

# Fabric on a Chip: Towards Consolidating Packet Switching Functions on Silicon

Brad Matthews<sup>1</sup>, *Student Member, IEEE*, Itamar Elhanany<sup>1</sup>, *Senior Member, IEEE*, Vahid Tabatabaee<sup>2</sup>

<sup>1</sup>Electrical & Computer Engineering Dept.  
University of Tennessee

<sup>2</sup>Institute for Advanced Computer Studies  
University of Maryland at College Park

**Abstract**—To resolve the high memory bandwidth requirements presented by output-queued switches, several parallel shared-memory architectures have been previously proposed. In this paper, our goal is to extend existing shared-memory architecture results while introducing the notion of Fabric on a Chip (FoC). In taking advantage of recent advancements in integrated circuit technologies, FoC aims to facilitate the consolidation of as many packet switching functions as possible on a single chip. Accordingly, this paper focuses a novel pipelined memory management algorithm which plays a key role in the context of on-chip output-queued switch emulation. We discuss in detail the fundamental properties of the proposed scheme, along with FPGA-based implementation results that illustrate its scalability and performance attributes.

## I. INTRODUCTION

Output queued (OQ) switches offer several highly desirable performance characteristics, including minimal average packet delay, controllable Quality of Service (QoS) provisioning, and work-conservation under any admissible traffic conditions [1],[2]. However, the memory bandwidth requirements of such systems is  $O(NR)$ , where  $N$  denotes the number of ports and  $R$  the data rate of each port. This requirement significantly limits scalability of OQ switches with respect to their aggregate switching capacity. In an effort to mimic the desirable attributes of output-queued switches, while significantly reducing the memory bandwidth requirements, distributed shared memory architectures, such as the parallel shared memory (PSM) switch, have recently received attention [3]. In order to properly operate, the PSM switch must have sufficient bandwidth. At the core of distributed shared memory architectures is the memory management algorithm, which determines for each arriving packet, the memory unit in which it will be placed. However, the complexity of such algorithms found to date is  $O(N)$ , where  $N$  denotes the number of switch ports, thereby inherently limiting the scalability of the scheme.

Initial work has indicated that, assuming each of the shared memory units can perform at most one packet-read or -write operation during each time slot, a sufficient number of memories needed to emulate a FCFS output queued switch is  $K = 3N-1$  [3]. The latter can be proven by employing constraint sets analysis (also known as the "pigeon hole" principle). Although this limit on the number of memories is sufficient, it has not been shown to be necessary. In fact, a tighter bound was recently found, suggesting that at least  $2.25N$  memories are necessary [4]. Regardless of the

precise minimal number of memories used, a key challenge relates to the practical realization of the memory management mechanism, i.e. the process that determines the memories in which arriving packet are placed.

In [5],[6] Prakash, Sharif, and Aziz proposed the Switch-Memory-Switch (SMS) architecture as an abstraction of the M-series Internet core routers from Juniper. The approach consists of statistically matching input ports to memories, based on an iterative algorithm that statistically converges in  $O(\log N)$  time. However, in this scheme, each iteration comprises multiple operations of selecting a single element from a binary vector. Although the nodes operate concurrently from an implementation perspective, these algorithms are  $O(\log^2 N)$  at best (assuming  $O(\log N)$  operations are needed for each binary iteration as stated above). Since timing is a critical issue, the computational complexity should directly reflect the intricacy of the digital circuitry involved, as opposed to the high-level algorithmic perspective.

In relation to standard switching architectures, the Fabric on a Chip approach exploits recent improvements in the fabrication of VLSI circuitry to consolidate switching functions on a single die. If we first consider the recent advances in packaging technology, we can assert that it is plausible to consider that all data packets arrive at a FoC directly. This reduces the need for virtual output queueing [7] and some output buffers associated with standard switch architectures. Furthermore, we are now given the ability to embed multiple Megabits of dual-port SRAM on chip. From this, packets can be efficiently stored and switched internally. The crosspoint switches and scheduler, key components in input-queued switches, are avoided thereby substantially reducing chip count and power consumption. Correspondingly, much of the signaling and control information that typically spans multiple chips can be carried out on chip. Finally, the switch management and monitoring functions can be centralized since all the information is available at a single location. Acceptable costs, according to the paradigm fostered here, include fixed latency and a reasonable increase in the number of memory units used.

This paper addresses a core challenge of FoC design, the memory management algorithm, by introducing a novel multi-stage pipelined memory management architecture in which packets arrive to the system and are assigned to non-conflicting memories. In discussing the memory management algorithm,

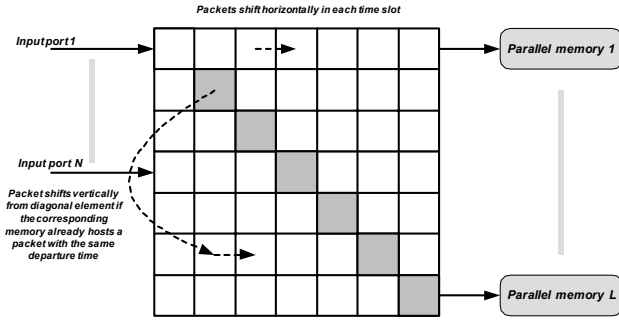


Fig. 1. Memory Management Pipeline Architecture

a sufficiency condition on the number of memories required is established. We conclude this paper by addressing the implementation issues and further establish the viability of FoC architectures by providing synthesis results of the critical design path.

## II. PACKET PLACEMENT ALGORITHM

### A. Switch Architecture

This section describes the proposed pipelined memory-management approach for PSM switches. The first step is to calculate the departure time of each arriving packet, a process governed by the output scheduling algorithm used. The most straightforward scheduler is the first-come-first-serve (FCFS), in which packets are assigned departure times in accordance with their arrival order. To provide delay and rate guarantees, more sophisticated schedulers [1],[2] can be referenced in order to determine the appropriate departure time assignments. The main contribution of this paper resides in the fact that the placement algorithm distributes the packet-placement process, thereby gaining execution speed at the cost of a fixed latency. In the proposed switch architecture, the memory-management algorithm is implemented using a multi-stage pipeline architecture, as depicted in Figure 1.

The pipeline architecture consists of  $L \times L$  cell buffering units arranged in a square structure. Each row is associated with one of the parallel shared memory units. Hence, the architecture requires  $L$  parallel shared memories. Incoming packets from input port  $i$  are initially inserted into row  $i$ . The underlying mechanism is that at every time slot, packets are horizontally shifted one step to the right, with the exception of the diagonal cells of the structure. These cells have the ability to shift (move) vertically to another row of the same column. A cell moves vertically if any of the following two conditions are met: (1) the memory associated with the row in which it is currently located already contains a packet with the same departure time; (2) there is another cell ahead of it in the same row or the associated memory with the same departure time. Therefore, vertical moves are used as means of resolving memory placement contentions. The goal of the scheme is that once a packet reaches the last column of the pipeline, it is guaranteed to be located in a row which is associated with

a memory that does not contain any packets with the same departure time.

### B. Memory Management Algorithm and Related Resources

It should be apparent that each row, with the exception of the diagonal elements, can be considered as a simple shift register whereby packets are shifted one stage to the right at each time step. In each stage of the pipeline, a single packet assignment is attempted per row. These assignment attempts are only made on the diagonal elements of the pipeline structure, so that there can be at most one assignment per column as well. The motivation for doing so is to isolate memory assignments, thereby reducing the complexity of the placement mechanism.

Each row maintains a mapping that specifies pre-allocated (or reserved) departure times of packets that have successfully passed the diagonal position. This mapping, which is essentially a binary mask, is referred to as the row's *availability map*. The diagonal element refers to this map in order to check if its departure time is available at that row. If it is available, the cell will horizontally shift to the right and mark the corresponding departure time in the availability map as reserved. If it is already reserved by another cell, it will first shift vertically down to another row where the corresponding element position is empty and its departure time is not reserved. Note that in this manner, following the first vertical jump, a cell may still have contention with (1) packets that are ahead of it with the same departure time but have not reached the diagonal element, or (2) packets having the same departure time that have leaped in the same time slot to the same row. To resolve contention originating from the first scenario mentioned, a second binary availability map called the *reservation map* is maintained. Upon shifting to a new row, each cell must update the row's reservation map by asserting the bit corresponding to its departure time. This allows future packets to guarantee that the row they are moving into does not contain packets that have arrived in a previous time step with the same departure time.

For pragmatic reasons, both maps (the availability and reservation) cannot be unbounded in size. In all practical switching systems, once a buffer reaches (or is close to reaching) its limit, flow-control signaling is provided to the sources, indicating the need to either slow down or temporarily stop the flow of packets to a certain destination. Such a mechanism is always required since instantaneous data-traffic congestion may occur at any router or switch. In fact, even if the traffic is said to be statistically admissible, meaning that no input or output is oversubscribed, it may still be the case that for short periods of time a given output port, for example, is oversubscribed. To address such scenarios, and in an effort to reduce the probability of dropping any packets, the linecards typically host large memory spaces. Traffic is regulated as if flows through the fabric in order to ensure that no packets are lost.

In order to obtain an estimate of how much memory is required, one can refer to the pure output queued switch model. The approach taken is to obtain the expected queue size for

each output port, from which a good estimate of the memory need can be obtained.

Consider an  $N \times N$  output queue in which buffering occurs only at the output ports. During each time slot, the switch must transfer all arriving packets into their respective output ports. Let us first assume that arriving packet traffic is uniformly distributed across the output ports and obeys a (memoryless) Bernoulli i.i.d. process. Consequently, a packet arrives at a given input port with probability  $p$  and equal chance of being destined for any of the  $N$  output ports, i.e. uniformly and at random. Due to the inherent symmetry of the system, we shall analyze a single output port (e.g. output port 1) from which we will be able to derive the behavior at all other ports. Let  $Q_t$  denote the number of packets buffered in the output port buffer at time  $t$ . Since in each time slot, a maximum of  $N$  packets may arrive to a given output port and at most a single packet can depart, the following Markov chain can be used to accurately portray the evolution of the output buffer:

$$Q_{t+1} = Q_t + A_{t+1} - D_{t+1} \quad (1)$$

where  $A_t \in [0, N]$  denotes the number arrivals during time slot  $t$  and  $D_t \in [0, 1]$  is an indicator function representing a departure event. Provided the queue size is not allowed to take negative values, then  $D_t = 1$  if and only if  $Q_t > 0$ . At is clearly i.i.d. (as defined above), and its distribution is given by

$$P(A_t = k) = \binom{N}{k} \left(\frac{p}{N}\right)^k \left(1 - \frac{p}{N}\right)^{N-k} \quad (2)$$

which converges to

$$\lim_{N \rightarrow \infty} P(A_t = k) = e^{-p} \frac{p^k}{k!} \quad (3)$$

Moreover, since the arrival process is independent of the departure process,  $A_t$  is independent of  $Q_t$ .

Taking the expectation at both sides of (1), we consider the steady-state result by omitting the subscript  $t$  yielding the expected expression  $E[A] = P(Q > 0)$ . However, we are primarily interested in  $E[Q]$ , so we square both sides of (1) and take the expectation, resulting in

$$E[Q^2] = E[Q^2] + E[A^2] + P(Q > 0) + 2E[AQ] - 2E[Q|_{Q>0}] - 2E[A|_{Q>0}] \quad (4)$$

Since  $A$  and  $Q$  are independent, we obtain

$$E[Q] = \frac{E[A^2] - E[A]}{2(1 - E[A])}. \quad (5)$$

At this point, all that remains is to determine  $E[A^2]$  and  $E[A]$ . In view of (3), we assert that  $E[A] = p$  and  $E[A^2] = p + p^2$ , therefore

$$E[Q] = \frac{p^2}{2(1 - p)}. \quad (6)$$

By multiplying the above term by  $N$ , we obtain the total average memory consumed by a pure output queued switch. This provides us with a reasonable estimate of the amount of memory the PSM switch is to host on chip. As an example,

for a load of 90% (i.e.  $p = 0.9$ ) the per-port average memory requirements are merely 4.95 packets.

Similar analysis may be applied to obtain the average queue size when bursty traffic is considered. It is widely acknowledged that real-life traffic tends to be correlated, or bursty, on many levels [8],[9]. Accordingly, we consider a discrete-time, two-state Markov chain generating arrivals modeled by an ON/OFF source that alternates between the ON and OFF states. Let the parameters  $\alpha$  and  $\beta$  denote the probabilities that the Markov chain remains in states ON and OFF, respectively. An arrival is generated for each time slot that the Markov chain spends in the ON state. Letting  $f_n$  denote the inter-arrival times distribution, the probability of two consecutive arrivals occurring is identical to the probability that following an arrival the Markov chain remains in state ON, i.e.  $f_1 = \alpha$ . Similarly,  $f_2$  is the probability that following an arrival, the chain transitions to the OFF state and then returns to the ON state. For  $n > 2$ , it is apparent that following a transition from the ON state to the OFF state, there are  $n - 2$  time slots during which the chain remains in the OFF state before returning to the ON state. Accordingly, we obtain the following general expression for  $f_n$ :

$$f_n = \begin{cases} \alpha & n = 1 \\ (1 - \alpha)\beta^{n-2}(1 - \beta) & n > 1 \end{cases} \quad (7)$$

From (7) the average queue size,  $B = (1 - \alpha)^{-1}$ , and  $E[A^2]$  can be directly obtained, from which we conclude that the average queue size is given by

$$E[Q_{(ON/OFF)}] \approx \frac{Bp}{(1 - p)} \quad (8)$$

Note that in this case the average memory size is linearly proportional to the mean burst size. Figure 2 illustrates the aggregate amount of on-chip memory needed for a switch of 100 ports whereby packets are 64 bytes in size. The mean burst size is 16 packets. The reader should note that at 10 Gbps per port, this switch has an aggregate capacity of a Terabit/sec.

### C. Sufficiency Condition on the Number of Memories

In this section, we provide some complexity results for the memory requirements of the pipeline architecture. In the proposed model, rows of the pipeline are arranged in  $P$  sequential blocks, whereby there is one row per input port (for total of  $N$  rows) in the first block. As such, every input port writes its packets to one row. A cell in block  $r$  can only jump vertically to a cell in block  $r + 1$ . In order to illustrate the underlying memory-management principal, we shall refer to the following example.

*Example 1:* Consider the simple scenario depicted in 3. The state of the pipeline for four consecutive time slots (i.e.  $t, t+1, t+4, t+5$ ) is shown. At time  $t$ , there are two packets in the second row and three packets in the third row, all with the same departure time  $D$ . Two packets are located at diagonal positions, and since there are two packets ahead of them with the same departure time, they shift down to a row in the second block (row 6) and then move one position to the

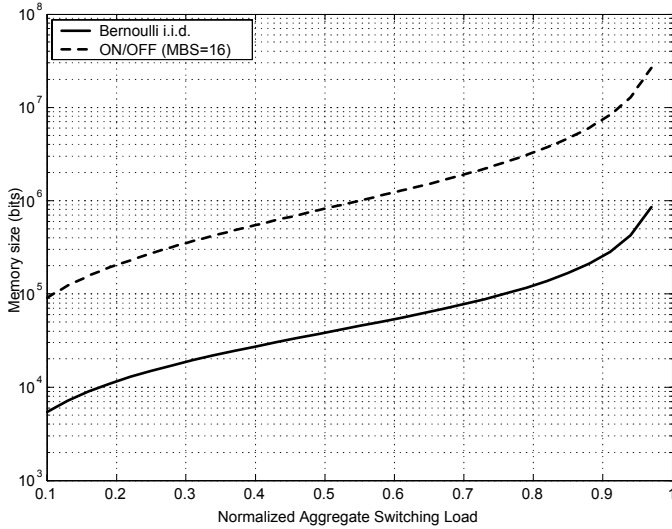


Fig. 2. Average Memory Requirements from a FoC Device.

right, as shown in the pipeline diagram for time  $t + 1$ . Note that the packets were not permitted to move to row 5 since their position is already occupied with two other packet with departure times  $X$  and  $Y$ . At time  $t + 4$ , the second packet with departure time  $D$  in row 6 reaches the diagonal element, and, since there is another packet ahead of it with same departure time, it moves vertically to a row in the third block (row 7), followed by a shift one step to the right, as shown in the  $t + 5$  diagram. In this example, the pipeline consisted of three blocks of rows with 4, 2 and 1 rows in each, respectively. We emphasize that the notion of blocks is used only for illustration purposes. As will later be discussed, partitioning the structure into these blocks facilitates the complexity analysis as well as memory requirements of the architecture.

*Lemma 2:* There should be at least  $2N - r$  rows in block  $r$ , for  $r \in [2, 3, \dots, P]$ .

*Proof:* Consider a cell that is shifting vertically from block  $r$  to block  $r + 1$ . It will find at most  $N - 1$  rows blocked by other packets, since there have been at most  $N - 1$  other packets that may have arrived at the same time as this packet and may have shifted down in prior time slots. Moreover, it will find at most  $N - r - 1$  other rows having packets with the same departure time. Note that there could be at most  $N - 1$  other packets in the system with the same departure time; however,  $r$  of them have already been accounted for having caused this packet to jump to block  $r + 1$ . Therefore, there are at most  $2N - r - 2$  rows (or locations) that this packet cannot move to in block  $r + 1$ . Using the pigeon-hole principle, we conclude that  $2N - r$  is a sufficient number of rows for block  $r \in [2, 3, \dots, P]$ . ■

From lemma 2, for a switch with  $N$  ports and  $P$  blocks, the total number of rows (parallel memories) can be expressed

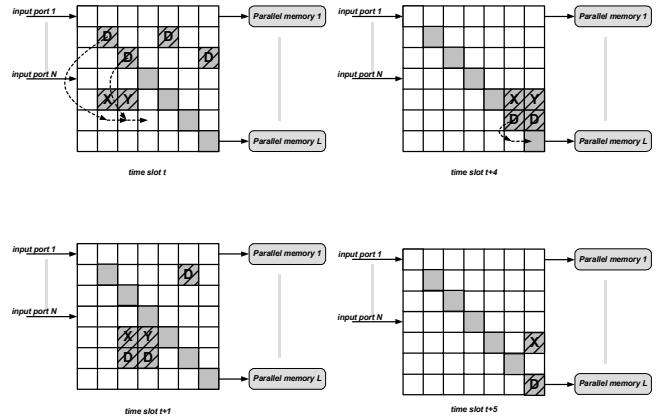


Fig. 3. Example illustrating the proposed memory-management algorithm for a 4-port switch. The state of the pipeline structure is depicted for 4 consecutive time slots.

as:

$$\begin{aligned}
 L(N) &= N + (2N - 2) + (2N - 3) + \dots + (2N - P) \\
 &= N + 2N(P - 1) - (P + 2)(P - 1)/2 \\
 &= (2P - 1)N - (P + 2)(P - 1)/2
 \end{aligned} \tag{9}$$

In order to compute number of the rows, we have to find  $P(N)$ , which directly reflects the maximum number of vertical shifts a packet can perform prior to being successfully assigned to a memory. We shall refer to Example 1. and derive an upper limit on the number of conflicting packets with the same departure time in the fourth block, i.e. after three vertical shifts have occurred. We would like to find an upper limit on the number of conflicting packets after three shifts and use that recursively to obtain the maximum number of jumps, and hence blocks, that are sufficient to guarantee that each arriving packet will be successfully assigned a memory.

*Lemma 3:* The maximum number of packets with the same departure time in the fourth block is  $(\sqrt{N} - 1)$

*Proof:* Suppose that there are  $N_1$  packets with the same departure time in the first block. Throughout the proof, we shall refer to this set of packets having the same departure time. Note that  $N_1 \leq N$ , since there cannot be more than  $N$  packets with the same departure time in the system. Let us assume that these packets are located in  $M_1$  ( $M_1 \leq N$ ) rows of the first block. Therefore, the number of packets that move vertically to the second block will be  $N_2 = N_1 - M_1$ . Next, we compute the number of rows in the second block that contain one of these packets. There are  $N_2 = N_1 - M_1$  packets that have moved to the second block, and the maximum number of packets that can shift simultaneously to the same row is  $M_1$ . Hence,

$$M_2 \geq \left\lfloor \frac{N_1 - M_1}{M_1} \right\rfloor \tag{10}$$

Therefore, the maximum number of packets with the same departure time that can move to the third block is given by

$$N_3 = N_2 - M_2 \leq N_1 - M_1 - \left\lfloor \frac{N_1 - M_1}{M_1} \right\rfloor \tag{11}$$

If  $N_1 - M_1$  is divisible by  $M_1$ , then

$$N_3 \leq N_1 \left(1 - \frac{1}{M_1}\right) - M_1 + 1 \quad (12)$$

otherwise, since  $N_4 \leq N_3 - 1$ , we have

$$\begin{aligned} N_3 &\leq N_1 \left(1 - \frac{1}{M_1}\right) - M_1 + 2 \\ N_4 &\leq N_1 \left(1 - \frac{1}{M_1}\right) - M_1 + 1 \end{aligned} \quad (13)$$

The maximum value of the expression in 13 is reached when  $M_1 = \sqrt{N_1}$ . Substituting  $N_1 = N$ , yields the inequality

$$N_4 \leq \left(\sqrt{N} - 1\right)^2 \quad (14)$$

Note that if  $N$  is a complete square we have,

$$N_3 \leq \left(\sqrt{N} - 1\right)^2 \quad (15)$$

In the following corollary, we exploit these results to determine the order of the memory blocks.

*Corollary 4:* A sufficient number of parallel memory blocks required for an  $N \times N$  switch, employing the proposed architecture, is  $O(\sqrt{N})$

*Proof:* Equation 14 shows that for an  $N$ -port switch, the maximum number of conflicting packets with the same departure time in the fourth block is  $\left(\sqrt{N} - 1\right)^2$ . Let  $P(N)$  represent the number of stages required for an  $N$ -port switch. We thus have,

$$\begin{aligned} P(N) &= P(N - 2\sqrt{N} + 1) + 3 \\ P(1) &= 1 \end{aligned} \quad (16)$$

from which we conclude that  $P(N) = O(\sqrt{N})$ . ■

*Theorem 5:* For an  $N$ -port switch, where  $N \leq k^2$   $k \in \{1, 2, \dots\}$ , the number of memories is

$$L(N) \leq 4k^3 - 5k^2 + k + 1 \quad (17)$$

with equality if  $N = k^2$

*Proof:* We prove the equality for  $N = k^2$ , suggesting that the general case trivially follows. We first show by induction that the number of required row blocks are  $P(k^2) = 2k - 1$ . For  $k=1$ , the result is trivial. We assume that the result holds for  $k$  and infer it for  $k+1$ . For  $N = (k+1)^2$ , using lemma 3 and (10) (given that  $N$  is a complete square), we have

$$P((k+1)^2) = P(k^2) + 2 = 2k - 1 + 2 = 2(k+1) - 1 \quad (18)$$

Then, we utilize the relationship proven for  $L$  in (9) to obtain

$$\begin{aligned} L(k^2) &= (2P - 1)N - (P + 2)(P - 1)/2 \\ &= (4k - 3)k^2 - (2k + 1)(2k - 2)/2 \\ &= 4k^3 - 5k^2 + k + 1 \end{aligned} \quad (19)$$

The above theorem states that the number of parallel memories required in this architecture is  $O(N^{1.5})$ . Given that the minimum number of memories is  $O(N)$ , the increased

memory requirement observed is the price paid in order to make the assignment problem parallel and feasible from an implementation perspective. For example, with  $N = 16$  (i.e.  $k=4$ ), the number of parallel memory elements is 177, arranged in 7 blocks.

### III. HARDWARE REALIZATION

This section provides a detailed discussion on the different implementation aspects pertaining to the proposed architecture. In particular we will focus on the timing requirements from a practical realization of the proposed scheme, as well as derive related scalability properties.

#### A. Architecture

It becomes apparent that the critical path in the design is the memory assignment process that takes place at the diagonal elements, or decision cells, of the pipeline structure. Each decision cell, residing on row  $r_i$  ( $i \in [1, 2, \dots, L]$ ) must determine whether a packet,  $p_i$ , can be placed at the memory associated with row  $i$ . The memory is considered available for packet placement if it does not already contain a packet with the same departure time,  $d_i$ , as the packet residing in the decision cell. If the memory is occupied with a packet with the same departure time, then the packet is shifted to a new row  $r_n$ .

In light of these placement rules, an availability vector of length  $k$  is created to indicate locations available for the placement of a packet at time  $d_i$ . In a distributed shared memory switch, each shared memory will contain  $k$  cells representing  $k$  consecutive departure times. This dictates the size of the availability map, which consists of  $L$  availability vectors. Note that in order to perform the appropriate placement selection for packet  $p_i$  with departure time  $d_i$ , the column of bits pertaining to the  $d_i$  position in the availability map should be examined. Based on lemma 2, since a packet always shifts into the next block, we note that the maximal number of rows that is to be examined by each diagonal location for shifting a packet is  $2N - 2$ . This inherently bounds the critical path of the design, since it defines an upper bound on the search space that must be considered at each diagonal element.

While the availability map provides information as to the memories contents, it does not provide any information regarding the location of packets that arrived during the same time slot as  $p_i$ . There are at most  $N - 1$  packets that arrived at the same time as  $p_i$ . Having stated that a vector of length  $2N - 2$  is sufficient to locate an available row, an occupancy vector must be created to determine the location of  $N - 1$  packets within the  $2N - 2$  subset reflecting potential adequate rows. Hence, the bitwise-OR of the occupancy vector and the availability vector provides a single decision vector representing viable rows for packet placement. The decision vector defines rows which (1) are not associated with memories that contain a packet with departure time  $d_i$ , and which (2) do not contain a packet with the same arrival time as  $p_i$ .

The memory-management algorithm is based on selecting an available memory from one of  $2N - 2$  memories, reflected

by the decision vector, such that all  $N$  packets are placed at the end of the  $O(N^{1.5})$  phases. To accomplish this, we first determine whether the packet in the decision cell is blocked by a packet with the same departure time as  $p_i$ , as indicated by the availability vector. If  $p_i$  is blocked, the decision vector is then used to select an available row from one of  $2N-2$  rows. The result of this decision process is that  $p_i$  is either written to the memory associated with the row in which it resides or placed in a row that is free of conflict.

While placement is certainly the more difficult task, some thought must be given to retrieving packets from a shared memory structure. One practical solution is to present a single shared packet bus to each shared memory unit. As packets are read from memory, each egress port will have at most one packet destined for it at any given time. Correspondingly, we can present the shared packet bus, of size  $N * packet\_size$ , to each shared memory unit such that each  $j^{th}$  bus slice, of size  $packet\_size$ , represents a packet destined for output  $j$ . Each shared memory unit determines the destination ( $j$ ) of the packet it is transmitting to an egress port and only drives the  $j^{th}$  bus slice of the packet bus. As such, each egress port will only be responsible for transmitting the packet located at the  $j^{th}$  bus slice, corresponding to the port enumeration of the packet bus. The implementation cost of this approach is reflected in the requirement for an  $N$ -to-1 multiplexor to be located in each memory cell.

### B. FPGA-based Results

To establish the viability of the proposed multistage PSM memory management architecture, we have implemented the critical path in VHDL and synthesized its components targeting a Xilinx Virtex-4 XC4VLX200-10-FF1513 FPGA device. This implementation consists of 16 ports operating at 40 Gbps each, representing a switch with an aggregate capacity of 640 Gbps. The maximum departure time value configured to be 64. With no logic speedup, the physical resource requirements for the system are as follows: physical memories – 176, decision cells – 176, registers  $\approx$  15.4k, pipeline depth – 176. After packets are placed into shared memories, they are presented at the output of the PSM switch every 12.5 ns. Given a link speed of 40 Gbps, and considering 64-byte packets, the scheme is only realizable if packet placement decisions can be made in less than 12.5 ns. We therefore consider the worst-case path, along with associated delays attributed to each decision made along this path.

First, we have identified the worst-case path origin to be the availability vector, which is obtained as an output of a  $k$ -to-1 multiplexer applied to the availability map. The maximum departure time value, having previously stated to be 64, requires the  $k$ -to-1 multiplexer design multiplex 64 potential availability vectors for presentation at the decision cell. The resulting 64-to-1 multiplexer was found to have an overall delay of 2.57 ns.

The selected availability vector is then used by the decision cell to determine the correct row in which to place conflicting packets. The time required to identify a memory conflict,

followed by locating an available row, was found to be 3.823 ns. Once the available row is located, a bitwise OR is applied to the departure time and the potential  $2N-2$  packets that may be attempting to relocate to the same row. This is required, as discussed earlier, to maintain the integrity of the availability map. The delay associated with this wide OR was 1.52 ns. Registering of the departure time denotes the terminating point of the worst-case path, from which we can assert that the worst-case delay is 7.913 ns – well below the 12.5 ns limit stated above. As a final note, the overall latency contributed by the architecture with respect to a pure output-queued switch is 2.2  $\mu$ s (176 stages of 12.5ns each).

## IV. CONCLUSION

The notion of designing a packet switching fabric on a chip was introduced and discussed from a theoretical as well as technological perspective. It has been argued that in the context of emulating an output-queued switch, a core challenge pertains to the memory-management algorithm employed. A proposed packet-placement algorithm and related architecture were described in detail, emphasizing the feasibility attributes. Future work will focus on further reducing memory requirements and the incorporation of quality of service (QoS) provisioning. The switch model and framework presented here can be broadened to further investigate the concept of consolidating multiple switch fabric functions on silicon.

## V. ACKNOWLEDGEMENTS

This work has been partially supported by the Department of Energy (DOE) under research grant DE-FG02-04ER25607, and by the Woodrow W. Everett, Jr. SCEE Development Fund in cooperation with the Southeastern Association of Electrical Engineering Department Heads.

## REFERENCES

- [1] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344–357, June 1993.
- [2] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *Proc. of ACM SIGCOMM '95*, pp. 231–242, September 1995.
- [3] R. Z. S. Iyer and N. Mckeown, "Routers with a single of buffering," *ACM Computer Communication Review (SIGCOMM'02)*, pp. 251–264, 2002.
- [4] H. Liu and D. Mosk-Aoyama, "Memory management algorithms for dsm switches," *Stanford Technical Paper*, July 2004.
- [5] A. P. A. Aziz and V. Ramachandra, "A near optimal scheduler for switch-memory-switch routers," in *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 343–352, July 2003.
- [6] A. A. A. Prakash and V. Ramachandra, "Randomized parallel schedulers for switch-memory-switch routers: Analysis and numerical studies," *IEEE INFOCOM 2004*, 2004.
- [7] Y. Tamir and G. Frazier, "Higher performance multiqueue buffers for vlsi communication switches," in *15th Annual Symposium on Computer Architecture*, pp. 343–354, 1988.
- [8] C. Williamson, "Internet traffic measurement," *IEEE Internet Computing*, no. 5, pp. 70–74, 2001.
- [9] G. I. C. Barakat, P. Thiran and C. Diot, "On internet backbone traffic modeling," *ACM Sigmetrics*, 2002.