Dynamic Random Access Memory: A Survey

Tulika Mitra
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
mitra@cs.sunysb.edu

February 25, 1999

Abstract

The increasing gap between processor speed and main memory latency has accelerated the development of various innovative architectures from dynamic random access memory(DRAM) vendors. We note that recent DRAM innovations have taken place along three dimensions: reduce access latency, develop new interfaces to achieve higher bandwidth, and add simple logic to the DRAM core for application specific memories, particularly for graphics. The goal of this report is to provide a comprehensive survey of DRAM operations and the recent DRAM architectures from a computer system designer's point of view. We also propose certain mechanisms to exploit the on-chip bandwidth of DRAM array to support certain memory bandwidth-intensive operations that are essential to overall system performance. We show that the current DRAM architectures can be extended to perform those operations by simple modification to the on-chip self testing logic (Built-In Self-Test) circuit on high-density DRAM chips.

1 Introduction

The growing performance gap between the processor and memory has accelerated the development of many interesting Dynamic Random Access memory(DRAM) architectures from the memory manufacturers. However, to the best of our knowledge, there has been no recent comprehensive survey of DRAM operations, technology and architectures. Most of the surveys on DRAM [Bur97, Jon92, Kat97, KOI95] are either too old to cover more recent architectures, or are intended for DRAM users to choose the right memory for their system. This report is intended to fill that gap by providing a comprehensive survey of DRAM operations, and various new DRAM architectures from a computer system designer's point of view.

This report is divided into two parts. In the first part, we describe in details, the internal structure and components of DRAM memory, and how the basic operations are performed. We also discuss the meaning of different performance specification metrics used for DRAM. This provides the background for the subsequent sections. Next, we follow the evolution of DRAM architecture from single transfer mode to burst mode, and from asynchronous to synchronous mode. We notice that DRAM architecture has evolved along three independent dimensions: reduce access latency, develop revolutionary interfaces to achieve higher bandwidth, and add simple logic to the DRAM core for application specific memories, particularly for graphics. We conclude this part with a

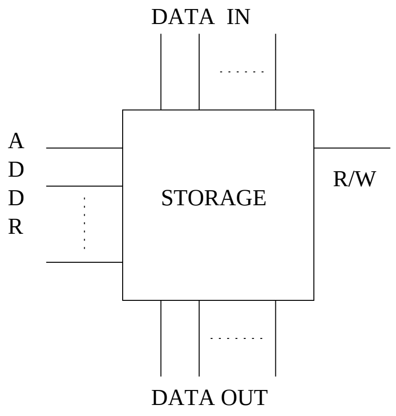

Figure 1: Basic Dynamic Random Access Memory Configuration

brief description of the memory controller architecture which is an integral part of modern memory systems.

In the second part, we try to exploit the on-chip bandwidth of DRAM core for performing simple operations, like block memory copy and memory clear. We notice that future DRAM chips, with its high density, are expected to have on-chip self-testing logic (Built-In Self-Test (BIST)), that is only used during power-up for testing the chip. This logic consumes some silicon area on the chip; however it is never used during normal memory operation. We propose to modify BIST logic to perform the above mentioned block memory copy and clear operations. Note that, including logic inside memory chip has been proposed for a long time [P$^+$97, OCS98]. However, these proposals require significant change to both the processor and memory. The goal of our proposal is to show that with a simple modification to existing BIST logic, the current DRAM architecture can be extended to support certain memory bandwidth-intensive operations that are essential to overall system performance.

2 DRAM Architecture, Interface and Operation

2.1 Basic DRAM Architecture

The basic dynamic random access memory (DRAM) configuration is shown in Figure 1. In DRAM, any storage location can be randomly accessed for read/write by inputting the address of the corresponding storage location. The block diagram of the internal architecture is shown in Figure 2. A typical DRAM of bit capacity $2^N \times 2^M$ consists of a *memory array* of *memory cells* arranged in 2^N rows (*word-lines*) and 2^M columns (*bit-lines*). Each memory cell has a unique location represented by the intersection of word and bit line. Each memory cell consists of a transistor and a capacitor. The charge on the capacitor represents *0* or *1* for the memory cell.

The support circuitry for the DRAM chip is used to read/write to a memory cell. It includes
a) *Address decoders* to select a row and a column.
b) *Sense amplifiers* to detect and amplify the charge in the capacitor of the memory cell.
c) *Read/Write logic* to read/store information in the memory cell.

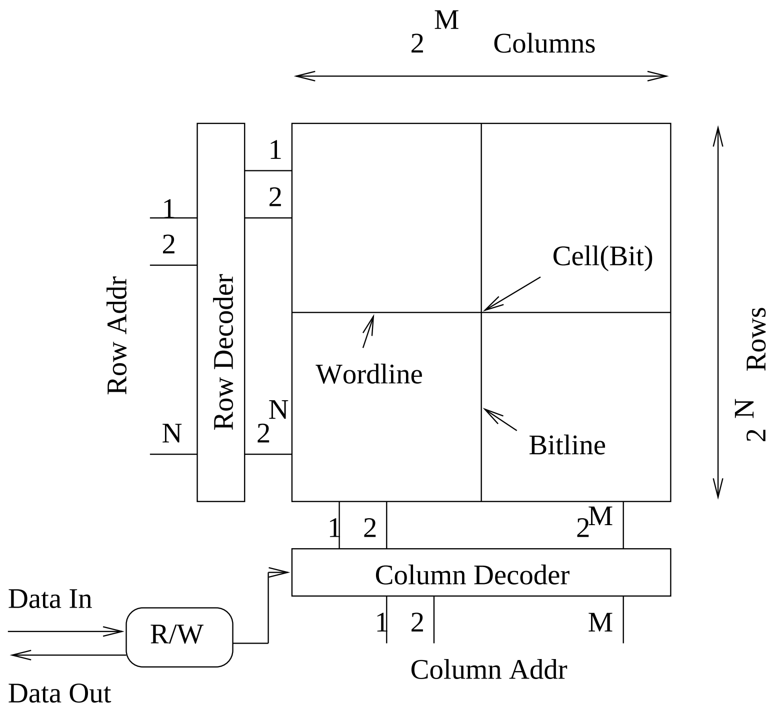

Figure 2: Basic Dynamic Random Access Memory Architecture

d) *Output Enable logic* that controls whether data should appear at the outputs.
e) *Refresh counters* to keep track of refresh sequence.

The address of a memory location comes in two parts. First, a row address appears on the address bus, and decoded, which selects one row/word-line. This activates all the 2^M cells along that word-line. Next, the column address appears on the address bus and decoded. This selects one bit out of the 2^M bits that have just been activated. Note that the row and column address are not required at the same time. Hence, the two addresses are multiplexed on the address lines to save the precious pin count of the DRAM chip. The charge stored in the chosen memory cell is amplified using the sense amplifier and then routed to the output pin. Read/Write is controlled using the read/write logic.

2.2 Detail DRAM Architecture

In this section, we look at some of the interesting components of the DRAM chip in detail.

2.2.1 Memory cell

The heart of a DRAM chip are the one-transistor dynamic memory cells consisting of a select transistor and a storage capacitor as shown in Figure 3 [Pri91]. This low component count of the cell gives DRAM the advantage of small chip size. When the row address is decoded, the selected word-line is turned on. This in turn activates the gate of the transistor, and the charge stored on the capacitor appears on the bit-line. The capacitance of the bit-line C_b is normally much higher

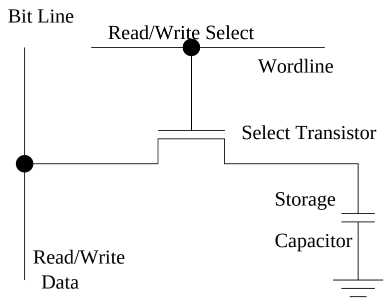

Figure 3: One Transistor DRAM cell

than that of the storage cell capacitor C_s. The charge stored on a capacitor is the product of its capacitance and voltage level. As the charge of a DRAM cell is to be distributed between itself and the bit-line when the cell is accessed, the voltage signal that eventually appears on the bit-line, V_b, Hence when the voltage V_s stored in the capacitor is read out to the bit-line, is $V_b = V_s \times \frac{C_s}{C_b}$, where V_s is the original voltage level at the DRAM cell's capacitor. That is why sense amplifiers are required to amplify this signal before it appears on the output level pin. It also results in the destruction of the charge stored in the cell, and it must be restored back after the read operation. Also, since the charge is stored in a capacitor, it tends to leak away with time, and needs to be refreshed periodically.

2.2.2 Bit-line Architecture

The main problem with DRAM till date remains about detecting the small charge associated with the capacitor. The sense amplifiers should be sensitive enough to detect the charge while they should fit in the area left between two bit-lines. The bit-line architecture is about the number of cells, the length and resistivity of the bit-line and the pitch or spacing between the bit-lines. The following are taken into consideration to determine the number of cells on the bit-line [Pri91]:

1. amount of charge stored in the capacitor cell and the minimum signal on the bit-line
2. power dissipation due to the activated sense amplifiers
3. time interval between two refresh operations which is determined by the capacitor's signal decay
4. access time determined partly by the propagation delay associated with the bit-line

The physical layout of the array architecture is also important to properly operate the sense amplifiers. Initially the cross-point architecture as shown in Figure 2 was used. Differential sense amplifiers was used to sense the small capacitor charge that appears on the bit-line by comparing it against a reference level provided by dummy cell structure. Figure 4 shows this concept. Storage cells C(L) and C(R) represent memory cells (there are many of them in parallel). Before reading the cell, *bitline* and $\overline{bitline}$ lines are precharged to the power supply voltage V_{dd}. Also, the reference voltage level (V_{dd}) is stored in the dummy cells on both sides of the sense amplifier. When a word line, let's say W(L) is selected, dummy cell $\overline{DUMMY}$ is also selected. This way, the signal from the storage cell C(L) appears on one side of the sense amplifier, while reference

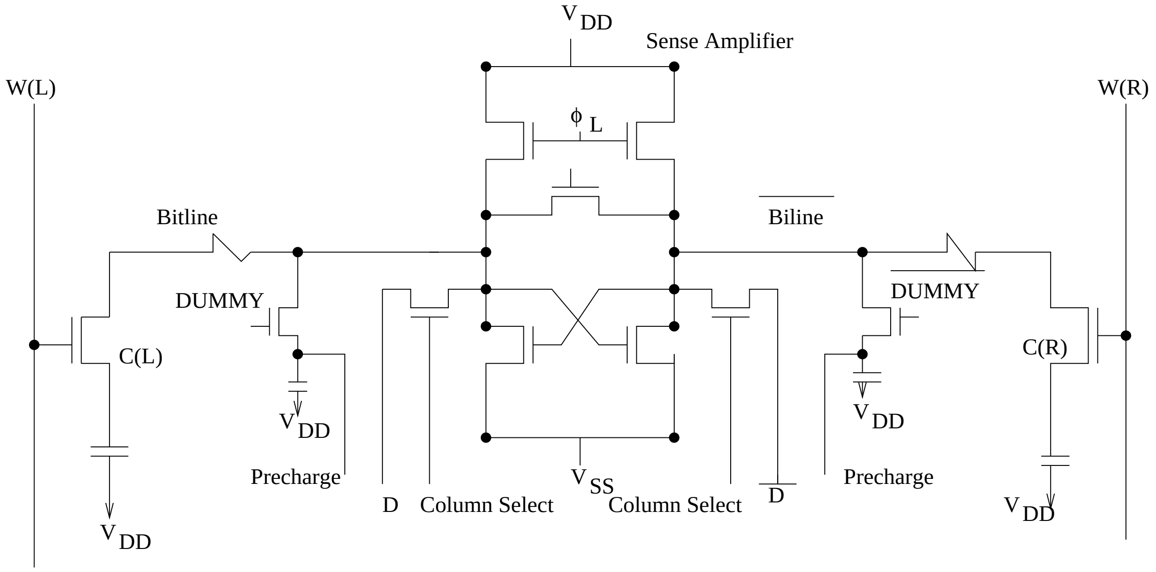

Figure 4: A DRAM differential sense amplifier with dummy cell structure

signal from the $\overline{DUMMY}$ cell appears on the other side. This creates a voltage differential across the sense-amplifier. Now the load transistors are turned on by Φ_L. Both nodes try to rise, till the higher side turns on the opposite transistor. This gradually results in amplification of the signal to full logic level, and the charge of the storage cell is also restored. This was known as *Open Bit-line Architecture.*

Today, open bit-line architecture has been replaced by *Folded Bit-line Architecture.* The later improves the balance of the line, reduces bit-line noise and results in less capacitance coupling to the sense amplifier. The difference between the two architectures are shown in Figure 5. In conventional bit-line architecture, the bit and $\overline{bit}$ lines were placed physically to the left and right side of the sense amplifier. In folded bit-line architecture, the bit-lines are folded and are placed side by side. Now the sense amplifiers and the pass transistors to the data-bus along with the column decoders can be placed at the opposite end of the bit-line pair. This way any fluctuation on the column decoder will become common mode noise on the bit-line pair and will not contribute to the differential signal.

Figure 6 shows a typical DRAM with folded bit-line architecture [Kat97]. Note that with lithography feature λ, folded bit-line takes $8\lambda^2$, as opposed to $4\lambda^2$ for the open bit-line architecture, assuming bit-line and word-line pitch of 2λ.

2.3 Basic DRAM Interface

DRAM chip has only a few signals that control the operation of the DRAM.

1) *Row Address Select Strobe* ($\overline{RAS}$) $\overline{RAS}$ is used to latch the row address and to initiate the memory cycle.

2) *Column Address Select Strobe* ($\overline{CAS}$) $\overline{CAS}$ is used to latch the column address and to initiate the read/write operation.

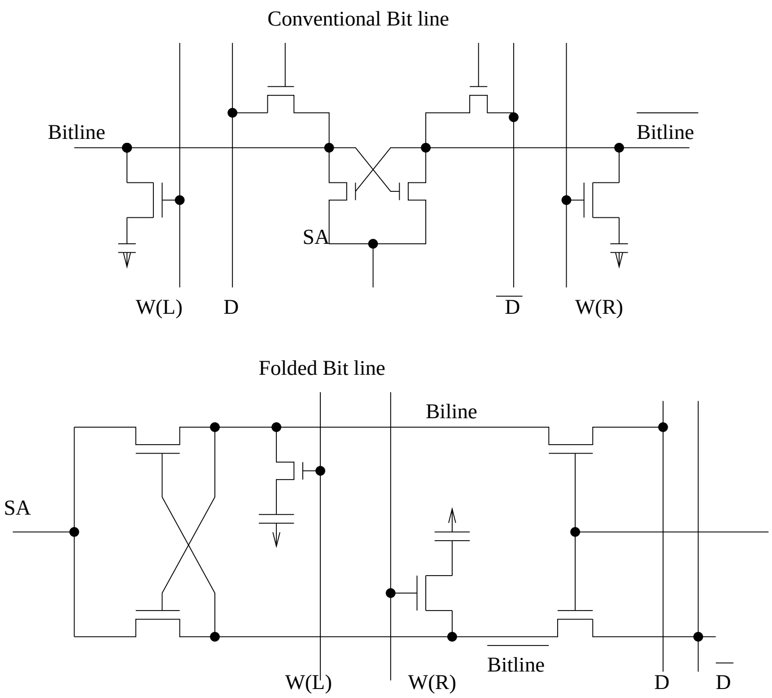

Figure 5: Conventional and folded bit line architecture

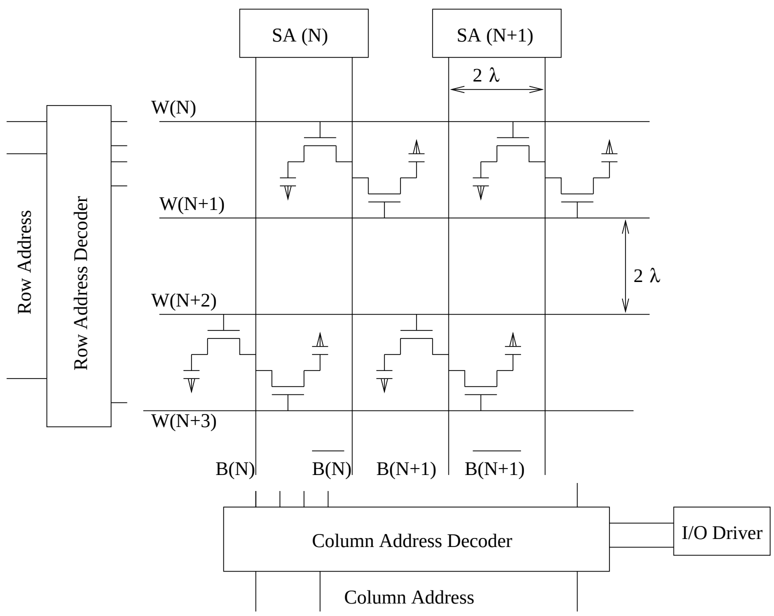

Figure 6: A DRAM with folded bit line architecture

3) *Row and Column Address* Addresses are used to select a memory cell.

4) *Write Enable ($\overline{WE}$)* Write Enable signal is used to choose read or write operation.

5) *Output Enable ($\overline{OE}$)* During a read operation this signal prevents the data to appear at the output till needed.

6) *Data In and Out (DQs)* These pins are used for input/output of data.

2.4 Basic DRAM Operations

DRAM interface allows the user to read data from a memory cell, write data to a memory cell, and refresh a memory cell.

2.4.1 Read operation

DRAM read operation consists of row access, column access, write-back and precharge [IBMd]. A simplified read cycle is illustrated in Figure 7 and consists of the following steps:

1. The row address is applied to the Row Address input for a time t_{ASR}.
2. $\overline{RAS}$ goes from high to low and held low for a time t_{RAS}.
3. Row Address should be applied for t_{RAH} after $\overline{RAS}$ goes low. By this time, row address is latched and decoded to activate the appropriate word-line. This connects the memory cells of the word-line to the bit-line pairs, and transfers the charge stored in the memory cell capacitor to the bit-line which is detected and amplified by the sense amplifiers.
4. The column address is applied to the Column Address input for a prescribed amount of time t_{ASC} before $\overline{CAS}$ goes low.
5. $\overline{WE}$ is set high to indicate a read operation for time t_{RCS} before $\overline{CAS}$ goes low.
6. $\overline{CAS}$ switches from high to low and remains low for t_{CAS}.
7. Column Address and $\overline{WE}$ should be held for time t_{CAH} and t_{RCH} respectively, after $\overline{CAS}$ goes low. By now Column Address is latched and decoded to select the particular bit-line from the sense amplifier. *Since reading the memory cell destroys the charge stored in the capacitor, the data in the selected row has to be written back from the sense amplifier in parallel to the column access.*
8. $\overline{OE}$ goes low within the prescribed window of time. Transition of $\overline{OE}$ is optional. It could be tied to low.
9. Data appears at the DQ pins after a prescribed amount of time.
10. $\overline{CAS}$ and $\overline{RAS}$ returns to their inactive states. They should remain high for time t_{CRP} and t_{RP} for the memory array to be precharged and prepared for the next memory operation.

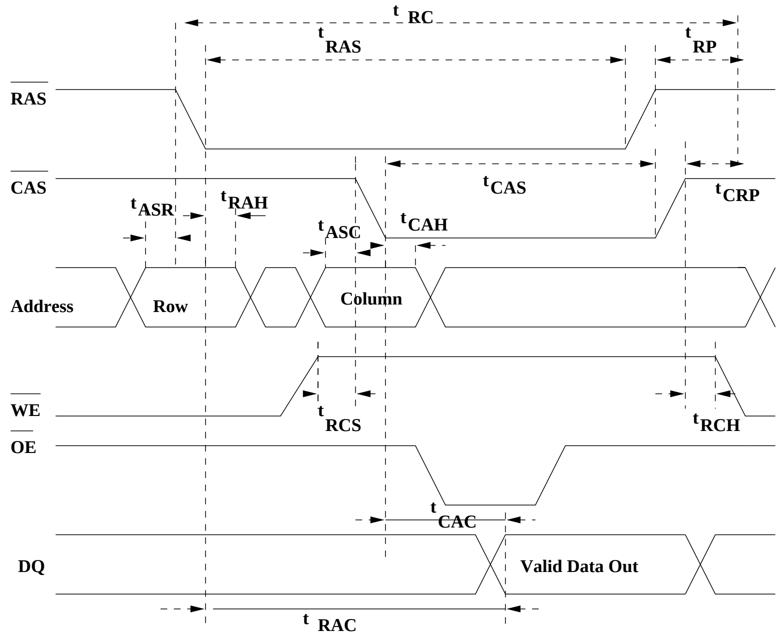

Figure 7: Timing diagram for basic DRAM read operation

2.4.2 Write operation

DRAM write operation is similar to the read operation except that the selected bit-line pair is connected to external circuitry and forced into a new logic state according to the external data [IBMd]. Figure 8 shows the DRAM write operation. Read and write has identical operation up to step 4.

5. $\overline{WE}$ is set low to indicate a write operation for time t_{WCS} before $\overline{CAS}$ goes low, and t_{WCH} after $\overline{CAS}$ goes low. $\overline{WE}$ needs to be set low for a total amount of time t_{WP} where $t_{WP} >= t_{WCS} + t_{WCH}$ for the write operation to proceed successfully.

6. Data must be applied to the Data input pins for t_{DS} before $\overline{CAS}$ goes low and should be held for t_{DH}.

7. $\overline{CAS}$ switches from high to low and remains low for t_{CAS}.

8. Column Address should be held for time t_{CAH} after $\overline{CAS}$ goes low. By now Column Address is latched and decoded to select the particular bit-line. The selected memory cell is either charged or discharged depending on whether *1* or *0* needs to be stored.

9. $\overline{CAS}$ and $\overline{RAS}$ returns to their inactive states. They should remain high for time t_{CRP} and t_{RP} for the memory array to be precharged and prepared for the next memory operation.

2.5 Fast Page Mode and Extended Data Out DRAM operations

2.5.1 Fast Page Mode (FPM)

Ordinary DRAM operation as described above allows random access to any memory location. However, in most applications, there is spatial locality in the reference stream to the memory,

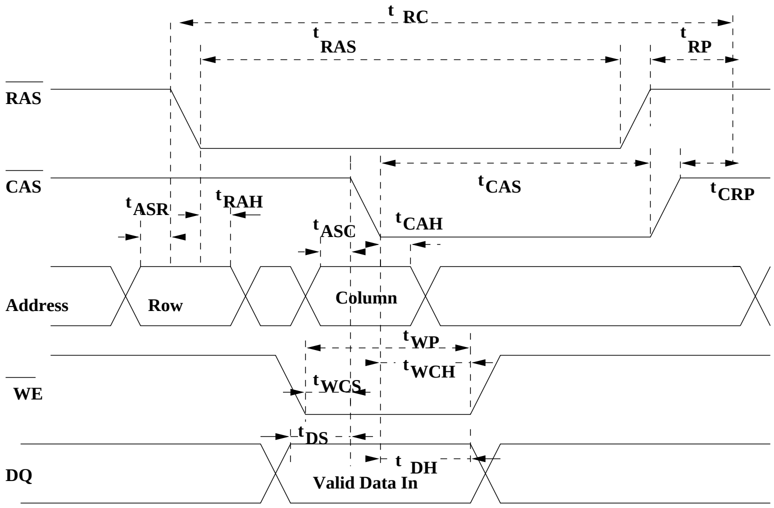

Figure 8: Timing diagram for basic DRAM write operation

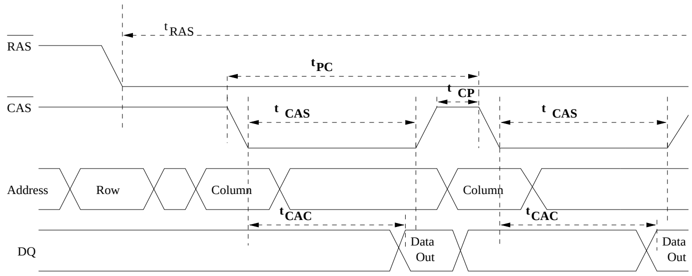

Figure 9: Timing diagram for Fast Page Mode DRAM read operation

i.e. whenever a memory location is accessed, with high probability the next sequential location is accessed next. Recall that a DRAM read operation reads in a complete row in the sense amplifier but only one bit-line is accessed. This row is called a page in the DRAM literature. A page is defined as all the memory locations that have a common row address and are wired to the same DQ. If the next accessed memory cell belongs to the same row, then it can be retrieved just with a new column address without precharging the array. Fast Page Mode DRAM (FPM DRAM) uses exactly this idea. Once the first data is obtained $\overline{CAS}$ is deactivated but $\overline{RAS}$ is not. After the required interval $\overline{CAS}$ is again activated along with the new column address [IBMd]. This cycle continues till the memory address is a page miss when the memory array has to be precharged and a new row has to be read in to the sense amplifier. Notice that this improves the throughput of the memory as well as the access time of subsequent memory accesses. However the access time for the first location is the same as ordinary DRAM. Figure 9 shows the timing diagram of a simplified fast page mode read operation.

2.5.2 Extended Data Out (EDO)

Extended Data Out (EDO) DRAM improves the throughput of the FPM DRAM with a small change to the output buffer control. In FPM DRAM, output data becomes invalid when $\overline{CAS}$ is

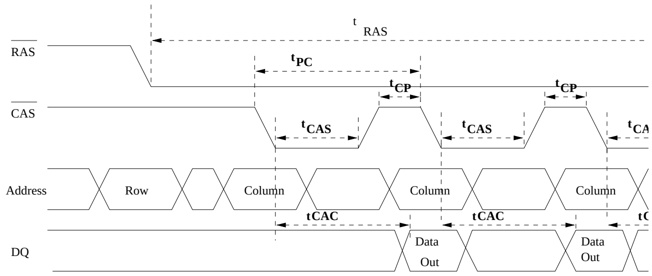

Figure 10: Timing diagram for Extended Data Out DRAM read operation

deactivated. This leaves a very small window to perform the data read and hence it is difficult to increase the operating frequency. EDO allows the data from the current read cycle to be present at the output while the next read cycle begins. This gives more room to read the data and results in faster read cycle [IBMd]. EDO DRAM required minimal change to the existing DRAM interface and became very popular as system main memory. Figure 10 shows the timing diagram of a simplified extended data out read operation. For EDO, $t_{CAC} > t_{CAS}$, where t_{CAS} determines the minimum interval between consecutive reads.

2.6 DRAM Performance Specification

In this section we define the terms used in specifying DRAM performance. The two main performance metrics for DRAM are *Access Time* and *Cycle Time* [IBMe].

2.6.1 Access Time

Access time is defined as the maximum delay from an address probe event to the time when the data appears on the output of the memory device. Memory vendors use two access times: Random Access Time and Column or Page Access Time corresponding to ordinary DRAM and FPM or EDO DRAM.

Random Access Time (t_{RAC}), is defined as the maximum delay from $\overline{RAS}$ going low to the time data actually appears on the output pins of the memory device. (Refer to Figure 7). Note that t_{RAC} is the most important metric for a system when there is little or no locality in the address stream. In that case, even if FPM or EDP RAM is used, most of the accesses will still encounter this random access delay.

Column or Page Access Time (t_{CAC}), is defined as the time between $\overline{CAS}$ going low and the data appearing on the output pin. (Refer to Figure 7) again. Note that with FPM or EDO RAM and high spatial locality in the address stream, column access time will be the dominant metric. (Figure 9, 10).

2.6.2 Cycle Time

Cycle time is defined as the time interval between two consecutive operations. It is the time required to perform the operation plus the time required for recovery (precharging in DRAM context), before another operation can begin. Cycle time is defined differently for ordinary DRAM, FPM

DRAM and EDO DRAM.

Random Access Cycle Time (t_{RC}), is associated with random access for DRAM. It is the sum of the minimum time that $\overline{RAS}$ needs to be low (t_{RAS}), plus the minimum $\overline{RAS}$ precharge or recovery time (t_{RP}), plus the time required for $\overline{RAS}$ transitions from high to low and low to high. Figure 7 shows the random access cycle time. When multiple random access occurs consecutively, t_{RC} will determine the minimum time between two accesses.

Fast Page Mode Cycle Time (t_{PC}), is define for FPM DRAM when data is read consecutively from the same memory row/page. As discussed in Section 2.5.1, FPM allows the user to write to different locations within a row without providing the row address. t_{PC} is defined as the sum of the minimum time that $\overline{CAS}$ should remain low (t_{CAS}), the minimum $\overline{CAS}$ precharge time (t_{CP}), and the time required for $\overline{CAS}$ transitions. For FPM, $\overline{CAS}$ going high terminates the FPM cycle, and hence for this mode, the following invariant should always hold

$$t_{CAS} >= t_{CAC}$$

Extended Data Out Cycle Time (t_{PC}), is defined in the same way as Fast Page Mode Cycle Time. However, as discussed in Section 2.5.2, EDO DRAM cycle time is less than FPM by design. In this case, $\overline{CAS}$ going high does not terminate the data access, but the output drivers continue to drive the data, while the next column address is set up. Notice that now we can have (Figure 10)

$$t_{CAS} <= t_{CAC}$$

and thereby reducing t_{PC} which is sum of t_{CAS}, t_{CP}, and $\overline{CAS}$ transition time.

2.7 Refresh Operation

Since the charge stored in the capacitor decays over time, all the memory cells in the DRAM array have to be refreshed periodically. Refreshing involves reading the memory cell and storing it back internally and is performed a row at a time.

There are various ways in which refresh can be accomplished. We can categorize DRAM refresh according to (i) Refresh Rate, (ii) Refresh Cycle Distribution, and (iii) Refresh Cycle Operation [Mic].

2.7.1 Refresh Rate

Refresh rate is determined by the silicon technology used to manufacture the memory cell and design of the memory cell. For a DRAM array to operate correctly, all the rows need to be refreshed within a certain interval of time called **Refresh Time**. If the refresh time divided by the number of refresh cycle (equal to the number of rows in the DRAM array), is 15.6μsec, it is called a **Standard Refresh** device. If the result is 125μsec, it is called **Extended Refresh** device. Table 1 shows the refresh time for different DRAM technologies.

2.7.2 Refresh Cycle Distribution

Refresh cycles can be distributed in two fashions.

DRAM	Refresh Time	Number of Cycles	Refresh Rate
4Mx1	16ms	1024	15.6μs
256Kx16	8ms	512	15.6μs
256Kx16	64ms	512	125μs
4Mx4	32ms	2046	15.6μs
4Mx4	64ms	4096	15.6μs

Table 1: Standard and extended refresh rate

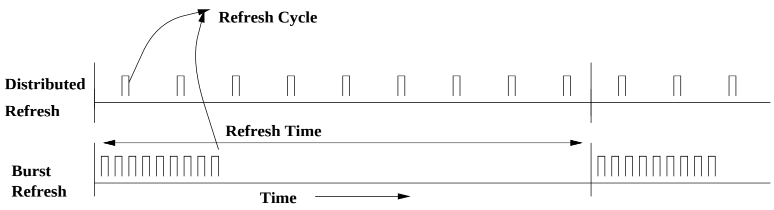

Figure 11: Distributed versus. Burst Refresh

1) **Distributed Refresh**: In this case, refresh cycles are evenly distributed within the refresh time/interval. For example, for a standard refresh device, a refresh cycle needs to be performed every 15.6μsec, so that all the rows get refreshed within the refresh time. The memory is available for read/write during the rest of the time.

2) **Burst Refresh**: Memory array is refreshed in a burst by performing refresh cycle for all the rows consecutively. For the rest of the time, memory will be available for read/write. For example, for standard $4Meg \times 1$ DRAM, 1024 consecutive refresh cycles, with each cycle being $130ns$ (t_{RC}), will require

$$1,024cycles \times 130ns = 133,120ns = 0.133ms$$
$$16ms - 0.133ms = 15.867ms \qquad ,$$

which leaves 15.867ms for performing all the other read/write operations.

Figure 11 shows the difference between two methods. Distributed refresh is more popular since DRAM controller just needs to perform refresh once every 15.6μs independent of the size and configuration of the DRAM.

2.7.3 Refresh Cycle Operation

There are four kinds of memory refresh cycle operations used by DRAM vendors.

1) **RAS Only Refresh(ROR)** In this case the address of the row along with $\overline{RAS}$ signal is applied to the DRAM chip. $\overline{CAS}$ is held high throughout the cycle. It is the responsibility of DRAM controller to generate the row addresses so that all the rows get refreshed within the refresh interval. Figure 12 illustrates the RAS Only Refresh cycle.

2) **CAS before RAS Refresh(CBR)** This the most prevalent refresh mechanism in DRAM industry. In a CBR refresh cycle, first the $\overline{CAS}$ signal is lowered, followed by the $\overline{RAS}$ signal. $\overline{WE}$

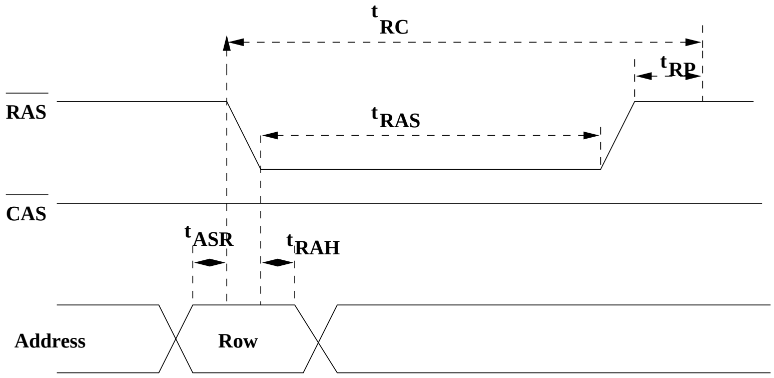

Figure 12: Timing diagram for Read Only Refresh Cycle

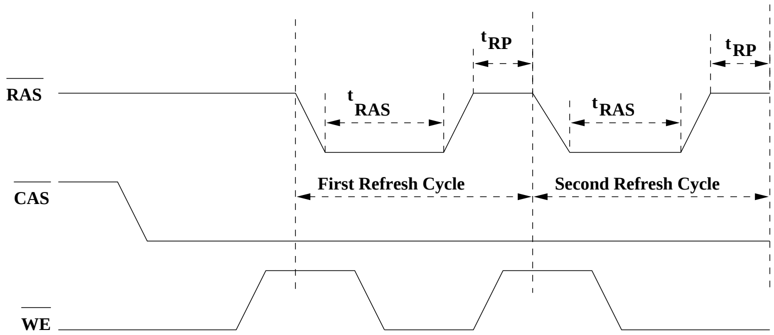

Figure 13: Timing diagram for CAS before RAS Refresh Cycle

must be held high while $\overline{RAS}$ is lowered. Each transition of $\overline{RAS}$ from high to low does one refresh. The row address is not explicitly provided. An internal refresh counter generates the address of the row to be refreshed.Figure 13 shows two consecutive CBR refresh cycles. This method is simple since the memory controller need not generate any address, and also saves power as the transitions on the address bus is avoided.

3) **Hidden Refresh** In this case the read/write operation is followed by a refresh operation. After a read/write cycle, $\overline{CAS}$ is left low, while $\overline{RAS}$ is toggled to high (for t_{RP}), and then brought back to low. Since $\overline{CAS}$ was already low, when $\overline{RAS}$ is lowered, this performs a CBR refresh cycle. Note that the time taken to refresh a row is not "hidden", it still requires two cycle time, $2 \times t_{RC}$. (Refer Figure 14). But the advantage is that data remains valid for a longer time on the output pin.

4) **Self Refresh (SR)** Also known as sleep mode uses an internal oscillator and a counter to keep track of the refresh interval as well as the row address. This is mostly used for battery-backup mobile applications since the device uses extremely low current in this mode. This refresh is initiated by CBR cycle with $\overline{RAS}$ active for a minimum of 100μsecond. To exit the refresh mode, $\overline{RAS}$ and $\overline{CAS}$ are brought high.

Note that all these refresh cycles can be performed in burst mode, or distributed evenly across the refresh time.

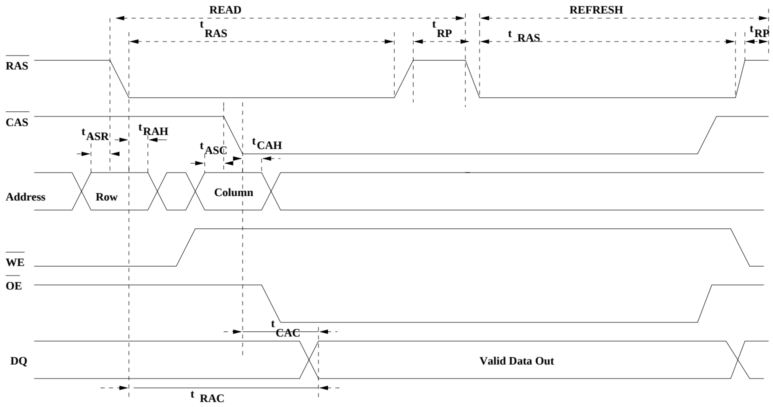

Figure 14: Timing diagram for Hidden Refresh Cycle

3 Different DRAM Architectures

Until very recently, there was nothing interesting in the field of DRAM architecture. For a long time, DRAM industry saw a phenomenal increase in the DRAM chip density and decrease in access time due to process and photo-lithography advances. However, there was no serious rethinking of the DRAM interface to achieve higher bandwidth. As processor speed increased enormously, and new applications, like multimedia and graphics started to emerge, they pushed the limit of ordinary dynamic random access memory. These processors and applications demanded higher bandwidth and shorter latency from the memory subsystems. This resulted in both evolutionary and revolutionary DRAM architectures that appeared in the market in the last 5 years. The improvement in performance has resulted from changing the base DRAM architecture without any change in the die size. This is indeed a very fascinating time to watch the dynamic random access memory industry. In this section, we examine the evolution of the DRAM architecture.

3.1 Page Mode Operation: Fast Page Mode and Extended Data Out DRAM

Originally, DRAM mode of operation was single read and write. Then it was noticed that there is lot of spatial locality in the address stream, specially because of cache line read. To exploit this locality, the architecture was changed to incorporate page mode operation, where an entire page can be written or read by changing the column address and column address strobe. This is called the **Fast Page Mode (FPM)** DRAM which we have discussed in Section 2.5.1. The *random access* time of the fastest FPM DRAM is 60 ns for 66 MHz bus rate. The *page mode cycle* time for this DRAM is *35ns*(28.5 MHz), a significant performance difference with respect to 66 MHz bus. When the DRAM is accessed in burst mode (during cache line fill), usually 4 consecutive locations are addressed either in sequential or interleaved fashion. Assuming the page is already open (*page hit*), the peak burst rate using 66 MHz system bus and 60 ns FPM DRAM will be 5/3/3/3 cycles.

Extended Data Out (EDO) DRAM was introduced in 1996 and is described in Section 2.5.2. In this architecture, during read cycle, the output data is latched in the buffers, so that when output drivers are putting the data on the bus, next column address could be decoded. This improved the page mode cycle time from 35ns for FPM to 25ns(40MHz) for EDO, a 40% increase in the page mode performance. However the random access time still remained the same. The peak burst rate using 66MHz system bus, and 60ns EDO is 5/2/2/2 cycles. This implies reduction of 3 wait states; however the wait states for the first and critical data still remains the same. This architecture became very popular among the system architects since it required little or no change in the memory controller. However, EDO RAM can work at maximum bus speed of 66 MHz. As faster CPUs demanded faster system bus rates, moving from 66 MHz to 75 MHz and finally 100 MHz, EDO DRAM, with its asynchronous mode of operation (address, data and signal setup and hold delay) was not a viable alternative any more.

3.2 Burst mode operation: Burst Extended Data Out (BEDO)

For many applications, it is required to perform a long sequential access within an open page. It would be nice if the DRAM can be architected so that it can generate the sequential addresses by itself, thereby reducing the page mode cycle time further. This bursting technique requires the memory controller to provide only *the starting address*, *burst type* and *burst length.* The burst length determines the number of addresses to be generated. An internal address counter is used to generate the sequential or interleaved memory addresses depending on the burst type. This internal address generation eliminates the delays associated with the detection and latching of external column addresses on DRAM pins.

The first DRAM that supports burst mode operation was Synchronous DRAM or SDRAM which we will discuss in the next section. In this section, we describe *Burst Extended Data Out (BEDO)* RAM that was a simple modification of EDO RAM [IBMc].

BEDO can support a burst length of only 4 as in either sequential or interleaved manner. BEDO differs from EDO in two ways. First, an additional latch stage is added to the output so that in the second page mode cycle, data will appear in a shorter time from the application of $\overline{CAS}$. The delay in receiving the first data in not increased however from EDO since that delay is dominated by t_{RAC} or random access time rather than page mode cycle time. The second difference with EDO is the addition of internal address generation counter. These features together decreased the page mode cycle time to 15ns and hence the burst rate is 5/1/1/1 cycles.

BEDO RAM never became popular among the vendors. This architecture was introduced as a less expensive alternative of SDRAM. As SDRAM moved to volume production quite quickly, BEDO never became popular since it has the same limitation of 66MHz system bus rate as EDO.

3.3 Synchronous Operation: Synchronous DRAM (SDRAM) and Double Data Rate SDRAM (DDR SDRAM)

Until early 1997, all DRAMs were asynchronous, i.e., they were not synchronized to the system clock, and the CPU had to wait for a specific period of time to receive the data and send the next request. As CPU speed, as well as the system bus speed increased, it was difficult to reduce cycle time in asynchronous mode of operation due to noise margin etc. Also, $\overline{RAS}$ had to be held active throughout the entire access period. This implied the accesses were completely non-pipelined; DRAM cannot accept new control or address information during this time. To solve this problem, system architects resorted to multiple banks of DRAM, and switched between them in complex

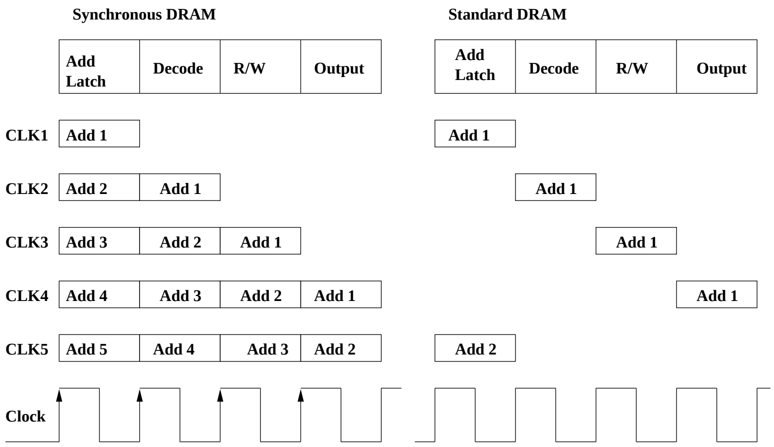

Figure 15: Pipeline nature of Synchronous DRAM compared to non-pipelined nature of Standard DRAM

fashion to hide the inherent latencies.

The **Synchronous DRAM (SDRAM)** represented a significant re-thinking of the functions of DRAM. Internally, the basic operations and architecture remained unchanged with word-line, bit-line, cells, and sense-amplifiers. However the interface was changed from asynchronous to synchronous mode of operation. All address, control and data inputs were referenced according to a master clock. This increased noise-immunity since the input pins are accessed during a narrow setup and hold window around the rising edge of the clock. The key parameters are now expressed in terms of system clock(latency) rather than as time. As the control and address signals need to be asserted only for one clock, SDRAM could accept new address and control information in every clock [Hit]. This improved interface allowed the memory system to meet the throughput requirement of today's high-speed microprocessors.

In addition to the synchronous interface, SDRAM employed some other optimizations to further improve the performance [IBMc].

1. **Burst Mode:** SDRAM employs bursting technique as described for BEDO so that once the burst has started, all the remaining data can be supplied at the rate of *one per clock cycle*.

2. **Mode Register:** A mode register was added so that *burst length*, *burst type* as well as the *column access latency* ($\overline{CAS}$ latency) can be programmed by the memory controller according to the need of the particular processor or application. Burst length of 2,4,8 or full page is supported. $\overline{CAS}$ latency can be set to 1,2 or 3 cycles.

3. **Multiple Banks:** It includes multiple independent memory banks (currently 2 per chip), to increase the page hit rate.

4. **Auto Precharge:** An open page in DRAM bank, should be closed properly before another row/page can be accessed. In case of FPM/EDO, user issues signals to precharge the row and column explicitly. SDRAM employs an auto-precharge technique that can automatically close the page after it has been accessed. The user decides whether to use the auto-precharge or

not for every access. By interleaving accesses to two banks and using auto-precharge, DRAM can hide all the precharge and access latency, and provide seamless stream of data at peak bandwidth.

Another difference between SDRAM and previous DRAM is that, even though same signals ($\overline{RAS}$, $\overline{CAS}$ etc) are used for SDRAM, they are interpreted as bits of an opcode forming a command set rather than as signals. SDRAM employs a state machine that decoded these commands, changes its state and controls actual memory access. We will briefly describe a few important commands here [Hit].

1) **ACTV:** The active command is issued with the bank and row address. As a result of this command, row address is latched and decoded, and the appropriate row is activated i.e. read into the sense amplifiers. After ACTV command, read, write or precharge operation can be performed to that row.

2) **READ and WRITE:** These commands are used to read/write from the row previously activated with ACTV command. These commands specify the column address within the open row. For READ command, data becomes available after a number of clocks as defined by $\overline{CAS}$ latency, and then subsequent data appears at every clock for the size of burst length.

3) **READA and WRITEA:** These commands are similar to READ and WRITE except that after the access is complete, an automatic precharge operation is performed to close the active row, and prepare the bank for subsequent access.

4) **PRE and PALL:** These commands are used to explicitly precharge one or both the banks.

With synchronous interface, SDRAMs have achieved clock rate of 66MHz, 83MHz, 100MHz and recently 133MHz. A 256Mbit SDRAM(8Mx32) chip at clock rate of 133MHz, can achieve a theoretical peak bandwidth of 532MB/sec. Even though initially SDRAM implementation was quite challenging, it has been solved by now, and today most of the high-end workstations are based on SDRAM chip. Note that, at 66 MHz system bus rate, SDRAM and BEDO has the same burst rate of 5/1/1/1. However, BEDO cannot be used for 100MHz bus rate, whereas SDRAM can. Also, we should note that all the architectures that succeeded SDRAM are synchronous, even though the nomenclature always does not state so explicitly.

Double Data Rate SDRAM (DDR SDRAM) is an upgrade of SDRAM that allows the chips to deliver data twice as fast by using transferring data on both rising and falling clock edges [Bur97]. This high data transfer data satisfies the bandwidth requirements of high-end desktop PCs, high-performance graphics adapters, workstations and servers. Internally, the architecture is same as the standard SDRAM, with a chip area overhead of 4% over SDRAM due to the circuitry for strobe generation and synchronization. The main differences between single and DDR SDRAM are reduced $\overline{CAS}$ latency, reduced number of burst length choices, and addition of one-cycle write latency. In addition, DDR uses differential rather than single ended clock. A 256Mbit DDR SDRAM(8Mx32) chip at clock rate of 133MHz, can achieve a theoretical peak bandwidth of 1064MB/sec. The main advantage of DDR is that it has the same packaging and organization as SDRAM and hence, does not require a new interface, and scalable with the current generation of

SDRAM.

3.4 Reducing Access Latency: Enhanced Synchronous DRAM (ESDRAM) and Cache DRAM (CDRAM)

Most new DRAM architectures try to obtain higher burst rates. However, the initial latency of a page hit or miss remains the same; so does the penalties caused by random access cycle time(t_{RC}), and the precharge time(t_{RP}). Enhanced Synchronous DRAM (ESDRAM) from Enhanced Memory Systems Inc. and Cache DRAM (CDRAM) from Mitsubishi Electric Corporation are two architecture that use on-chip SRAM cache memory to decrease the latency of memory access.

3.4.1 Enhanced Synchronous DRAM(ESDRAM)

ESDRAM is an evolutionary modification to standard SDRAM. It is a superset of SDRAM and remains pin and command compatible with standard SDRAM [Enh]. The goal of ESDRAM is to reduce the latency, increase the bandwidth and allow for concurrent operations to the same bank. SDRAM with two banks allows the user to perform simultaneous operation on both the banks, thereby hiding precharge and cycle time delays. However, if order for this to be useful, data should be stored in such a manner that the SDRAM can ping-pong between two banks continuously, and is difficult to achieve. ESDRAM allows to start a new row access on a page miss, while bursting out data from the previous row of the same bank. This hides the precharge and row access latency, and can provide seamless data burst from the same bank instead of alternating between two banks as in the case of standard SDRAM.

The new features of ESDRAM compared to SDRAM are the following.

1. **Faster DRAM Core:** ESDRAM has faster DRAM array ($t_{RAC} < 27ns$) than standard SDRAM. This fast array reduces the latency of page misses as well as the precharge and cycle times.

2. **SRAM cache:** 16Mbit ESDRAM integrates a 8Kbit high speed SRAM cache with the DRAM array. This 8Kbit SRAM cache consist of two 4Kbit integrated row registers for each DRAM bank. The first advantage of the cache is that random column access can be performed at SRAM speed. Also, when a DRAM row is activated, the data gets latched in the sense amplifiers. Once a READ command is issued to the bank, the entire row is automatically transferred from the sense amplifier to the SRAM cache. Subsequent READ operations access the data from the SRAM cache, and do not touch the DRAM array. Hence, the DRAM bank can be precharged and another row can be activated. The data corresponding to the new row will exist in the sense amplifier and hence, will not interfere with the data in the SRAM cache. This way the precharge time and the row activation time can be completely hidden for a memory bank.

3. **Write Transfer Mode versus No Write Transfer Mode:** The ESDRAM incorporates a programmable mode register to select either write transfer mode or no write transfer mode for a particular application, thereby optimizing the memory performance and compatibility.

   In the **Write Transfer Mode**, a write miss causes the new row of sense amplifiers to be activated and transferred to the cache, thereby overwriting the previous information in the

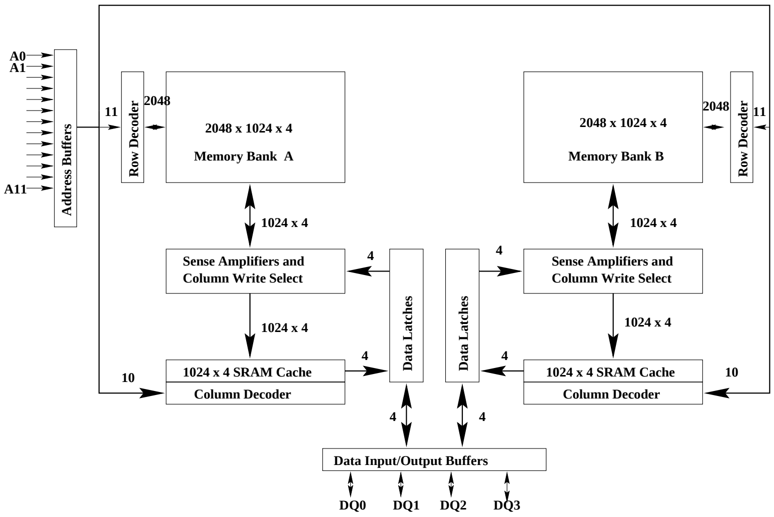

Figure 16: Block Diagram of ESDRAM

cache. Since, the contents of the sense amplifier and the cache are the same, this mode allows only one DRAM page per bank to be available for reading/writing.

In the **No Write Transfer Mode**, a write miss leaves the new row in the sense amplifiers, and does not transfer the row to the cache. Since the content of the cache remains unaffected, this mode allows two pages to be open simultaneously per bank, one (in the cache) for reading, and one (in the sense amplifier) for writing. This mode is ideal for systems that reads data from one page of the memory, processes the data, and writes the result back to another page of memory, and also for application that does frequent data copying from one page to another. In both cases, ESDRAM can have both the READ and WRITE page open simultaneously in the same bank.

3.4.2 Cache DRAM(CDRAM)

The main idea of Cache DRAM(CDRAM) was to provide a cheap second-level cache on memory chip that complements the on-chip memory on the processor. Unfortunately, CDRAM was only available from Mitsubishi and the architecture never became popular. The goal of the architecture was to provide high cache hit rate, and low access time so that it becomes effective as secondary cache.

The simplest form of memory based cache is the sense amplifiers. However, this cache is direct mapped with a huge block size of more than 1Kbits. Hence, the hit rate is not high. CDRAM employs a more flexible cache architecture [H$^+$90]. A Cache DRAM with 1Mbit (256Kx4) is organized as 4 memory planes of 256Kbits each. In addition, the RAM contains 256Kbits of DRAM cells for ECC. The SRAM array consists of four 2Kbit planes. Each SRAM plane is divided into 4 blocks to provide 4-way associativity. The block size is 8bits per I/O, i.e. the cache is organized as 8bit X 64 sets X 4 ways X 4 I/O for 8Kbits. The vendors expected to achieve 90% hit rate with this simple implementation. At the beginning of the access cycle, SRAM cache is

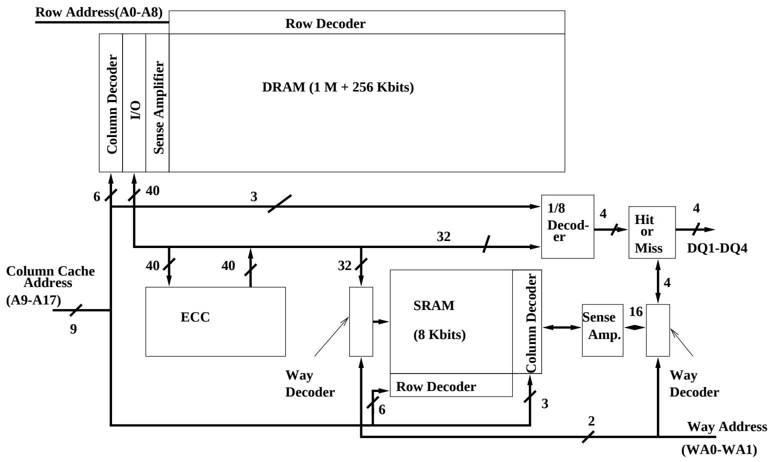

Figure 17: Block Diagram of Cache DRAM

looked up first. In case of a cache hit, cache controller provides the way address to select the right block. In case of a cache miss, DRAM is accessed with $\overline{RAS}/\overline{CAS}$ cycle. Eight bits of data from each DRAM, 40 bits in parallel are transferred to the SRAM and ECC unit via the 40 bit wide internal bus. After error checking and correction, 32 bits are transferred to the proper location in the SRAM plane (using column address), and 40 bits are transferred back to the DRAM. In case of a write hit, data is written both to the cache memory as well as DRAM core, and write-miss writes to the DRAM core, and transfers or does not transfer the data to the cache, depending on the user choice. The disadvantage of the cache DRAM is that the cache controller and the cache tags have to be implemented in the memory controller for Cache DRAM to work.

3.5 Revolutionary Interface

By the end of this century microprocessors with clock frequency of 500 MHz will be very common among the desktop PCs. To keep up with this high speed processors, the memory bus frequency has to be increased manifold from 133MHz SDRAM available today. Future applications will demand a high bandwidth of 3.2GB/sec. To reach this bandwidth and close up the gap between processor and memory, a revolutionary memory solution is necessary. In existing technology, bandwidth is increased by widening the data path and adding more memory devices in parallel. The main disadvantages of this approach is that wide data buses are expensive, difficult to go beyond 200MHz bus rate, and as the size of the minimum memory units increases, system architects are forced to put more memory than necessary just to achieve the bandwidth.

In the future approach, memory devices will use a narrow bus of 16bit. The high bandwidth is achieved by increasing the bus frequency up to 800MHz. Memory upgrades are done by adding more memory devices in series instead of the conventional approach where the devices are added in parallel. This way, memory can be increased incrementally, thereby improving the flexibility of memory system design. Also, in today's system memory devices are accessed using various signals like $\overline{RAS}$, $\overline{CAS}$, and so on. In this new approach, memory controller sends request packets that activates the read or write operation. Essentially, the interface between CPU and memory is becoming more like an open network, rather than a closed bus.

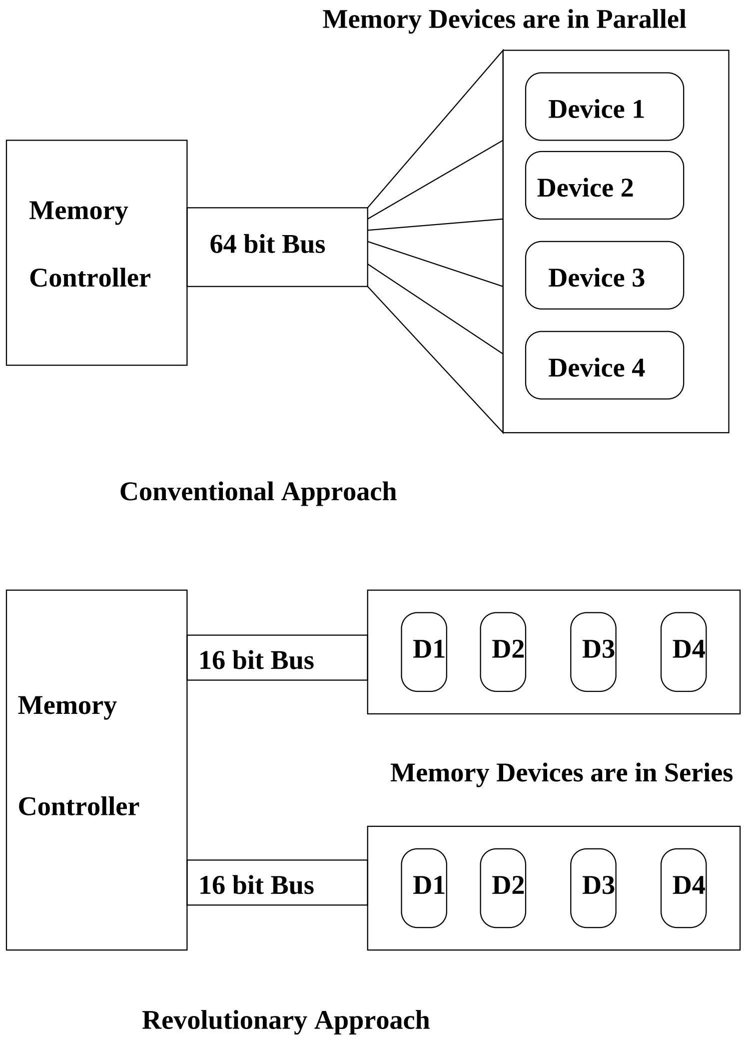

Figure 18: Revolutionary versus Conventional Interface

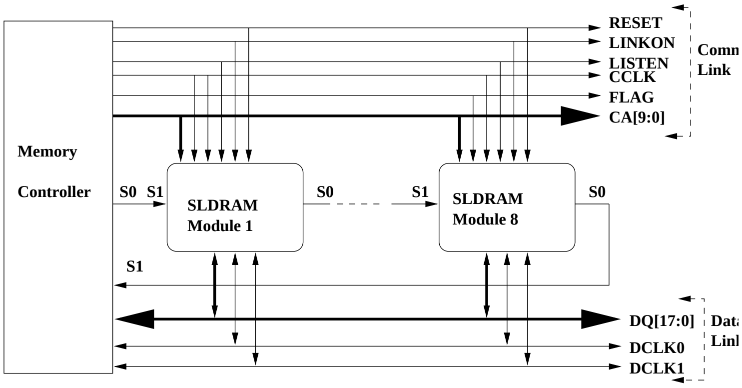

Figure 19: SLDRAM Memory System

In this section, we describe two architectures that use this new approach.

3.5.1 Synchronous-Link DRAM (SLDRAM)

Synchronous-link DRAM (SLDRAM), is a new open memory interface specification developed through the cooperative efforts of leading semiconductor memory manufactures. SLDRAM decouples the internal DRAM address and control signals from the data interface, thereby allowing the memory to achieve high-bandwidth data transfer. Like DDR SDRAM, SLDRAM uses both the rising and falling edge of the clock to transfer data. The first SLDRAM devices are already available as 72Mbit chips with 18 bit I/O port (16 bit for data and 2 bits for ECC), operating at 200MHz clock (400Mbps/pin). This chip achieves a peak bandwidth of 800MB/sec. Interfaces with bandwidth of 600, 800 Mbps/pin or even 1Gbps/pin are expected to be manufactured as they become cost-effective [GV97].

Bus Topology: SLDRAM multi-drop bus has one memory controller and up to eight loads, where each load can be either a single SLDRAM device or a buffered module with many SLDRAM devices. There is a unidirectional command link to transfer command, address, and control information, and a bidirectional data link to transfer read/write data. Both the the links operate at the same rate (say. 400 Mbps/pin). The command link consists of the following command signals (CA[0..9]), differential free-running clock($CCLK/CCLK^*$), LISTEN to put the device in power-saving standby mode, FLAG to indicate the first word of a command, LINKON to put the device in shutdown mode, and RESET to put SLDRAM in a known state on power-up. An SLDRAM command packet consists of four consecutive 10 bit words on the command signals (CA[0..9]). The bidirectional data link consists of data signals DQ[0..17], and two differential clocks ($DCLK0/DCLK0^*$ and $DCLK1/DCLK1^*$). Read and write data packets consist of minimum of four consecutive 18 bit words on the DQ[0..17] lines. A daisy-chained serial bus (serial in and serial out on each chip) is used during power-up to synchronize the SLDRAMs and assign unique IDs to each.

Internal Architecture: A 64Mbit DRAM array is organized as eight banks, each structured as 128K words by 72 bits. A bidirectional data path shared among the eight banks provide 72 bits of read/write data at the 100MHz core frequency. The 72bit word is transferred over the 18 bit

I/O interface in a burst of four 18 bit words. All transactions begin with a request packet. The SLDRAM compares the chip ID field of the request packet with its local ID register. If there is a match, the SLDRAM passes the data/bank commands on to the address sequencer and immediately executes the control and event commands. Data from either one or two columns in a page may be accessed with a single request packet, with a two column request resulting in a continuous burst of eight 18 bit data words.

3.5.2 Direct Rambus (RDRAM)

Direct Rambus (RDRAM) is another revolutionary memory system developed by Rambus Incorporate, which provides more bandwidth from a single chip than possible with any other DRAM memory technology available today [Cri97]. It replaces memory chips, bus, interface as well as memory controller with a single standard specification. The mail difference between Direct RDRAM and SLDRAM is that the former is not an open standard as opposed to the later. The main elements of this solution are the Rambus Memory Controller, Rambus Channel,and the RDRAM devices on the channel.

Rambus Memory Controller: Memory controller is the bus master of the memory subsystem. It consists of two separate logic blocks: the Rambus ASIC Cell(RAC), which acts as the physical and electrical interface to the channel, clocked at channel frequency which can be as high as 400MHz, and the Rambus Memory Controller(RMC), that acts as an interface between the processor or other bus masters and the RAC. The responsibilities of the RMC are taking requests from the processor or any other bus master and converting them into the Direct Rambus protocol, keeping track of the RDRAM states and refresh, and optimizing channel performance.

Direct Rambus Channel: RDRAM channel creates an electrical connection between the controller and the RDRAM devices. This channel uses 30 high-speed signals that use both edges of a 400MHz clock, creating a data transfer rate of 800Mbps/pin. The signals can be divided into 4 categories: 16/18 bit bidirectional data bus, 3 bit unidirectional row control bus, 5 bit unidirectional column control bus, and the differential clock pairs CTM (Clock To Master and its complement), and CFM (Clock From Master and its complement). This is different from SLDRAM where row and column buses are not separate. All read data signals are aligned to CTM and all write data as well as column and row packet are aligned to CFM clock. This way packets and the corresponding clock always flow in the same direction, making it easier for the controller, and RDRAM device to reliably latch all the inputs irrespective of the location of the device on the channel. All the buses operate independently of each other, allowing a row command, a column command, and data to be transferred at the same time to different banks of an RDRAM or to different RDRAMs. This concurrency, in addition to the pipeline nature of the RDRAM allows data bus to achieve utilization of up to 95%. A single channel can support up to 32 Direct RDRAMs on up to 3 Direct Rambus RIMMs. To support more devices, up to two repeater chips can be used to make it 128 devices.

Packet Characteristics: Data and control information is transferred across the channel in a packet-oriented protocol. Each packet consists of a burst of 8 bits over the corresponding signals of the channel, so that row command has 24 bits, column command 40 bits, and data packet is 128 bits.

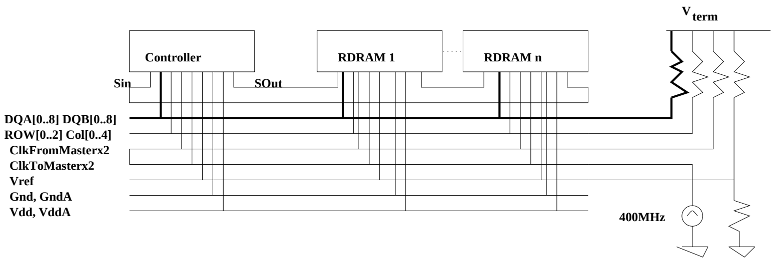

Figure 20: Direct RDRAM Memory System

Direct RDRAM Core: Direct RDRAM core is same as any standard SDRAM. The 64Mbit DRAM core of the Direct RDRAM consists of 16 independent banks with each bank organized as a 32k X 128 bit DRAM. The wide (x128) core operates at the single edge of 100MHz clock, and the interface logic converts this 128 bit data packet, to transfer across the 16 bit data channel on 8 clock edges of a 400MHz clock. The DRAM core uses standard $\overline{RAS}/\overline{CAS}$ signals which are issued by interface logic. Direct RDRAM also introduces the concept of dependent banks. The sense-amplifiers are shared between two adjacent banks, giving room to allow more banks in a smaller die area. However, the drawback is that once a bank is activated, it needs to be precharged before either of the two adjacent banks can be activated, so that only half of the total number of banks can be open simultaneously. In addition, Direct RDRAM has a write buffer inside the chip, so that latency sensitive READ operation need not be pending for a WRITE operation to complete.

3.6 Other Innovative Architectures

In this section, we discuss two other innovative memory architectures that try to improve the performance efficiency to the current memory cores. One is Virtual Channel Memory (VCM) from NEC Semiconductor (it has recently become JEDEC standard), and the other is Multibank SDRAM (MDRAM) from MoSys Incorporation.

3.6.1 Virtual Channel Memory (VCM)

Virtual Channel SDRAM is designed to improve the performance of the prevalent multitasking, multiple masters, interleaved access scenarios in desktop PCs. In this kind of environment, the access stream of each memory master has spatial, and temporal locality, as well as fixed block size, and a burst length. However, interleaved memory accesses from different masters destroy this locality. VCM architecture provides each memory master with its own programmable virtual memory channel. Each channel is equipped with a data row buffer and its independent operating modes. The system memory controller associates these channels to the memory masters. Hence, each channel constitutes a fast, dedicated memory path for each memory master to access the memory, and hence reduces the conflicts from other memory masters.

VCM is a superset of standard SDRAM and pin, and command compatible with SDRAM [NEC]. Internally, the memory core of VCM is same as the SDRAM. However, logically each row is divided into fours segments, where a segment is the unit of operation between the memory core and the virtual channels. Current 64Mbit VCM provides 16 virtual channels and one dummy channel to be used as a write buffer. In addition, it provides various prefetch and restore commands to transfer

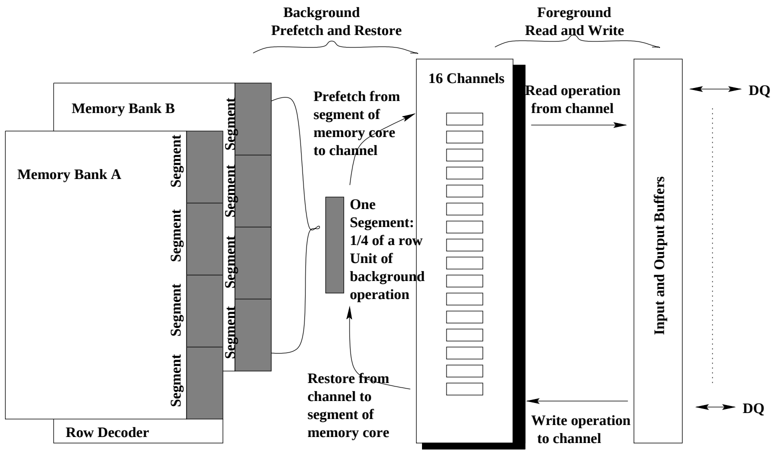

Figure 21: Memory Operations of Virtual Channel Memory

data between the memory core to the virtual channels. After a row in a bank is activated, prefetch commands can transfer one/multiple segments of that row to one/multiple virtual channels. After this transfer, all the read/write operations can take place between the virtual channel, and the output pins of the chip. A restore operation transfers data from one/multiple virtual channels to one/multiple segments in a row. After the data is transferred, that particular row can be activated to transfer the data back to the DRAM array. Any channel can store data from any row and vice versa, i.e., they are fully associative.

The read/write operations on the channel (foreground operations), can operate concurrently and independently with the DRAM bank operations like Active, Precharge, Prefetch and Restore (background operations), thereby hiding the latency associated with the memory array operations. The memory controller should try to schedule the read/write operations with the high-speed channels as much as possible to minimize the overhead associated with the DRAM bank operations. Apparently, VCM can improve performance as much as 25% at the systems level.

3.6.2 Multibank DRAM (MDRAM)

Multibank DRAM (MDRAM), is again an extension of SDRAM optimized for high-bandwidth applications. MDRAM with 16bit interface can operate at frequencies up to 166MHz(333Mbps/pin) providing 666MB/sec of peak bandwidth [Mosb].

Internally, MDRAM can be viewed as an array of independent 256Kbit DRAMs, each with a 32bit interface.This allows the DRAM to be incremented in granularity of only 256KB, thereby allowing low cost solution for systems that require high bandwidth with small amount of DRAM. The control of each bank is local to the bank than being centralized. Programmable bank ID registers are used to map banks to the logical address space, and perform internal address recognition within the bank just as in SLDRAM or direct RDRAM. All the banks are tied to a common internal bus. The external bus is simply a buffered version of the internal bus seen through a bus repeater.During data transfer, both clock edges are used for synchronous transfer of 32 bits per clock cycle on 16 pins. The small bank size, and the simplicity of the repeater allow extremely short CAS access latency of 13.8ns, and multiple independent banks hide the precharge and row access penalty. The main disadvantage of MDRAM is that it requires short buses, thereby allowing only up to 8MB of memory on a single bus.

3.7 Application Specific DRAM Architectures

Workstation and PC users are looking for interactive, realistic 3D graphics from their games and multimedia applications. However, visually appealing 3D graphics images put a tremendous demand for high memory bandwidth. A typical high-end subsystem with a 1028 X 1024 pixel display refreshed at 72Hz and using true color (24 bits/pixel), requires a data transfer of 284 MB/sec only to keep the screen refreshed. In addition, 3D graphics subsystems require Z buffer for depth comparison, as well as texture memory. Together, they can easily place a bandwidth demand of 500MB/sec. Standard DRAM was not suitable to these bandwidth hungry applications. This pressure for high bandwidth resulted in application specific memories for graphics applications. Today, once again, SDRAM, ESDRAM, SLDRAM, and Direct RDRAM are capable of handling multimedia applications as well as system memory specific applications. However, in this section, we will take a quick look at some of the innovative architectures for application specific memory.

3.7.1 Video RAM (VRAM)

To maintain high quality image on the screen, the screen data must be refreshed at regular interval from the frame-buffer memory. At the same time, processor needs to update the frame-buffer and concurrent request for the frame-buffer memory will create a bottleneck. A conventional solution to this bottleneck has been dual-port Video RAM (VRAM). VRAM combines DRAM and SAM (Serial Access Memory) as frame buffer on a single chip, and separates the refreshing bus from the rendering bus. Video RAM includes a series of special registers called SAM registers tied to a serial port. These registers access and store a portion of a page of data from the DRAM array, then the data is clocked out of the serial port to a D/A converter at very high speed. This allows high speed serial streams of data to the video monitor continuously, while the random port is available for almost 100% of the time for update [IBMf].

The SAM register in the Video RAM can be either *Half Depth SAM* or *Full Depth SAM*. In the case of Full Depth SAM, the size of the SAM is equal to the size of a DRAM page, while for Half Depth SAM, it is half the size of the DRAM page. The tradeoff is shorter time to transfer data from DRAM to SAM, as opposed to larger chip size. The transfer operation itself, can again be of two types. In *Full Transfer Mode*, the whole SAM is filled up in one cycle, whereas in *Split Register Transfer Mode* half the SAM is filled up, while data is being read from the other half. In this way, there is no possibility of overlap between transfer from DRAM to SAM, and the reading from the SAM and seamless serial data can be provided to the video monitor [IBMb].

In addition to the serial port, Video RAM provides some other nice features that helps to improve the performance of graphics applications. Among these are:

1. Color Register: stores the color data for filling up large area with block write operation.
2. 8-Column Block Write: writes the content of the color register to 8 columns of the DRAM in single cycle. This is useful for filling large areas such as polygon with a single color.

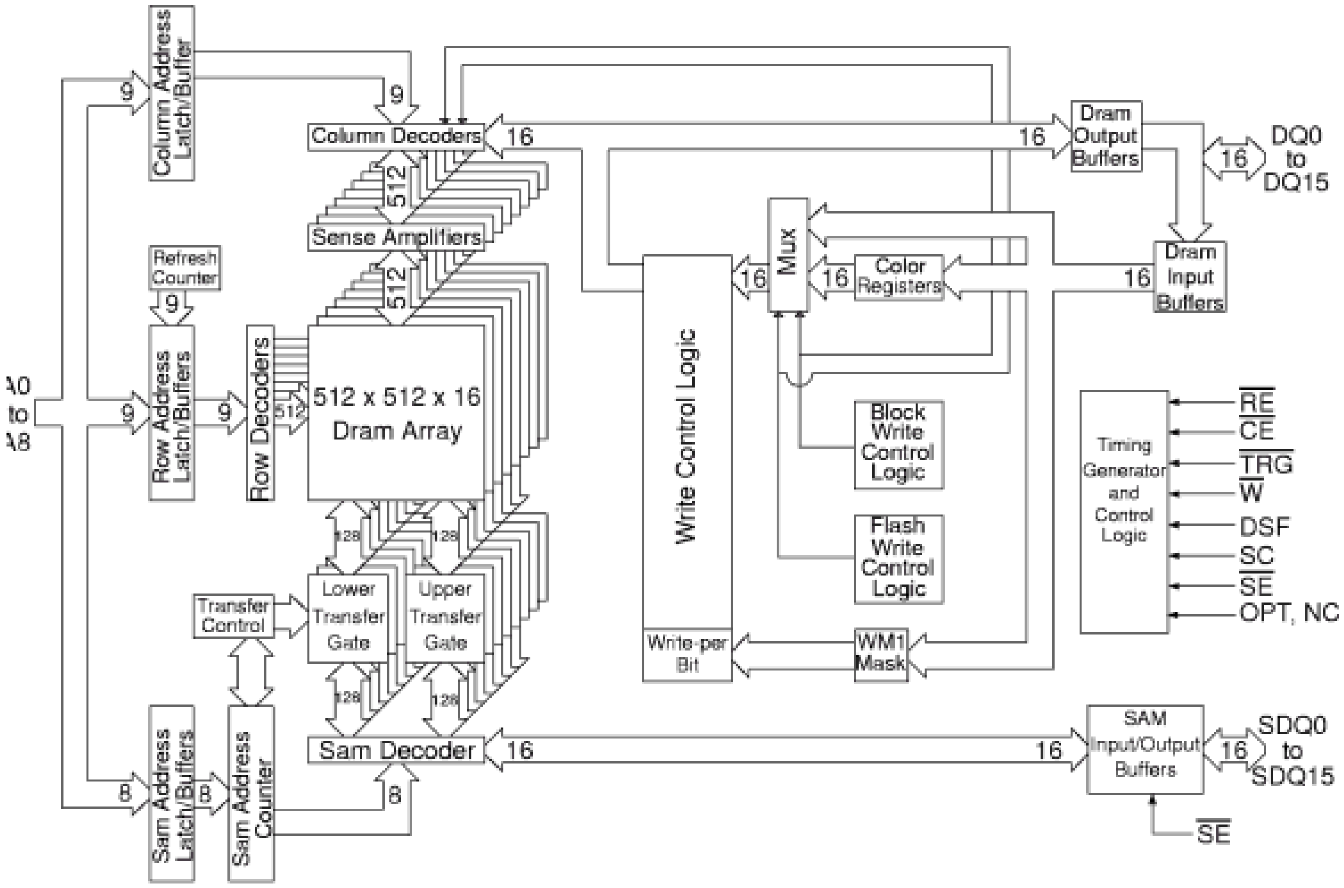

Figure 22: Block Diagram of a standard 4Mb VRAM

3. Mask Register: stores mask data to prevent certain memory locations from getting written during block write.

4. Write-per-Bit: used in conjunction with mask register, it is a temporary masking operation to decide which memory locations get written.

5. Flash Write: clears an entire DRAM row in a single operation.

The die area for Video RAM is almost two times that of standard DRAM due to the addition of serial access port, and hence it is very expensive compared to standard DRAM. As a result, it is only used in very high-end systems. High performance low-cost frame buffers required alternative solutions.

3.7.2 Window RAM (WRAM)

Window RAM (WRAM) developed by Samsung Semiconductor is a modified version of the conventional VRAM specifically for window-based graphics user interface operations [Bur93].

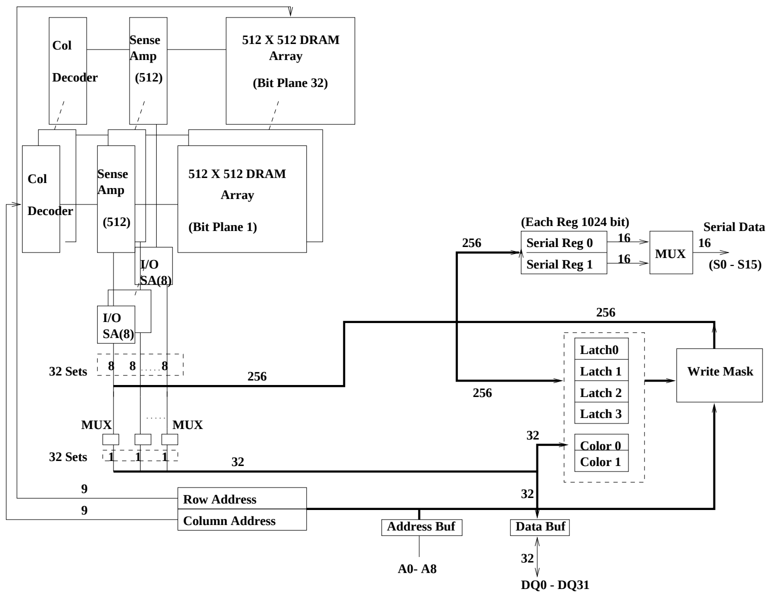

Figure 23: Block Diagram of a standard Window RAM

Internally, a 8Mbit WRAM is organized as 32, 512x512 bit planes, and is connected through a 256 bit bus to the rest of the logic on the chip. These 256 bits are multiplexed to 32bits for host port random access operations. However, 256 bits are used directly to feed two 64x16 bit *serial registers*. In turn, the serial registers are used to feed 16 bit serial port through a 2:1 multiplexer. While one serial register is getting loaded, the other could be used for serial transfer. It requires 4 cycles to load a serial register with 256bit wide bus. Unlike VRAM, data can only be transferred out of the video port, instead of bi-directional operation, thereby simplifying the I/O structure. In addition to the serial registers, data from the memory can be routed to either *data-latch blocks* that holds four 256 bit words, or two 32bit *color registers* for foreground/background color, or two 32-bit *mask registers*. WRAM performs scroll and block move operations quite fast by first moving the data to the data-latch blocks in 4 cycles, and writing them back to DRAM in a new location in 4 more cycles. In addition, these operations can be performed in the background while the serial port, or the host random port is getting accessed. The small size of the on-chip serial register together with the elimination of serial write operation reduces the storage size by 50% compared to VRAM. Hence, WRAM provides frame buffer operations twice as fast as VRAM at a much lesser cost. Unfortunately, it offers little cost advantage over VRAM, because it is only provided by one manufacturer.

3.7.3 Frame Buffer RAM (FBRAM)/3D RAM

Mitsubishi first designed Cache DRAM and further optimized it in cooperation with Sun Microsystems Computer Corporation for 3D graphics rendering with the addition of an on-chip arithmetic logic unit (ALU), SAM video buffers and other functions. This enhanced version became known as 3D RAM.

The chip has three portions: DRAM, pixel ALU and video buffers [Mit]. The 10Mbit DRAM

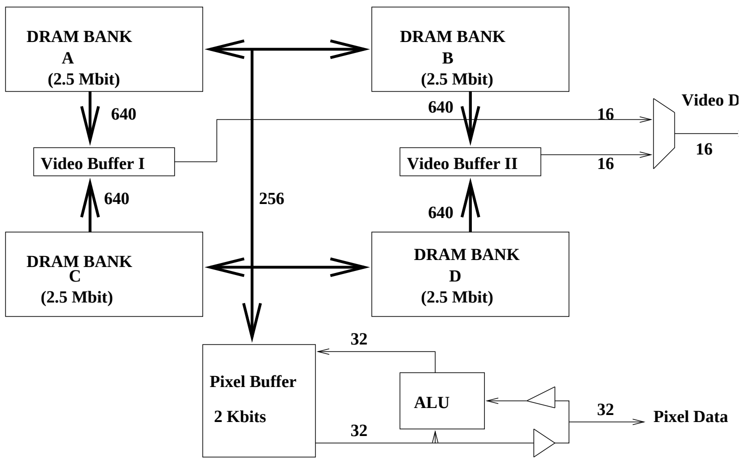

Figure 24: Block Diagram of a standard 3D RAM

consists of 4 independent banks, with each bank consisting of 256 memory pages, where each page is organized into 10X4 blocks. Each block is 256 bits, and can be viewed as 8X4 rectangular pixels, where each pixel is 8bits. The advantage of rectangular memory blocks is that it reduces page misses, since 3D objects frequently occupy more than one scan line. The independent banks can be interleaved to facilitate almost uninterrupted frame buffer update and, at the same time can transfer pixel data to the dual video buffer for screen refresh. Each video buffer is 80X8 bits, and is loaded in a single DRAM operation. One video buffer can be loaded while the other is sending out video data at the rate of 16 bits per 14ns. The pixel buffer consists of 8 blocks, and each block can be updated in a single 256 bit transfer on the global bus. Pixel buffer acts as a level-one cache, and has three ports: 256 bit read/write port to the global bus, 32bit read port, and 32 bit write port to the on-chip ALU unit. All three ports can be accessed simultaneously so long as they are not accessing the same memory location.

In most 3D graphics systems, Z buffer is used to perform depth comparison. When a new depth and color value is obtained for a pixel, the depth value is compared against the old depth (Z) value. If the new Z is behind the old Z, the color value remains unchanged, otherwise it is replaced with the new value. Similarly, for alpha-blending, the new color value is calculated as the weighted sum of the old and new color value using alpha as the weight factor. Both these operations require read-modify-write for the memory, and might become a bottleneck. 3D RAM uses the on-chip compare and alpha-blending unit in the ALU to convert this read-modify-write operation to a single write-only operation at the data pins [DSL94].

The pixel cache along with the ALU makes 3D RAM ten times faster than conventional VRAM. However, it is optimized for high-performance system, and is not very cost-effective for low-cost systems.

3.7.4 Synchronous Graphics RAM (SGRAM)

Synchronous Graphics RAM (SGRAM) is designed to address the high-speed wide I/O requirements of multimedia applications. Its architecture and operation is based on Synchronous DRAM. We will only describe the features that has been added to enhance performance of graphics applications. These features are the same as the features added to the Video RAM for block write and write per bit operation. These features are selected via a special mode of registers and additional command pins that can be loaded with appropriate information [IBMf].

1. Color Register: the 32 bit register is used to store the color for one or more colors for block write operation.
2. 8-Column Block Write: writes the contents of the color register to 8 columns of the SGRAM in a single operation.
3. Mask register and Write-per-Bit(WPB): mask data in the mask register along with the WPB data present at the inputs (DQs) prevent certain memory locations from getting written during block write.

SGRAM is the single most popular DRAM for graphics memory today just as its counterpart SDRAM is ubiquitous for system memory.

4 Memory Controller

Memory controller is the component that connects the CPU and the memory together. Since CPU and memory have their own history of evolution, their signals are not comparable, and hence memory controller is needed to bridge the gap between these two essential components. Memory controller is an integral part of any memory system, and hence it is imperative that the controller itself be optimized and does not add a significant latency to the memory subsystem.

4.1 Basic Components of Memory Controller

In this section, we describe the basic components of a memory controller. The main task of the memory controller is to decode the request coming from the bus master (which can be the CPU or the PCI bridge), convert the request to a form that the DRAM can understand, and finally chanelling the output from the DRAM pin to the requesting agent. In addition, it is the controller's responsibility to take care of the DRAM refresh cycles.

A generic memory controller for DRAM consists of the following major blocks:

1. **Address Multiplexer:** The memory controller accepts an address from the address bus, and converts it to row and column addresses for DRAM.
2. **Memory Sequencer:** This is the main logic in the memory controller. Memory sequencer is responsible for overall transaction flow. It starts all transactions, and issues all memory commands/signals. These are usually implemented as state machines.
3. **Refresh Controller:** It is the responsibility of the memory controller to send refresh requests at required intervals. A refresh timer provides refresh requests at regular intervals, typically every 15.6μsec. Most of the memory controllers treat the refresh request as low priority, till either there are couple of them pending, or a *considerable* portion of refresh period has expired. Then the requests are treated as high priority, and controller holds all the read/write requests till the refresh requests complete [Sch97].

4. **Memory Interleaving:** In case of word interleaving, memory controller need not perform anything intelligent. It just blindly issues the requests. It is the responsibility of the application to interleave the data and the requests over several memory devices. However, for non-sequential accesses, memory controller can take the responsibility of scheduling the accesses in such a way, so as to overlap the operations of different memory devices as much as possible.

5. **Fault Tolerance:** Most of the memory subsystems in low-end PC and workstations do not have parity or ECC bits. In that case, the memory controller does not need to do anything. However, for servers and mission-critical systems memory subsystem should implement some kind of error detection and/or correction logic. In case of simple **parity** based memory subsystems, the memory controller needs to calculate the parity bit. The simple type of **ECC** is one in which single-bit error in a data word can be detected and corrected, and double-bit error can be detected but not corrected. This requires 7 check-bits over 32-bits, and 8 check-bits over 64-bits. ECC and simple parity costs are equal for 64-bit systems [IBMa]. The memory controller needs to calculate the check-bits during memory write operation, and perform a comparison between read, and calculated check-bits during read operation. Even though the logic is not complex, this affects the memory latency during both read and write operation. Also, on a partial word write, first the entire word should be read, new check-bits should be calculated, and then the entire word should be written back. This is the case where read-modify-write operation provided by some of the DRAM vendors is used.

6. **Cache Coherence:** In a system with multiple bus masters, like DMA controller, and a processor with on-chip cache, the memory controller needs to ensure that data in the memory and the cache is the same after DMA write. Memory controller can just request the cache to invalidate its entry if the DMA write covers the entire cache line. For a partial write of a cache line, the cache line data needs to be merged with the DMA data before it is written back to the memory [Sch97].

4.2 Memory controller design for Synchronous DRAM

The memory controllers for Synchronous DRAMs have much more options than early generation DRAMs. The controller can schedule memory operations so as to obtain optimized performance. Also, starting with SDRAMs, memory controllers now have the option of keeping a row "active" after the operation so that if the next access is to the same page, precharge and activation time is saved. Depending on how the controller handles the active rows, they can be divided into four categories [Mosa]:

1. **Autoprecharge Controller:** This is the simplest controller that issues a precharge command *after every memory access*. The advantage is simplicity of logic, and the disadvantage is of course lost performance, when the next access is to *the same row*. This approach is useful only when: 1) bursts are so long that the precharge and activation overhead is too small compared to actual data cycles 2) the accesses are random so that leaving the row open does not help, and 3) when the requests are interleaved between two banks.

2. **Single Comparator Controller:** The controller has a single register that can map to any row of any bank. The register contains the bank and the row last activated. If there is a

match, the precharge and row activation time is saved. Otherwise, the open row should be closed with a precharge command, and activation command has to be issues to the new row and bank. This controller usually performs well compared to autoprecharge controller; only if all the accesses are quite random, single comparator controller wastes time by waiting to issue the precharge command, and might perform worse than auto precharge controller.

3. **Comparator-per-Bank Controller:** This controller maintains one register per bank that has the last activated row. If the row to be accessed matches the register content corresponding to the bank, controller can just issue read/write request. Otherwise, the row needs to be closed, and the new row activated. Again, this architecture will perform very well when one task is constantly accessing one bank, and another task the other, and each has lot of locality. Single comparator controller will miss every time context switch occurs and will unnecessarily precharge and reactivate the rows. However, for applications with poor locality it might again perform worse than autoprecharge controller due to precharge latency.

4. **Register-per-task Architecture:** If the controller can distinguish between accesses from different tasks, and has knowledge of the access patterns, the register-per-task architecture will perform extremely well. The main difficulty of this architecture is bank contention, when two or more tasks are trying to access different rows of the same bank. The controller should try to schedule the accesses such that memory performance is optimized as well as the latency/task is also minimized.

5 Adding Intelligence to Memory Using BIST

With the growing gap between processor and memory speed, current applications are demanding more and more bandwidth from the memory chips. DRAM vendors have responded to that demand by developing new architecture like Direct RDRAM and SLDRAM. Still it is not clear whether those new architectures will be able to satisfy the bandwidth requirement of future applications. It was noticed long ago that a huge bandwidth is available in the internal data path of the DRAM array. IRAM [P+97] philosophy tries to exploit this bandwidth by shifting all computation to memory system through integration of a processor onto a DRAM chip. This results in dramatic improvement of DRAM bandwidth and latency; however, this approach requires a complete paradigm shift. An interesting alternative is to integrate specialized logic into DRAM to perform some simple operation like Read-Modify-Write [B+97], or to integrate reconfigurable logic to perform a set of different operations [OCS98]. However, all these approaches require significant change to the DRAM core and hence, it is difficult to convince the DRAM vendors to integrate those logic inside the commodity DRAM chip. On the other hand, we think that implementing some simple operations, like block memory copy, or block memory clear, inside the memory chip have significant advantage since these operations seem to be quite important from operating system point of view as any read/write operation from/to a device needs a copy from the kernel to/from user space and is a significant performance bottleneck for memory system [R+95, TGH92, CB93]. Memory clear operations are frequently needed since new memory pages are often zeroed before handling it down to application. We also observe that with increasing memory chip density, most DRAM chips in the future are expected to include some built in logic circuit that already has the capability of performing this kind of operations. This logic is called **Built-In-Self-Test (BIST)** circuit, and is only used during the power-up to test the DRAM chip. Even though this logic consumes precious silicon area of the chip, it is never used after the power-up. In this section, we explore the possibility of modifying this logic a little bit to implement simple operations like block memory copy and zeroing operations. We argue that the modifications are simple enough so that it is easy to include the changes in the BIST circuit of DRAM chips. We also show that it is easy to use these new memory features without any change to the existing instruction-set-architecture of current processors. It will only require significant change to the memory controller, which we think is acceptable. Finally, we use these new functionalities to suggest some improvement in the performance of PCI bandwidth and latency for scatter-gather operations frequently used to DMA between PCI device and system memory.

Rest of this section is organized as follows. First, we describe BIST concept and two specific BIST architectures which we use as base for our proposed design. Next, we show how we implement the simple operations by modifying the BIST architectures. We also discuss how these features can be used in the overall system architecture. We conclude by discussing the impact of these operations on operations systems, compilers and applications.

5.1 BIST Architectures

As memory density is increasing by four times for every new generation, test cost of memory is increasing proportionately. Built-in Self-testing techniques are gaining popularity since they offer a cost-effective way to test high density random access memories. The main concept of BIST is to include logic circuit on the memory chip that can be used for manufacture testing as well as in-circuit testing, and the overhead is less than 1% for high-density DRAM. There are a number of proposed BIST algorithms that captures different levels of memory faults. However, from the architectural point of view, Franklin et. al. [FS90] classified the BIST implementations as

1. **Single-array Single-bit (SASB):** Single array of memory chip is tested and single bit of the memory array is accessed at a time.

2. **Single-array Multiple-bit (SAMB):**: Tests a single array at a time, but multiple bits of the single array are tested simultaneously. To exploit large bandwidth of DRAM array, we will use SAMB architecture for our purpose. In general, multiple bits from the same memory page(row) are accessed simultaneously, and this is achieved by modifying the column address decoder. When a SAMB architecture accesses all the bits of a memory page simultaneously, it is referred to as *line-mode testing.*

3. **Multiple-array Single-bit (MASB):**: Tests multiple arrays simultaneously, but single bit from each array is tested at a time.

4. **Multiple-array Multiple-bit (MAMB):** A number of arrays are tested simultaneously, with multiples bits from each array accessed at a time.

Since, our design will be based on SAMB architecture, the following description of BIST logic will assume SAMB architecture, even though the main components are more or less same for all the four architectures. In general, BIST logic consists of *control logic*, *address-generation logic*, *data-generation logic*, *response-verification logic* and *test trigger logic*. Control logic is the main component of BIST and controls the flow of the test algorithms. There are two distinct approaches

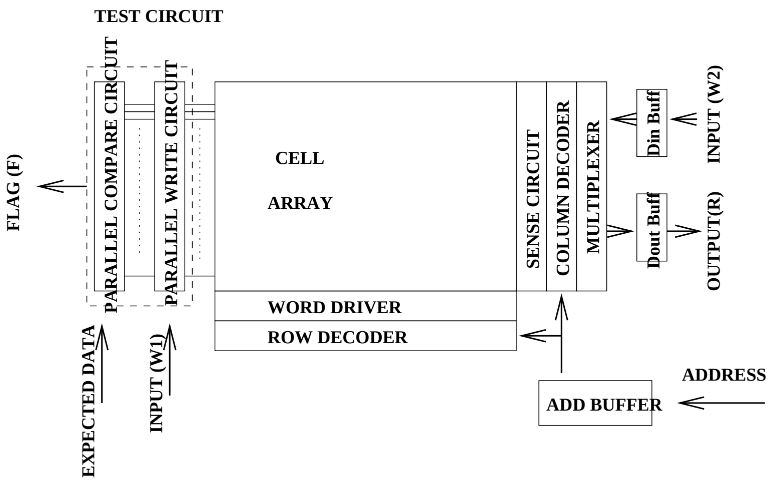

Figure 25: Block Diagram of a SAMB BIST Circuit

for implementation of control logic: ad-hoc random logic, and micro-code based control. Micro-code based logic provides the flexibility of implementing more than one test algorithms and the user can choose from that set. Our design is based on micro-code based control since it offers us to easily include our operations as new micro-procedures, and allows the memory controller to choose the particular operation. Address-generation logic generates the addresses using linear-feedback shift registers or counters. Data-generation logic generates the data patterns for the particular test algorithm, and response-verification logic does the comparison between the value read from the memory array and the expected value. Test-trigger logic is required to switch the DRAM from normal mode to BIST mode and vice versa. Even though this can be achieved using over-voltage or extra-package pins, we would prefer the unique timing sequence mechanism with the Chip Enable, Write Enable, $\overline{RAS}$ and $\overline{CAS}$. In this method, particular test mode as well as some other information necessary for that test mode can be obtained using the normal address inputs.

Figure 25 shows the block diagram of a memory device with an on-chip test circuit that uses the SAMB architecture [Sri85]. It consists of parallel write and compare circuits. Parallel write circuit is used to write data into memory cells connected to a word-line. The parallel comparator circuit is used to compare all data from the cells with the expected data and to raise pass/fail flag. This requires the modification of the column decoder and data buffers.

Figure 26 shows a micro-code based BIST circuit for a SDRAM device as proposed in [HPC94]. This circuit implements SASB architecture for all the test algorithms since they do not modify the column decode and data buffers of the existing DRAM core. However, we will use this circuit as the base of our design, and we propose to add parallel write circuit as shown in Figure 25 to the micro-code based design to perform memory page copy and clear operations. We also note that not all memory copy and clear operations are page aligned and equal to page size. For such un-aligned and small size operations, we use the operations provided by the micro-code based BIST circuit.

The functional blocks of the BIST circuit are shown in Figure 26. The 64 x 12 ROM contains the test algorithms. The timing generator uses the information from the ROM to generate timing

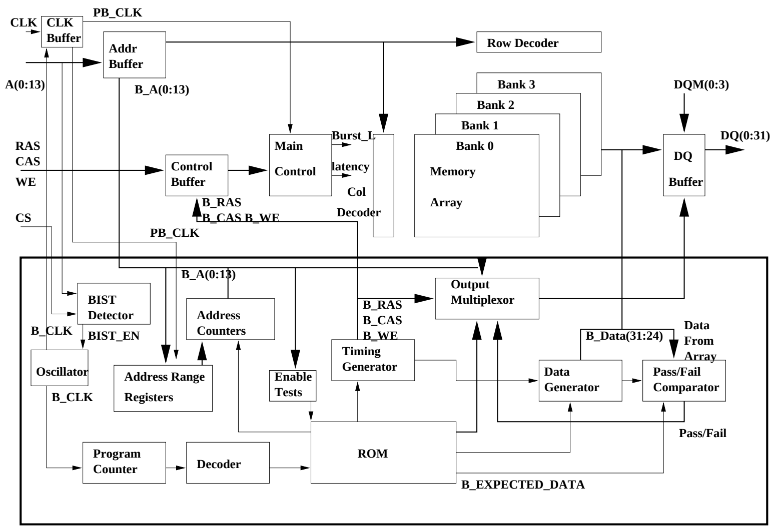

Figure 26: Block Diagram of a micro-code based BIST Circuit

signals for control, address and data. The data generator uses ROM information to generate test data patterns. The pass/fail comparator circuit does the response-verification. BIST detector is used for detecting BIST requests. This scheme includes a method to give the user the flexibility of choosing some or all of the available tests. This information is provided on the address line during BIST request. The BIST circuit brings in this information to an internal register. The information on the address lines contain test selection as well as BIST operation options as shown in the Figure 27. The address lines are multiplexed to perform different functions during normal mode, BIST mode, and initialization of BIST request. During normal mode, the address buffer uses external signals. During BIST initialization, the address lines are used to load test selection information. In addition, if BIST is in sub-array option mode (tests only a portion of the array), the address ranges to be tested are loaded in using the address lines. During the BIST mode, external signals are replaced by the signals generated from the BIST circuit. There are two type of instructions, *program control* and *memory access*. For program control instructions, first six bits specify the type of the conditional jump, and last six bits indicate the new PC address. For memory access instructions, first six bits indicate READ, WRITE, READ-MODIFY-WRITE operations, and whether the memory access is by row, or by column. The last six bits define the timing sets and data patterns to be generated for that memory access. A shift register is used to store the information regarding whether or not to execute a test. The first instruction of each algorithm checks whether the least significant bit of the shift-register is *1* or *0*. If it is *1*, an increment PC signal is generated and instructions corresponding to that algorithm are executed one by one. Otherwise, a "LOAD PC" signal is generated to load the start address of the next algorithm in the PC, thereby skipping the current algorithm (Refer to Figure 28. The last six bits of the first instruction provides the start address of the next algorithm. In total, the scheme implements 10 algorithms with total of 64 instructions.

BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Int/Ext Clk	Sub Array	Output Enable	T10	T9	T8	T7	T6	T5	OV	T4	T3	T2	T1

OPTIONS TEST ALGORITHMS

Figure 27: Test Selection at Power-Up

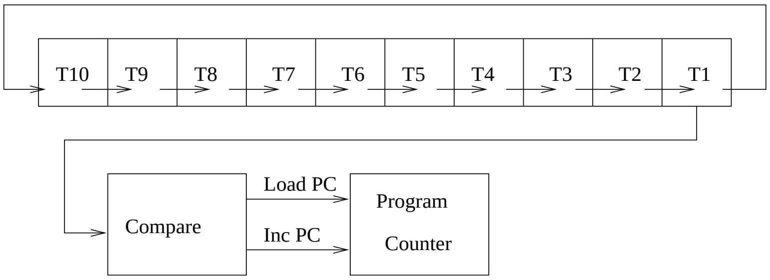

Figure 28: Implementation of Enable Instruction

5.2 BIST algorithm

To illustrate how a BIST algorithm is implemented using the micro-code based circuit described above, we choose one specific test algorithm, known as **Marching Test**. The test initializes the memory array to all 0's, and then scans the memory cells in ascending and descending orders. For each cell, scanning involves reading the cell for the expected value, writing the complement value, and reading it again. The idea behind this algorithm is that, while scanning the memory in ascending order, any coupling fault (a fault in which changing the content of one cell affects the content of a cell) between the current cell and the higher address cell is detected while reading the later. Similarly, scanning the memory in descending order, any fault between the current cell and the lower address cell is detected. The formal algorithm is given below:

```
S1: W[i] <- 0 for i=0,1,.... n-1

For i=0,1,... n-1
S2: R[i] (=0)   /*== Check whether 0 */
S3: W[i] <- 1   /*== write the complement */

For i=n-1,n-2,... 0
S5: R[i] (=1)   /*== Check whether 1 */
S6: W[i] <- 0   /*== write the complement */
S7: R[i] (=0)   /*== check whether 0 */
```

This algorithm is implemented in the micro-code based design using seven instructions.

Instruction 1: This is a control flow instruction that checks whether the algorithm needs to be executed (as discussed in the previous section). If not, the PC jumps to the next algorithm.

Instruction 2: This instruction performs WRITE 0 to the full array (S1).

Instruction 3: This instruction performs READ-MODIFY-WRITE for a memory cell. The data written is the opposite of the read pattern (S2 and S3).

Instruction 4: This instruction performs the marching pattern for the whole array. The address in incremented, and checked for overflow. If overflow has not occurred, the PC is loaded with the address of instruction 3, so that previous instruction is executed on a new memory cell. For overflow, the PC is loaded with the address of instruction 5 (implements the For loop).

Instruction 5: This instruction if effectively same as instruction 3, with the pattern inverted (s5 and S6).

Instruction 6: This is same as instruction 4 in that all the addresses are processed (implements the second For loop).

Instruction 7: This instruction verifies that the array now contains 0's(S7).

5.3 Our Design

We propose to add four algorithms to this scheme corresponding to four operations we plan to introduce:

1. **PAGE COPY:** Copy memory pages/rows from one address to another.
2. **PAGE CLEAR:** Clear memory pages/rows.
3. **COLUMN COPY:** Copy data corresponding to memory columns from one address to another.
4. **COLUMN CLEAR:** Clear memory columns.

We will refer to these operations collectively as **MODIFY** operations from now on. Column mode operations are necessary to implement memory copy and clear operations, that are not page alligned and/or smaller than a page size. We propose to modify the micro-code based scheme to implement our simple MODIFY operations as follows.

The BIST detector in the scheme above uses over-voltage at power-up to detect BIST request. We propose to modify it with a logic that can detect an unique timing sequence on the input pins as described by [V$^+$89].

The BIST circuit includes start and stop addresses of the rows and columns for performing sub-array operations. These addresses are stored in the address range registers. This is sufficient for PAGE CLEAR and COLUMN CLEAR operations. However for COPY operations, we need address range for both the source and the destination. Hence, we propose to duplicate the address range registers to hold both the source and destination address range. Also, the initialization logic has to be modified to read in both the address ranges from the address lines for COPY operations instead of one as done in sub-array mode or CLEAR operations. Finally, we duplicate the address counter to support two address increments simultaneously.

The test selection logic as shown in Figure 27 has to be modified to include the MODIFY operations. Note that the current scheme has already used up all the address pins available for performing the test selection. However, pin A4 is used to request BIST mode by providing over-voltage on that pin. Since we use timing sequence to detect BIST mode instead, this pin is free for us. We use this pin to detect whether BIST operations or MODIFY operation is requested. In case of MODIFY operations, the address pins are interpreted as shown in Figure 29. Address range option indicates whether one or two address ranges need to be obtained from address lines

BA1	BA0	A11		A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Address Range	PAGE/COL Mode	X		X	X	X	X	X	X	BIST/ MODIFY	COL CLEAR	COL COPY	PAGE CLEAR	PAGE COPY

OPTIONS ALGORITHMS

Figure 29: Operation Selection for our approach

Memory OR Control	READ	WRITE	DATA	REG1	REG2	ROW	COL	-	-	- -	-		1
	TEST					N	E	X	T	A	D	D	2

1: Memory Reference Instruction
2: Program Control Instruction

Figure 30: ROM instruction decoding for MODIFY

(one for CLEAR and two for COPY). The PAGE/COLUMN mode will be described shortly.

As described before, some MODIFY operations operate on complete page/row and some use single column. Hence, we need to implement the parallel write circuit as described in [Sri85] to perform page wise operations. Note that adding a row buffer to the existing DRAM architecture is nothing new. ESDRAM [Enh] approach already implements some kind of row buffer within the memory chip. Chiueh et. al. [Chi92, CVP95] has also proposed adding two row buffers to DRAM architecture, along with simple control logic to implement efficient vector memory systems. The PAGE/COLUMN mode option chooses between page-wise or column-wise operation. When in page mode (PAGE CLEAR/PAGE WRITE), the column decoder is ignored, and the contents of the sense-amplifier is transferred to the parallel write circuit, or copied from the parallel-write circuit to sense-amplifiers. When in column mode, the existing I/O path is used to read and write data to the array as done in the original BIST circuit.

We need to add instructions corresponding to the MODIFY operations. This will increase the set of instructions to more than 64 and hence we need one more bit per instruction to accommodate the target address of program control instructions (7 bits for target address, and hence 13 bits per instruction). Also, when in MODIFY operations mode, the instruction bits are interpreted differently than in BIST mode. Figure 30 shows the instruction format. For memory reference instruction, Read/Write fields indicate READ, WRITE, READ-AND-WRITE operation. The Data bit indicates the data pattern to be used for a WRITE operation (can be set to 0 or 1). It is ignored for READ, and READ-AND-WRITE operation. The Register1-2 bits indicates which register and counter to use for address range. For READ-AND-WRITE, Register1 indicates the source register, and Register2 indicates the destination register. For READ and WRITE operation, Register2 is ignored. Finally, a 1 in Row bit indicates that operation is to be executed on all row addresses within the address range and, 1 in Col indicates operation on all column addresses within the address range. A program control instruction is indicated with a 1 in the first bit of the instruction. The last 7 bits of the program control instruction indicates the target address. A 1 in the Test field indicates that this instruction should check whether an operation should be performed; if not it jumps to the target address. A 0 in the test bit indicates unconditional jump.

Operation	Read	Write	Data	Register1	Register2	Row	Col
PAGE COPY	1	1	X	0	1	1	0
PAGE CLEAR	0	1	0	0	X	1	0
COLUMN COPY	1	1	X	0	1	0	1
COLUMN CLEAR	0	1	0	0	X	0	1

Table 2: Instructions corresponding to different MODIFY operations

Table 2 show the instruction corresponding to each MODIFY operation. For example, for PAGE COPY, it is required to read a page from a row address, and write it to a different row address. We represent that with a READ-AND-WRITE operation, which is indicated by 1 in both READ and WRITE fields. Data field is ignored for this operation. A 0 in Register1 indicates that register0 should be used as source address and a 1 in Register2 shows that register1 should be used as destination address. A 1 in field Row indicates that the operation should be performed for all rows within the address range.

Also, note that in MODIFY mode, only one operation will be performed. Hence the first instruction for each MODIFY operation is a program control instruction with Test field set. It checks whether the last bit in the shift register is 1. If yes, the next instruction (the actual operation) is executed; otherwise it skips and goes to the next MODIFY operation. The instruction after the actual operation is always an unconditional jump instruction to the last instruction in the ROM that exits the execution in BIST mode.

5.4 Advantage of on-chip memory copy and clear operation

The support of on-chip memory copy and clear operation is quite useful from overall system point of view. As discussed before, operating system as well as application programmers frequently use the block memory copy and clear operation. In existing system, these operations are extremely expensive, and how to avoid those operations is an active research area. Even though the operations does not involve any computation, processor still gets involved consuming precious processor cycle times and bus bandwidth. For example, with 64 bit system bus, a 256 byte memory copy will involve 32 memory read and 32 memory write operations. Assuming it results in cache miss (which is quite likely for kernel to user space copy or vice versa for example), this operation will block the memory bus for a long time, preventing other devices to access the bus as well as the memory. With out proposed model, if the data belongs to a single page, the operation can be achieved with a single command to the memory system. Even otherwise, the data will cover two consecutive pages, and the operation can be performed with two commands to the memory system. This saves bus bandwidth as well as takes advantage of on-chip parallelism to perform faster operation.

5.5 Scatter-Gather Operation

Today, most of the PCI devices come with built-in DMA engine and those DMA engines are capable of performing scatter-gather operation. The idea of scatter-gather is quite simple. Suppose, some data needs to be transferred from the system memory to the memory on the PCI device. However, the data is scattered over multiple locations in the memory. Originally, it will be the responsibility of the processor to issue multiple DMA requests to transfer the data. However, PCI commands have significant overhead and it would be nice if this request can be issued in a single command. Gather mode of DMA allows the system programmer to achieve exactly this effect. The processor will supply the addresses of all the memory locations, and size of data chunk at each location, and then ask the device to perform a gather DMA. The device will go through each entry and perform the DMA. Similarly for scatter, data is transferred from the PCI device to multiple locations in the memory using a single command. The scatter-gather mode performs significantly better than original individual transfer mode. However, notice that the transfer still takes place as multiple small PCI operations. Hence, the performance is much worse than a single big transfer to/from one memory location. Theoretically, the PCI device can transfer to a single memory location and then memory copy can be performed to transfer data to multiple memory locations. However, memory copy has significant overhead and hence the approach is not feasible. We notice, that with our MODIFY operations, memory copy can be performed quite fast. Hence, we propose that for gather, processor can request multiple MODIFY operations to first transfer data to a temporary free memory location, and then PCI device can perform a single DMA from that temporary location. Similarly for scatter, PCI device transfers data to the temporary location, and memory controller, can later transfer that data to multiple memory locations. Of course the memory controller needs to keep track of these operations so that data consistency is maintained.

5.6 Overall System Considerations

In this section, we show how the proposed mechanism can be used with almost negligible modification to the instruction set architecture of current processors. We propose to change the memory controller so that it can issue MODIFY operations to the memory chips. Also, it is the responsibility of the memory controller to determine whether PAGE mode or COLUMN mode operations need to be used. The processor requests MODIFY operations through a set of memory mapped registers inside the memory controller. This is a standard method of communication between the processor and the memory controller for almost all existing architectures. One can view this as if the processor is requesting the memory controller to perform some DMA, only in this case it is from system memory to system memory. Note that it is the responsibility of the processor to issue the requests using physical addresses. This is not a significant problem when the block copy or block clear operation is performed by the operating system (transferring data from user space to kernel space or vice versa) since operating system has complete knowledge of the virtual to physical address translation. However, for application programs to use this feature, it is necessary to export the page tables to the user space so that user process does not need to make system calls to convert virtual address to physical address. This can be easily done if the kernel exports the page tables as READ-ONLY to the user space, and a library is provided in user level that can consult these tables to convert addresses from virtual to physical. Finally, we have to consider the cache effect in this case. Since the copy or clear operation takes place inside the memory chip, the processor cache might contain dirty data. It is the responsibility of the processor to delete those cache entries. For this, we need to add a new instruction to the processor that can flush all the cache entries corresponding to a particular page. Otherwise, the whole cache needs to be flushed and that overhead is not acceptable, or the page should be marked uncachable (pages that are destination of DMA are marked this way today)

Finally, all the memory copy and clear operations are not nicely aligned according to memory pages [Fro], and our design can perform only intra-bank copy operations. We do not consider inter-bank copy since that involves significant silicon area. However, we hope that if this feature is available, compiler and operating system designers will try to take advantage of these features and try to place data such that the block copy and clear operations can be performed.

6 Conclusion

With the increasing gap between processor and memory performance, along with the advent of virtual reality and multi-media applications, the accepted DRAM architectures were pushed to a limit. This need for a change accelerated the appearance of various innovative DRAM architectures from the memory industry. Initially, DRAM chip used to be accessed *one bit at a time*. Then, the spatial locality of memory reference stream was exploited by using *page mode*, where multiple bits from the same page are accessed without precharging the array in between. The *burst mode* was invented next, where the chip generates addresses of multiple bits internally, given the starting address and the burst length. With the advent of Synchronous DRAM, industry moved from asynchronous to *synchronous mode*, once and for all, making faster operations feasible. Finally, Direct RDRAM and SLDRAM architecture are trying to change the wide and slow signal-based processor-DRAM interface to a *narrow and fast packet-based interface*. ESDRAM and Cache DRAM are on the other hand, orthogonal architectures, attempting to *reduce the memory latency*, by putting a small on-chip cache. On the other spectrum, VRAM, 3D-RAM, WRAM and SGRAM are all *applications specific memories* that provide simple logic on memory chip to perform certain operations frequently required by graphics applications. In this report, we have presented a comprehensive survey of the recent DRAM architectures. Finally, we have proposed a simple modification to the Built-In Self-Test logic circuit on high-density DRAM chips, to support certain memory-intensive operations. We believe, these operations are easy to incorporate within existing DRAM architectures, and are essential for overall system performance.

References

[B+97] A. Brown et al. Using MML to Simulate Multiple Dual-Ported SRAMs: Parallel Routing Lookups in an ATM Switch Controller. In *Workshop on Mixing Logic and DRAM*, 1997.

[Bur93] D. Bursky. Dual-Port DRAM Accelerates Window. *Electronic Design*, November 1993.

[Bur97] D. Bursky. Advanced DRAM Architectures Overcome Data Bandwidth Limits. *Electronic Design Online*, November 1997.

[CB93] J. Chen and B. Bershad. The Impact of Operating System Structure on Memory System Performance. In *ACM Symposium on Operating System Principles*, 1993.

[Chi92] T. Chiueh. A Vector Memory System Based on Wafer-Scale Integrated Memeory arrays. In *Supercomputing'92*, 1992.

[Cri97] R. Crisp. Direct Rambus Technology: The New Main Memory Standard. *IEEE MICRO*, 17(6), 1997.

[CVP95] T. Chiueh, M. Verma, and S. Padubidri. Efficient Implementation Techniques for Prime-Degree Interleaved Memory Systems. In *International Symposium on Parallel Algorithms, Architectures, and Networks*, 1995.

[DSL94] M. F. Deering, S. A. Schlapp, and M. G. Lavelle. FBRAM : A new Form of Memory Optimized for 3D Graphics. In *SIGGRAPH Conference*, 1994.

[Enh] Enhanced Memory Systems Inc. *16Mbit EDRAM Design Guide*. `http://www.edram.com/products/appnotes/16mdgrev1_4.pdf`.

[Fro] R. Fromm. Utilizing the on-chip IRAM bandwidth. `http://www.cs.berkeley.edu/~rfromm`.

[FS90] M. Franklin and K. K. Saluja. Built-in Self-Testing of Random-Aaccess Memories. *IEEE Computer*, 23(10), 1990.

[GV97] P. Gillingham and B. Vogley. SLDRAM: High-Performance, Open-Standard Memory. *IEEE MICRO*, 17(6), 1997.

[H+90] H. Hidaka et al. The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory. *IEEE MICRO*, April 1990.

[Hit] Hitachi. *Benefits of Synchronous DRAMs*. `http://www.semiconductor.hitachi.com`.

[HPC94] F. Hii, T. Powell, and D. Cline. A Built In Self Test Scheme for 256Meg SDRAM. In *International Workshop on Memory Technology, Design and Testing*, 1994.

[IBMa] IBM Microelectronics. *Fault Tolerance Decision in DRAM Application*. `http://www.chips.ibm.com/products/memory/fault/fault.html`.

[IBMb] IBM Microelectronics. *Half SAM and Full SAM Compatiblity*. `http://www.chips.ibm.com/products/memory/vramshalfsam/vramhalfsam-1.htm%l`.

[IBMc] IBM Microelectronics. *Synchronous DRAM: The DRAM of the Future*. `http://www.chips.ibm.com/products/memory/sdramart/sdramart.html`.

[IBMd] IBM Microelectronics. *Understanding DRAM Operations*. `http://www.chips.ibm.com/products/memory/dramoperation/dramop.html`.

[IBMe] IBM Microelectronics. *Understanding DRAM Performance Specification*. `http://www.chips.ibm.com/products/memory/dramperf/dramperf.html`.

[IBMf] IBM Microelectronics. *Understanding VRAM and SGRAM Operation*. `http://www.chips.ibm.com/products/memory/vramsgramop/vramsgram.html`.

[Jon92] F. Jones. A New Era of fast dynamic RAMs. *IEEE Spectrum*, October 1992.

[Kat97] Y. Katayama. Trends in Semiconductor Memories. *IEEE MICRO*, 17(6), 1997.

[KOI95] M. Kumanoya, T. Ogaowa, and K. Inoue. Advances in DRAM Interfaces. *IEEE MICRO*, December 1995.

[Mic] Micron Semiconductor Products Inc. *Various Methods of DRAM Refresh.* `http://www.micron.com/mti/msp/pdf/datasheets/tno432.pdf`.

[Mit] Mitsubishi Semiconductors. *3D-RAM Datasheet.* `http://www.mitsubishichips.com/data/datasheets/memory/mempdf/ds/3d-ram3%.pdf`.

[Mosa] Mosys Inc. *High-Speed Synchronous Controller Design Techniques.* `http://www.mosysinc.com/products/an_sg01.pdf`.

[Mosb] Mosys Inc. *Multibank DRAM (MDRAM) 128Kx32 TO 656Kx32.* `http://www.mosysinc.com`.

[NEC] NEC Electronics Inc. *64M-bit Virtual Channel SDRAM Datasheet.* `http://http://www.necel.com/home.nsf/ViewAttachments/M13022EJ9V0DS00/$f%ile/13022e90.pdf`.

[OCS98] M. Oskin, F.T. Chong, and T. Sherwood. Active Pages: A Computation Model for Intelligent Memory. In *International Symposium on Computer Architecture*, 1998.

[P+97] D. Patterson et al. A Case for Intelligent RAM. *IEEE MICRO*, 17(2), 1997.

[Pri91] B. Prince. *Semiconductor Memories: A Handbook of Design, Manufacture and Application.* John Wiley and Sons, 1991.

[R+95] M. Rosenblum et al. The Impact of Architectural Trends on Operating System Performance. In *15th ACM Symposium on Operating Systems Principle*, 1995.

[Sch97] R. C. Schumann. Design of the 21174 Memory Controller for DIGITAL Personal Workstations. *Digital Technical Journal*, 9(2), 1997.

[Sri85] T. Sridhar. A New Parallel Test Approach for Large Memories. In *IEEE International Test Conference*, 1985.

[TGH92] J. Torellas, A. Gupta, and J. Hennessy. Characterizing the Caching and Synchronization Performance of a Multiprocessor Operating System. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992.

[V+89] P.H. Voss et al. A 14-ns 256Kx1 CMOS SRAM with Multiple Test Mode. *IEEE Solid-State Circuits*, 24(4), 1989.