System-Level Modeling of a Network Switch SoC

JoAnn M. Paul, Christopher P. Andrews, Andrew S. Cassidy, Donald E. Thomas
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{jpaul, cpa, acassidy, thomas} @ ece.cmu.edu

Abstract

We present the modeling of the high-level design of a next generation network switch from the perspective of a Computer-Aided Design (CAD) team within the larger context of a design team consisting of an experienced network switch designer and an experienced VLSI hardware designer. After facilitating the design process, the CAD team observed how designers approach high-level designs, beyond RTL. We motivate the need for CAD support that allows designers to effectively manipulate what we refer to as Memory Visualization Level (MVL) design.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems — *Modeling techniques, Design studies, Performance attributes.* I.6 [**Computing Methodologies**]: Simulation and Modeling.

General Terms

Performance, Design, Experimentation.

Keywords

Computer-Aided Design, Network Switch, Performance Modeling, System Modeling, Memory Visualization Level Design.

1. Introduction

A primary goal of system level modeling has been to formulate a near-universal modeling representation within which a broad class of designs can be developed. Ultimately, this allows designers to efficiently implement instances of systems within a single modeling style as supported by common design tools and methodologies. Of course, hardware design has been quite successful at this. Register-transfer level (RTL) design is based upon the premise that all designs that can be specified in the RTL formulation can be synthesized and simulated in the same way. Clearly the RTL representation imposes some restrictions on the broader class of systems that might be represented by gate-level hardware design. But the restrictions are minor compared to the benefits of the tools and methodologies that support RTL-based designs. A primary reason for RTL design becoming popular was that an elegant formalism fit the modeling already being done by designers.

Interestingly, convergence on a near-universal representation of digital systems beyond RTL is still elusive. Function/architecture or computation/communication orthogonalization [4], and the description of all systems by uniting specific models of computation [5] are perhaps the most conspicuous research efforts toward system representation. These carry a premise that certain approaches to design are inherently more formal than others, and so are better for designers to follow than others. For instance, the notion that all computation systems can be distilled to reactive models, or that shared memory is a poor modeling practice, are strong sentiments within some parts of the CAD community related to these and other efforts.

The authors of the paper were a part of a system design team. The goal was to explore the design of a Tera-bit/sec switch-on-a-chip with Quality of Service (QoS), taking advantage of technology predicted by the SIA roadmap [13] to be available in the year 2008. The team included an experienced network switch architect and an experienced VLSI architect. The goal of the authors was to develop CAD tool support for the team and thus to better understand the important modeling issues of next generation designs from the perspective of system designers.

In this paper, we discuss the network switch being designed and present our observations on the design process and the modeling needed to support it. We observe that these designers conceive systems at higher levels of abstraction than those currently promoted in research or commercial tools, and break many of their premises. We agree with [6] that the memory architecture of embedded designs can be chosen more or less freely. Indeed, in concurrent computation, memory organization has the greatest impact on overall performance [9]. The shift from computer systems as discrete components to SoCs with on-chip memories requires memory-based organizing principles for design tools. We illustrate the design space these next generation tools must capture.

The most significant challenges the team's designers faced in performance-oriented design were ones that pertained to effective visualization and manipulation of memory access, which we refer to as Memory Visualization Level (MVL) design. At this level, the issue is concurrently evolving, from their earliest conception, the functionality, the architecture and the access to shared information based on technological limitations. None of these is known first; they evolve together.

We start with a brief background in network switching and proceed to describe the network switch architecture we developed and its simulation model. Results from the simulation are presented to illustrate the level of abstraction used and the types of questions that needed to be answered. We conclude with our observations on the design process and the modeling needed to support it.

2. Networking Background

Following is a very brief background of networking principles so that a reader unfamiliar with the networking industry will be able to understand our design scenario and experiments. For a more complete background, see [10]. The basic function of a network packet switch is to route packets received on a set of input ports to a set of output ports. The correct output port is determined by the address contained in the header of the packet.

The design complexity of the switch increases when trying to maximize the number of packets per second routed by the switch or when the traffic is not well behaved. Research into the general characteristic of packet traffic on local area networks and wide area networks has revealed a fundamentally bursty nature [1]. Typical network applications such as email, file transfer, and web page downloading result in data transfers of a burst or stream of back-to-

ISSS'02, October 2-4, 2002, Kyoto, Japan.

back packets, followed by long periods of inactivity [2]. The aggregation of many of such sessions intensifies the bursty traffic model [1]. Given the extreme behavior of such traffic models, the true test of a network switch is the ability to handle these cases.

Bursts of packets are typically handled by buffering the excess packets until the burst dies away. The result of this buffering is a smoothing function on the traffic, but it can also lead to an increase in packet latency as the packets must wait in buffers until they can be sent. If the traffic burst is too large, the buffers will overflow, and some packets will be discarded or dropped.

Network engineers realized it would be commercially beneficial to make distinctions between packets such that a customer paying a premium would be offered a service with lower latency and lower drop rate. In order to implement this, the notion of *priority class* was created. Packets with a higher priority class are guaranteed lower latency and a lower drop probability by the network switch — the QoS. Thus, a major metric for evaluating the performance and quality of a network switch architecture has become the measure of QoS offered and delivered by the architecture.

3. Proposed Network Switch Architecture

From the goals of the project, the architects determined a number of technological constraints and proposed a baseline architecture to model. The baseline architecture was still quite broad. Thus, it was not possible to consider any single aspect of the model synthesizable, unlike [6]. Given the complexity of the design, a wide range of architectural modeling was necessary to support design exploration.

A primary goal was to maximize the bandwidth of the switch within the limits of a single chip design. A 128x128 port switch, i.e. a switch with 128 input ports (or inports) and 128 output ports (or outports) was proposed by the architects. A full crossbar that provided a single connection between every inport and outport was deemed impractical as well as a scheme based on a central shared memory. The team investigated an on-chip routing fabric based upon simple, autonomous routing nodes.

The nodal-based design forms a hierarchical network as a connection fabric between ports as in Figure 1, which shows a 2-D mesh at the top and expanded node view at the bottom. Each node has a primary external input and a primary external output; thus it is both a potential off-chip source and destination for packets. Each node also has several internal inputs and outputs; these connect to neighboring on-chip nodes. A packet can travel across the switch, as shown by the gray line, by hopping from node to node until it reaches its destination.

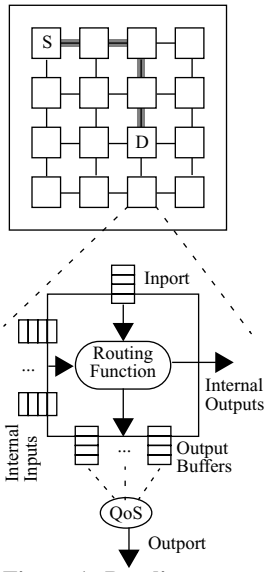

Figure 1 Baseline Architecture

For a 128 node switch, the memory available to buffer data was the greatest resource limitation. To compensate, a variation on virtual cut-through routing [9] was used to route packets across the fabric. This method breaks the packet up into a number of smaller units called *flits,* typically a 32-bit piece. The first flit in the packet opens a path across the switch which is dedicated to the packet until the tail is streamed down the path, freeing the path. If the path is blocked, the flits start to be buffered in place in the node's transit buffer (these buffer flits in transit between on-chip nodes). If the node at the head of the path runs out of room, the back-pressure causes previous nodes in the path to begin to use their transit buffer space as well. This method allows a blocked packet to be buffered across multiple nodes, creating a form of shared memory that distributes memory usage more evenly across the fabric. (This is unlike a store-and-forward scheme that stores a complete packet at each node in the path.) If this back-pressure reaches a primary input, the packet in the inport is lost; it is dropped.

Another goal was to keep the functionality of the nodes simple with local decision making. Nodes only know the conditions of neighboring nodes by the back-pressure of packets. This desire for simplicity also affected the QoS implementation (described in more detail in section 4.1). Every time there is contention for a resource, it would be optimal for a QoS scheduler to differentiate between priority classes to resolve the contention. However, due to the limited resources available, it was decided that it would be sufficient to place QoS schedulers at the site of the most contention — the output ports. Essentially, this requires an assumption that the switching fabric on the chip appears ideal to the QoS — that the fabric's limitations do not interfere with effective QoS.

While the above architecture is fairly detailed, it also leaves a great deal unspecified. For instance, how are the nodes connected together? a 2-D mesh? a 2-D or 3-D torus? How much of the available memory should be used for transit buffers? output buffers? All of these questions are a part of the larger design space remaining after the definition of the nodes.

4. Simulation Model

The simulation model was built using our frequency interleaved simulation environment [7]. SystemC [11] and Superlog [12] were considered. Since they have aspects of an unconstrained programming language, each could have been used to construct our (or any) simulation by using the general programming constructs as opposed to the specific design principles supported by the languages. Since they are extensions of RTL-level modeling, they tend to constrain state access to port-like models consistent with RTL, which would have limited the definition of the modeling style and its parameterization.

Conceptually, the model resembles Figure 2. A set of parameters is applied to the model to explore numerous aspects of the unspecified design space implied by the baseline architecture. A traffic pattern stimulates the design allowing the designer to examine how this particular instance of the architecture performs.

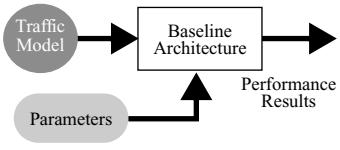

Figure 2 Simulation Model

Structure in the model was implied by modeling each component as a periodically-activated thread. One advantage of this style of specification is that the communication channels act as a buffer between neighboring nodes, preserving the notion of concurrent execution of the nodes without having to utilize a discrete event scheduler to maintain variable update order. This has the further advantage that the model can be distributed on a multi-processor machine to decrease simulation time.

4.1 The Node

A routing function included in each node directs all of the traffic. Along with the inport and outport, the node also contains a number of internal inputs and outputs that connect it to its neighbors. The number of neighbors that the node has is determined by the topology of the routing fabric; neighboring

nodes can have any number of channels connecting them. The QoS scheduler [3] resides on the primary output and decides the order in which packets leave the node. All of the inputs and outputs have buffers that are modeled using FIFO queues that allow the packets to be buffered in order as the flits from the packet each arrive separately. The memory of each node is split between transit buffers, which hold flits transiting across the node, and an output buffer for the output port.

The main loop of the node cycles through each of the primary and internal inputs. If a connection exists, the main loop attempts to move a single flit from the current input to its destination. If there is no connection and a packet is waiting in the buffer, the routing function uses the packet address to either route the packet to the output buffer or forward it to the appropriate internal output. The connections are circuit switched; they are dedicated to the packet until the tail arrives and closes the connection.

Once all of the inputs have been serviced, the primary outport is examined. Here, the QoS scheduler gives priority to the more important packets. As each packet arrives in the output buffer it is given a unique timestamp that is a function of the current timestamp of the scheduler, the class of the packet and the packet's length. Whenever the outport is free, the QoS scheduler merely has to pick the packet in the buffer that has the smallest timestamp.

4.2 Parameterization

The model has several categories of parameterization (see Table 1). One determines the actual physical structure of the switch, including the number of nodes, the topology of the interconnect, and the connection strength (number of connections) between neighbors. The second category deals with the node's memory architecture — the size and how it is distributed. Other parameters allow the designer to see the effects of running the model without a QoS scheduler in place and the difference that changing the size of a flit makes to the model. Even though the core architecture model remains fixed, the parameters allow switches with radically different performance characteristics to be modeled.

Table 1 Parameters of Switch Simulation Model

Architecture	Topology - 2-D mesh, 2-D or 3-D torus, hypercubes, etc...; Conn.Strength - number of connections between neighbors; Number of Nodes - actual number of nodes in the fabric
Memory Sizes	Inport; Outport; Total transit; Individual transit
Other	Flit Size - physical size of flit (usually 32 bits); QoS / FIFO - choice of scheduler on outport

The parameters are not completely independent; they capture interacting design concerns. The most obvious example of this interaction is the two parameters governing the transit buffers. One parameter limits the length that individual transit buffers can reach. The other limits the total overall space allowed for all transit buffers. These two parameters allow the model to capture a number of different memory architectures depending on their ratio. If the two numbers are the same, this models a single piece of memory that is shared between all of the node's internal ports. At the other end of the spectrum, if the individual buffers are only allowed one N^{th} of the total memory available for N buffers, this models an architecture that uses completely separate memories for each port. The space between these two extremes captures a number of intermediate architectures, including individual memories, a memory arbiter or even virtual channels with shared buffers.

4.3 Traffic Models

At high levels of design, the model of a design's testbench — in this case network traffic — must be parameterizable so that designs may be evaluated against specific scenarios. Further, there are few publicly available traces of actual network traffic. Thus, we built a traffic generator that allowed the designer to simulate the two major aspects of network traffic: the arrival characteristics, and the destination distribution. The first is a measure of the burstiness of the traffic. A lack of burstiness results in traffic that arrives according to a Poisson process. One extreme of the second aspect is hotspot traffic — where the majority of the network traffic is directed to a single output, overwhelming it. The other extreme is a completely uniform traffic model with an even load on all outputs. Our generator allows the designer to create traffic that combines these attributes in varying amounts and overlay them with an average load on the entire switch.

5. Simulation Results

The research team used simulation to explore the proposed switch architecture. In this section we show a subset of the experiments, pertaining to a 2-D torus, developed to show the implications and trade-offs of coarse grain design decisions. Other results can be found in [8]. The model used there was derived from the one described here. Further, it only included the fabric; QoS was not included. Thus, QoS-fabric interactions could not be modeled. By contrast, the results in this paper show the impact of the fabric on the effectiveness of QoS.

The limitations and design questions posed are: Can we implement a simplified QoS algorithm and save node resources? Can the experimental fabric approximate an ideal one? (An ideal fabric is an unrealizable one with complete connectivity — as would be the case for a 128x128 crossbar — and unlimited memory to buffer packets.) What is an adequate connection strength to approximate ideal performance? Given a finite amount of memory per node, how should it be distributed to maximize performance?

From these high level design questions, we created four experiments to test these hypotheses. The switch performance is evaluated using two primary metrics, packet latency and drop rate.

5.1 QoS Algorithm Complexity

The first experiment investigated the trade-off of reducing QoS functionality in order to reduce the node resources taken up by the QoS logic. We varied the number of sessions (streams) that the QoS scheduler had to choose from to make scheduling decisions. This is intended to simulate QoS algorithms of varying complexity. A session is a logical connection across the switch formed by multiple packets associated with a particular application. Each session is assigned a traffic class priority — multiple sessions may be assigned the same class. We compared the latency of the overall traffic on a per class basis of these cases to the cases of no QoS scheduler, and a QoS scheduler with access to all of the sessions. These are shown in Figure 3. The traffic was Poisson distributed with a 97% load, 128 sessions, and eight traffic priority classes.

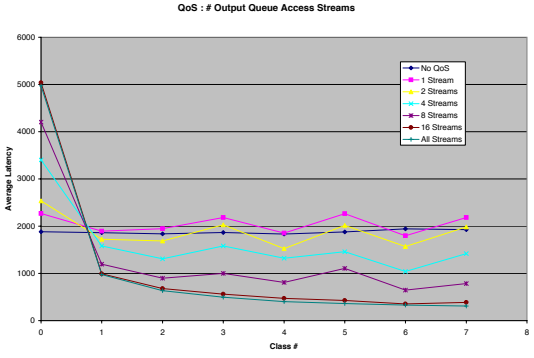

Figure 3 QoS Classes: Latency Distribution

When there is no QoS scheduler, the average latencies of the classes are roughly constant, no packet has priority over another. If the scheduler has access to all of the sessions, the average latencies are distributed in a smooth curve. The highest priority traffic has the lowest latency, and the lower the priority of traffic, the greater the latency. The results show that the greater the number of sessions, the more effective the QoS scheduler, and the greater the latency distinction between classes. Also, the results show that it may be possible to use a reduced complexity QoS scheduler and have performance close to that of the full implementation. For our particular design, under the given traffic conditions, the graph in Figure 3 shows 16 sessions to be close to the best case.

5.2 Experimental Fabric

The next three experiments explore two parameters of the experimental fabric architecture, the connection strength, and the memory allocation. For QoS to work properly on the output ports, the fabric should appear as ideal as possible.

5.2.1 Fabric Connection Strength

The connection strength between nodes is a design trade-off between performance and implementation complexity. The greater the connection strength, the closer the torus would be to the ideal fabric in terms of performance. On the other hand, the smaller the connection strength, the simpler the node logic and the easier the implementation. The traffic latency is compared between experimental switch fabrics of varying connection strength, and the ideal fabric. The connection strength of the experimental fabric is varied from one to sixteen. The experiments were run using Poisson Traffic with a 85% load and two traffic priority classes, where class 1 has higher priority than class 0.

The general trend we found was that for low connection strengths, overall traffic latency increases as the connection strength decreases. For connection strengths above a certain point, the overall traffic latency was approximately the same as the ideal fabric. Further, looking at the traffic on a class by class basis, we found that the differentiation between the latency of the high and low priority classes decreased as the connection strength decreases. The results in Table 2 show this occurring for our design at connection strengths below four.

Table 2 Avg. Latency Ratio vs Connection Strength

Conn. Strength	1	2	4	8	16	Ideal
Avg. Latency: (class 0 / class 1)	2.0	3.0	3.8	3.8	3.8	3.9

This effect occurs because decreasing the connection strength decreases the shared data paths on the fabric. As a result, packets are blocked on the fabric. Since blocking on the fabric occurs without differentiation between class, the ability of the QoS scheduler at the output port of the node is compromised by the latencies already incurred while the packets were on the fabric. This implies that for low connection strengths, the experimental fabric performance no longer resembles that of the ideal fabric.

5.2.2 Fabric Memory Allocation

Similarly, given the finite amount of memory per node, its distribution should optimize the performance of the experimental fabric to as close to the performance of the ideal fabric as possible. Holding the total fixed memory size constant, we varied the percent of memory allocated to the output queue, allocating the remainder of the memory to the internal transit buffers. The switch configurations were tested using a 10x hotspot traffic model of finite duration and two QoS classes of traffic. The results look at both classes of traffic at the output port that is the target of the hotspot congestion.

First we found that larger output queues have a lower percentage of dropped packets. This result is reasonable given that larger buffers are better able to absorb bursts of traffic, and the output queues do not block internal traffic traveling to other nodes. Secondly, we found that QoS scheduling requires a minimum output queue size in order to effectively differentiate between classes of traffic. The results in Table 3 show that for our design, output queue sizes of greater than 35% of the total memory perform the same as the ideal fabric.

Table 3 QoS Performance vs. Output Queue Size

% Output Queue Memory	2	18	36	51	69	(Ideal)
% Class1 packets Tx	50	67	100	100	100	100

For a finite duration hotspot, the QoS scheduler should transmit the traffic for the higher priority class before the lower priority class. For a short duration and high congestion, it is expected that only high priority traffic will be transmitted. This is verified by the ideal fabric where 100% of the traffic transmitted out of the port is from the high priority class.

For the experimental fabric the results show that for small output queue sizes the performance of the QoS scheduler degrades. For our design, output queue sizes below 20% are too small to always have a high priority packet for the scheduler to transmit, so a lower priority packet is sent instead. As the output queue size decreases, the number of low priority packets that are sent increases. This implies the transmitted packet distribution can only be shaped by QoS if the output queue size is sufficiently large.

5.2.3 Fabric Packet Dropping

A fourth experiment demonstrates the limits of approximating the experimental torus fabric as an ideal fabric. Instead of a finite duration hotspot, packet statistics were measured after the hotspot traffic had saturated the switch. We again varied the percentage of memory allocated to the output queue versus the transit buffers and tested using a 10x hotspot traffic model and two QoS classes of traffic. Table 4 shows the number of dropped packets for two classes and overall as the percentage of memory modeled in the output queues varies.

Table 4 Experimental Fabric: Dropped Packets

	% Output Queue Memory	10%	50%	90%
Packet Class	class 0	115	115	128
	class 1 (higher)	107	108	120
	total	222	223	248

As the distribution of memory is varied, the total number of dropped packets does not change appreciably. Furthermore, there is little distinction between the number of class 0 or class 1 packets dropped. The effects of QoS have been lost in this design under these traffic conditions.

5.3 Results Summary

The breakdown of the model in this final experiment caused us to pose new high level design questions: Should the internal nodes of the fabric have a notion of QoS and not just the output queues? Alternatively, should the packet drop function have a notion of QoS? Or, should our design have fewer than 128 ports allowing more memory per node and the possibility for a cross-bar interconnect to improve QoS?

These questions were not investigated, but are indicative of the broad range of high level questions appropriate for system level design and simulation. In summary, from the second and third experiments, we drew the conclusion that it may be possible to

approximate an ideal fabric with a torus fabric with certain architectural parameters. However, the fourth experiment showed a point at which the model broke down and the experimental fabric could not approximate the ideal fabric. Thus when QoS is implemented only at the node outputs of the proposed experimental fabric design, it becomes ineffective due to dropping decisions that are made upstream, in the fabric.

6. Observations

These experiments answer design questions that cannot be answered by pure analytical methods or by designer's intuition. In this section we make some observations about the model. Our goal is to point out characteristics of modeling at this level that support the creative process in the context of design space exploration and yet may be inconsistent with traditional views.

6.1 Level of Design

Performance modeling in high level design is enabled by a level in between the purely functional and the physical. The network switch model exists at such a mid-level; we call it *flit accurate.* At a purely functional level, a model could capture a very large design space. But since the model is completely functional, there is no way to examine the physical implications of the implied design space; these models are too high level for automatic synthesis. At a lower level, the actual physical structure of the design or a particular implementation could be modeled. However, system alternatives are difficult to explore at this level; it is useful mainly for detailed timing and verification.

For the network switch, neither the whole function, the architecture, the model of computation, nor the boundary between computation and communication were obvious. The model has structure and a number of implied design decisions built into it, but much of the model still exists at a more functional level. This combined with the parameterizable aspects of the model's structure and function allow a designer to explore the physical consequences of a fairly large design space without having to commit to other attributes of the design.

The important issue at the system level of design is having a model that is just accurate enough to allow the designer to define and manipulate how the significant system features will interact to have the most impact on overall system balance. Clearly, for performance modeling to take place, some notion of timing must be included in the specification. Implicitly, the level of time granularity of the model dictates the types of concurrent architectural elements that are manipulated by the designer as well as the functionally atomic actions that happen between them. For instance, the assumption made for all of the node threads was that a simulation cycle was just enough time to process and move a single flit. The communication between the testbench and the fabric was done at one fifth of this rate to simulate an off-chip/on-chip communications ratio, enabling consideration of on/off chip trade-offs without requiring excessive physical detail.

6.2 Accommodation

At the system level, designers prioritize the physical implementation of one set of behaviors over others. The less prioritized behavior is not ruled out, but is implemented in a less optimal manner than the common case. However, traditional, automatic synthesis begins with a representation of the required behavior of the system which is a specification of what the system does. In this specification, there is no notion of meeting some portions of the behavior better than others. All of the behavior in the specification is met. By contrast, the network switch started with a notion of what an ideal behavior might be, for instance, that no packets are dropped under any circumstances, and that latencies are always guaranteed to be met. However, switch designers realize that this ideal will be met only under some situations, but not all. Thus, in some cases the switch model might be considered imperfect or less than ideal.

Imperfection in the design process is recognition that a broad class of ideal behaviors might be implemented very effectively, while less-ideal ones must be *accommodated*. In other parts of the system *compensation* is used to handle the less ideal behaviors. Because performance is a primary factor, function and architecture are affected, together, in such designs. Often this is far superior to designing for the true worst case. For instance, the network switch is designed to optimize most operating conditions, but to drop packets or increase latency in others. Dropped packets are compensated for by higher levels of network protocols that will resend the packets — this solution is implemented in a completely separate part of the system. A wholly different example is where software memory is ideally modeled as infinite. But, it is accommodated for being finite in other parts of the model such as memory allocation schemes.

Compensation is not automatically synthesizable — a synthesis tool would not decide, in general, when it is appropriate to consider some design cases rare, or how to most effectively accommodate them. Rather, these are creative decisions of a designer which, only after being invented and set down in behavior, can be synthesized. Significantly, the acceptability of accommodation of rare design cases is an example of how function and architecture interact; design representations and methodologies must support the interaction.

6.3 Design Elements

System level designers conceptualize both the design elements of a system as well as how they can be manipulated so that their interactions can be understood. In the case of the switch, the design elements were the nodes (which coupled output and transit buffer memory) which were manipulated by the model parameters, some of which were interdependent. Other designs might have utilized different design elements; some were considered in the early parts of the design process. Importantly, the phase in which the design elements were defined had the greatest impact on the overall nature of the design. Limiting the designer's ability to think about the design elements of the system would have hampered the ability of the designers to reach a potentially elegant solution.

For instance, some have proposed all computation can be distilled to a reactive system, thus forming a basis for system level CAD. From a formal point of view, it is hard to argue with such approaches; after all, all computation can also be modeled as a collection of interconnected NAND gates. All physical systems are really structural and finite. But, from that point of view all other modeling constructs are superfluous.

A network switch is an embedded system — but not a reactive system. The fact that packets can be dropped by the interaction of the particular traffic situation is a singular notion not well supported by a reactive model. A reactive model is similar to a hardware model in that signals are processed by interconnected elements. The notion that some signals may not be processed — they might be dropped due to complex interactions between the inputs and the architecture — is difficult to conceive in a hardware model. And yet, the fact that a packet can be dropped and accommodated at higher levels of design — higher layers of the network stack protocol — is a powerful notion in networking. At the system level, the most elegant formalisms are often inconsistent with the most elegant designs — those from a designer who effectively manipulates the physical design space

with an understanding of how functionality interacts with the most significant architectural features to form an elegant result.

6.4 Shared Memory

The method of implementation of shared memory illustrates the way functionality and architecture as well as computation and communication are strongly interrelated in such a design; they are not independent at this higher level.

Network switch designers seek two goals simultaneously — a completely connected topology to minimize latency, and an ideally shared, multiport memory to reduce bursty and hotspot traffic effects. Architecturally, these two goals are opposites. One is about complete connectivity while the other is about the need to develop central arbitration for a conceptually global memory space. Since memory was at a premium on the switch and a 128-port memory was out of the question, a memory sharing scheme was important to investigate. The modeling issue is not simply a question of the presence or absence of a shared memory behavior in the switch. Rather, its method of implementation had an impact on virtually all other aspects of the design; alternatives had to be considered.

6.5 Unbounded Models of Hardware

An unbounded model allows the designer to explore architectures without having to build each case. Designers of the switch needed to consider the size of the architecture as driven by the way the system is utilized, such as how often a resource is needed. Thus, while the network switch is a hardware model, it was important to include the ability to consider computation, communication and state within the model unbounded at various times so that the designers could understand how rare design cases might actually affect the amount of hardware required.

Some test cases used traffic models that placed heavy demand on one or more output nodes, and measuring the amount of memory that would be required for *no* packets to be dropped. The amount of memory was significant enough so as to prohibit the possibility of more than a few nodes on the switch, if that traffic case had to be met under all possible circumstances. Similarly, the designers ran the model with an unbounded connection strength between nodes on the switch. Even under normal traffic models, as many as 80 channels were utilized at one time in support of little or no dropping of packets, but the times that many channels were required was so rare it was possible to easily rule out the need for excessive interconnect on the switch.

7. Conclusions

The experience described in this paper motivates the need to develop system modeling strategies around higher level concepts. At the early stages of the switch design, there was no notion of function, architecture nor how information (state) in the system would be shared. Although none of these were initially known, they evolved concurrently in response to technological limitations, a set of application data against which to model performance, creativity, and technical intuition. The issue is how to support this important phase of the system design process.

We observed that the system designers were carrying out two separate steps in the design process: defining design elements, resulting in a modeling style, and exploring how performance is impacted by manipulation of designs within that style. Unlike RTL designs where computation, communication and state are well defined *a priori*, high level system designs do not start with a premise of what the design elements of the system are. In the switch design, the designers initially thought in terms of a node that processes information in packets where some behavior is implied by the node definition. It remains an open question if the node was the best design element around which to organize the design; however, the fact that optimal design elements are not always obvious suggests the need for the CAD community to support designers in discovering them.

We observed that this level of design can be thought of as Memory Visualization Level design. MVL is pre-behavior design. The behavior must be discovered in light of how the system is going to be used. State in any computation system includes that used to construct the system's behavior and that used to store application data. The primary organizing principle of most system-level design is that of application state — memory. Thus, in MVL design, the design elements are formed around how application data is effectively organized for performance-optimized behavior.

Designers must effectively manipulate memory access on two levels: by defining design elements, and by defining the way the elements interact. With MVL design, boundaries of shared memory must be easily movable, allowing for alternate partitionings of function (even accommodation). Levels (speeds) of memory access must be easily changeable, allowing for easy consideration of alternate functional partitioning. The result of MVL design is then a behavior which represents how the overall system computation, communication and state will form the system. At that point, many design decisions have been made, facilitated by this higher level of exploration. Divide-and-conquer can then be applied to implementing each part.

8. Acknowledgments

This work was supported in part by the Pittsburgh Digital Greenhouse, NSF Award EIA-0103706, the General Motors Collaborative Research Lab at CMU, and ST Microelectronics. We thank the other members of the Switch on a Chip design team at CMU: Hyong Kim, Herman Schmit, Ece Guran and Dave Whelihan. We also thank Chris Eatedali for his suggestions.

9. References

[1] W. Leland, M. Taqqu, W. Willinger, D. Wilson. "On the self-similar nature of Ethernet traffic," *ACM/SIGCOMM '93*, pp 183-193.

[2] M. Crovella, A. Bestavros. "Self-Similarity in World Wide Web Traffic Evidence and Possible Causes," *ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems*, pp. 160-169, May 1996.

[3] D. Stephens, J. Bennett, H. Zhang. "Implementing scheduling algorithms in high-speed networks," IEEE Journal on Selected Areas in Communications, pp. 1145-1158, 1999.

[4] K. Keutzer, S. Malik, A. R. Newton, et. al. "System-Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Trans. CAD*, pp. 1523-1543, Dec. 2000.

[5] E. Lee, A. Sangiovanni-Vincentelli. "A Framework for Comparing Models of Computation," *IEEE Trans. CAD.* Vol. 17, pp. 1217-1229. December 1998.

[6] S. Meftali, F. Gharsalli, F. Rousseau, A. Jerraya. "An optimal Memory Allocation for Application-Specific Multiprocessor System-on-Chip," *ISSS 2001*, pp. 19-24.

[7] J.M. Paul, A.J. Suppé, D.E. Thomas. "Modeling and Simulation of Steady State and Transient Behaviors for Emergent SoCs," *ISSS 2001*, pp. 262-267.

[8] D. Whelihan, H.Schmit. "Memory Optimization in Single Chip Network Switch Fabrics," *DAC 2002,* pp. 530-535.

[9] Culler and Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann. 1999.

[10] J. Walrand. *Communication Networks: A First Course* McGraw-Hill. 1991.

[11] http://:www.systemc.org

[12] http://www.co-design.com/

[13] http://public.itrs.net/files/1999_SIA_Roadmap/Home.htm