University of Texas at Austin  
jacome@ece.utexas.edu

## Abstract

*In this paper, a new system-level exploration method called architectural probing is proposed. It embodies the widely accepted concepts of behavior/architecture codesign, and allows to selectively gather functional information from a functional diagram and feed it into the architectural diagram allowing to reduce the amount of details that are simulated at the architectural level, while still maintaining a high degree of accuracy. The effectiveness of this novel method is demonstrated by a case study of exploring data queuing management schemes for packet switches with input queues. As shown in the experimental results, the method enables us to evaluate the impact of different data management algorithms and different system parameters such as DMA size and packet payload size, in terms of power and performance tradeoff by estimating the average energy per bit transferred, on each architectural component, such as memories, buses and bus arbiters, involved in the implementation of the queuing management system.*

## 1 Introduction

Simulation of packet switch systems is one of the challenging problems in developing new effective schedulers for them. The inherent high concurrency of these systems, together with a large number of parameters make the simulation and performance analysis difficult.

So far, most of the research work on various architectures and protocols have concentrated on high level design, with few exceptions [1, 2]. However, evaluation of packet processing time and scalability to a large number of interfaces can only be accurately done by a hardware level design. Therefore, an integrated high level design and synthesis is extremely valuable in the design process of such systems.

Although the concept of behavior/architecture codesign is

relatively well established, its deployment into industrial design flows is happening at a slow speed. We think that one of the reasons could be the lack of comprehensive examples of *behavior/architecture codesign in practice* that could allow designers to approach smoothly the new methodology. This paper tries to reduce this gap. In particular, we propose a system-level simulation methodology called *architectural probing*, which allows to selectively collect functional information from a behavioral description of the system, and feed it into an architectural description in order to perform the architectural simulation for the selected sub-system. In this way, the amount of details that are simulated at the architectural level is reduced, while a high degree of accuracy is still maintained by appropriately selecting a useful set of functional information. The effectiveness of this simulation methodology is demonstrated by the simulation of an input queue packet switch system with different queuing management schemes.

This paper is organized as follows. Section 2 describes our proposed architectural probing technique. Section 3 provides some further motivation for system level exploration of queuing management schemes. Section 4 provides in-depth details of the functional and architectural specification of the input queue packet switches. Section 5 defines the power and performance tradeoff metric which we use to evaluate different queuing management schemes and different system parameters. Experimental results demonstrating the effectiveness of our methodology are presented in Section 6. Finally, Section 7 concludes the paper.

## 2 Architectural Probing

Figure 1 shows an example of application of the *architectural probing* method to a 16x16 input queue switch system. The functional specifications are shown on the top of the figure, while the architectural specifications are shown on the bottom. The architecture is the one used for implementing the virtual

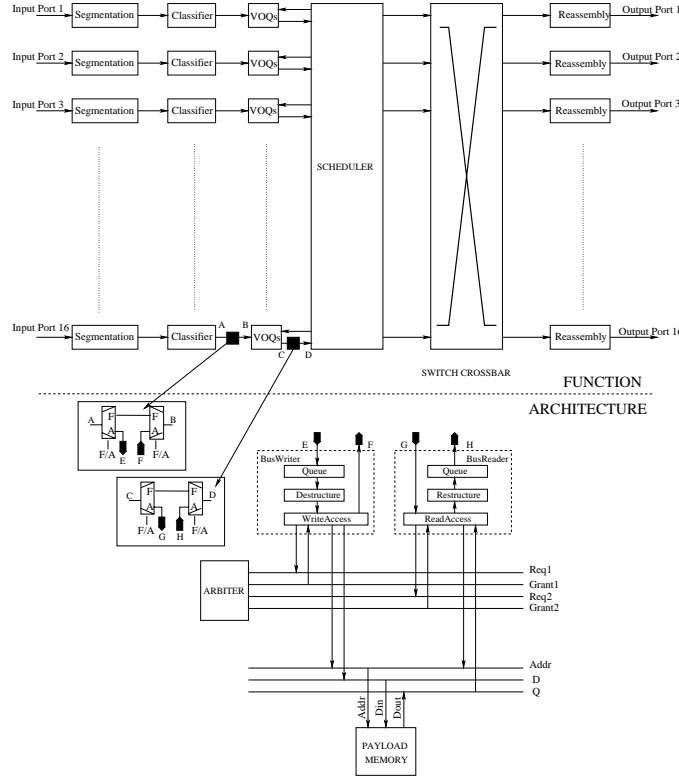


Figure 1: Architectural probing.

output queuing system on one input line card, which includes shared memory, memory bus, bus arbiter, and bus interface modules.

The system is described graphically by introducing an explicit architectural diagram together with the simulatable functional specifications as it is done in commercial tools like [3]. The two diagrams are kept as much separated as possible in order to decouple functionality from architecture and computation from communication as it is dictated by a true behavior/architecture codesign approach.

As it is possible to see, there are no explicit connections between the functional and the architectural diagrams. The mapping of the functional communication onto architectural links is performed graphically by inserting a black box (*probe*) on the nets in the functional diagram that need to be refined<sup>1</sup>. This is what we call the *architectural probing* phase. In particular, in our 16x16 switch example, we have inserted two probes onto two nets on the 16<sup>th</sup> input line card, since we wanted to analyze the communication with the architectural memory implementing the virtual output queues (VOQs) on one single input line card and hence avoiding to simulate the entire switch at the architectural level.

The probe is a hierarchical module that has the structure shown in the small boxes represented in Fig. 1. It is composed

of a sequence of a multiplexer and demultiplexer, both controlled by a boolean signal ( $F/A$ ) representing the simulation style ( $F$ : *Functional* simulation,  $A$ : *Architectural* simulation).

The functionality of the probe is the following. In the functional mode, the probe is completely transparent: the event coming from the left passes directly through the multiplexer and demultiplexer and exits from the right where it reaches the next module in the functional diagram. In the architectural mode, instead, the event coming from the left goes into an output porthole and is then captured by an input porthole in the architectural diagram. The association (mapping) between input and output porthole is done by name. The event in the architectural diagram can then activate all the architectural resources involved in its manipulation (bus, arbiter, memory, ...) allowing to estimate performance and power consumption more accurately at the architectural level. Eventually, the event is sent back to the probe through another name-based association output/input porthole and can then be propagated to the next functional module.

As in [3], we support two simulation styles: *functional* and *architectural*. In the functional mode, the probes capturing events in the functional diagram are simple wires and no event is passed in the architectural diagram. In the architectural mode, a probe in the functional diagram is connected to a corresponding probe in the architectural diagram. The association is done by name: each probe has a parameter called *name* that can be set by the user.

<sup>1</sup>With *refinement* we mean capturing architectural effects from the architectural diagram into the functional diagram.

It is evident that the architectural simulation style results in a lower simulation speed since the number of events transferred in the netlist is higher than in the functional complexity due to the additional details simulated in the architectural view of the system.

### 3 Memory Management Exploration and Optimization

Let us consider the example system shown in Figure 2 that constitutes the basis of the application that will be used throughout the rest of this paper.

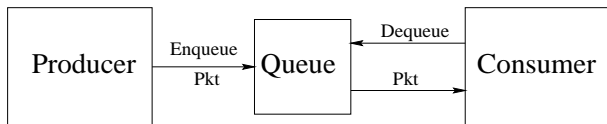


Figure 2: Example system.

The system is composed of two modules: *Producer* and *Consumer* that respectively produce and receive basic pieces of information called *packets*. The communication is performed through a shared *Queue* in which *Producer* stores new packets and *Consumer* retrieves new packets to be processed.

Depending on the type of functionality performed in *Producer* and *Consumer*, the implementation of the shared *Queue* can vary. When the *Producer* and *Consumer* can be implemented in software by micro-controllers, the shared *Queue* can be mapped into a shared bus based architectural implementation which may include shared memory, memory bus, bus arbiter, and so on. From a system implementation point of view, a set of commercial memory components will need to be selected and purchased for the final SOC implementation. Rarely, if ever, does a single memory architecture ideally address all memory needs within a given implementation. In networking equipment, for example, the look-up table, linked-list, and packet buffer memories have different read-versus-write access profiles, random-versus-sequential address patterns, and burst access lengths.

In this work, we show some of the tradeoffs involved when evaluating at the architectural level two different memory management schemes (array and linked-list) in a queuing system like the one shown in Fig. 2, with consideration of different system parameters such as DMA size and packet payload size.

### 4 System Specifications

In this work, we have used the 16x16 input queue switch system of Fig. 1 in order to analyze the system-level tradeoffs involved in two different queuing management schemes of the VOQs for one input line card: the array with circular pointers (fig. 3) and the linked list (fig. 4).

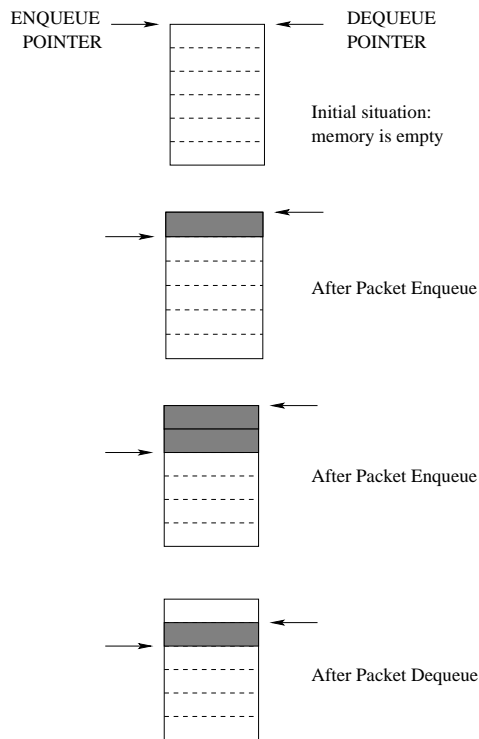


Figure 3: Array with circular pointers.

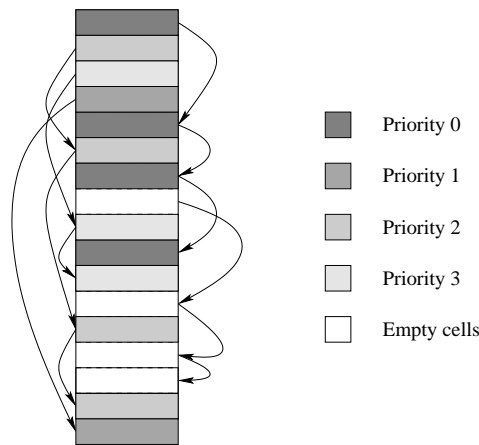


Figure 4: Multiple Doubly Linked-list organization.

In this system, each input port receives packets that can be destined to any output port. In particular, we have considered a switch supporting 4 priorities of packets, so that each input line card has a total of 16x4 VOQs.

The incoming packet is first classified and stored in a queue according to its destination output port and priority. A scheduler then attempts to deliver up to one packet per output port in a time slot. For each output port, the scheduler selects one of the 16x16x4 virtual output queues from which to dequeue and transfer the packet.

In the case of the array with circular pointers the entire

address space is equally distributed among the 4 priorities (each priority gets two memory banks).

In the case of the linked-list organization, the entire data storage space is available for each priority and each VOQ can grow up to occupy the entire memory space (in the context of this work we did not consider backpressure). Memory allocation for the packets of each priority is implemented using a dedicated doubly linked-list. An additional doubly linked-list is used in order to manage the empty (available) memory cells. So in total there are 5 doubly linked-list to be managed. When a packet arrives at an input port, the required number of cells is removed from the empty list and used to store the packet. The cells are then linked at the bottom of the list of the priority associated with the packet.

## 4.1 Functional specifications

On the top of Fig. 1 it is shown the functional description of the system that includes 16 line cards composed of:

- **TrafficGen:** this is a behavioral traffic generator used as a test bench for the system. It can generate packets of fixed and variable size with different statistical distributions like Bernoulli and Exponential. Pre-generated traffic can also be read from an input file.
- **Segmentation:** this module takes variable size packets and splits them into several different cells of fixed size. The size of the fixed size cells is parametrizable. Packets larger than a maximum size can also be dropped.
- **Classifier:** this module receives a fixed size packet and stores it in the virtual output queue corresponding to its destination port.
- **VOQs:** this is the functional representation of all the Virtual Output Queues available on one input port.
- **Scheduler:** this is the scheduler module that selects among all the VOQs of all the input ports, the packets to be transferred.
- **Switch crossbar:** this is the actual cross-connection able to transfer a scheduled packet from one input port to the destination output port.
- **Reassembly:** this is the module performing the reassembly of the fixed size cells into variable size packets.
- **OutputAnalyzer:** this is a module used for gathering statistics during a simulation sessions like number of packets transferred and average waiting time of a packet in the switch.

## 4.2 Architectural specifications

The architectural diagram on the bottom of Fig. 1 shows the architectural template that we have used for the evaluation of

the VOQs of one input line card. The architecture consists of the following modules:

- **BusWriter:** bus interface unit for the packet producer, which will buffer the input packets, destructure the packets, request write access to the bus, and generate the destination address of the input packets after receiving the grant signal from the bus arbiter.
- **BusReader:** bus interface unit for the packet consumer, which will request read access to the bus, generate the source address of the packets to read after receiving the grant signal from the bus arbiter, and restructure and buffer the packets read from the bus.
- **Arbiter:** the module that arbitrates the read and write accesses to the bus. It supports a variety of arbitration protocols such as round-robin, static priority and dynamic priority, etc. In this study, the bus arbiter uses the round-robin arbitration scheme. The arbiter also supports the DMA (direct memory access) transactions, i.e. it can grant multiple-word access to the BusWriter or BusReader.
- **Bus:** architectural abstraction of data, address and control busses. Access to the shared memory can be only performed through the bus module. The bus module supports DMA transactions. To compute the energy dissipation on the bus module, the switching activities on the following signals are taken into account: address and data from the BusWriter, address from the BusReader and data to the BusReader, address to the memory, and data to/from the memory.
- **Payload Memory:** multiple-banked memory modules connected to the bus with an address decoder, which is used to store the packet payload. For the array-based memory scheme, packets of each priority can occupy one or two memory banks. For the linked-list scheme, the whole memory space can be used for packets with any priority.

The bus access for writing is handled by the module *BusWriter* which sends requests (*Req1*) to the *arbiter* controlling the bus and waits for the grant (*Grant1*) to access the bus.

The reading from memory is performed in a similar fashion. The module *BusReader* sends the request (*Req2*) to the *arbiter* and waits for the grant (*Grant2*) to access the bus. The bytes read from memory are then reassembled and enqueued in a high-level packet data structure through the module (*Restructure*).

The BusWriter, BusReader, and the Arbiter modules are implemented in ASICs. The queues for both the BusWriter and BusReader are one-position queues (i.e. they can only hold one packet). That means, new packets generated by the producer will be dropped if the queue for the BusWriter is

not empty, i.e. an old packet is still being processed in the architectural template. Also, the new incoming packets will be dropped by the BusWriter when the memory partition for the priority of the incoming packet is full. Similarly, new requests generated by the consumer will be dropped if the BusReader's queue is not empty, or if there are no packets in memory with the requested priority.

In the context of this work, we have analyzed the behavior of the switch under the two memory management schemes with consideration of a set of parameters that may affect the system performance and power consumption, such as the packet length and the DMA size.

The same graphical description of the system can be used for exploring both memory access schemes. Only the bus interface modules in the architectural diagram (*BusWriter* and *BusReader*) behave differently under the two schemes, by generating different address patterns and the user can change the memory implementation by simply changing a parameter in the system.

## 5 Energy Per Bit Transferred

In our study, the metric used to compute the power efficiency of different queuing management schemes and different system's parameters is the average energy per bit transferred ( $\bar{E}_{ub}$ ), which is defined as:

$$\bar{E}_{ub} = \frac{E_t}{n_t} \quad (1)$$

where  $E_t$  is the total energy consumed in the architectural components (bus, payload memory, arbiter, BusWriter and BusReader):

$$E_t = E_{bus} + E_{mem} + E_{arb} + E_{bw} + E_{br} \quad (2)$$

and  $n_t$  is the total number of useful bits transferred. When the length of the packets is fixed, we have:

$$n_t = N_b N_{pl} N_{pkt} \quad (3)$$

where  $N_b$  is the number of bits in a word,  $N_{pl}$  is number of words in the payload of one packet, and  $N_{pkt}$  is the total number of packets transferred.

The energy per bit transferred indicates the tradeoff between the energy dissipation and the system throughput. It is a particular version of the widely known energy-delay tradeoff [4], and we think it is appropriate for evaluating the energy efficiency of packet switch systems.

### 5.1 Energy analysis for bus and memory

In equation 2, the energy consumed in the bus is computed with the formula:

$$E_{bus} = \frac{1}{2} C_{bus} V_{dd}^2 N_{bus}, \quad (4)$$

where  $C_{bus}$  is the load capacitance of the bus lines,  $V_{dd}$  is the supply voltage and  $N_{bus}$  is the switching activity (i.e. total number of bit transitions) on the bus lines.

Similarly, based on [5], the expected memory power consumption can be calculated as a function of frequency, utilization, and output loading:

$$E_{mem} = E_{DC} + E_{AC} = \frac{V_{dd} I_{dd}}{f} + \frac{1}{2} C_L V_{dd} Q^2 N_{mem}, \quad (5)$$

where  $V_{dd}$  is the operating supply voltage,  $I_{dd}$  is the operating supply current,  $C_L$  is the output loading capacitance,  $V_{dd} Q$  is the supply voltage, and  $N_{mem}$  is the total number of output transitions.

For a 2-word burst QDR (the one we have used),  $I_{dd}$  can be expressed as:

$$I_{dd} = f(0.96 + 1.10R + 0.74W)[mA], \quad (6)$$

where  $f$  is the clock frequency,  $R$  is the read duty cycle (1=100%),  $W$  is the write duty cycle (1=100%)

The worst case for energy consumption is when the device is reading and writing continuously at 166.67 MHz. In this case we have:

$$I_{dd} = 166.67(0.96 + 1.10 + 0.74) = 467[mA], \quad (7)$$

and

$$E_{DC} = \frac{V_{dd} I_{dd}}{f} = 6[nJ], \quad (8)$$

### 5.2 Energy analysis for hardware components

The modules BusWriter, BusReader, and arbiter, are implemented in hardware using a cell-based 0.25  $\mu\text{m}$  technology library. In the context of this work, for the power analysis of hardware modules, we have adopted the power macromodeling methodology that has been proposed in [6].

Power macromodeling refers to the characterization of the power consumed by a logic macroblock (i.e. adder, multiplexer, ...) by looking only at the switching activity at its primary inputs and outputs.

Let us consider a logic macroblock with  $n$  primary inputs ( $I_1, \dots, I_n$ ) and  $m$  primary outputs ( $O_1, \dots, O_m$ ),

Given a sequence of input vectors at different points in time ( $t_1, t_2, \dots$ ):

$$\begin{aligned} t_1 &: I_1^{t_1} \dots I_n^{t_1} \\ t_2 &: I_1^{t_2} \dots I_n^{t_2} \\ &\dots \end{aligned} \quad (9)$$

If we consider only the switching activity at its primary inputs, the power at time  $t_i$  can be computed as:

$$P(t_i) = K_0 + K_1(I_1^{t_i} \oplus I_1^{t_i-1}) + \dots + K_n(I_n^{t_i} \oplus I_n^{t_i-1}) \quad (10)$$

where  $K_0, \dots, K_n$  are the characterized parameters which are determined by the implementation technology of the module.

Given the power of the ASICs, the energy for those ASICs can be computed as

$$\begin{aligned} E_{arb} &= \int_0^t P_{arb}(t) dt \\ E_{bw} &= \int_0^t P_{bw}(t) dt \\ E_{br} &= \int_0^t P_{br}(t) dt \end{aligned} \quad (11)$$

where  $E_{arb}$  is the energy consumed by the arbiter,  $E_{bw}$  the energy consumed in the BusWriter and  $E_{br}$  the energy consumed by the BusReader.  $P_{arb}$ ,  $P_{bw}$  and  $P_{br}$  are the power for the arbiter, the BusWriter and the BusReader, respectively.

## 6 Experimental Results

The simulation methodology that we have used in this work is the Discrete Event (DE) domain in the PTOLEMY [7] simulation platform. Each functional and architectural module in the queuing packet switch system is implemented as a parameterized module in the PTOLEMY DE domain.

The energy dissipation for each architectural module is computed during the simulation run. The system throughput, i.e. the number of packets transferred, is also recorded, and the average energy per bit transferred is calculated for all cases.

The system is simulated under both uniform and non-uniform traffic. With uniform traffic, each input receives packets with equal probability for each output port and each priority. With non-uniform traffic, each input receives packets with equal probability for each output port, but of one single priority.

The input queue packet switch system has 16 ports, and supports 4 different packet priorities. For the array-based memory scheme, packets of each priority can occupy two memory banks. For the linked-list scheme, the whole available memory of 8 banks is used for packets of any priority.

To compute the energy consumed, the voltage supply is assumed to be 2.5 V. The bus line load capacitance used is 5 pF, a typical value for 1 cm long wires with 0.25  $\mu\text{m}$  technology. According to [5], the typical value of the output loading capacitance of the memory cells is 5 pF. The parameters used in power macromodeling for the ASICs to implement the BusWriter, BusReader and arbiter are typical values from standard logic component libraries.

Figure 5 and Figure 6 show the energy consumption for each architectural component, with given DMA size and packet size, in presence of uniform traffic and non-uniform traffic,

respectively. The energy results presented for both the array-based and the linked-list schemes assume that the same packet throughput is obtained for both schemes. From these figures, we see that under uniform traffic, the array-based memory management scheme outperforms the linked-list scheme in terms of energy efficiency, while under non-uniform traffic, the linked-list scheme is more energy efficient.

Figure 7 shows the energy per bit transferred for both array-based and linked-list organizations, with different DMA sizes and fixed payload size of 16 words, under both uniform (left) and non-uniform (right) traffic. From this figure, we can see that, under both traffic scenarios, when the DMA size increases, the average energy per bit transferred decreases, and hence the total energy dissipation for the same packet throughput also decreases. This is because, when the DMA size increases, the packet can be transmitted in fewer bus transactions, resulting in fewer bus arbitrations and more sequential bus accesses. This implies that, with a given packet size, larger DMA size can result in more efficient power utilization for the same packet throughput, or a higher packet throughput with the same energy consumption. This effect could not be easily predicted and evaluated without performing system-level simulations as we propose in this paper.

Figure 7 shows also that in the case of uniform traffic, the array-based scheme outperforms the linked-list scheme in terms of energy efficiency, since it is more effective for sequential accesses. That is, less bit transitions occurs with array-based memory scheme for all the address and data signals. Another reason is that for the bus interface units, i.e. BusWriter and BusReader, extra energy is dissipated for the more complex logic used to handle the pointers under the linked-list schemes.

However, in presence of non-uniform packet traffic, the linked-list scheme outperforms the array-based scheme in terms of energy efficiency, since it better utilizes the whole available memory space. Indeed, under non-uniform traffic, with the same execution time, the linked-list scheme achieves much higher packet throughput, though the total energy consumed is higher than in the array-based scheme. This is because, in the array-base scheme, under non-uniform traffic, the memory bank dedicated to one single priority becomes full more frequently than in the linked-list scheme and results in a much higher rate of packets dropped. Indeed, in the array-based scheme, when the memory banks dedicated to a priority are full, new packets of that priority will be dropped until free space appears again in those banks. Instead, in the linked-list scheme, new packets will be dropped only when the whole memory space is full until free space shows up again. As a result, the linked-list outperforms the array-based scheme in terms of the energy efficiency, i.e. average energy per bit transferred.

Figure 8 shows the effect of different packet sizes on the energy efficiency for both memory management schemes under both uniform (left) and non-uniform (right) traffic. Note that in these cases, the DMA size is equal to the packet size, which

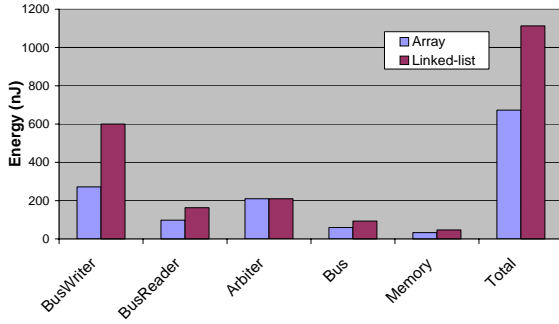


Figure 5: Energy consumption in presence of uniform traffic (DMA size = 2 words and packet size = 16 words).

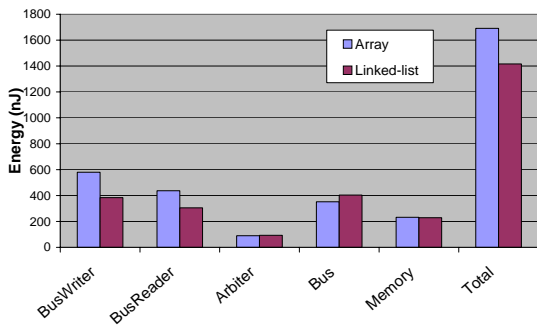


Figure 6: Energy consumption in presence of non-uniform traffic (DMA size and packet size = 16 words).

is optimal in terms of energy efficiency for that packet size as it has been shown earlier. In this context, the packet size implies the number of words in the payload, excluding the packet overhead such as the checksum, destination output port, etc. It is possible to see that the average energy per bit transferred decreases when the packet size increases. This is because, when the packet size increases, the portion of useful data transferred increases, and the overhead due to the negotiations between the bus interface units (BusWriter and BusReader) and the bus arbiter decreases.

Similarly to the results obtained with the previous experiment (Figure 7), even here we see that the array based scheme is preferable in presence of uniform traffic, while the linked list scheme is better in presence of non-uniform traffic. The difference in terms of energy per bit transferred between the two memory management schemes is even more significant with respect to the previous experiment.

Figure 9 and Figure 10 show the 3-dimensional relationship among the energy per bit transferred, packet size and DMA size under uniform traffic. Similar relationship can be obtained under non-uniform traffic. It is possible to see that when the DMA size becomes larger than the packet size, the energy efficiency does not improve any further. This is because it is not possible for the arbiter to take advantage of the larger DMA

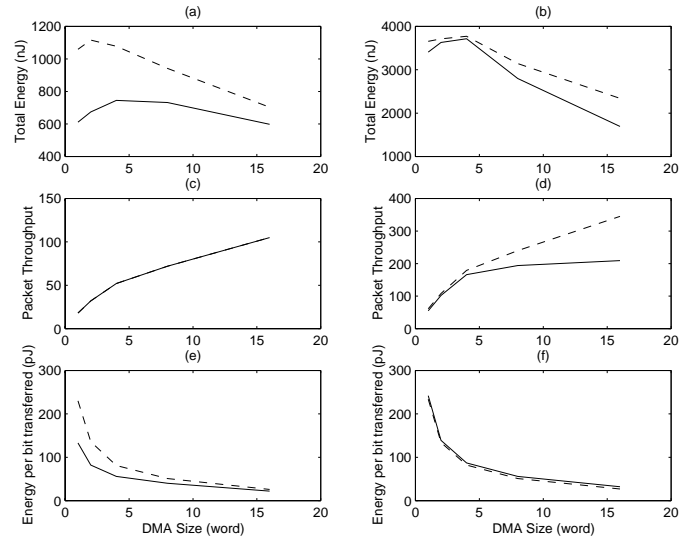


Figure 7: Energy per bit transferred vs. DMA size (packet size = 16 words). Solid line: Array scheme; Dashed line: Linked-list scheme. (a) Total energy under uniform traffic (b) Total energy under non-uniform traffic (c) Packet throughput under uniform traffic (d) Packet throughput under non-uniform traffic (e) Energy per bit transferred under uniform traffic (f) Energy per bit transferred under non-uniform traffic.

size in order to accommodate more bytes of a packet in one DMA transaction.

In summary, from these results, the following conclusions can be reiterated:

- In presence of uniform packet traffic, the array-based memory management scheme outperforms the linked-list scheme in terms of energy efficiency.
- In presence of non-uniform packet traffic, the linked-list scheme outperforms the array-based scheme in terms of energy efficiency.
- Increasing DMA size can result in better energy efficiency. Given the packet size, the optimal energy efficiency is achieved with the largest possible DMA size which is equal to the packet size. When the DMA size becomes larger than the packet size, the energy efficiency does not improve any further.
- The average energy per bit transferred decreases when the packet size increases, given the largest possible DMA size is used for that packet size.

The simulation results demonstrate the effectiveness of the proposed simulation methodology, i.e. *architectural probing*, as some of the above conclusions could not be easily predicted and evaluated without such system-level simulations.

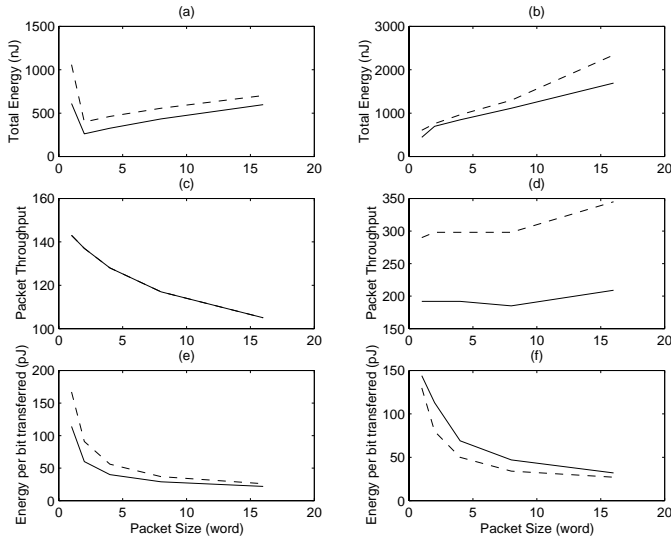


Figure 8: Energy per bit transferred vs. packet size (DMA size = packet size). Solid line: Array scheme; Dashed line: Linked-list scheme. (a) Total energy under uniform traffic (b) Total energy under non-uniform traffic (c) Packet throughput under uniform traffic (d) Packet throughput under non-uniform traffic (e) Energy per bit transferred under uniform traffic (f) Energy per bit transferred under non-uniform traffic.

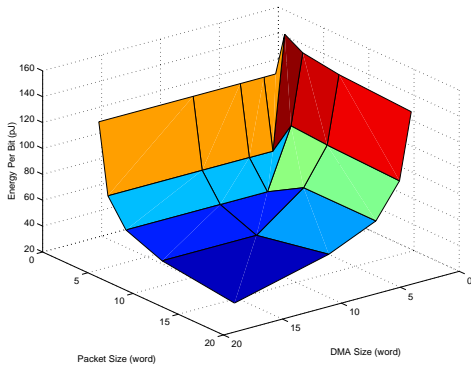


Figure 9: Array-based scheme: energy per bit transferred vs. packet size vs. DMA size.

## 7 Conclusions

Data queuing is one of the most difficult parts in embedded switch design. Its implementation must be highly optimized to combine high speed, low power, large data storage, and high memory bandwidth. It must also be optimized to support worst case situations as efficiently as possible. In this paper, such data queuing was used as a case study to demonstrate the effectiveness of a new system-level exploration method, *architectural probing*, for evaluating architectural effects at the system level. In the case study presented, two different memory management schemes, banked and doubly linked-list, have been compared in terms of the energy per bit transferred, a particular flavour

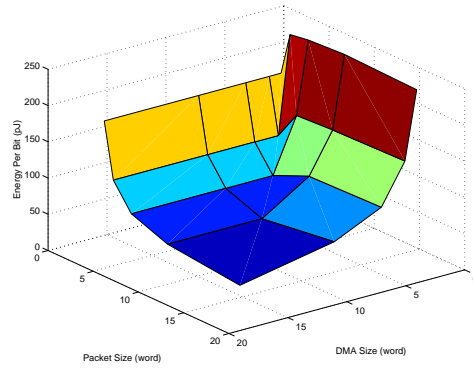


Figure 10: Linked-list scheme: energy per bit transferred vs. packet size vs. DMA size.

of the classic energy-delay tradeoff. Meanwhile, the effect of DMA size and packet size on the energy efficiency metric has also been explored with the help of the proposed system-level simulation methodology.

## References

- [1] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, “Tiny Tera: A Packet Switch Core,” *IEEE Micro*, vol. 17, pp. 26–33, Jan. 1997.
- [2] P. Gupta and N. McKeown, “Designing and Implementing a Fast Crossbar Scheduler,” *IEEE Micro*, vol. 19, pp. 20–28, Jan. 1999.
- [3] Cadence VCC  
<http://www.cadence.com/products/vcc.html>.
- [4] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power CMOS digital design,” *IEEE J. Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.
- [5] Micron QDR-SRAM Data Sheet  
<http://www.micron.com/products>.
- [6] N. R. Potlapally, A. Raghunathan, G. Lakshminarayana, M. S. Hsiao, *et al.*, “Accurate power macro-modeling techniques for complex RTL circuits,” in *Proc. Int. Conf. VLSI Design*, pp. 235–241, Jan. 2001.
- [7] J. Buck, S. Ha, E. A. Lee, and D. D. Masserchmitt, “Ptolemy: A framework for simulating and prototyping heterogeneous systems,” *International Journal on Computer Simulation Special Issue on Simulation Software Management*, Jan. 1994.