Improving Power and Data Efficiency with Threaded Memory Modules

Frederick A. Ware and Craig Hampel, *Member, IEEE*
Rambus Inc.
Los Altos CA, 94022
{ware, champel}@Rambus.com

***Abstract*—The technique of module-threading utilizes standard DDR DRAM components to build modified memory modules. These modified modules incorporate one or more additional control signals. The modification permits the module to operate at higher performance levels and at lower power levels than standard modules. The modified modules are also capable of finer granularity transactions while still operating at full bandwidth.**

***Index Terms*— CMOS memory integrated circuits, Distributed memory systems, Memory management, Memory architecture, MOS memory integrated circuits, MOSFET memory integrated circuits, Shared memory systems,**

I. Introduction

The interface speeds of DRAM (dynamic random-access memory) components have improved dramatically in the last decade. However, DRAM core speeds have seen much smaller improvements. This is because DRAM components are optimized for low cost per storage bit and not for core performance.

DRAM storage arrays are designed to be as large as possible, so that the row and column support circuitry occupies a relatively small fraction of the chip area. A consequence of this is that the row and column access times are relatively large because of the heavily loaded word lines, bit lines, and column IO lines.

II. Burst Length and Row Granularity

One of the timing parameters used by the DRAM is the column cycle time (t_{CC}). This is the interval required by a column access to transfer a block of information between a sense amplifier in the DRAM core and a pipeline register in the DRAM interface.

Another timing parameter used by the DRAM is the DQ bit time (t_{BIT}). This is the interval of time occupied by a bit of information on a data signal.

The ratio t_{CC}/t_{BIT} is called the burst length (BL). It represents the number of parallel bits that are accessed during a t_{CC} interval, and which are transferred serially though a DQ signal in sequential t_{BIT} intervals. The burst length is also called the prefetch length.

The DDR3 DRAM used in the timing examples of this paper has a t_{CC} value of 5.0ns, and a t_{BIT} value of 0.625ns. The burst length is thus [5.0/0.625] or 8, as may be seen in the last row of Table 1. Historically, the t_{BIT} parameter has changed much more rapidly than the t_{CC} parameter. The doubling of burst length every three years is due mostly to corresponding reductions in the t_{BIT} parameter.

The row-to-row access time (t_{RRD}) is the time interval between commands to access rows in different banks. Traditionally the minimum t_{RRD} value is twice the t_{CC} value, meaning that two column accesses may be performed during each row access. This leads to the following module row granularity relationship (i.e. data transferred during a row access):

$$\text{RowGranularity} = \text{BL} * (t_{RRD}/t_{CC}) * (\text{DQ/module}) \qquad (1)$$

Or

$$\text{RowGranularity} = (t_{RRD}/t_{BIT}) * (\text{DQ/module}) \qquad (2)$$

The row granularity has increased steadily, and this has created a performance issue for many applications. Some applications simply can't utilize this much data from each random access. One solution to this problem is the use of two or more independent access threads on standard memory modules, a technique referred to here as module-threading.

TABLE 1. TREND OF MODULE ROW GRANULARITY

Module Component	Year	Row Granularity[1] (bytes)
SDRAM (BL[2]=1)	1998	16
DDR (BL=2)	2001	32
DDR 2 (BL=4)	2004	64
DDR 3 (BL=8)	2007	128

[1] Two column accesses (with the indicated burst length) per row access is assumed.

[2] BL (burst length) refers to the number of bits transferred on each data wire in response to each column access.

III. Single-Thread module

Figure 2 shows a block diagram for a standard (single-thread) memory controller and memory module. The memory controller consists of a logic block and an interface that occupy part of an integrated circuit. The memory controller creates internal read and write interfaces (labeled “R” and “W”) that allow other logic blocks on the integrated circuit to access the external memory.

The memory module consists of eight DDR3-1600 components. The -1600 designation means that data is transferred at the rate of 1600 Mb/s. Each DRAM connects to just eight of the 64 DQ data signals – each DQ signal is routed from a controller pin to a DRAM pin in a single rank topology.

Each DRAM also connects to all of the control and address signals (CA) on the module. These signals carry the command code and the bank, row, and column addresses used by each memory transaction. Each CA signal is routed from a controller pin to a pin on each DRAM in a “flyby” connection topology (also know as “multi-drop” topology). Each CA signal communicates at the rate of 800 Mb/s.

A chip select signal (CS) is routed with the same flyby connection topology as the CS signals. This CS signal is shown separately because it will be modified slightly in the next section.

Write transactions are received from the transaction interface and are accumulated in a queue that consists of an address block (WA) and eight data blocks (WD). Read transactions accumulate in a queue that consists of an address block (RA). The returning read data is de-multiplexed by eight register block (RD).

The controller accepts write transactions and accumulates them in the write queue. Read transactions are accepted into the read queue and are executed in order. When the write buffer is filled to a predetermined threshold, the controller will stop issuing transactions from the read queue, and will instead wait for an appropriate read-write turnaround interval, and then issue a burst of write transactions. After an appropriate write-read turnaround interval, the controller will again issue transactions from the read queue.

This controller policy is a relatively simple one to implement, but can achieve good performance results. An improved policy will be described in a later section.

Figure 3 shows the transaction timing for the memory subsystem of Figure 2. The top diagram shows a single read transaction. It begins with an ACTIVE command, which causes one row of one bank to be sensed and held (there are eight banks altogether).

After a t_{RCD} interval a READ command is issued, and after a BL (t_{CC}) interval a second READ command with auto-pre-charge option is issued. After a CL interval, two bursts of read data are returned. Each burst is a time interval of BL containing 64 bytes. The two bursts occupy a time interval of t_{RRD} and contain a total of 128 bytes.

The CS signal is asserted for the ACTIVE command as well as the READ commands. Also, this timing example uses DDR3-1600 components from the 9-9-9 timing bin (see reference [3]).

The bottom diagram shows interleaved read transactions. Each transaction is like the one in the top diagram, but is directed to a different bank. The controller issues a transaction during each t_{RRD} interval to five different banks (A,B,C,D,E). A block of 128 bytes is returned during each t_{RRD} interval.

The bank used in transaction A may be re-used in transaction F. This is determined by the row cycle time interval (t_{RC}). If the bank in transaction F used the banks of E, D, C, or B, then a delay (bubble) must be inserted so t_{RC} is met.

The bubble size will be different depending upon which transaction uses the same bank as transaction F. Here it is assumed that there is an average bubble size of $t_{BUB-AVG}$ between each transaction. Figure 1 shows the previous three transactions (C, D, E) with the average bubble “bub” in between each. The bubble size that must be added between E and F will be one of the three sizes shown:

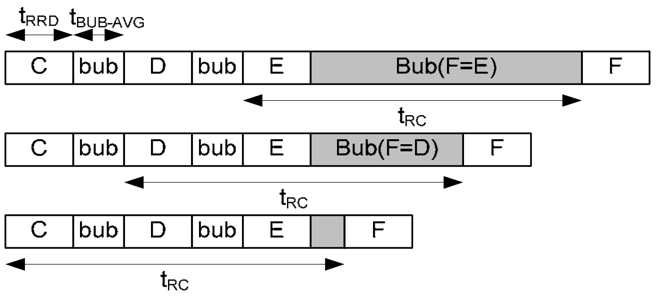

Fig. 1. Summary of bubble delay cases when the bank in F matches E, D, and C, and when there is an average bubble size of $t_{BUB-AVG}$ between the previous transactions.

Because t_{RC} is $5*t_{RRD}$, and because there are eight banks in the memory device, the following closed-form expression can be generated:

$$\begin{aligned} t_{BUB-AVG} = \; & 0.125*(5*t_{RRD}-1*t_{RRD}) \\ & +0.125*(5*t_{RRD}-2*t_{RRD}-1*t_{BUB-AVG}) \\ & +0.125*(5*t_{RRD}-3*t_{RRD}-2*t_{BUB-AVG}) \end{aligned} \quad (3)$$

In other words, the average bubble size is one of three sizes, each with a probability of 0.125 (because of eight banks). The t_{RC} delay is equal to $5*t_{RRD}$, and is reduced by the indicated amounts ($3*t_{RRD}+2*t_{BUB-AVG}$, $2*t_{RRD}+2*t_{BUB-AVG}$, $1*t_{RRD}$) in each of the cases.

Solving for $t_{BUB-AVG}$ yields a value of $0.82*t_{RRD}$ for $t_{BUB-AVG}$, resulting in a bandwidth efficiency of 54% for random, in-order read transactions. Here bandwidth efficiency is defined as:

$$\text{BW Efficiency} = t_{RRD} / (t_{RRD} + t_{BUB-AVG}) \quad (4)$$

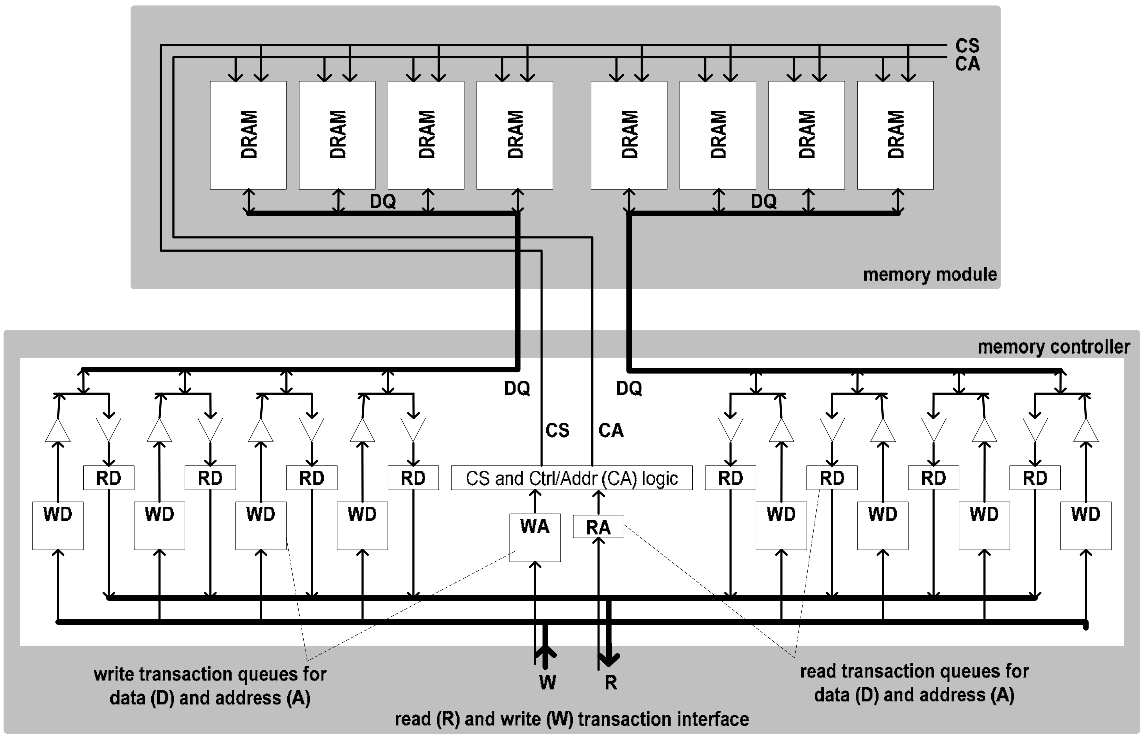
CS
CA
DRAM
DQ
memory module
memory controller
DQ
DQ
CS
CA
RD
WD
CS and Ctrl/Addr (CA) logic
WA
RA
write transaction queues for data (D) and address (A)
W
R
read transaction queues for data (D) and address (A)
read (R) and write (W) transaction interface

Fig. 2. The block diagram for a single-thread memory controller and memory module. Write transactions accumulate in a queue that consists of an address portion (WA) and eight data slices (WD). Read transactions accumulate in a queue that consists of an address portion (RA). The returning read data is de-multiplexed by eight data slices (RD). Each transaction affects all eight DRAM components on the memory module.

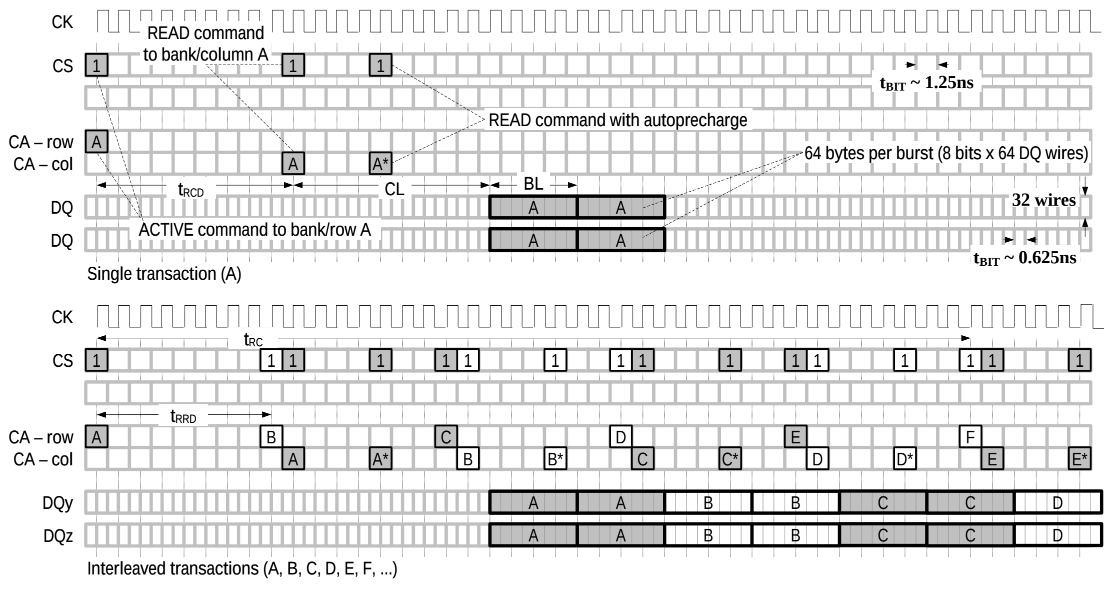
CK
CS
READ command to bank/column A
READ command with autoprecharge
t_{BIT} ~ 1.25ns
CA – row
CA – col
64 bytes per burst (8 bits x 64 DQ wires)
t_{RCD}
CL
BL
DQ
DQ
32 wires
ACTIVE command to bank/row A
t_{BIT} ~ 0.625ns
Single transaction (A)
t_{RC}
t_{RRD}
DQy
DQz
Interleaved transactions (A, B, C, D, E, F, ...)

Fig. 3. Transaction timing for a single-thread memory controller and memory module. The top diagram shows a single read transaction. It begins with an ACTIVE command, followed by two READ commands (the second with an auto-pre-charge option). Two bursts of read data are returned, each with 64 bytes, for a total of 128 bytes. The bottom diagram shows interleaved read transactions. Each transaction is like the one in the top diagram, but is directed to a different bank. A block of 128 bytes is returned during each t_{RRD} interval. The bank used in transaction A may be re-used in transaction F (this is the t_{RC} interval). This timing example uses DDR3-1600 components from the 9-9-9 timing bin.

IV. Dual Thread module

Figure 5 shows a block diagram for a dual-thread memory controller and memory module. It is similar to the single thread subsystem of Figure 2; this section will focus on the modifications that were made to support dual-thread operation.

The memory controller now has four internal interfaces (labeled “Ry”, “Wy”, “Rz” and “Wz”) that provide parallel access to two independent memory spaces.

The memory module consists of two sets of four DDR3-1600 components. Again, the -1600 designation means that data is transferred at the rate of 1600 Mb/s. Each DQ signal is still routed from a controller pin to a DRAM pin in a “point-to-point” connection topology. Each CA signal is still routed from a controller pin to a pin on each of the eight DRAMs in a “flyby” connection topology. Each CA signal communicates at the rate of 800 Mb/s.

There are now two chip select signals (CSy and CSz) that are routed with the same flyby connection topology as the CS signals. However, each of the chip select signals connects to only four of the eight DRAMs, allowing them to be separately controlled.

Write transactions are accumulated in one of two write queues (WAy/WDy and WAz/WDz). An address bit is used to determine which queue to use. This address bit will be chosen to ensure that there is an approximately equal number of transactions directed to the “y” and “z” memory spaces.

Read transactions also accumulate in one of two read queues (RAy and RAz). The returning read data is de-multiplexed by one of two sets of four register slices (RDy and RDz).

The controller policy is the same as that described for Figure 2. The one difference is that read transactions are paired for the two memory spaces, and write transactions are paired. In other words, the controller does not attempt to perform a read transaction in one space and write transaction in the other.

Figure 6 shows the transaction timing for the memory subsystem of Figure 5. The top diagram shows a single pair of read transactions. Transaction A begins with an ACTIVE command to a bank in the “y” space. After a t_{RCD} interval a READ command is issued, and after three BL (t_{CC}) intervals a second, third, and fourth READ command is issued. The fourth READ includes the auto-pre-charge option. After a CL interval, four bursts of read data are returned. Each burst is a time interval of BL containing 32 bytes. The four bursts occupy a time interval of t_{RRD} and contain a total of 128 bytes. Note that the t_{RRD} value is twice the value used in Figure 3. This is because the same amount of data (128 bytes) requires twice as much time to transfer with half the number of wires (the 32 DQy wires).

The “z” memory space is accessed by a second transaction B. This transaction is offset by a delay of BL/2 relative to transaction A, but otherwise is identical, performing an ACTIVE command and four READ commands. The commands are steered to the two sets of DRAMs using assertions on the CSy and CSz signals.

The bottom diagram shows interleaved pairs of read transactions. Each transaction pair is like the pair in the top diagram, but is directed to different banks. The controller issues a transaction during each t_{RRD} interval to three different bank pairs (A/B,C/D,E/F). Two blocks of 128 bytes is returned during each t_{RRD} interval, one on the DQy signals and one on the DQz signals. The banks used in transaction A/B may be re-used in transaction G/H. This is determined by the row cycle time interval (t_{RC}). The t_{RC} value in Figure 6 is 48 CK cycles (3*t_{RRD}), compared with a value in Figure 3 of 40 CK cycles (5*t_{RRD}).

The bubble size will be different depending upon which transaction uses the same bank as transaction F. Again, it is assumed that there is an average bubble size of $t_{BUB-AVG}$ between each transaction. Figure 4 shows the previous three transaction pairs (A/B, C/D, E/F) with the average bubble “bub” in between each. The bubble size that must be added between E/F and G/H will be one of the three sizes shown. As before:

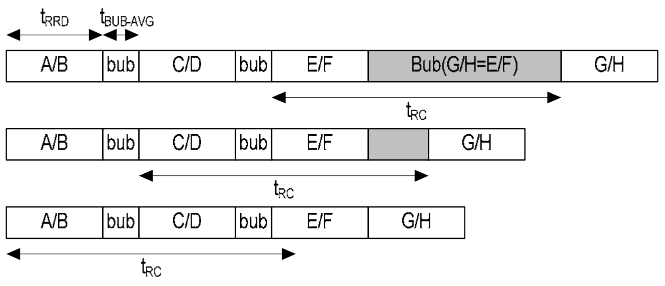

Fig. 4. Summary of bubble delay cases when banks in G/H match E/F, C/D, and A/B, and when there is an average bubble size of $t_{BUB-AVG}$ between the previous transactions.

Since t_{RC} is 3*t_{RRD}, and because there are eight banks in the memory device the following closed-form expression can be generated:

$$t_{BUB-AVG} = 0.125*(3*t_{RRD}-1*t_{RRD}) + 0.125*(3*t_{RRD}-2*t_{RRD}-1*t_{BUB-AVG}) \quad (5)$$

In other words, the average bubble size is one of two sizes, each with a probability of 0.125 (eight banks). The t_{RC} delay is equal to 3*t_{RRD}, and is reduced by the indicated amounts (2*t_{RRD}+2*$t_{BUB-AVG}$, 1*t_{RRD}) in each of the cases.

Solving for $t_{BUB-AVG}$ yields a value 0.33*t_{RRD} for $t_{BUB-AVG}$, resulting in a bandwidth efficiency of 75% for random, in-order read transactions. Again bandwidth efficiency is defined as:

$$\text{BW Efficiency} = t_{RRD} / (t_{RRD} + t_{BUB-AVG}) \quad (6)$$

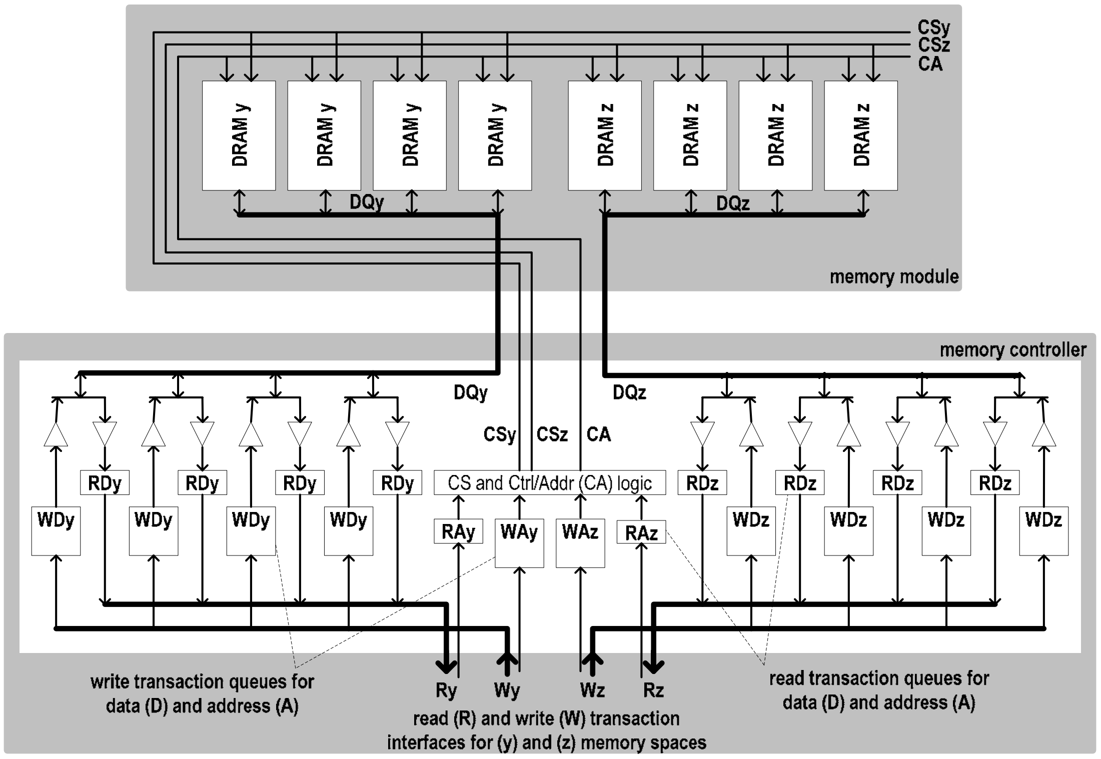
CSy
CSz
CA
DRAM y
DRAM z
DQy
DQz
memory module
memory controller
CS and Ctrl/Addr (CA) logic
RDy
WDy
RAy
WAy
WAz
RAz
RDz
WDz
write transaction queues for data (D) and address (A)
read transaction queues for data (D) and address (A)
Ry
Wy
Wz
Rz
read (R) and write (W) transaction interfaces for (y) and (z) memory spaces

Fig. 5. The block diagram for a dual-thread memory controller and memory module. One address bit is statically selected to steer each transaction to either the "y" or "z" memory space. The controller maintains separate read and write queues for each of the independent memory spaces. The module in this example has eight DRAMs, with four DRAMs in each memory space. There are two chip select signals (CSy and CSz) used to direct commands on the CA signals to the two sets of DRAMs.

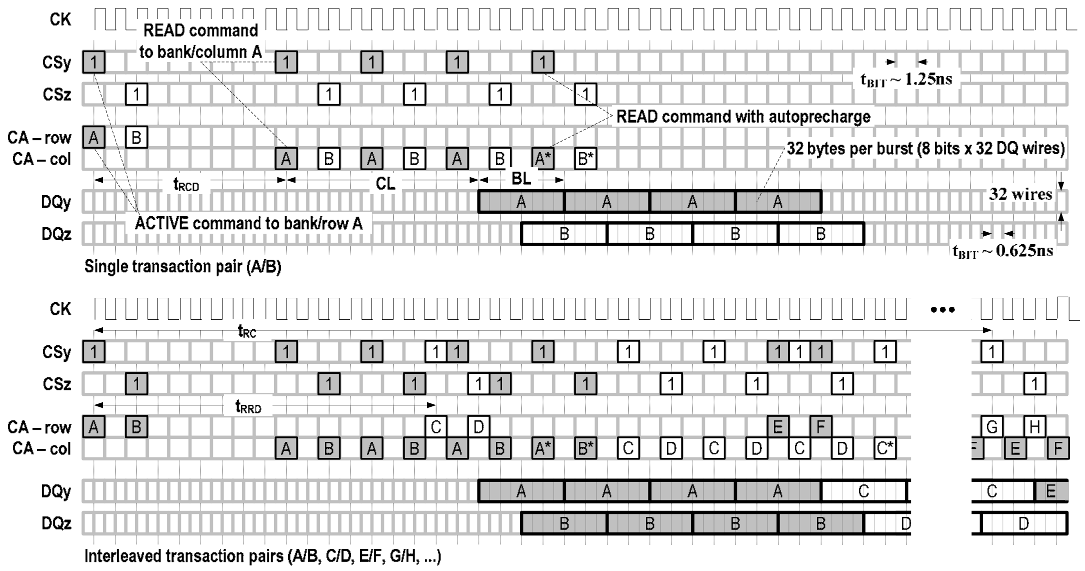
CK
CSy
CSz
CA – row
CA – col
DQy
DQz
READ command to bank/column A
READ command with autoprecharge
32 bytes per burst (8 bits x 32 DQ wires)
ACTIVE command to bank/row A
t_BIT ~ 1.25ns
32 wires
t_BIT ~ 0.625ns
t_RCD
CL
BL
Single transaction pair (A/B)
t_RC
t_RRD
Interleaved transaction pairs (A/B, C/D, E/F, G/H, ...)

Fig. 6. Transaction timing for a dual-thread memory controller and memory module. The top diagram shows a single pair of read transactions. The "A" transaction is directed to the "y" DRAMs using the CSY signal. It begins with an ACTIVE command, followed by four READ commands (the fourth with an auto-pre-charge option). Four bursts of read data are returned, each with 32 bytes, for a total of 128 bytes. The "B" transaction is offset by two CK cycles, and is directed to the "z" DRAMs. The bottom diagram shows pairs of interleaved read transactions. Each pair is like the one in the top diagram, but is directed to different banks. Two blocks of 128 bytes is returned during each t_{RRD} interval. The banks used in transactions A/B may be re-used in transactions G/H (this is the t_{RC} interval). This example uses DDR3-1600 from the 9-9-9 timing bins.

V. QUAD-THREADING

The multi-threading concept can be extended by adding two additional chip select signals (four total) to the memory module to provide four independent memory spaces. The DRAM components would also need an option for a burst length of 16. There would then be enough command bandwidth on the CA signals to interleave four concurrent transaction streams.

The interleave factor for each stream would be two (compared with three and five for dual- and single-threading), resulting in an effective read bandwidth of 88%. These results are summarized in Table 2.

VI. MODULE POWER

The use of multi-threading also reduces the total power for each transaction. Typically, the module power required for row accesses (the ACTIVE command) accounts for 0.25 to 0.50 of the total power, with the rest consumed by the column operation (the READ command).

Dual-threading has the same total number of column accesses as single-threading, but only one-half as many devices are accessed for each row transaction. This reduces the total power to 0.75 to 0.875 that of a single-threaded module..

Likewise, quad-threading has the same number of total column accesses as single-threading, but only one-quarter as many row accesses. This reduces the total power to 0.625 to 0.813 that of a single-threaded module. These results are summarized in Table 2.

TABLE 2. MULTI-THREADING BENEFITS

Threading Factor	Read BW (larger is better)	Power per Transaction (smaller is better)
single	0.54	1
dual	0.75	0.750 – 0.875
quad	0.88	0.625 – 0.813

VII. QUEUE DETAILS

Figure 7 shows more detail for the queue elements used in Figure 2 and Figure 5. As previously described, write transactions are accumulated in the write queue. Read transactions are accepted into the read queue and executed in order. When the write buffer is filled to a predetermined threshold, the controller will stop issuing transactions from the read queue, and will issue a burst of write transactions. The controller will then return to issuing transactions from the read queue.

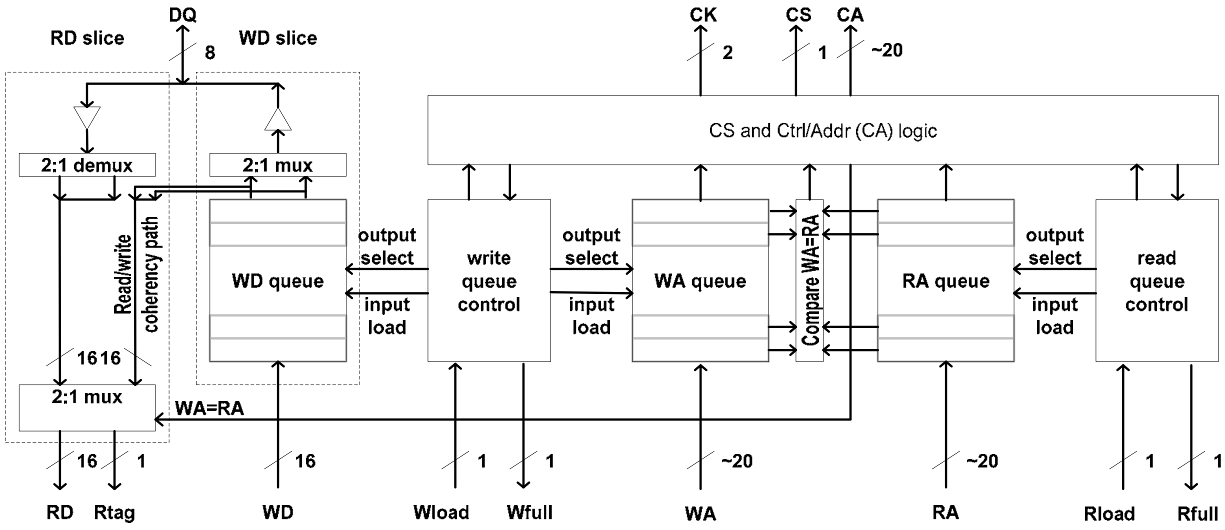

Fig. 7. Memory controller queue details. Write transactions are accumulated in the write queue. Read transactions are accepted into the read queue and executed in order. When the write buffer is filled to a predetermined threshold, the controller will stop issuing transactions from the read queue, and will issue a burst of write transactions. The controller will then return to issuing transactions from the read queue. The Wfull and Rfull signals provide flow control, so transactions from the controller can be held off. Comparison logic checks read addresses against write addresses to ensure coherency. The Rtag signal frames the read data.

The Wfull and Rfull signals are provided so transactions from the controller can be held off if the associated queue becomes full. The comparison logic checks read addresses against write addresses to ensure coherency. For example, a read of a pending write will return the write queue data. This ensures that the results do not depend upon the order in which read and write transactions are performed. The Rtag signal is used to frame the read data. In the case of in-order read transactions, it simply serves as a timing mark for each block of data.

Note that the WD and RD logic blocks are shown as slices – there is one for each each DRAM in Figures 2 and 5. In the case of Figure 2, the eight slices are operated together. In the case of Figure 5, there are two groups of four slices operated independently for each of the two memory spaces.

Note also that the 16-bit input of the WD queue is loaded many times for each load of the WA queue – 8 times for single threading and 16 times for dual threading. This is because each slice must store 16 bytes and 32 bytes, respectively, for each 128 byte write transaction.

VIII. OUT-OF-ORDER WRITE EXECUTION

When the write queue becomes full or nearly-full, some or all of the transactions are written to memory. It is not necessary to empty the write queue in the same order it was filled. As long as read/write coherency is maintained, the write transactions may be issued in any order.

In fact, there is a performance benefit to issuing the write transactions in an order which avoids unnecessary bank conflicts with previous transactions.

For example, after the final read transaction has issued and a read/write turnaround bubble has been observed, a first write transaction is selected. The selection criteria will include choosing a bank address that doesn't conflict with any of the remaining read transactions that are still in progress.

Each subsequent write transaction will be chosen in the same manner, so the smallest number of bank-interference bubbles are added to the transaction stream.

An approximate relationship for out-of order efficiency of the write buffer unloading process is as follows:

$$\text{Efficiency} = t_{RRD} /(t_{RRD} + t_{BUB}) \qquad (7)$$

Where

$$t_{BUB} \sim t_{RRD}\, [N^{Q+1}]/ [Q*B^{Q}] \qquad (8)$$

With the following definitions:

$N = t_{RC} / t_{RRD}$ (9)
B = banks per memory space (10)
Q = transaction entries per queue (11)

The out-of-order efficiency is plotted in Figure 8 as a function of queue depth for the single thread (N=5) and dual thread (N=3) cases, with the number of memory banks set to 8.

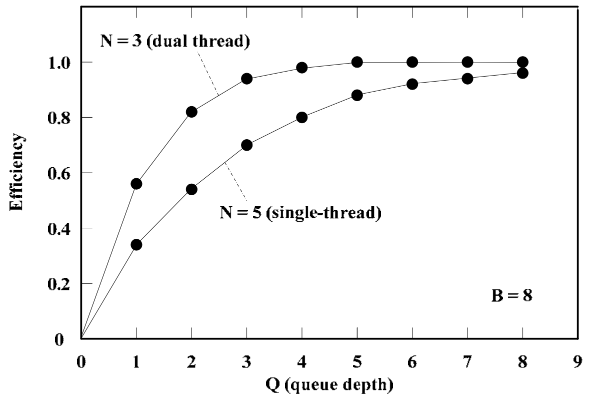

Fig. 8. Queue Depth versus Efficiency. The expression for efficiency is evaluated for queue depths of 1-8 transactions for an eight bank memory for the single thread and dual thread examples previously discussed.

As expected, the dual-threading case is much more efficient for small queues because there are a smaller number of transactions which can potentially interfere with one another. A benefit of the smaller queue size is less controller area, and simpler controller logic for detecting the conflicts.

IX. OUT-OF-ORDER READ EXECUTION

The same out-of-order benefits may be realized for read transactions. This requires that read transactions be tagged with a sequence tag (the Rtag signal) in Figure 7). This requires that the transaction generating logic be able to accept the returning read data out-of-order – this may not be possible in some applications

X. FINE GRANULARITY TRANSACTIONS

Finally, note that the 128 byte transaction size could be reduced to 64 bytes in the case of the dual-threaded module in Figure 6. This would increase the number of concurrent transactions, but might be a benefit for some applications which didn't need as much data per access.

Conclusions

The historical trend of increasing row and column access granularity of memory modules will increasingly limit the performance in certain classes of applications. The architectural technique of module-threading may be applied to conventional memory modules with relatively low incremental cost. This technique permits the module to provide greater performance with reduced power consumption. It also permits the option of lower granularity transactions at full bandwidth.

References

[1] A Performance Comparison of DRAM Memory System Optimizations for SMT Processors. Zhichun Zhu. The 11th International Symposium on High-Performance Computer Architecture Palace Hotel, San Francisco, February 12-16, 2005
http://www.hpcaconf.org/hpca11/papers/19_zhuperformancecomparisonofdram_updated.pdf

[2] Micro-threaded Row and Column Operations in a DRAM Core. Frederick A. Ware and Craig Hampel. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS05), Austin TX, March 20, 2005.
http://www.rambus.com/news/technical_docs/MicroThread.pdf

[3] DDR/DDR2/DDR3 SDRAM Tutorial. JEDEX Conference 2006, San Jose, CA, April 16, 2006.
http://www.jedex.org

[4] Building DRAM-based High Performance Intermediate Memory System. Junyi Xie and Gershon Kedem Department of Computer Science Duke University Durham, North Carolina 27708-0129 May 15, 2002.
http://www.cs.duke.edu/~junyi/papers/dram/techrpt.pdf

[5] Reducing DRAM Latencies with an Integrated Memory Hierarchy Design. Wei-fen Lin, Steven K. and Doug Burger. The 7th International Symposium on High-Performance Computer Architecture, January 2001.
http://www.eecs.umich.edu/~stever/pubs/hpca01.pdf

.