Design and Implementation of High-Performance Memory Systems for Future Packet Buffers

Jorge García, Jesús Corbal*, Llorenç Cerdà, and Mateo Valero
Polytechnic University of Catalonia - Computer Architecture Dept.
{jorge,jcorbal,llorenc,mateo}@ac.upc.es

Abstract

In this paper we address the design of a future high-speed router that supports line rates as high as OC-3072 (160 Gb/s), around one hundred ports and several service classes. Building such a high-speed router would raise many technological problems, one of them being the packet buffer design, mainly because in router design it is important to provide worst-case bandwidth guarantees and not just average-case optimizations.

A previous packet buffer design provides worst-case bandwidth guarantees by using a hybrid SRAM/DRAM approach. Next-generation routers need to support hundreds of interfaces (i.e., ports and service classes). Unfortunately, high bandwidth for hundreds of interfaces requires the previous design to use large SRAMs which become a bandwidth bottleneck. The key observation we make is that the SRAM size is proportional to the DRAM access time but we can reduce the effective DRAM access time by overlapping multiple accesses to different banks, allowing us to reduce the SRAM size. The key challenge is that to keep the worst-case bandwidth guarantees we need to guarantee that there are no bank conflicts while the accesses are in flight. We guarantee bank conflicts by reordering the DRAM requests using a modern issue-queue-like mechanism. Because our design may lead to fragmentation of memory across packet buffer queues, we propose to share the DRAM space among multiple queues by renaming the queue slots. To the best of our knowledge, the design proposed in this paper is the fastest buffer design using commodity DRAM to be published to date.

1. Introduction

Nowadays, router design follows two clearly different trends. Firstly, advances in optical transmission technologies and the sustained growth of Internet traffic have led to research efforts focused on supporting higher line-speed rates and a larger number of line interfaces [15]. Secondly, the pervasive use of the Internet and the introduction of multimedia applications demanding more functionality (e.g. stateless and stateful classification of packets, Quality of Service, security support, etc.), has resulted in an increased interest in Network Processor design. Clearly, in order to follow these developments, current router design needs to be reconsidered and innovative research in all router subsystems is needed. In this paper we focus our attention on packet buffer design for future high-speed Internet routers.

A router is a network node connecting several transmission lines whose basic function is to forward *Internetworking Protocol* (IP) packets across the lines depending on the packet's destination IP address and the information stored in a routing table. The main functional units of a router are: (i) *Line interfaces*, which connect the router to each transmission line, (ii) *Network processors*, [22, 2, 1] which process the packet headers, look up routing tables, classify packets, and perform related tasks, (iii) *Packet buffers*, which store the packets waiting to be forwarded, (iv) *the switch fabric*, which interconnects the router's packet processing units, and (v) *the system processor*, which performs the control functions such as routing table computation, configuration tasks, etc.

Packet buffers for the next generation routers will require a storage capacity for several Gb (giga bits) of data and a bandwidth of several hundreds of Gb/s, and managing internal data structures of almost one thousand queues. Moreover, the design must be able to handle any input pattern, and not only traffic patterns that can be present in average. This restrictive condition is usual in networking equipment, and leads to design choices that optimize the worst case and not the most common case. Currently proposed packet buffer architectures do not meet these strict requirements.

Traditionally, fast packet buffers were built using low-latency SRAM. However, with the increasing capacity requirements, high density DRAMs have become the preferred choice. DRAM-based packet buffers can easily provide for a bandwidth of up to around 1 Gb/s, but if we increase the required bandwidth to several Gb/s the design becomes difficult. For instance, [9] addresses the design of a packet buffer using a single-chip 16 Mb SDRAM with a 16 bit data interface and a 100 MHz clock. Even though the peak bandwidth is of 1.6 Gb/s, the guaranteed bandwidth drops to 1.2 Gb/s, due to activate and precharge overhead. A multiple chip design would increase the buffer

* Jesús Corbal is currently working at BSSAD, ILB, INTEL.

Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36 2003)
0-7695-2043-X/03 $17.00 © 2003 IEEE

bandwidth, but the increase in bandwidth would not be proportional to the total number of chips. Using, for instance, the same SDRAM parameters, an 8-chip configuration with a 8x wider bus would provide a guaranteed bandwidth of only 5.12 Gb/s. Increasing the number of chips and widening the data bus therefore yields diminishing returns, while creating problems [6] such as higher memory granularity, more memory components in the line card, wider data paths, etc.

The low efficiency of multichip DRAM buffers can be improved using some special techniques oriented to reduce bank conflicts in a DRAM buffer using pipelining and out-of-order access techniques [24, 23, 16], or exploiting row locality whenever possible in order to enhance average-case DRAM bandwidth [4, 10]. Using faster DRAM components (e.g. RLDRAM [21], FCDRAM [7], etc) would also lead to faster buffers. However, from the previous discussion it is clear that for supporting a line-rate as high as OC-3072 alternatives to DRAM-only buffers should be considered.

To our knowledge, the fastest packet buffer with worst-case bandwidth guarantees that can be found in the literature is the hybrid SRAM-DRAM design proposed in [13]. In this hybrid DRAM-SRAM design, in each line interface the arriving cells are stored in two SRAMs which cache the front and back of the queues (tail and head SRAMs), and in one central DRAM. Cells that come into the buffer are placed in the tail SRAM, whereas cells that will leave the system in the near future are placed in the head SRAM. The availability of room in the tail SRAM and the availability of cells to be served in the head SRAM are controlled using a Memory Management Algorithm (MMA), which must *guarantee* that there is always room in the head SRAM for an incoming packet and that any packet to be output is always present in the tail SRAM before the outputting needs to be done (i.e. the cache never misses). When the occupancy of the tail SRAM reaches a given threshold, a transfer from SRAM to DRAM of a group of cells addressed to the same output interface is ordered by the MMA. Conversely, when the head SRAM needs to serve a cell that currently resides in DRAM, the MMA orders a group transfer from DRAM to SRAM.

Bank conflicts are avoided by spacing consecutive DRAM accesses by a DRAM random access time. Thus, group size must be set to the ratio of DRAM random access time to the transmission time of a cell. As this factor directly influences the SRAM size, large SRAMs are needed to sustain high line rates and a large number of interfaces. This, in turn, limits what access times are attainable. This buffer design would support line rates up to OC-3072, but only for a reduced number of interfaces.

Currently available high-speed routers support up to 16 interfaces at OC-192 (10 Gb/s) or OC-768 (40 Gb/s) line rates. It is devised, however, that next generation high-end systems will support a much more larger number of interfaces (e.g. 624 or even more) at OC-192, OC-768 or even OC-3072 (160 Gb/s) line rates. The goal of this paper is to reduce the SRAM size of [13] while supporting a large number of interfaces. The key observation we make is that we can reduce the effective DRAM access time by overlapping multiple accesses to different banks, allowing us to reduce the granularity of the accesses thereby reducing the SRAM size. The key challenge is that to keep the worst-case guarantees of [13] we need to guarantee that there are no bank conflicts while the accesses are in flight.

In the proposal presented in this paper we maintain the same SRAM/DRAM structure and MMA subsystem as in [13], but we completely redesign the DRAM system. We propose a DRAM storage scheme and associated access method that achieves a conflict-free access memory organization with a reduction of the granularity of DRAM accesses. As we show, we obtain peak memory performance while reducing SRAM size by an order of magnitude. This basic scheme would lead to DRAM memory fragmentation, which is dealt with by means of renaming of queues, although for some quite specific traffic patterns, some degree of memory fragmentation still could appear

To our best knowledge, the design proposed in this paper is the fastest buffer design using commodity DRAM that has been published up to date. A technological evaluation presented in this paper shows that our design can support up to 800 queues for line rates of 160 Gb/s using commodity DRAM.

2. System assumptions

During the next few years, aggregate router throughput will probably grow by increasing the number of interfaces rather than increasing line rates [15]. Firstly, the current industry standard supports 16 ports. However the use of Dense Wavelength-Division Multiplexing (DWDM) increases the number of channels available on a single fiber (without increasing the individual line rates), leading to a number of interfaces in the order of several hundreds. Secondly, although line rates have increased rapidly over the past years (up to OC-192 or OC-768), it seems that this increase is close to its limits: around OC-3072.

In this paper we address the design of packet buffers for router supporting line rates as high as OC-3072 and almost one thousand line interfaces. We can therefore set several parameters that are of utmost importance in the packet buffer design: required bandwidth, buffer size, basic timeslot and number of internal data structures internal to the buffer.

Required bandwidth: We assume that most buffering is placed at the input line interfaces (input-queuing architecture) as this leads to minimum packet buffer bandwidth requirements. For input-queuing architecture the required packet buffer bandwidth is twice the line rate, as every packet must be both written and read from memory before being forwarded. We do not consider any further speeding-

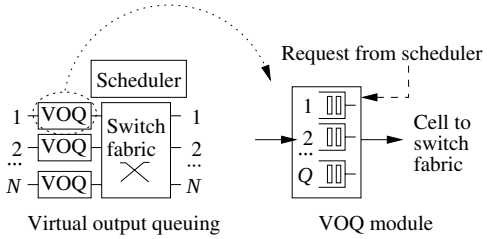

Figure 1. Input-queueing router architecture using VOQ buffer.

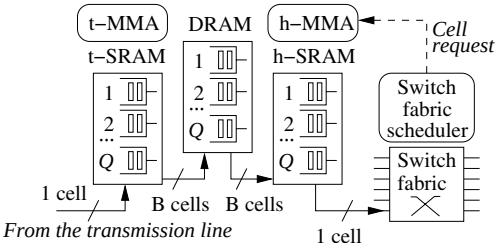

Figure 2. RADS memory architecture of the packet buffer.

up that could be needed in order to compensate for header and segmentation overhead. We should also point out that most of the results in this paper could be applied to the design of other types of packet buffers, such as shared-memory routers with a high aggregated throughput.

Buffer size: The amount of required buffering would be fairly large. As a rule of thumb, router manufacturers usually employ packet buffers of a size equal to an estimate of a typical packet round-trip-time over the Internet times the line rate [6]. Taking a typical round-trip-time of 0.2 sec, the required buffer size for a line rate of 160 Gb/s is of 4 GB.

Basic time-slot: We assume that packets in the router are internally fragmented into fixed-length 64 byte units that we call cells [3]. Cells are handled as independent units, although they are reassembled at the output port before packet transmission. The system operates synchronously into fixed time-slots, which correspond to the transmission time of a cell at the line rate. For instance, for a line rate of 160 Gb/s the basic time-slot is of 3.2 ns.

Number of internal data structures: As is well known, in order to achieve full link utilization, input-buffered routers require the use of *Virtual Output Queuing* (VOQ) [20]. In VOQ, (see Figure 1) the input buffer maintains Q separate logical FIFO queues. Each logical queue corresponds to an output line interface and a class of service. When a cell arrives to the input line interface, it is placed at the tail of the queue corresponding to its outgoing interface. When an input port receives a request for a cell addressed to a given output, the cell is taken from the head of the corresponding queue in the VOQ buffer. We will assume that our packet buffer incorporates this mechanism. We assume that the number of Virtual Output Queues to be supported is around one thousand.

3. Random Access DRAM System (RADS)

In [13] a VOQ buffer design targeted at worst-case bandwidth is discussed. The system (see Figure 2) consists of two fast but costly SRAM modules (t-SRAM and h-SRAM), a slow but low cost DRAM system, and two Memory Management Algorithm modules (t-MMA and h-MMA). t-SRAM and h-SRAM respectively cache the tail and head of each VOQ logical queue. The rest is stored in DRAM. The SRAM memory bandwidth must fit the line rate, which means that the SRAM access time must be less than or equal to the transmission time of a cell.

In order to match DRAM/SRAM access times, transfers between DRAM and SRAM occur in batches of cells. We shall refer to the batch length as the *data granularity* of the memory scheme. The design in [13] considers that transfers between SRAM and DRAM are done every random access time of the DRAM. We shall refer to this system as *Random Access DRAM System* (RADS). We define B as the minimum granularity that can be used in RADS. Therefore, using RADS, the DRAM-SRAM transfers consist of batches of B cells that begin every random access time of DRAM. Thus, the random access time of DRAM must be equal to B time-slots.

Every B time-slots, the t-MMA must select a queue from which B cells are to be transferred from t-SRAM to DRAM. This algorithm should guarantee that the t-SRAM does not fill up before DRAM. Otherwise, losses would occur before the DRAM is full. A t-MMA that would avoid these losses is simple: transfer B cells to DRAM from any queue with an occupancy counter (i.e. the number of cells of the queue present in the SRAM) higher than or equal to B. In this case, the required tail SRAM size would be $Q\,(B-1)+1$ cells.

The h-SRAM is a more complex system. This algorithm has to guarantee that cells transferred between DRAM and h-SRAM can accommodate the sequence of cells requested by the external scheduler. Otherwise, the cell requested by the scheduler may not be present in the h-SRAM as it may not yet have been transferred from the DRAM. We shall refer this condition as a *miss*. In rest of the paper we will focus our attention on the h-MMA and h-SRAM, and we shall refer to them simply as MMA and SRAM.

Figure 3 shows the general MMA scheme: an arbiter (e.g., the switch fabric scheduler) issues a cell request every slot. This request is stored in the tail of a *lookahead shift register* of l positions. At every slot, one cell from the queue demanded by the head of the lookahead is read from the SRAM and granted to the arbiter. In order to guarantee that the requested cell is always in SRAM, every B slots a queue is selected by the MMA and a group of B cells of this queue are transferred from DRAM to SRAM.

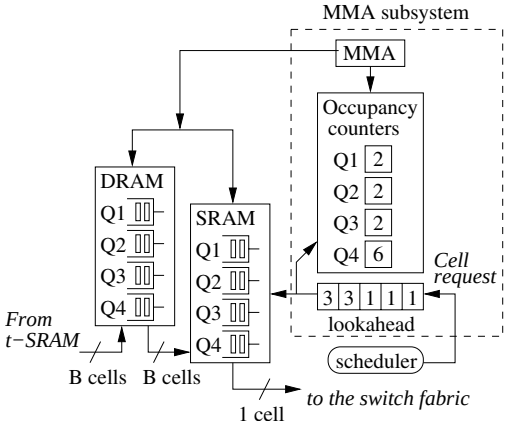

Figure 3. RADS Memory Architecture.

The decisions of the MMA take into account (i) the SRAM occupancy counters, and (ii) the arbiter requests stored in the lookahead. The lookahead allows the MMA to select the queue to be replenished as it knows the l requests that are going to be issued in the future. Armed with this information, the MMA can make better decisions. This means the SRAM size can be reduced while still guaranteeing zero miss probability.

An intuitive insight into why RADS works is as follows[1]: the worst-case scenario for RADS is where the scheduler requests goes through the queues (in the SRAM) in a round-robin manner removing one packet per queue and going to the next queue. The effect of this pattern is that all the SRAM queues will empty at about the same time (to be precise, one transfer time apart), putting the most pressure on the algorithm. Intuitively, because all the queues empty about the same time, RADS needs about $Q \times B - Q$ time slots to fill the Q queues before any one of them is completely empty. $Q \times B$ is the total number of cells, but by the time the first queue drains completely there is still one cell left in each of the other queues, hence $Q \times B - Q$, assuming perfectly synchronized hardware where as the last cell drains out of a queue, the next batch of B cells enters the queue.

For example, suppose that the parameters of the system shown in Figure 3 are: $Q = 4$, $B = 3$, $l = 5$. Suppose also that the MMA is called with the SRAM occupancy counters and the lookahead values shown in the figure. The MMA should select the queue 1. This queue would be replenished with 3 cells after 3 slots, and would remain with 2 cells after 5 slots. If the MMA had selected queue 3, a miss would occur for queue 1 after 5 slots.

In [13] various proposals for MMAs are studied. In this section we summarize the main dimensioning results for the Earliest Critical Queue First (ECQF) MMA, which minimizes SRAM size. The ECQF-MMA algorithm works as follows: read the lookahead from the head (slot 1) to the tail (slot l). For every request, read from the lookahead and decrease the occupancy counter of the corresponding queue. If this *modified* occupancy counter is less than zero, then the queue is said to be critical. The first queue found to be critical is the queue selected by the ECQF-MMA. The minimum SRAM size necessary to guarantee zero miss probability is $\text{SRAM}_{min} = Q\,(B-1)$ and the required lookahead is $l = Q\,(B-1)+1$. Note that this lookahead value guarantees that there is always at least one critical queue. Other MMAs reduce the required lookahead and in turn pay the cost by having to increase SRAM size. rads_sram_size$(l,\ q,\ b)$ is the required SRAM size that would be needed using the scheme described in this section as a function of the lookahead l, the number of queues q and the granularity b. We refer the reader to [13] for a solution of this formula.

As we can see, t-SRAM and h-SRAM sizes are roughly proportional to $Q \times B$ cells. Decreasing the value B would lead to smaller and hence faster SRAMs, leading to a VOQ design suitable for faster input line rates. Unfortunately, commodity DRAM random access times decrease at a relatively low pace (around 10% every 18 months). Therefore, in order to decrease the value B, we cannot rely on purely technological improvements. In Section 7 we perform an evaluation of RADS in order to assess its limitations.

4. Potential of bank interleaving

In response to the growing gap between processor and memory speed, DRAM manufacturers have created several new architectures that address the problem of latency, bandwidth and cycle time (e.g. DDR-SDRAM [12] or RAMBUS DRDRAM [5]). All these commercial DRAM solutions implement a number of memory banks–as many as 512–that can be addressed independently. RADS's SRAM size is proportional to the DRAM access time and does not scale for high packet rates. We exploit banking in DRAM to reduce the effective DRAM access time. The main advantage of having independent banks is that we can begin to access one bank while the other is still busy. In practical terms, this means we can potentially reduce the 'random' access time of a DRAM memory system by performing several on-the-fly requests to different banks. Reducing the effective DRAM access time allows us to reduce the SRAM size needed by RADS.

Figure 4 illustrates the concept of a memory bank and an interleaved memory system. A memory bank is a set of memory modules that are always accessed in parallel with the same memory address. The number of memory modules grouped in parallel is dictated by the size of the data element we want to address. This size in cells is the *data granularity*.

Figure 4 also shows a possible memory bank configuration (similar to that of a DRDRAM-like memory system [5]). In a conventional DRAM memory system, the data is interleaved across all memory banks using a specific pol-

[1]The worst case scenario described here applies for the ECQF-MMA explained later. For other MMAs the worst case pattern could be different.

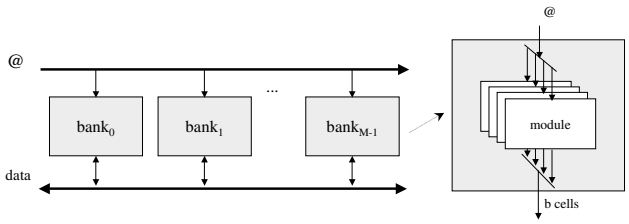

Figure 4. Organization of a DRAM in banks: configuration of a DRDRAM-style memory and internal structure of a memory bank.

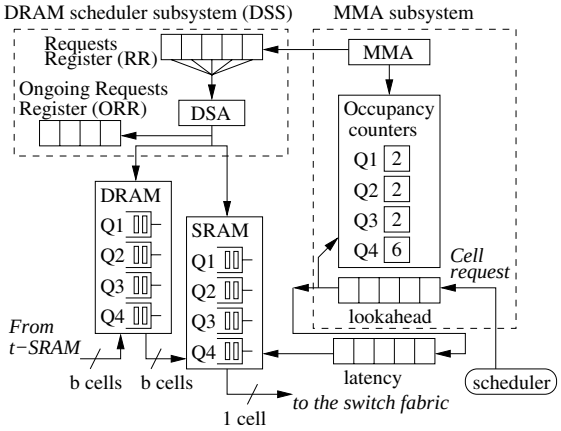

Figure 5. CFDS Memory Architecture of the packet buffer.

icy, and the memory controller is simply in charge of broadcasting the addresses to them. Each memory bank has a special logic that determines whether the address identifies a data item that the bank contains or not.

In the RADS scheme described in Section 3 the data granularity (B) was given by the DRAM random access of a single bank. Now, given an array of M memory banks and a random cycle time of T seconds per bank, it is theoretically possible to initiate a new memory access every T/M seconds. Therefore, the data granularity can be potentially reduced by a factor of M (as we can perform sequential accesses at an M times faster rate).

There are two fundamental limits to the bank interleaving exploitation. The first is the bus address speed, that is, the cycle time required to broadcast again an address to all memory banks. The second is the problem of bank collisions. In order to fully exploit the potential bandwidth of an interleaved memory system, we need to guarantee that the same bank is not accessed twice within its random access time (T).

The implementation of conflict-free mechanisms is especially relevant in the context of fast packet buffering, as we need to make sure that no bank collision is ever produced. This is because a collision would result in the loss of a packet. For instance, the RADS memory system cannot take advantage of banks, as there is no guarantee that a series of requests produced by the arbiter will not produce a conflict. Therefore, it is forced to rely on the worst-case scenario (the random access time of a single DRAM bank).

5. Conflict Free DRAM System (CFDS)

While the previous section shows the potential of banking, worst-case guarantees needed in packet buffer design requires a conflict-free banking scheme. In this section we describe a novel DRAM memory system that guarantees conflict-free access along with affordable cost. The system is based on a special memory bank organization coupled to a reordering mechanism that schedules the different MMA requests so as to guarantee no bank conflicts. Figure 5 summarizes the Conflict-Free DRAM System (CFDS) memory architecture. The following items are particular to CFDS: (i) CFDS exploits the DRAM bank organization. (ii) The *MMA Subsystem* works in exactly the same way as the MMA subsystem of the RADS memory architecture described in Section 3. It uses, however, a granularity of $b < B$ for cell transfers between DRAM and SRAM. (iii) The *DRAM Scheduler Subsystem* (DSS) hides the DRAM bank organization from the former MMA Subsystem. The MMA operates under the illusion that the DRAM access time is b time-slots, although in reality the DRAM access time remains B time slots (as in RADS). It is this illusion that reduces the SRAM size. Both DSS and MMA subsystems order the transfer of b cells from the same queue every b time-slots. However, transfers requested by DSS can take place in a different order than that requested by the MMA. Reordering these cells implies an additional cost in terms of latency and SRAM size. It can be shown, however, that introducing an additional delay and storage, an exact delivery of cells to the arbiter can be guaranteed: Thus, our banking can decrease the effective DRAM access time to allow smaller SRAM while guaranteeing worst-case bandwidth through conflict freedom. Moreover, the benefits of decreasing the granularity outweigh the additional cost introduced by the reordering process. These items are discussed in the following subsections.

5.1. DRAM Bank Organization

Let M be the number of DRAM banks. We organize these banks into $G = M/(B/b)$ groups of B/b banks per group (see Figure 6). Each group stores cells of Q/G queues. Banks are accessed by transferring b cells in the same queue. Furthermore, in order to avoid bank conflicts, the cells in each queue are stored in blocks of b cells following a round-robin configuration across all the banks of the group (block-cyclic interleaving). This way, we can perform B/b consecutive accesses to the same queue (transferring B cells overall) without bank conflicts. The distribution of the queues among the maximum number of groups maximizes

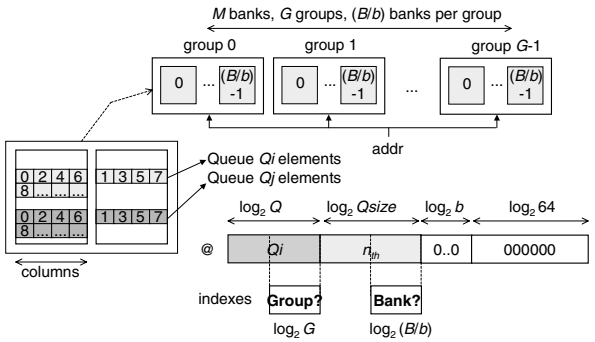

Figure 6. Proposed memory bank interleaving. The example on the left illustrates the block-cyclic interleaving using $B/b = 2$ banks per group.

the likelihood of finding independent accesses, reducing the hardware requirements of providing conflict-free access.

Figure 6 shows the mapping function used to obtain the bank and group indexes. The $log_2 b + log_2 64$ lowest-order bits of the memory address are always set to 0, since we want to address data with a granularity of b cells (i.e., $b \times 64$ bytes). The higher-order bits contain two different fields: one determines the queue identifier and another determines the relative order of the group of b cells inside that same queue. The bank group index we need to address is obtained using the low-order bits of the queue field while the bank index inside that group is obtained using the low order-bits of the ordinal field. The other bits are used to determine row and column addresses of the specified DRAM bank.

Table 1 summarizes the terms used to explain RADS and CFDS.

5.2. MMA Subsystem

The *MMA Subsystem* shown in Figure 5 works in the same way as the MMA subsystem of the RADS memory architecture described in Section 3, but it is assumed that $b < B$ cells are transferred every DRAM memory access. Every b slots the MMA decides whether the queue is to be replenished, issues the request to the *DRAM Scheduler Subsystem*, and increases the *occupancy counter* of this queue accordingly (by adding b). Every time a cell request leaves the lookahead register, the occupancy counter of the corresponding queue is decreased.

Note that in this scheme the occupancy counters of the MMA subsystem do not concur with the number of cells in the physical SRAM. This is because: (i) the requests leaving the lookahead still suffer an additional latency before being issued to the SRAM, and (ii) because the replenish requests issued by the MMA are delayed and possibly delivered in a different order by the *DRAM Scheduler Subsystem*.

Q:	Number of Virtual Output Queues.
B, b:	Granularity used in access to DRAM. B is the value used in Random Access DRAM System (RADS), while b is the value used in Conflict Free DRAM System (CFDS).
l:	Lookahead shift register size.
M:	Number of memory banks.
T:	Random Access Time to DRAM measured in seconds.
G:	Number of memory bank groups used in CFDS.
L:	Request Register (RR) size.
R_{max}:	Maximum number of times a request can be delayed by the DRAM Scheduler Algorithm (DSA).
Q_i^l, Q_i^p:	Logical and physical queue names used in the renaming scheme used in CFDS.

Table 1. RADS and CFDS legend.

5.3. DRAM Scheduler Subsystem

The *DRAM Scheduler Subsystem* (DSS) shown in Figure 5 manages the transfers between the DRAM and SRAM in order to fulfill the requests issued by the MMA. The DSS uses a *DRAM Scheduler Algorithm* (DSA) to avoid bank conflicts, using two registers: the *Requests Register* (RR) and the *Ongoing Requests Register* (ORR).

The RR is a shift register that stores the requests made by the MMA that have not been fulfilled yet. Every b slots, the DSA chooses a request of the RR, which can be located at any position of the register. Once a request has been chosen, it is removed from the RR and the requests from this position to the tail of the RR are shifted ahead, making room for the new request that will be issued by the MMA b slots later. The ORR is a shift register that stores the identifiers of the banks that are currently being accessed. Should a new request be issued to any of these banks, a bank conflict would arise. Hence, the banks with identifiers stored in the ORR are locked and the DSA never initiates a new transfer of the cells that reside in these locked banks. Taking into account that a bank is locked during B slots, we need to consider the latest $B/b - 1$ ongoing requests. The size of the ORR is hence $B/b - 1$.

The DSA chooses the *oldest request in the RR addressed to a bank which is not locked*, starting a new transfer of b cells and placing the memory bank identifiers at the tail of the ORR. It can be proved (see [8]) that the DSA can always find a non-locked request[2] provided that the RR has a size of:

$$L = (2\,Q/G - 1)\,(B/b - 1) + 1. \qquad (1)$$

An intuitive explanation of equation (1) is as follows: Because within one bank there are at most Q/G queues and because the next access to the same queue will go to the modulo next bank within the group, the maximum number of consecutive accesses to the same bank is roughly Q/G.

[2]Empty requests are considered as requests to a special queue.

Moreover, because each access takes B time slots to deliver b cells, at most roughly B/b requests can accumulate in one access. Therefore, the RR need to be as large as $(Q/G-1)(B/b-1)+1$ to guarantee conflict freedom.

Now we estimate the delay experienced by a request to go through the RR. A request in the RR has two kinds of delays: (i) the delay of $(L-1)\,b$ slots to go from the tail of RR to the head, assuming the DSA empties the RR in a FIFO manner, and (ii) the delay due to the DSA skipping over requests. The maximum number of times a request can be skipped over is:

$$R_{max} = (2\,Q/G-1)\,(B/b-1), \tag{2}$$

and thus the maximum delay due to skipping over is given by $R_{max} \times b$.

The intuitive explanation for equation (2) is as follows: Because the maximum number of requests to any one bank is Q/G and because a bank takes B time slots to access, the DSA can skip over a request $(Q/G-1)\,(B/b-1)$ times.

Finally, in formulas (1) and (2) we use $2\,Q$ instead of Q because the DRAM Scheduler Subsystem manages both reads and writes to Q queues in DRAM.

5.4. The Latency Register

In the proposed conflict-free access mechanism, the DRAM subsystem may deliver cells out of order. Therefore, we have to introduce a reordering mechanism which introduces an additional delay and causes the SRAM size to increase somewhat:

Firstly, an additional delay, equal to the maximum delay that a replenish request can suffer due to the DSA reordering, has to be added to the lookahead of the MMA. This is introduced by the *latency* shift register shown in Figure 5. From the previous section's discussions we can see that the size of this register must be equal to:

$$\begin{aligned}\text{latency (in slots)} &= b\,((L-1)+R_{max}) \\ &= 2\,b\,(2\,Q/G-1)\,(B/b-1).\end{aligned} \tag{3}$$

Finally, note that requests are delivered to the DRAM when they leave the RR register, but cells are removed from the SRAM when they leave the *latency* shift register. The mismatch between these two events requires an increase in SRAM size (in order to store the cells downloaded to the SRAM before they are granted to the arbiter). From that, we conclude that there are two factors that contribute towards SRAM dimensioning: (i) the size required by the *MMA Subsystem* given by rads_sram_size$(L,\,Q,\,b)$ (see Section 3), and (ii) the additional SRAM size required to cope with the mismatch described above. Summing both terms we have:

$$\begin{aligned}\text{SRAM size (cells)} = \text{rads\_sram\_size}(L,\,Q,\,b) + \\ b\,R_{max}.\end{aligned} \tag{4}$$

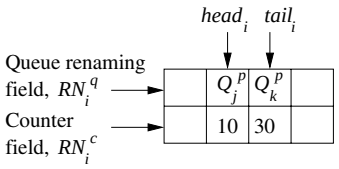

Figure 7. Circular "renaming register" used for the logical queue Q_i^l.

6. DRAM Fragmentation

The memory scheme previously described statically assigns queues to memory groups. This may prevent the full usage of the DRAM. For instance, if the DRAM size is S and we are using G groups, the Q/G queues assigned to a certain group can only use a S/Gth of the DRAM. Thus, if the other queues are empty, only a S/Gth of the DRAM would be used. We shall refer to this problem as *DRAM fragmentation*. In this section we explain a mechanism designed to cope with this problem.

We reduce fragmentation by renaming. We allow each *logical queue name* (Q_i^l) to be associated with more than one *physical queue name* (Q_j^p). By doing so cells from a given logical queue can reside in more than one memory group, and can potentially occupy the whole DRAM system. Q_i^ls are the names used by the scheduler to identify the VOQ logical queues, whereas Q_j^ps are the names used internally for identifying queues assigned to a certain group.

In order to perform this renaming process, we use Q circular *renaming registers* RN_i (one for every Q_i^l) to translate Q_i^l into Q_j^p. Each element of this register has two fields (see Figure 7): (i) a queue renaming field (RN_i^q) that stores the Q_j^p that will be used to access the DRAM and (ii) a counter (RN_i^c) with the number of cells of Q_i^l that have been stored in Q_j^p.

This register is initialized as follows. When b cells from a Q_i^l arriving to DRAM find the circular register RN_i empty (i.e., Q_i^l has no cells in DRAM), the first Q_j^p is chosen to store the cells of this queue (this value would be stored in RN_i^q and RN_i^c would be set to b cells). In order to balance DRAM occupancy, the assignment algorithm could select a Q_j^p from the group with the least cells. Every time more cells from Q_i^l are transferred to DRAM, the queue stored in RN_i^q is used and the counter RN_i^c is increased. If cells arriving to this queue find that the DRAM assigned to the group is full, a new Q_k^p would be chosen in another group that could offer free DRAM space. The value of Q_k^p would be stored in a new element added to the RN_i register. Now, the $tail_i$ index of this register would point to the last Q_k^p associated with Q_i^l, and the $head_i$ index would point the the first one. Since cells are stored in FIFO configuration, scheduler requests to the logical queue Q_i^l would be translated to the physical queue Q_j^p that $head_i$ is pointing to. Thus, each time a request for a Q_i^l is issued by the scheduler,

the element of the RN_i register pointed to by $head_i$ would be accessed. The content of RN_i^q would then be stored in the lookahead register and the RN_i^c counter would be decreased. If RN_i^c reached zero, $head_i$ would be advanced to the next position.

Each active logical queue needs to be associated with at least one physical queue. Therefore, if we want to guarantee that at least Q logical queues can be active at any given time, we need to oversubscribe the number of physical queues (P) to $P > Q$.

This mechanism allows the occupancy of the entire DRAM by any logical queue. However, there are situations in which memory fragmentation can still arise. For instance, fragmentation is possible if we have logical queues with cells scattered through many physical queues, so that all the physical queue identifiers are used, even though the overall DRAM occupancy is low. However, we believe that using a reasonable value of P, the probability of this DRAM fragmentation problem arising may be very low.

Finally, note that this renaming scheme is hidden to the rest of the memory management algorithm described in the previous sections. Those algorithms deal only with the physical queues. Thus, all previous results remain the same, although P is used instead of Q.

7. Evaluation of RADS

In this section we will perform an evaluation of the viability of the RADS memory architecture described in Section 3 for two different link rates, taking into account two restricting factors: area and access time. Throughout this section, we shall assume OC-768 (40 Gb/s) and OC-3072 (160 Gb/s) links for an input-buffer architecture. For the OC-768, we will assume $Q = 128$. This could correspond to 16 interfaces with 8 service classes. For OC-3072 we have assumed a more aggressive design with $Q = 512$. On the other hand, taking into account the link rates of OC-768 and OC-3072, we will set the RADS data granularity (B) to 8 for the former system and 32 for the latter (assuming 48 ns of main DRAM random access time).

7.1. Design of SRAM buffers

In order to show how significant the benefits of our proposed CFDS system are, we would like to evaluate whether a RADS SRAM buffer will face technologic hurdles in the near future. We have used CACTI 3.0 [19] to estimate the access time (in ns) and the area (in cm^2) of different implementations of the t-SRAM and h-SRAM buffers using a 0.13 μm technological process. CACTI is an integrated cache access time, cycle time, area, aspect ratio, and power analytical model. The main advantage of CACTI is that, as all these models are integrated, tradeoffs between the different parameters are all based on the same assumptions and hence are mutually consistent.

We assume that the t-SRAM and h-SRAM are shared by all the queues, as the unified SRAM leads to smaller memories. However, the design of a unified (shared) SRAM buffer is not as trivial as the design of a distributed (isolated) SRAM buffer, where each queue has its own partition of the available memory.

The second kind of SRAM buffer could be easily implemented as a set of circular queues implemented with simple direct-mapped SRAM structures. On the other hand, in the shared SRAM buffer, we need a mechanism to identify where exactly the n_{th} element of a given queue q_i is placed. Intuitively, this seems analogous to the design of Q linked lists, where the next cell to be accessed by the scheduler is located at the head of the corresponding list, and the next cell to store coming from the DRAM is placed at the tail of the corresponding list.

In [8] we proposed several RAM organizations to handle the management of cells in a shared buffer. In this paper, we are going to describe and focus on two of the designs: the one targeted at minimum area and the one targeted at getting the shortest access time. The first would be the most suitable for moderately high link rates while the second would be required for very high link rates.

The design targeted at the shortest access time is the *global CAM*. The *global CAM* consists of a full content-addressable memory where all the cells are stored together. Each cell has a tag that identifies the queue related to that cell, and the relative order within the list of cells of that same queue. When the address (queue identifier and order) of the cell is sent, the CAM searches across all entries related to that cell. This implementation requires two ports (one for reading and another for writing cells). Note that we assume that the refreshes from the DRAM are serialized along B time slots at a rate of one cell per slot.

Our design targeted at minimum area investment is the *unified linked-list*. The *unified linked-list* proposal is a straightforward implementation of Q linked lists in a direct-mapped memory structure. Each entry of that direct-mapped SRAM contains one cell and a pointer to the next cell (another entry of the same structure). In order to be able to identify the head and the tail of each linked list, we have another direct-mapped structure that stores the head and tail pointers for each of the queues. This special structure requires an additional write-port to store the position of the new tail onto the pointer field of the old tail. This translates into a direct-mapped SRAM with one read port and two write ports. If we assume that we do not have time constraints, we can serialize the three accesses (that is, time-multiplexing), thus requiring just a single read/write port and substantially reducing the area required.

7.2. RADS SRAM buffer performance

Figure 8 shows the access time and area of the two different SRAM implementations for OC-768 and OC-3072, as a

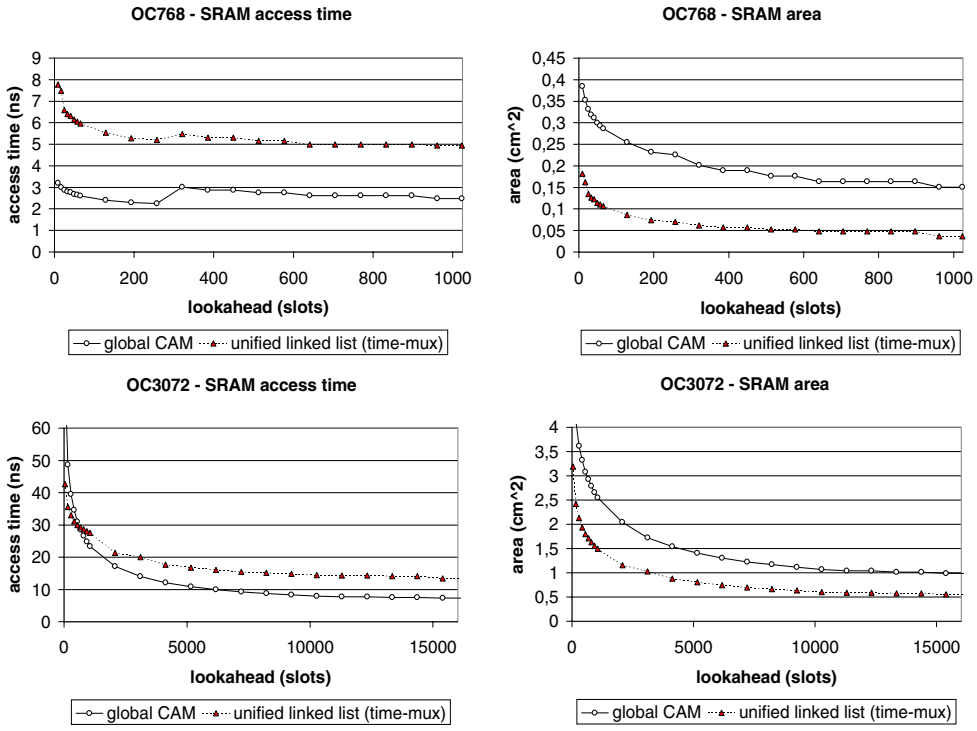

Figure 8. h-SRAM area and access time as a function of the lookahead for the RADS scheme (Q=128, B=8 for OC-768 and Q=512, B=32 for OC-3072).

function of the number of slots of the lookahead. Given a lookahead value, the SRAM size is obtained using the formulas given in [13]. In practice, it would be desirable to match the link-rate targets with the minimum look-ahead to minimize the average cell delay. The size of OC-768 system SRAM ranges from 300 kB (for minimum lookahead) to 64 kB (for maximum lookahead). The size of OC-3072 system SRAM ranges from 6.2 MB (for minimum lookahead) to 1.0 MB (for maximum lookahead).

For an OC-768 system, we need to access a new cell every 12.8 ns (assuming cells of a width of 64 bytes). We can observe from Figure 8 that the access times of both SRAM alternatives are far quicker, even for the shortest lookahead. Therefore, as access time is not a concern, the most suitable alternative is the small-area design (the *time-mux unified linked list*), which matches OC-768 time requirements with a modest investment in silicon (0.1 cm^2 even for the shortest look-ahead). In conclusion, RADS is an ideal way of providing fast packet buffering for OC-768.

For an OC-3072 system, we need to access a new cell every 3.2 ns, which is a significantly harder constraint to meet, taking into account that the SRAM buffers are now larger. Indeed, Figure 8 clearly shows that none of the SRAM implementation is able to comply with the 3.2 ns target (not even for the longest lookaheads), including the fastest one: *global CAM*. Furthermore, the area results show that only time-multiplexed, direct-mapped SRAM have an area smaller than 1 cm^2, which is already a significant fraction of the overall transistor budget of a high-end system (as we need both h-SRAM and t-SRAM buffers).

From the results, we can clearly see that RADS do not scale well beyond OC-3072 and suffer from severe implementation hurdles. Therefore, techniques focused on reducing both the area and the access time of the SRAM buffers are of the highest interest.

8. Evaluation of CFDS

In this section we will perform an extensive evaluation of various implementation issues that concern the design of the CFDS memory architecture described in Section 5. We will discuss the design of the request register scheduling logic, as well as the required modifications for the SRAM buffers. Finally, we will compare the performance (in terms of area and access time) of the RADS and CFDS memory architectures.

8.1. Implementation issues of the DRAM Scheduler Subsystem

As already described in Section 5.3, the *Requests Register* (RR) is a special lookahead register for requests to the DRAM memory system. The main function of this register is to store the requests so the DSA can schedule them to

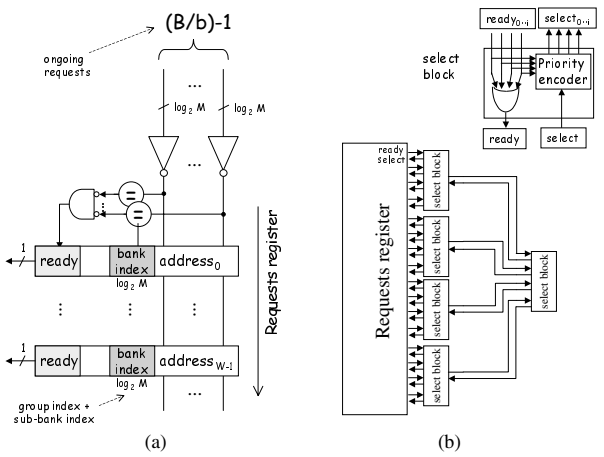

Figure 9. Implementation of the request register: (a) wake-up logic, (b) selection logic.

guarantee conflict-free access to the DRAM memory banks. The scheduling algorithm for selecting the request to be serviced is very simple. The *Ongoing Requests Register* (ORR) contains the banks that are currently accessing a block of b cells ($(B/b) - 1$ banks). The DSA scheduler is responsible for finding the oldest request from the RR that will access none of the banks contained in the ORR.

This logic problem is actually equivalent to the design of instruction issue windows in out-of-order superscalar processors [17, 11]. Figure 9 shows a possible design for the RR scheduling logic, based on a conventional issue window implementation. The logic is based on sending the tags from the ORR (that is, the $(B/b) - 1$ indexes of the banks being accessed) across all the entries of the Requests Register. Each entry is responsible for determining whether the bank index corresponding to the request is different from all the indexes coming from the ORR. If it is different, the entry is marked as ready. This stage (wake-up) is performed every time we want to select a new request candidate. After the wake-up stage, the logic needs to select the oldest entry that is in ready state (i.e. that can access the DRAM array with no bank conflicts). In order to do so, we can use a hierarchical selection logic that first propagates the ready signal across the tree, and then sends the selection signal back using priority encoders. This stage is known as the selection stage. Finally, once the request has been selected, a mechanism to perform compaction of requests is required in order to maintain the ordering of the requests by age.

Table 2 shows the RR size for different values of b and the time available to perform the scheduling of a single request. In order to assess the technological hurdles of these implementations, it will be useful to study a current commercial processor: the *Alpha* 21264. This processor implements, with a 0.35 CMOS process, a 20-entry issue queue able to select up to four instructions in 1 ns approximately [14], using 0.05 cm^2 of the overall die area. From this basic result, we can conclude that the implementation of the RR logic for OC-768 is fairly trivial, since even for $b = 1$ we have 12.8 ns to search in an RR of a length of 64. For OC-3072, the design is attainable for values of b higher than 2, and possible (yet aggressive) for $b = 2$. The implementation of the RR scheduling logic for OC-3072 and $b = 1$ is certainly of difficult viability.

		$b = 32$	$b = 16$	$b = 8$	$b = 4$	$b = 2$	$b = 1$
OC-768	RR size	-	-	0	2	16	64
	Sched. time (ns)	-	-	-	51.2	25.6	12.8
OC-3092	RR size	0	8	64	256	1024	4096
	Sched. time (ns)	-	51.2	25.6	12.8	6.4	3.2

Table 2. Requests register size and time to perform the scheduling of a new request.

8.2. Design of CFDS SRAM buffers

In order to implement an SRAM buffer for any CFDS configuration, we have to tackle two main issues. Firstly, the cells coming from the DRAM memory system of any given queue may come out-of-order. Secondly, the SRAM must contain $b \times R_{max}$ additional entries to be able to hold elements before they are scheduled by the MMA. The first problem can be easily overcome by implementing some basic changes to our proposed SRAM structures in order to allow them to insert cells from a queue out of its natural order: (i) *global CAM*: The implementation of out-of-order writing operations is trivial in this configuration, since the order is already specified in the tag of each entry of the CAM which we use to find the correct element. (ii) *unified linked-list*: Out-of-order writing operations are complex within a linked-list. However, an easy solution is to implement $Q\,(B/b)$ linked-lists instead of Q, since B/b is the number of banks per group and two operations over the same bank are always performed in strict order.

8.3. Performance comparison

Figure 10 allows us to demonstrate the performance benefits of using CFDS instead of a basic RADS approach. The figure shows the area (of both h-SRAM and t-SRAM) and most restricting access time for OC-3072 as a function of the delay (measured in μ-seconds). The delay is the lookahead delay for RADS and the lookahead delay plus latency for CFDS. Again, the number of queues Q is 512. The curves with a data granularity $b = 32$ correspond to the RADS implementation. The rest of the curves correspond to different CDFS configurations with varying values of b. We assume the number of banks M to be 256.

There are two main conclusions that can be inferred from the results in Figure 10. Firstly, one can observe the evident advantages of CFDS over RADS. A CFDS system with $b = 4$ is compliant with the requirements of buffering packets at 160 Gb/s, as *the access time is lower than*

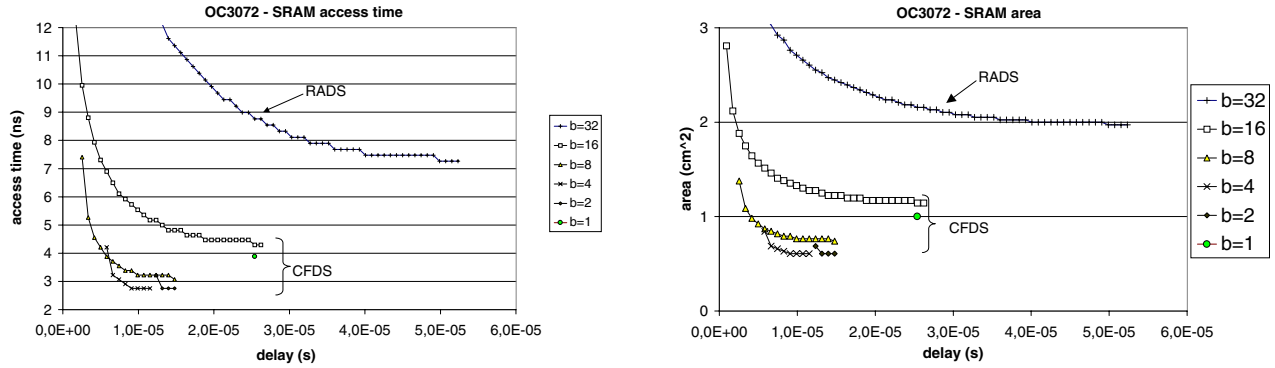

Figure 10. SRAM area (both h-SRAM and t-SRAM) and access time, as a function of the delay, for the RADS scheme and several CFDS configurations.

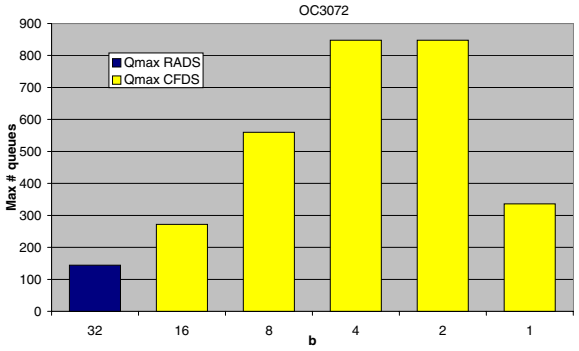

Figure 11. Maximum number of queues attainable for RADS and CFDS with different configurations (using maximum lookahead and complying OC-3072 time constraints).

3.2 ns. Moreover, this is accomplished with a modest lookahead delay (10 μs) and an affordable area (0.6 cm^2 overall). This contrasts heavily with its RADS counterpart, which is hardly able to access data in 7 ns, even with a delay of more than 50 μs. Another relevant issue is that the RADS system requires an area of 2 cm^2.

The second important conclusion is that there is an optimal value of b for any given CFDS implementation. This is due to the trade-off between the SRAM size required to tolerate the unpredictability of arrivals from the arbiter, which is proportional to b (see Section 3), and the SRAM size required to absorb the level of reordering of the accesses from the DRAM, which is proportional to $1/b$ (see equation (2)).

8.4. Potential number of queues

An important parameter is the number of queues that can be supported by our system [15]. Figure 11 shows the maximum number of queues that the different SRAM buffer approaches can afford, taking into account the access time constraints (for OC-3072, less than 3.2 ns). The first column bar (where $B = b$) corresponds to a RADS implementation, while the rest of the columns correspond to CFDS implementations as we reduce the data granularity b. As shown in the figure, CFDS allows 6 times more queues for OC-3072 (up to 850 queues). Note however, that this amount represents the number of physical queues available. The number of real queues is slightly lower taking into account the renaming process used to avoid the problem of fragmentation.

9. Related Work

Virtual Output Queuing was proposed for the first time in [20] (with the name of "dynamically-allocated multi-queue buffers"). The amount of buffering and the line rates considered in this seminal paper were far lower than those required for our target application: high-speed backbone routers. For OC192 (10 Gb/s) line rates, a time-slot is lower than the random access time of DRAM. [16] proposes a design using DRAM only for a VOQ buffer architecture working at this line rate. The proposed design uses out-of-order memory access in order to reduce the number of bank conflicts, although it does not guarantee zero miss loses.

[10] proposes techniques that exploit row locality whenever possible in order to enhance average-case DRAM bandwidth. However, this scheme does not guarantee zero miss probability, while the scheme proposed in this paper does.

For faster line rates, a combined SRAM-DRAM implementation of a VOQ buffer using ECQF for the h-MMA, is discussed in [13]. It assumes random access time to DRAM, which corresponds to what we call RADS.

There are many proposals dealing with mechanisms designed to alleviate the problem of memory conflicts [18] or to provide conflict-free access [23] This is especially true in the vector processor domain. The novelty of our technique resides in the application of these mechanisms to the context of fast packet buffering. In packet buffering, no bank collision can be produced as packets have to be guaranteed

within a bounded delay.

10. Conclusions

In this paper we have analyzed a novel architecture targeted at fast packet buffering. In order to overcome the bandwidth problems of current commodity DRAM memory systems, the use of SRAM modules coupled to DRAM modules have been proposed in the past. SRAM memories act as ingress and egress caches to allow wide transfers between the buffering system and its main DRAM memory system. The main drawback of this organization is that the data granularity of the DRAM accesses has to be enlarged in order to sidestep the high cycle times of DRAM. As a result, the SRAM memories are too large and slow for very high link rates.

We make the key observation that we can reduce the effective DRAM access time by overlapping multiple accesses to different banks, allowing us to reduce the SRAM size. The key challenge is that to keep the worst-case bandwidth guarantees we need to guarantee that there are no bank conflicts while the accesses are in flight. We guarantee bank conflicts by reordering the DRAM requests using a modern issue-queue-like mechanism. Because our design may lead to fragmentation of memory across packet buffer queues, we propose to share the DRAM space among multiple queues by renaming the queue slots.

We carry out an analysis that shows that the reordering introduced by the access scheme is bounded and that zero miss conditions can be guaranteed. Moreover, a technological study of the system implementation shows that our organization gives better results for area, access times, delay, and maximum number of queues than the previously proposed designs. For instance, OC-3072 line rates (160 Gb/s) require an access time $\leq$ 3.2 ns. This constraint is fulfilled by our proposed mechanism with a delay of 10 μs, while the baseline counterpart system would require an access time $\gtrapprox$ 7 ns with a delay of more than 50 μs.

To the best of our knowledge, the design proposed in this paper is the fastest buffer design using commodity DRAM to be published to date.

11. Acknowledgments

This work was supported by the Ministry of Education of Spain under grants TIC-2001-0956-C04-01 and TIC98-05110C02-01 and by a grant from IBM. We would like to thanks T.N. Vijaykumar for his comprehensive comments and help to improve the readability and quality of the paper. We would like to extend the acknowledgment to the anonymous reviewers for their insightful comments.

References

[1] Intel IXP2400 Network Processor.

[2] Power NP4GX network processor.

[3] V. Bollapragada, R. White, and C. Murphy. *Inside Cisco IOS Software Architecture*. Cisco Press, July 2000.

[4] J. Corbal, R. Espasa, and M. Valero. Command-Vector Memory System. In *IEEE Parallel Architectures and Compilation Techniques, PACT'98*, Paris, 1998.

[5] R. Crisp. Direct Rambus technology: The new main memory standard. *IEEE Micro*, 7:18–28, Nov/Dec 1997.

[6] W. Eatherton. Router/Switch Architecture with Networking Specific Memories. In *MemCon 2002.*, USA, 2002.

[7] Fujitsu. 256M bit Double Data Rate FCRAM, MB81N26847B/261647B-50/-55/-60 data sheet.

[8] J. García, L. Cerdà, and J. Corbal. A Conflict-Free Memory Banking Architecture for Fast Packet Buffers. Technical Report UPC-DAC-2002-50, UPC, July 2002.

[9] G. Glykopoulos. Design and Implementation of a 1.2 Gbit/s ATM Cell Buffer using a Synchronous DRAM chip. Technical Report 221, ICS-FORTH, July 1998.

[10] J. Hasan, S. Chandra, and T. Vijaykumar. Efficient Use of Memory Bandwidth to Improve Network Processor Throughput. In *ISCA 2003*, USA, June 2003.

[11] D. Henry, B. Kuszmaul, G. Loh, and R. Sami. Circuits for Wide-Window Superscalar Processors. *International Symposium on Computer Architecture, ISCA'00*, 2000.

[12] Hitachi. Hitachi 166 Mhz SDRAM. *Hitachi HM5257XXb series*, 2000.

[13] S. Iyer, R. Kompella, and N. McKeown. Designing Buffers for Router Line Cards. Technical Report TR02-HPNG-031001, Stanford University, Nov. 2002.

[14] R. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro*, pages 24–36, March-April 1999.

[15] C. Minkenberg, R. Luijten, W. Denzel, and M. Gusat. Current Issues in Packet Switch Design. In *Proc. HotNets-I*, Princeton, NJ, 2002.

[16] A. Nikologiannis and M. Katevenis. Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out of Order Execution Techniques. In *IEEE International Conference on Communications*, Helsinki, Finland, 2001.

[17] S. Palacharla, N. Jouppi, and J. Smith. Complexity-Effective Superscalar Processors. In *ISCA24*, 1997.

[18] B. Rau, M. Schlansker, and D. Yen. The Cydra 5 stride-insensitive Memory System. *International Conference on Parallel Processing*, pages 242–246, 1989.

[19] P. Shivakumar and N. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. *Technical report, Compaq Computer Corporation*, August 2001.

[20] Y. Tamir and G. Frazier. High-Performance Multi-Queue Buffers for VLSI Communication Switches. In *15th Annual International Symposium on Computer Architecture*, pages 343–354, Honolulu, Hawaii, 1988.

[21] I. technologies. RLDRAM. High density, high-bandwidth memory for networking applicacions.

[22] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, and K. Keutzer. *Network Processors Design: Issues and Practices*, chapter 7. Morgan Kaufmann Publishers, 1 edition, 2002.

[23] M. Valero, T. Lang, J. LLaberia, M. Peiron, E. Ayguade, and J. Navarro. Increasing the Number of Strides for Conflict-Free Vector Access. *ISCA-19*, May 1992.

[24] M. Valero, T. Lang, M. Peiron, and E. Ayguade. Increasing the Number of Conflict-Free Vector Access. *IEEE Transactions on Computers*, 44(5):634–646, May 1995.