Report Number: WUCS-01-26

2001-01-01

Synthesizable Design of a Multi-Module Memory Controller

Sarang Dharmapurikar and John W. Lockwood

Random Access Memory (RAM) is a common resources needed by networking hardware modules. Synchronous Dynamic RAM (SDRAM) provides a cost effective solution for such data storage. As the packet processing speeds in the hardware increase memory throughput can be a bottleneck to achieve overall high performance. Typically there are multiple hardware modules which perform different operations on the packet payload and hence all try to access the common packet buffer simultaneously. This gives rise to a need for a memory controller which arbitrates between the memory requests made by different modules and maximizes the memory throughput. This paper discusses... **Read complete abstract on page 2.**

Synthesizable Design of a Multi-Module Memory Controller

Sarang Dharmapurikar and John W. Lockwood

Complete Abstract:

Random Access Memory (RAM) is a common resources needed by networking hardware modules. Synchronous Dynamic RAM (SDRAM) provides a cost effective solution for such data storage. As the packet processing speeds in the hardware increase memory throughput can be a bottleneck to achieve overall high performance. Typically there are multiple hardware modules which perform different operations on the packet payload and hence all try to access the common packet buffer simultaneously. This gives rise to a need for a memory controller which arbitrates between the memory requests made by different modules and maximizes the memory throughput. This paper discusses the design and implementation of a SDRAM controller which satisfies both the requirements. The memory throughput depends on the burst lengths, the address pattern of the memory accesses and the type of memory access (read/write). Given the information about the current SDRAM access and the pending SDRAM access requests, the controller finds the memory access request among the pending requests which utilizes the data bus most efficiently and increases the throughput. This leads to the re-ordering of the memory requests between modules. Results show how this controller improves the overall throughput.

Synthesizable Design of a Multi-Module Memory Controller

Sarang Dharmapurikar and
John W. Lockwood

WUCS-01-26

October 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Synthesizable Design of a Multi-Module Memory Controller

Sarang Dharmapurikar
John W. Lockwood

WUCS-01-26

October 12, 2001

Department of Computer Science
Applied Research Lab
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130

Abstract

Random Access Memory (RAM) is a common resource needed by networking hardware modules. Synchronous Dynamic RAM (SDRAM) provides a cost effective solution for such data storage. As the packet processing speeds in the hardware increase memory throughput can be a bottleneck to achieve overall high performance. Typically there are multiple hardware modules which perform different operations on the packet payload and hence all try to access the common packet buffer simultaneously. This gives rise to a need for a memory controller which arbitrates between the memory requests made by different modules and maximizes the memory throughput. This paper discusses the design and implementation of a SDRAM controller which satisfies both the requirements. The memory throughput depends on the burst lengths, the address pattern of the memory accesses and the type of memory access (read/write). Given the information about the current SDRAM access and the pending SDRAM access requests, the controller finds the memory access request among the pending requests which utilizes the data bus most efficiently and increases the throughput. This leads to the re-ordering of the memory requests between modules. Results show how this controller improves the overall throughput.

Supported by: NSF ANI-0096052 and Xilinx Corp.

1 Introduction

The memory controller provides an interface between user modules and the SDRAM. It abstracts the memory by offering a simple model for the SDRAM. The controller maximizes the memory throughput.

Typically a memory access involves three phases:

- Initiation of the data access
- Data access
- Termination of the data access

The time spent in the actual data access determines the memory throughput. The more time spent in accessing data, the better the throughput. In case of two successive memory accesses, if the next operation waits till the current operation completes all the phases of the access then a latency equal to the time required for initiation and termination phases of the access is introduced. Thus for every access, some time is spent during which the memory data bus is not being used. This results in the reduction in memory throughput, especially when the burst lengths of the memory accesses are short. In order to improve the throughput, two accesses can be overlapped and the overhead for the initiation phase of the next access can be hidden. This is an example of a pipelined operation. Potentially there can be a contention for the SDRAM command bus access. Using an efficient command scheduling scheme, it is possible to interleave the commands and increase the overlap in the two memory operations thereby increasing the memory throughput.

The problem of memory throughput maximization is even more challenging when there are multiple user modules trying to access the SDRAM at the same time as shows in Figure 1. This type of scenerio is commonly seen in the platforms like Field Programmable Port Extender (FPX) [1] [2]. FPX is a platform for implementing the hardware modules in active networking. It loads the *hardware plugins* dynamically which operate on the packet payload stored in the off-chip SDRAM (Figure 2).

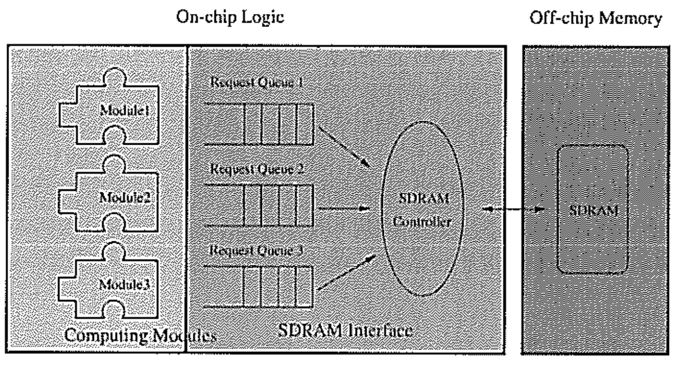

Figure 1: Multiple modules contending for SDRAM access

In this case, every module needs a fair access to the memory in comparison with the other modules. The order between the memory requests of an individual module has to be maintained. However at any time instance, there are multiple outstanding memory requests each of which corresponds to one of the modules and there is a choice between these requests for the next SDRAM access. This offers a good opportunity for maximizing the memory throughput since some of the pending memory requests are always favored by the ongoing memory access depending upon the

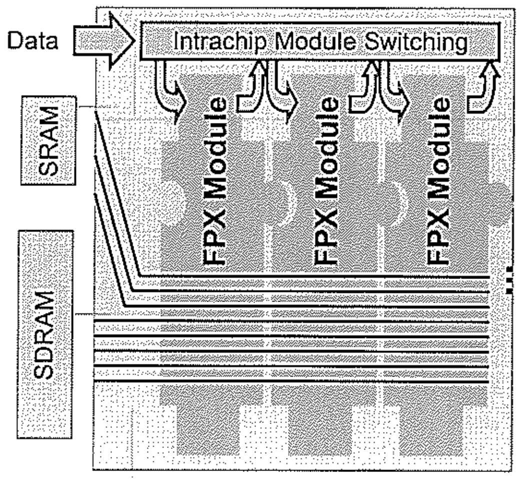

Figure 2: Hardware modules in the FPX

address pattern of the two requests and their operation type (read or write). Using this relationship between the address pattern and operation type of the pending operations and the address pattern and operation type of the ongoing operation, it is possible to determine which is the most suitable operation to be scheduled next. For instance, if the next memory access is on the same chipset, same bank and same row as the current access, then the current access doesn't need to enter termination phase and the next access doesn't need to enter the initiation phase and thus next access can execute immediately without delay. If the next access is on a different bank than the one current access is operating on then it is possible to overlap the initiation phase of the next access and the termination phase of the current access and reduce the latency in next access. Same is true if the access are on different chipsets on the memory module.

Thus, the memory access requests from different modules can be reordered in such a way that the time spent in the actual data access is maximized and the time spent in the initiation and termination phase is minimized. The proposed controller design exploits this fact and extracts the maximum throughput from the SDRAM.

2 SDRAM characteristics

SDRAM is partitioned in a certain number of memory banks. Within each bank, memory is arranged in rows and columns. Each intersection of a row and a column can be considered as a memory element. The size of the memory element is the size of the data bus. In this paper we explain the SDRAM controller design ideas and implementation with respect to the SODIMM MT8LSDT864 SDRAM memory module[3]. In this memory module a single chip has a data bus width of 16 bits [3]. Four chips are arranged in parallel on one side of the module to give a memory element size of 64 bits. We refer to this set of four chips as a *chipset* . These four chips share the same select pin, address bus and the bank address bus. Thus a row address and a column address correspond to a memory element which is eight bytes wide. Another set of four chips is multiplexed with this set of four chips so that the data bus,the address bus and the command bus is shared among these two sets of four chips and memory size is doubled. Each set of four chips share the same select pin. The commands on the command bus are applicable only to the set which has been selected using the select pin. Figure 3 shows the logical arrangement of memory in MT8LSDT864

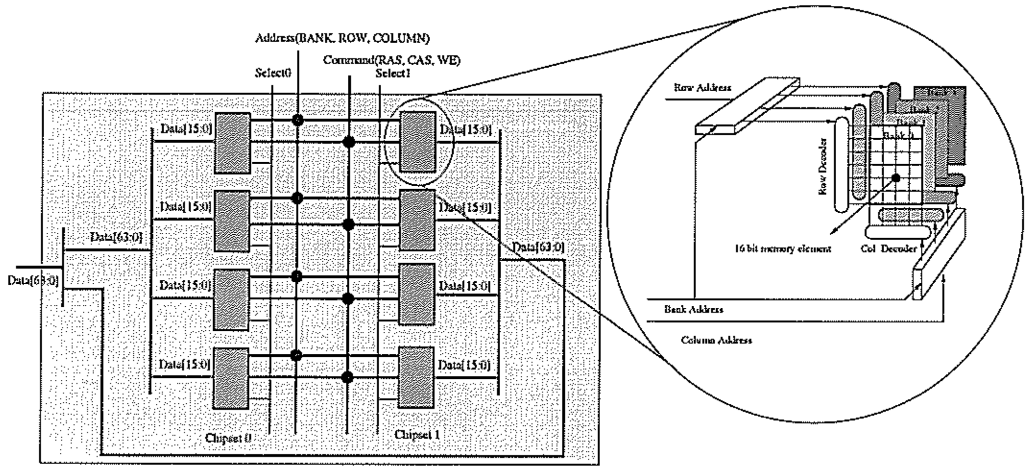

Figure 3: Logical arrangement of memory in MT8LSDT864 SDRAM memory module

SDRAM memory module. The memory module can be programmed to handle a burst length of 2, 4, 8 or a full page burst which is 256 words wide (each word is 8 bytes). To be able to do a burst access of arbitrary length, the burst length is programmed to the full page and the burst access is terminated using the BURST STOP command at appropriate time to get the required burst length.

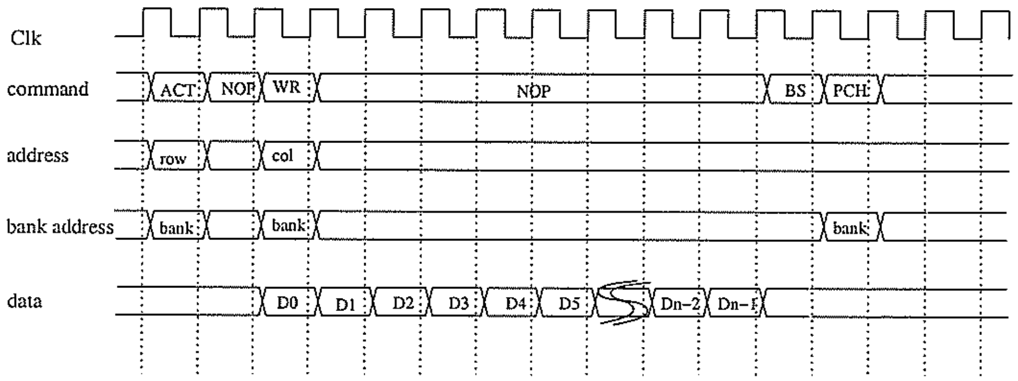

Figure 4: Write operation of burst length 'n'

Figure 4 shows the timing diagram for the write operation and figure 5 shows the timing diagram of the read operation. In case of the SDRAM, ACTIVE and READ/WRITE commands constitute the initiation phase of the access and BURST STOP and PRECHARGE constitute the termination phase of the access.

3 Using the SDRAM controller

Figure 6 shows the module interface for the SDRAM. This interface has two types of signals : one which participate in the arbitration for the SDRAM access and other which actually access

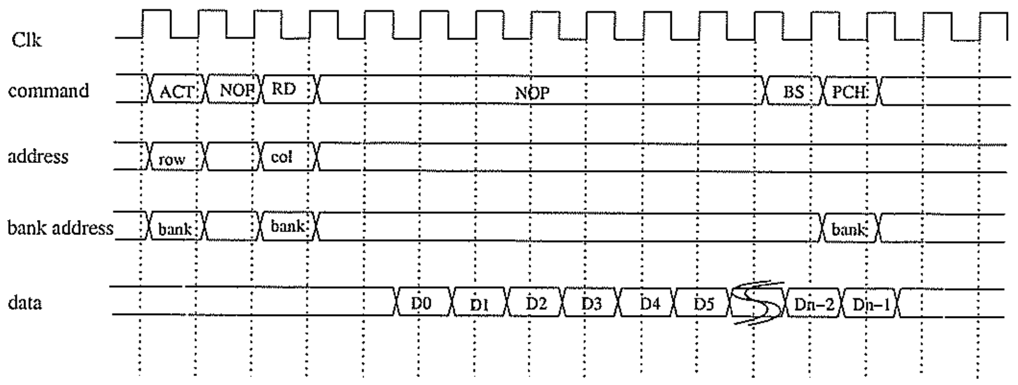

Figure 5: Read operation of burst length ‘n’

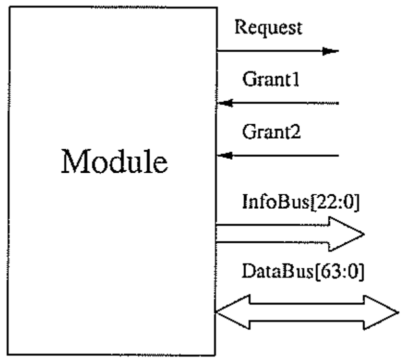

Figure 6: Module Interface for the SDRAM access

the data. The signals Request, Grant1 and Grant2 (as shown in the figure) are used for the arbitration while others are used to access the data. When a module wants to perform a SDRAM operation, it requests the access by asserting the Request signal high. A module can make multiple requests for multiple SDRAM operations by holding the Request signal high for multiple clock cycles. Each request corresponds to one SDRAM operation. After making the request(s), module waits for the Grant1 signal. When the Grant1 signal is asserted high, the module gives all the information regarding the transaction to the SDRAM controller on the InfoBus. This information includes the address, the operation type and the burst length. The SDRAM controller analyzes this information and schedules the actual SDRAM access for this module. At the scheduled time, the SDRAM controller asserts Grant2 high which means that the request (first request in the queue in case of multiple requests) has been granted access to the SDRAM and module can read the data from the DataBus in case of read request or put data on the DataBus in case of a write request. All the requests made my a module are serviced sequentially which means if information regarding request A was sent before the information regarding the request B then A will be serviced before B.

Figure 8 shows the exact protocol and timing for a burst read operation of burst length n and Figure 9 shows the timing of a burst write operation of length n . The discontinuation sign on the signals indicates that there is a variable clock cycles of latency between the Request and Grant1 as

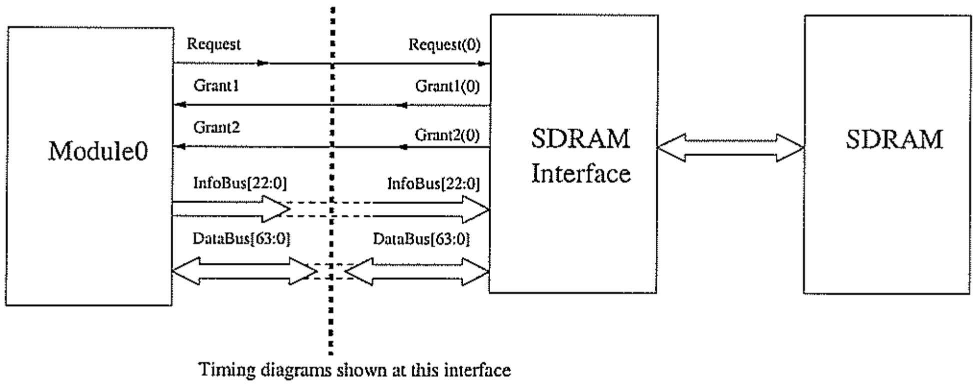

Figure 7: Connecting a module to the SDRAM Controller

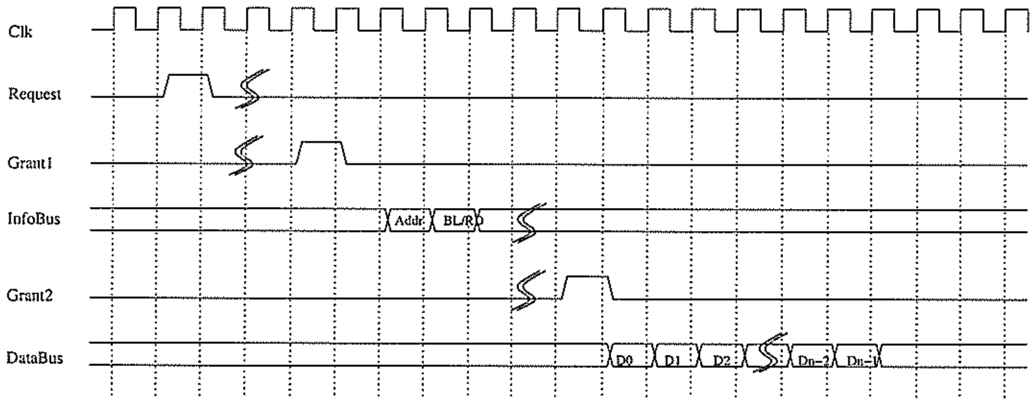

Figure 8: Timing diagram for a burst read operation

well as Grant1 and Grant2.

Figure 7 shows how the module is connected to the SDRAM controller. Request signal of module 'x' is connected to the Request(x) of the SDRAM controller. Similarly Grant1 signal of module is connected to Grant1(x) and Grant2 to Grant2(x) of the SDRAM controller.

Following list details the interface signals and their purpose.

- Request : This signal is used to make a request to the SDRAM controller for SDRAM access. Each request corresponds to one SDRAM transaction. The number of requests made is equal to the number of clock cycles for which the Request is asserted high. Every request gets corresponding grant. *The number of outstanding requests which haven't got the grant can not exceed 3* . Thus, if a module asserts the request signal high for 3 clock cycles then it means it has made 3 requests and if the first request gets a grant then the module can make only one more request.

- Grant1 : This signal is the input to the module from the SDRAM controller. When this signal is asserted high, it means that the module has been given grant to access the InfoBus(described below), onwhich it can put the information regarding *only one* SDRAM transaction it wants to do. This information includes the SDRAM address, the type of operation (read or write) and the burst length of this operation. First the 23 bit address should appear on the InfoBus at the time shown in the Figure 9 and Figure 8 then burst length and operation type should

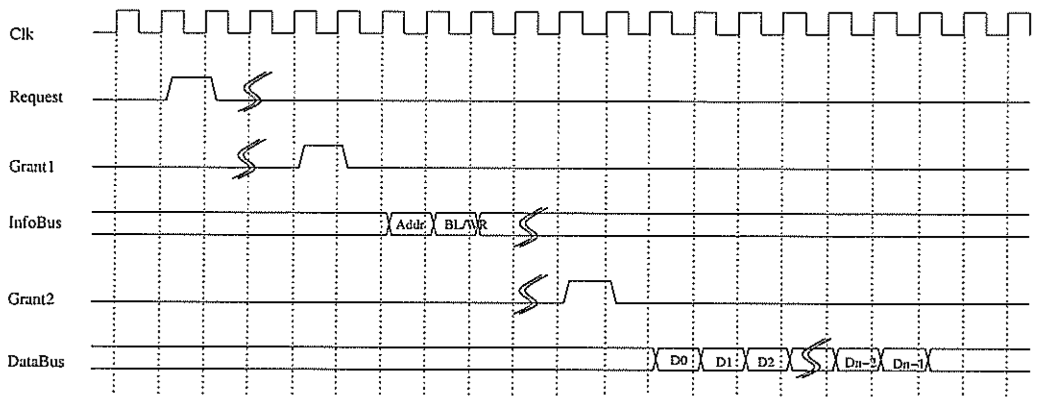

Figure 9: Timing diagram for a burst write operation

appear together on the bus in the next clock cycle.*The number of clock cycles between the Request and Grant1 is variable.*

- InfoBus[22:0] : This is a unidirectional tristate bus which is used to supply information regarding a SDRAM transaction to the SDRAM controller. After Grant1 is given to the module, it should put the 23 bit address on the InfoBus such that InfoBus[22:0] = Address[22:0]. In the next clock cycle it should put the 8 bit burst length and the operation type on the InfoBus such that InfoBus(8) = OperationType and InfoBus[7:0] = BurstLength[7:0]. The convention for the operation type is OperationType = '0' for a *read* operation and '1' for a *write* operation. *Except for these two clock cycles, the module should always drive HiZ on the InfoBus.*

- Grant2 : This signal is a input to the module from the SDRAM controller. For a read operation as shown in Figure 8, when Grant2 signal is asserted high the module should start taking in the data from the next clock cycle. This data corresponds to the read request which was at the head of the queue. Since all the requests made by the module are serviced in the same order in which they appear the module should keep track of the which data corresponds to which request. For a write operation as shown in Figure 9, when Grant2 signal is asserted high, then module should start putting data words on the data bus two clock cycles thereafter.

- DataBus[63:0] : This is a bidirectional tri-state data bus. The module should drive HiZ when it is not used by the module.

4 SDRAM controller architecture

The SDRAM controller is split up in the following submodules

- Request Collector
- Request Selector
- Request Executer

Following is the description of each of these modules.

4.1 Request Collector:

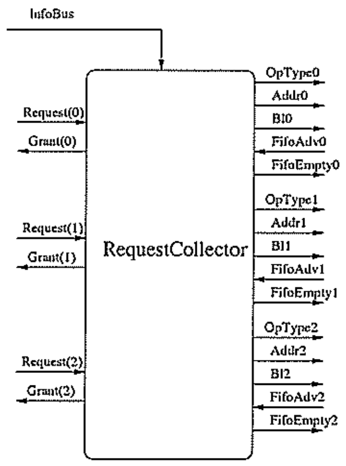

Figure 10: Request Collector Entity

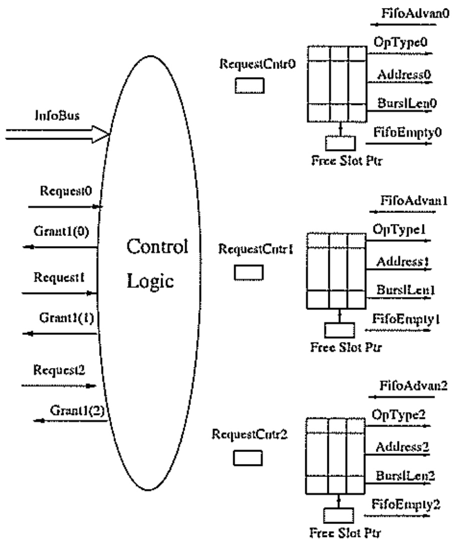

Figure 11: Request Collector Architecture

Figure 10 shows the entity of the Request Collector logic and Figure 11 shows the architecture. Request collector accepts the requests from different user modules, maintains their count, gives grants (Grant1) to the modules for supplying the memory transaction information and maintains the requests FIFO queues for the each module. The request signal coming from the module to the SDRAM controller is a point to point connection. Similarly the Grant1 signal is a point to point connection. The Request Collector counts the number of clock cycles for which the Request signal was asserted high and stores this count as the request count. One request counter is maintained for each module. When Request Collector gives Grant1 to one of the modules, the module puts the address and the burst length and the operation type on the bus called InfoBus. InfoBus is a tri-state unidirectional bus shared among all the modules. This information regarding the transaction the module wants to do is supplied in 2 clock cycles. In the first clock cycle the address is supplied and in the second clock cycle the burst length and the operation type (read or write) is supplied. The address received in the first clock cycle is stored in a temporary register and after the second clock cycle, the complete information is stored in the next free slot of a FIFO. One FIFO for each user module is maintained. The slot on the FIFO contains three fields: address, burst length and the operation type. Thus the requests from each module are queued up in independent FIFOs. The free slot pointer increments after a request has been added in the FIFO and decrements when a request has been dequeued from the FIFO.

Request Collector gives the Grant1 to the modules in round-robin order. Thus, a Grant1 is given to a module which has

- Non-zero outstanding requests in the request counter
- Empty slot in the request FIFO
- and the module is next in the round-robin order

The delay between the request and the Grant1 is variable. The requests at the head of the FIFOs contend for the next SDRAM access. When one of the requests at the head of the FIFOs is accepted for next transaction, it is removed from the FIFO and the FIFO advances by a slot. This FIFO advance signal is given as an input to the Request Collector by Request Executer. It is possible that a request is being added at the same time when the FIFO is advancing. This boundary condition has been handled. When the request FIFO is empty, the Request Collector conveys this to the next module through a FIFO empty signal. The empty FIFO module is dropped from the arbitration.

4.2 Request Selector:

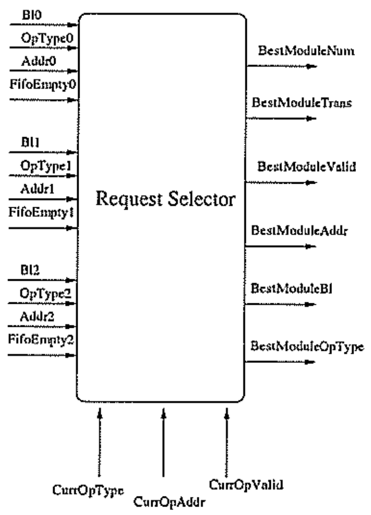

Figure 12: Request Selector Entity

Figure 12 shows the entity of the Request Selector entity and Figure 13 shows its architecture. Request selector logic takes the pending requests(one from each module) and selects a request among them which if executed at the appropriate time maximizes the SDRAM data bus utilization. The Request Selector makes use of a greedy method to find the best request among the pending

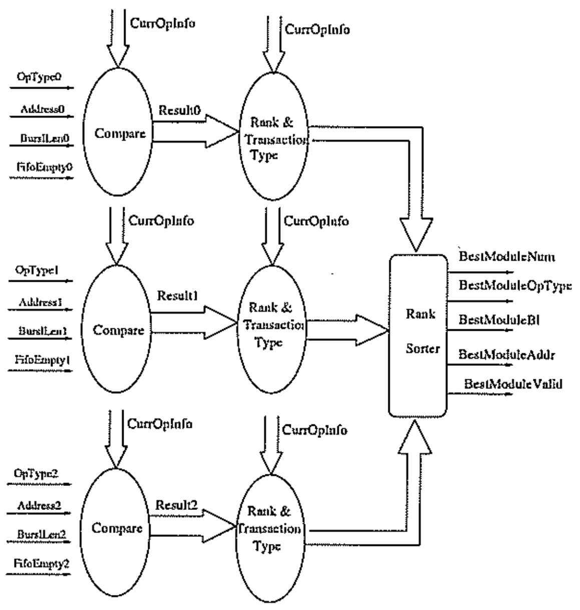

Figure 13: Request Selector Architecture

requests. It compares pending requests against the current memory operation and classifies the pending requests in one of the memory transaction types given in the table 1. The address of the pending request and the operation type (read or write) is compared against the address and the operation type of the current access. The result of the comparison is used as an index to a lookup table which stores the transaction type that a memory request represents. All the possible transactions are shown in table 1. Assuming that each memory request has a burst length of eight, the memory transactions in table 1 can be executed in the way shown in figure 14. This figure shows how the SDRAM commands corresponding to two successive memory operations can be arranged in order to increase the overlap between the initiation phase of one operation and termination phase of the next operation. The timings are taken with reference to [3] where CAS latency is programmed to 2.

Each transaction type in turn has a rank associated with it which tells how good that transaction is. As shown in figure 14 the number of idle clock cycles between two successive data accesses depends on the transaction type being executed. The fewer the number of idle clock cycles, the better is the throughput. Hence the rank of the transaction type can be set to represent the number of idle clock cycles in it. The lower is the rank of a pending request the better it is for starting next. The ranks are passed through a sorting circuit which gives the minimum rank and the corresponding module number. Once the module number which represents the best pending request is known, its address, burst length and the operation type is retrieved and this information is collectively passed to the next block. Every clock cycle, the combinational circuits finds the best pending request and its corresponding information out and latch it in the flipflops and supply it for further processing. The module which has an empty FIFO(i.e. invalid request) is dropped from all these computations.

4.3 Request Executer:

Figure 15 shows the entity of the Request Executer and Figure 16 shows its architecture.

Transaction	Type name
Read after a read on the same chipset, same bank, same row	RR1
Read after a read on the same chipset, same bank, different row	RR2
Read after a read on the same chipset, different bank	RR3
Read after a read on the different chipset	RR4
Write after a read on the same chipset, same bank, same row	RW1
Write after a read on the same chipset, same bank, different row	RW2
Write after a read on the same chipset, different bank	RW3
Write after a read on the different chipset	RW4
Write after a write on the same chipset, same bank, same row	WW1
Write after a write on the same chipset, same bank, different row	WW2
Write after a write on the same chipset, different bank	WW3
Write after a write on the different chipset	WW4
Read after a write on the same chipset, same bank, same row	WR1
Read after a write on the same chipset, same bank, different row	WR2
Read after a write on the same chipset, different bank	WR3
Read after a write on the different chipset	WR4

Table 1: Possible memory transactions

Request executer logic executes the selected request at the appropriate time. The information about the best available request is given by the request selector. The request executer operates on this information and issues commands to the SDRAM at appropriate clock cycles. First, the request executer calculates the scheduled start time of the memory request provided by request selector with reference to a global timer for the entire SDRAM controller. At the scheduled start time, the request executer starts executing the commands required for the selected request in a state-machine fashion. Since the basic scheme allows two requests to have overlapping operations (termination of one request and the initiation of the next request) two state machines are required which control the SDRAM interface. Hence the request executer runs two identical state machines. The state-machines have transaction diagram as shown in figure 17. The state-machines are used in a ping-pong fashion. When one state-machine has finished executing the commands of a request then the new request is dispatched to this state-machine for execution. Hence, whenever there are multiple pending operations, the state-machines are used alternately.

The state-machine goes in the NOP state by default. When triggered at the scheduled start time, the state-machine goes from NOP state to the ACT state. If the memory request needs to activate the row it wants to operate on then an ACTIVE command is issued, otherwise the state-machine stays in ACT without issuing ACTIVE. Note that the next request doesn't need to activate a row if the current request has already activated it. After two clock cycles the state-machine jumps to RD or WR state depending on whether the operation is read or a write. In these states the corresponding command (READ or WRITE) is issued to the SDRAM. The state-machine stays in this state for the clock cycles equal to the burst length of the memory access and then it jumps to the BS state. In the BS state, if the BURST STOP command is required then it is issued otherwise the state-machine stays in the BS state without issuing BURST STOP command. Note that the burst operation needs to be terminated only if the next the memory operation is of a different type or doesn't operate on the same row in the same bank. From BS the state machine jumps to the PCH state for precharging the bank. Note that the PCH command may or may not be required

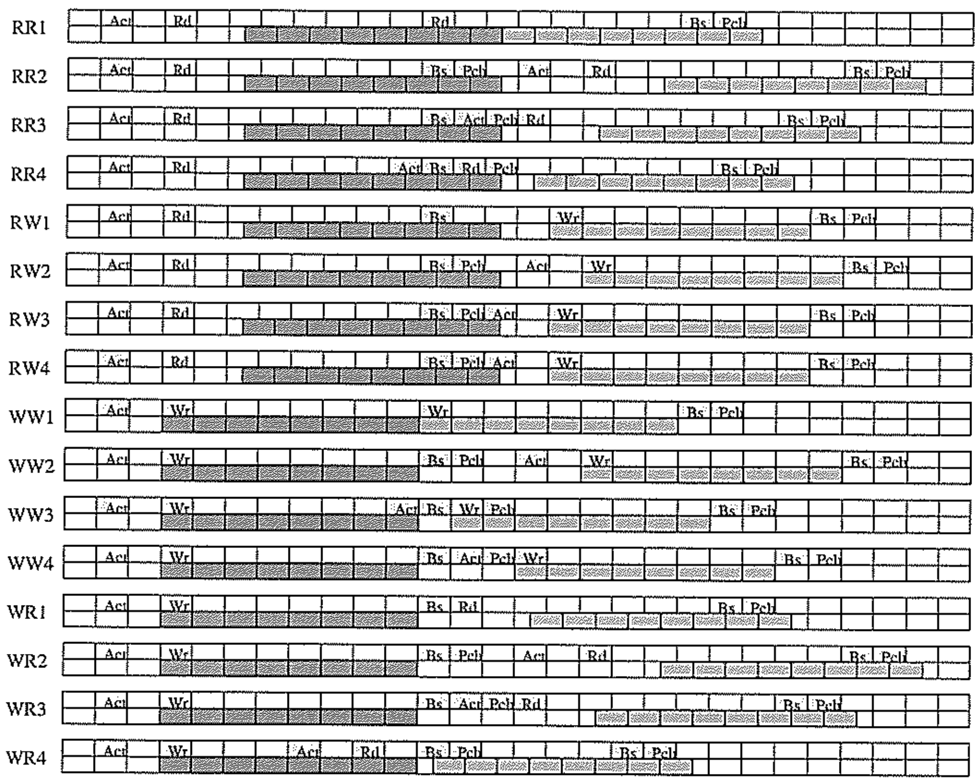

Figure 14: Execution of different type of memory transactions

for a transaction. For example, when the next access is on the same chipset, same bank and same row, there is no need of precharging the bank. On the other hand, in some cases the memory operation delays the precharging by a clock cycle than its prescheduled time in order to allow the next memory operation to apply its ACTIVE command.

From the above discussion, it is clear that the state-machine requires the following inputs

- Scheduled start time
- Operation type (read or write)
- ACT valid indication
- BS valid indication
- PCH valid indication
- Postpone PCH by one clock cycle indication
- Burst length = 1 indication

The context mentioned above is maintained for each state machine. Thus, there is context0 and context1 for state-machine0 and state-machine1 respectively each containing the above mentioned fields. In the beginning state-machine 0 is used for the first memory operation then state-machine1 is used for the second transaction and again state-machine0 is used for the third transaction and so on. This is possible since the basic scheme allows only two memory transactions to overlap at a

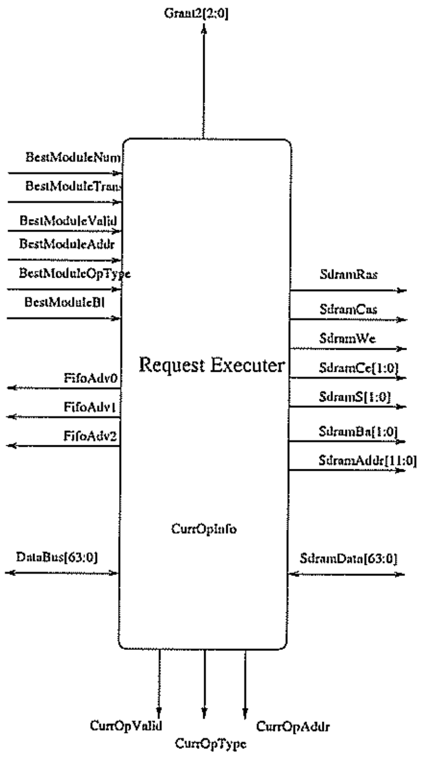

Figure 15: Request Executer Entity

time. Apart from the fields mentioned above, the state-machine also requires the Burst Length and the Address of the corresponding transaction. An independent piece of logic controls the Grant2 operation and the direction control of the SDRAM data bus and the user data bus.

5 Related work

The concept of maximization of SDRAM throughput by reordering the memory requests based on the addresses has been implemented previously in a number of projets. McKee et. al. [4] have presented a general classification of the methods for memory throughput maximization and a smart memory controller design which uses the technique of dynamic access ordering. The primary difference between this architecture and the FPX memory controller architecture is that the latter is more suitable for the networking applications whereas the earlier is suitable for larger systems with cache hierarchy. Different page interleaving schemes have been implemented for reducing the cache conflicts and cache misses [5] [6]. For the Impulse project [5], the controller is based on the similar concepts, however the architecture of Impulse is optimized for scientific computing where data structures are typically in the matrix form and exhibit a good deal of spatial locality.

A comprhensive set of algorithms for memory request reordering has been presented by Rixner et. al. [7] and McKee [8]. FPX memory controller uses a combination of some of the schemes for memory access scheduling presented here. However it differs considerably in the architectural details. The architecture of the FPX memory controller minimizes the implementation complexity without compromising on the benifits of complex memory request scheduling algorithms. The FPX memory controller uses a protocol for the data transfer between the controller and the requesting

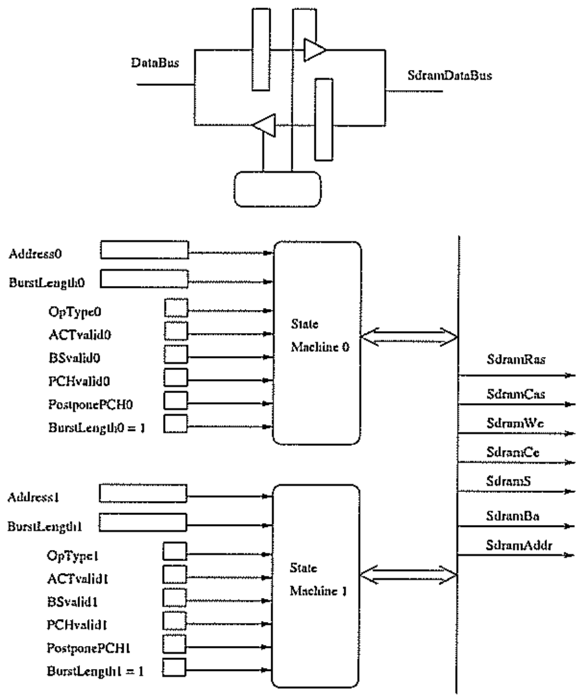

Figure 16: Request Executer Architecture

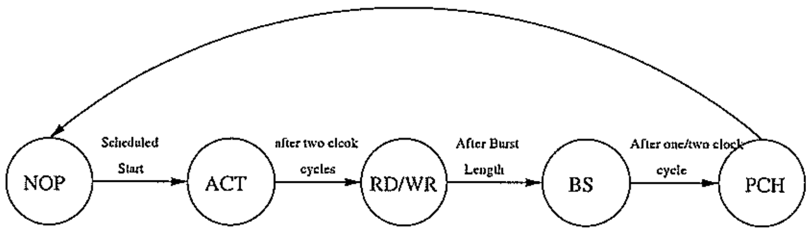

Figure 17: State transition diagram of the SDRAM controller

module and eliminates the need of the on-chip *data cache* which is used to restore the order of the memory requests in Rixner's architecture. Thus, the FPX memory controller saves on the hardware resources and makes it suitble for implementation as a core in the FPGA hardware. Some of the other general purpose SDRAM controllers have been discussed in [9]. Although they cover all the generic techniques to minimize the memory access latency, no special attention has been given to the problem of multiple-module memory requests.

Based on the existing work in smart memory controller design, we have chosen some of the controller designs to evaluate the performance of the FPX memory controller againts. Following is the description of these controllers.

The *trivial controller* activates each row before the access and precharges it after the access is over. This type of controller has low memory throughput since time is wasted in activating and precharging rows. This controller serves every request independently. In other words, it doesn't start the new transaction before the current transaction is finished completely. This controller can be improved by overlapping the successive memory transactions but without re-ordering any. We refer to it as *enhanced trivial controller* . To overcome the problem of frequent activation and precharging of the rows, row register can be used to latch the row address of the memory access. If the next memory access happens to be on the same bank and same row then the row does not need to be precharged after the memory access and needs no activation for the next memory access. This improves throughput if the access pattern exhibits locality. We refer to this type of controller as *single bank row register controller.* Further improvement in the throughput is possible by keeping one row register for each bank. Thus the row address of the memory access will be compared with the row register for that bank and if they match then there is no need to precharge the row and activate it again for the next access. If they are different then the row pointed by the row register is precharged and then the new row is activated for the access. We refer to this type of controller as *per-bank row register controller* .

A comparison of the efficiencies of all the controllers is shown in the next section.

6 Implementation results and Performance Comparison

The SDRAM controller has been implemented on the FPX. Multiple modules need to access the SDRAM simultaneously and the SDRAM controller satisfies this need. The controller has been implemented on a Xilinx VIRTEX-E XCV1000E FPGA of the FPX. The controller occupies 4% of the chip area and runs on 66 MHz.

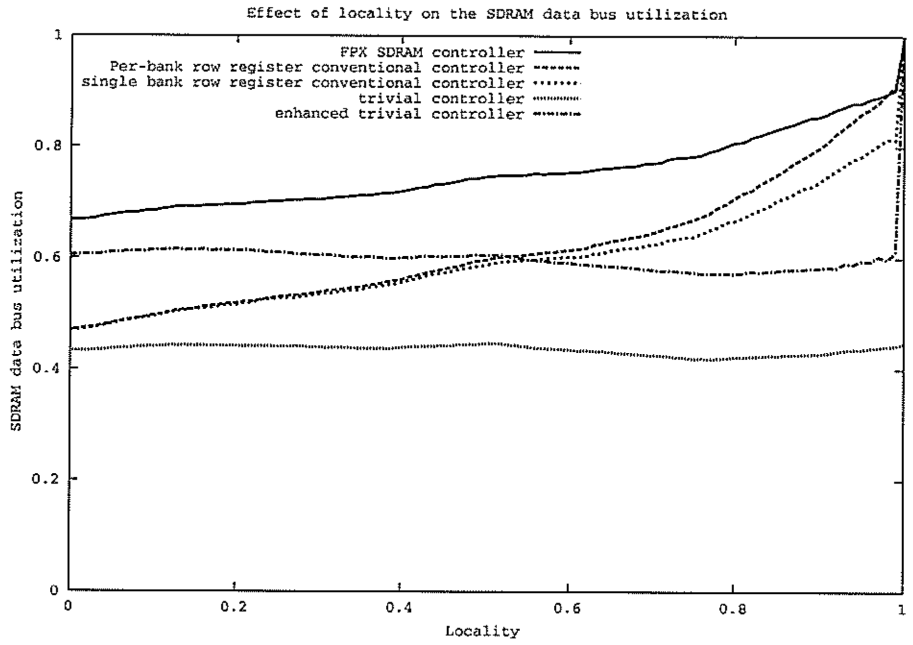

Figure 18: Performance of different SDRAM controllers against increasing locality in memory references

In order to evaluate the performance of the SDRAM controllers a request generation scheme has been devised. The request contains the requested address, requested access type (read/write) and the requested burst length. The address is generated using a markov model as shown in figure 19. In this model, a random address for access is generated and successive address vary in the range -100 to +100 randomly with probability p. The parameter p represents the locality of the memory references. Greater value of p implies greater locality of the memory references. Setting p = *0* gives completely random address and setting p = *1* gives the same address. The access type is also

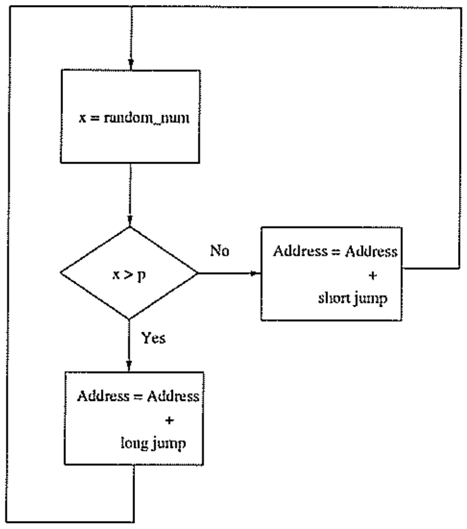

Figure 19: Markov model for the address and access type generation in the simulation

generated using the same Markov model. An access type (read/write) is generated and kept the same for the successive access with probability *p*. The burst length is generated using a poisson distribution. The mean of the poisson distribution is set to eight with the consideration that the typical memory usage is ATM circuits is of burst length eight(which gives 64 bytes, the size of an ATM cell with additional headers). This request generation scheme is used for each module. The number of modules has been fixed to five. The number of request generated was 100000. The number of clock cycles for unused data bus and clock cycles when the data bus is in use was obtained and the efficiency equal to (used clock cycles)/(Total clock cycles) was calculated. The efficiency was obtained for different values of *p*. The resultant plot is shown in the figure 18. As is clear from the graph, the *FPX memory controller* exhibits the best performance. The performance of the *trivial controller* (active precharge controller) is constant and at the lowest level. The *enhanced trivial controller* shows the improvement in the performance over the *trivial controller* , however has almost same performance irrespective of the spatial locality. The *single bank row register controller* shows better performance as the spatial locality increases. The *per-bank row register controller* surpasses the *single bank row register controller* only for hight locality values. The *FPX memory controller* provides better efficiency compared to the conventional controllers even at the lower locality. As the locality tends to 100% all the controllers almost converge into 100% efficiency except the trivial controllers.

7 Conclusions

A high performance SDRAM controller has been developed and implemented on the FPX. Conventional SDRAM controllers do not take the advantage of memory request re-ordering to improve the memory throughput. The proposed controller re-orders the memory access requests based on the address and the operation type (read or write). An intelligent permutation of the memory requests improves the memory throughput. The sequential consistency among the requests made by an individual module is maintained by queuing the requests in the FIFO. The request selection procedure is generic and can be applied to any kind of memory. The controller architecture is compact and easily implementable in the FPGA with insignificant resuorce usage.

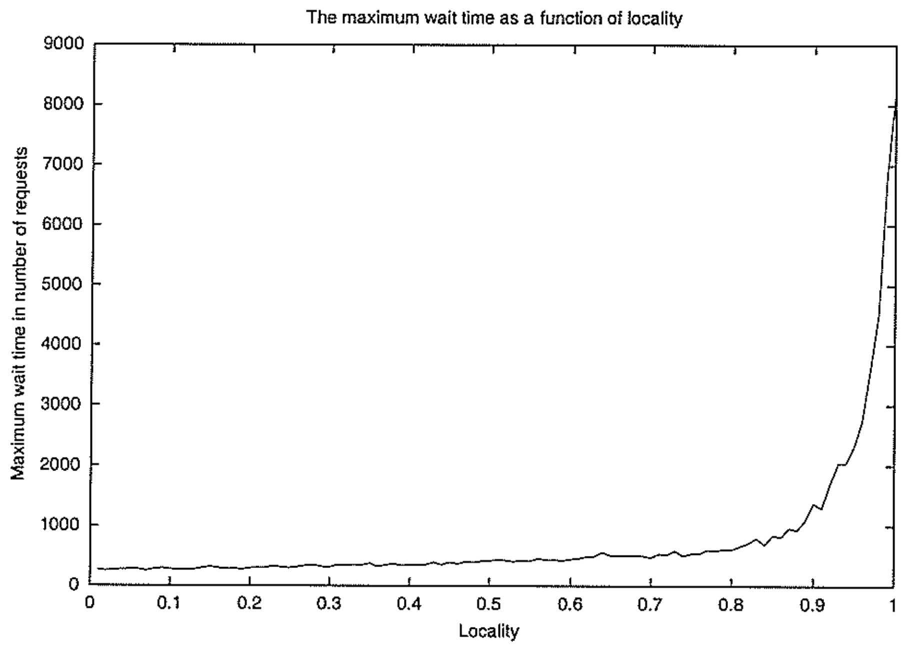

Figure 20: Starvation of the memory requests

8 Future work

Presently there is no provision to prevent the starvation of the memory requests if they wait for a long time in the queue. The controller might favor the requests of one particular module due to the high locality of references and ignore the memory requests from other modules. Without the implementation of any scheme to prevent starvation, the memory request might have to wait for exorbitantly long time in the queue. Figure 20 shows how starvation increases dramatically when the locality of the memory references increases. The x-axis in this graph shows the locality of the memory references made by five modules and the y- axis shows the maximum number of requests served by the controller between the time of arrival and the time of departure of a request of one particular module. The starvation can be avoided by giving priority to the requests which cross a certain age threshold. Hence one more field which indicates the age of the requests has to be added in the request context. When the age crosses a particular threshold, the request will be served. When all the modules have the request at the head of the queues which have crossed the age threshold then they will be serviced in round-robin fashion. The starting point of the round-robin search will be decided by a rotating token. This scenario was simulated and the SDRAM bus utilization for different age thresholds across varying locality of memory references was plotted. The age threshold is specified as the number of requests that can be served between the time of arrival and the time of departure of one request. In other words, a request can wait in the queue at the most the time required to serve the requests equal to the age threshold. As the figure 21 shows, the bus utilization increases as the threshold increases. However for the increasing value of the threshold, the increase in the throughput is not in proportion and the throughput almost saturates at the threshold value of 50 requests. The simulation can be run for different number of modules and suitable values of age threshold or maximum wait time can be obtained to enhance the performance of the controller.

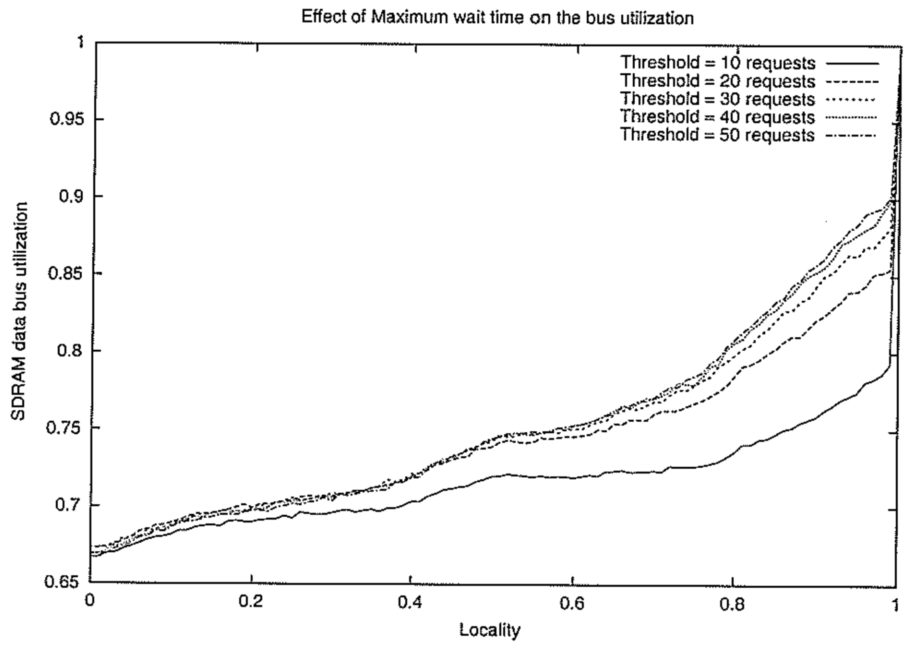

Figure 21: Memory throughput as a function of age threshold

References

[1] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, (Monterey, CA, USA), pp. 137–144, Feb. 2000.

[2] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, (Monterey, CA, USA), pp. 87–93, Feb. 2001.

[3] Micron Inc., "Small-outline SDRAM module MT4LSDT464(L)H, MT8LSDT864(L)H data sheet," 1999.

[4] S. A. McKee, W. Wulf, J. H. Aylor, R. H. Klenke, M. H. Salinas, S. I. Hong, and D. A. Weikle, "Dynamic access ordering for streamed computations," *IEEE Transaction on Computers*, vol. 49, Nov. 2000.

[5] J. B. Carter, W. Hsieh, L. Stroller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, and R. Kuramkote, "Impulse: Building a smarter memory controller," in *Proc. of the 5th IEEE Symp. on High-Performance Computer Architecture (HPCA-5)*, pp. 70–79, Jan. 1999.

[6] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, (California, CA), pp. 10–13, Dec. 2000.

[7] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens, "Memory access scheduling," in *ISCA*, pp. 128–138, 2000.

[8] S. A. McKee, "Hardware support for dynamic access ordering: Performance of some design options," Tech. Rep. CS-93-08, 9, 1993.

[9] C. Green, "Analyzing and implementing SDRAM and SGRAM controllers," *EDN Access*, Feb. 1998.