ABSTRACT

Title of dissertation: MODERN DRAM MEMORY SYSTEMS: PERFORMANCE ANALYSIS AND A HIGH PERFORMANCE, POWER-CONSTRAINED DRAM SCHEDULING ALGORITHM

David Tawei Wang, Doctor of Philosophy, 2005

Dissertation directed by: Associate Professor Bruce L. Jacob
Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies

The performance characteristics of modern DRAM memory systems are impacted by two primary attributes: device datarate and row cycle time. Modern DRAM device datarates and row cycle times are scaling at different rates with each successive generation of DRAM devices. As a result, the performance characteristics of modern DRAM memory systems are becoming more difficult to evaluate at the same time that they are increasingly limiting the performance of modern computer systems. In this work, a performance evaluation framework that enables abstract performance analysis of DRAM memory systems is presented. The performance evaluation framework enables the performance characterization of memory systems while fully accounting for the effects of datarates, row cycle times, protocol overheads, device power constraints, and memory system organizations.

This dissertation utilizes the described evaluation framework to examine the performance impact of the number of banks per DRAM device, the effects of relatively static DRAM row cycle times and increasing DRAM device datarates, power limitation constraints, and data burst lengths in future generations of DRAM devices. Simulation results obtained in the analysis provide insights into DRAM memory system performance characteristics including, but not limited to the following observations.

- The performance benefit of having a 16 banks over 8 banks increases with increasing datarate. The average performance benefit reaches 18% at 1 Gbps for both open-page and close-page systems.
- Close-page systems are greatly limited by DRAM device power constraints, while open-page systems are less sensitive to DRAM device power constraints.
- Increasing burst lengths of future DRAM devices can adversely impact cache-limited processors despite the increasing bandwidth. Performance losses of greater than 50% are observed.

Finally, This dissertation also present a unique rank hopping DRAM command-scheduling algorithm designed to alleviate the bandwidth constraints in DDR2 and future DDRx SDRAM memory systems. The proposed rank hopping scheduling algorithm schedules DRAM transactions and command sequences to avoid the power limiting constraints and amortizes the rank-to-rank switching overhead. Execution based simulations show that some workloads are able to fully utilize the additional bandwidth and significant performance improvements are observed across a range of workloads.

MODERN DRAM MEMORY SYSTEMS: PERFORMANCE ANALYSIS AND SCHEDULING ALGORITHM

by

David Tawei Wang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

Advisory Committee:

Associate Professor Bruce L. Jacob, Chair
Associate Professor Shuvra S. Bhattacharyya
Associate Professor Tsung Chin
Associate Professor Donald Yeung
Associate Professor Charles B. Silio Jr.

Table of Contents

CHAPTER 8 *Power-Constrained DDRx Scheduling Algorithm 183*

CHAPTER 9 *Concluding Remarks 212*

CHAPTER 1 Introduction

Performance of modern computer systems have seen dramatic improvements in the past thirty years due to advancements in silicon process technology. The advancements in silicon process technology have enabled the number of transistors on a single chip to roughly doubled every two years as suggested by Moore's Law. As a corollary to Moore's Law, processor performance has also doubled roughly every two years in the same time period due to a combination of the larger transistor budget and the increased switching speed of those transistors. However, increases in processor performance did not lead to comparable increases in performance of computer systems for all types of applications. The reason that increases in processor performance did not lead directly to comparable increases in computer system performance is that computer system performance is fundamentally constrained by the interaction between the processor and memory elements. Moreover, in contrast to the rapid improvements in processor performance, memory system performance has seen only relatively modest improvements in the past thirty years. The result of the imbalance in performance scaling trends between processor and memory is that modern computer systems are increasingly constrained by the performance of memory systems; in particular, the performance of DRAM based memory systems. The work in this dissertation is dedicated to the investigation of DRAM memory system performance characteristics, and the result of the investigation is then used to evaluate and support the design of future DRAM devices.

1.1 Problem Description

Computer system performance is increasingly limited by the performance of DRAM based memory systems due to the fact that the rate of DRAM memory system performance increase has lagged the rate of processor performance increase in the past thirty years. One reason that DRAM memory system performance has consistently lagged processor performance is that DRAM memory systems typically consist of one or more chips that are designed and manufactured separately from the processor, and the performance of the interconnected multi-chip DRAM memory system is difficulty to scale to achieve higher datarate and lower access latency. One apparent solution to the problem of access latencies introduced by system level interconnects between processors and memory systems is to integrate the memory system with the processor onto the same silicon die. However, in the case of the integrated memory system, the size of the silicon die limits the storage capacity of the memory system, and that capacity cannot be configured by the end user as needed for different operating environments. Moreover, the die area used by the memory system could have been used by performance enhancing features or more processor cores. In essence, the integration of processor and memory system onto the same silicon die is currently a viable solution for only a limited subset of high performance systems. As a result, high performance processors are keeping silicon die area for use by logic transistors, and memory transistors for main memory are still constructed separately from the processor chip. For example, high performance processors such as Intel’s Itanium and Pentium processors, AMD’s Opteron processor, and IBM’s Power5 processors are all moving toward multi-core designs or already contain multiple processor cores per chip, and the study of the memory system as a separate entity will continue to have great relevance for the foreseeable future.

A second reason that the rate of increase of DRAM memory system performance has lagged the rate of increase of processor performance is that while high performance processors are specialized parts and typically command high price premiums, standard DRAM devices are commodity items that can be freely purchased from multiple vendors. The commodity nature of standard DRAM devices means that DRAM device manufacturers are extraordinarily sensitive to manufacturing costs, and only features that provide substantial performance benefits for minimal cost increments are considered in each new generation of standard DRAM devices. However, there is great difficulty in the determination of performance impact for different performance enhancing features proposed for each new generation of DRAM devices, and that difficulty arises from the fact DRAM memory system performance depends on a large number of independent variables such as workload characteristics of memory access rate and request sequence, memory system architecture, and memory system configuration. As a result, system architects and design engineers will often disagree as to the impact of various performance enhancing features, since that performance impact depends on the configuration of specific systems.

Presently, DRAM device datarates are increasing with each new generation of DRAM devices at the rate of 100% every three years, and DRAM row cycle times are decreasing at a rate of approximately 7% per year[22]. The collective trends are increasing the ratio of row cycle times to the duration of data bursts on the data bus. As a result, to maintain a given utilization rate of memory system bandwidth, more requests must be issued to the DRAM memory system in parallel for each successive generations of higher data rate DRAM devices. Collectively, these trends form the larger picture that while DRAM based memory system performance are increasingly limiting system performance, it is becoming more difficult to maintain efficiency in each successive generations of higher data rate DRAM devices. Moreover, system architects and design engineers often disagree as to the desirability of various proposed DRAM device and system enhancements designed to increase DRAM memory system performance. With these considerations forming the background, the work in this dissertation is devoted to the creation of a common basis that system architects and design engineers can use to quantify the impact of various proposed performance enhancing features in modern DRAM devices, subjected to different workloads, system architecture and system configurations.

1.2 Contributions and Significance

The contribution of this dissertation is three-fold. Specifically, The contributions are the following:

- We create a parameterized and abstract DRAM memory access protocol. With proper definition of timing parameters, the DRAM memory access protocol accurately models DRAM device and system level interactions of SDRAM, DDR SDRAM, DDR2 SDRAM and DDR3 SDRAM memory access protocols. The creation of the abstract DRAM memory access protocol ensures that the analytical work performed for one memory system retains context for direct comparison against another memory systems. i.e. comparisons of an SDRAM memory system against a DDR2 SDRAM memory system.
- We derive a set of mathematical equations that establish the relationship between DRAM memory system configuration, timing parameters and the maximum achievable bandwidth. The formalized methodology is then utilized to examine the performance of future DRAM

memory systems given different DRAM system configurations, device data rates, row cycle times, DRAM device power limitations, rank-to-rank data bus turnaround overheads, read-and-write data bus turnaround overheads, cache line burst lengths, and the number of banks in a given DRAM device.

- We create a DRAM transaction and command scheduling algorithm that groups row activation commands and column access commands separately to ensure that maximum bandwidth can be maintained despite the existence of constraints such as data bus synchronization overhead in DDR, DDR2 and DDR3 SDRAM memory systems and mechanisms that limit peak power in DDR2 and DDR3 memory systems.

To aid the evaluations of performance and manufacturing cost trade-offs in modern DRAM devices, the work in this dissertation proceeds through a detailed examination of modern DRAM memory systems, starting from a description of modern DRAM devices and ending with the introduction of a high performance, power-constrained DRAM transaction and command scheduling algorithm. We believe that the performance evaluation methodology can contribute directly to the evaluation process of future DRAM device and memory system cost-performance trade-offs. We also believe that the DRAM transaction and command scheduling algorithm can contribute directly to the design of future high performance memory systems that must support high request rate access patterns with low spatial locality.

1.3 Organization of Dissertation

In this dissertation, the DRAM memory system is methodically examined from the transistor level to the system level. In this first chapter, a brief introduction to the dissertation is given. In Chapter 2, basic DRAM device architecture is described, and important details of DRAM device operations are examined in depth. In Chapter 3, typical DRAM based memory system topology and system architectures are described. The details provided in Chapters 2 and 3 are then used to create a generic DRAM memory access protocol in Chapter 4. The generic DRAM memory access protocol methodically examines the interactions between DRAM commands in a DRAM memory system. Then, from the description of the generic DRAM access protocol, a table of minimum scheduling distances between combinations of DRAM commands is summarized as table 4.3. Chapter 5 then examines DRAM controller designs and address mapping policies. The table of minimum scheduling distances is then used to form the foundation of a formalized methodology for the computation of maximum DRAM system bandwidth, hereafter referred to as the ***Request Access Distance*** methodology. The ***Request Access Distance*** methodology for the computation of maximum DRAM system bandwidth is formally defined in Chapter 6. In Chapter 7, results from studies based on the use of the ***Request Access Distance*** methodology are presented and analyzed. In Chapter 8, a unique rank-hopping memory scheduling algorithm is proposed and studied. The algorithm is designed to alleviate various constraints imposed upon high datarate DDRx SDRAM devices. Chapter 9 summarizes this work with concluding remarks. Finally, in Appendix A, the workloads used in the investigation of maximum DRAM system bandwidth in Chapter 7 are described in detail, and a glossary of terminology is enclosed in Appendix B.

CHAPTER 2

DRAM Device: Basic Circuits and Architecture

2.1 Introduction:

To facilitate the study of DRAM based memory systems, this chapter describes basic circuits and architecture of DRAM devices. For all practical purposes, it is impossible to provide a complete overview as well as an in depth coverage on the topic of DRAM circuits and architecture in a single chapter. The limited goal in this chapter is to provide a broad overview of functionalities of circuits and common functional blocks in DRAM devices sufficient to provide a basic understanding of internal circuits and architecture of modern DRAM devices. With the understanding of the fundamentals of DRAM device operations in place, more advanced discussions of architectural trade-offs at the DRAM device and system level would then be possible.

This chapter begins the examination of modern DRAM devices with the description of a basic fast page mode (FPM) DRAM device. Various components such as DRAM storage cells, DRAM array structure, voltage sense amplifiers, control logic and decoders are then examined separately.

2.2 DRAM Device Organization

Figure 2.1 illustrates the organization and structure of a **Fast Page Mode** (FPM) DRAM device. Internally, the array of DRAM storage cells in Figure 2.1 is organized as 4096 rows, 1024 columns per row, and 16 bits of data per column. In this device, each time a row access occurs, a 12 bit address is placed on the address bus and the **row address strobe** (RAS) is asserted by an external memory controller. Inside the DRAM device, the address on the address bus is buffered by the row address buffer, then sent to the row decoder. The row address decoder then accepts the 12 bit address and selects one of 4096 rows of storage cells. The data values contained in the selected row of storage cells are then sensed and maintained in the array of sense amplifiers. Each row of DRAM cells in this chip consists of 1024 columns and each column is 16 bits wide. That is, a 16 bit wide column is the basic

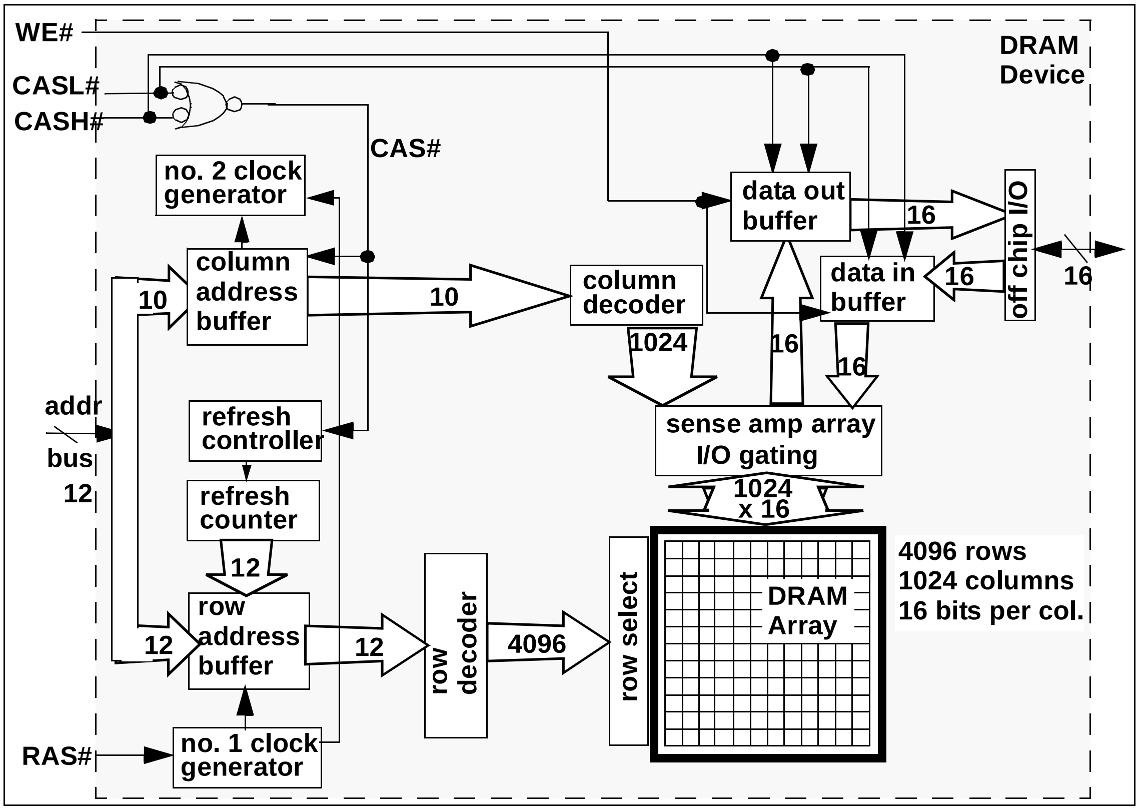


Figure 2.1: 64 Mbit Fast Page Mode DRAM Device (4096 x 1024 x 16).

addressable unit of memory in this device, and each column access that follows the row access would ordinarily read or write 16 bits of data from the same row of DRAM. The FPM DRAM device does allow each 8 bit half of the 16 bit column to be accessed independently through the use of separate **column access strobe high** (CASH) and **column access strobe low** (CASL) signals. The way that a column access is engaged is similar to the row access in that the memory controller would place a 10 bit address on the address bus, but then assert the appropriate **column access strobe** (CAS#) signals. Internally, the DRAM chip then takes the 10 bit column address, decodes it and uses it to select one column out of 1024 columns. The data for that column is then placed onto the data bus or overwritten with data from the data bus depending on the **write enable** (WE) signal.

All DRAM devices, from the FPM DRAM device to modern DDRx[*] SDRAM devices, possess similar basic organizations. All DRAM devices have one or more arrays of DRAM cells organized into a number of rows and columns, with a column being the smallest unit of addressable memory on that device. All DRAM devices also have some logic circuits that control the timing and sequence how the device operates. In the case of the FPM DRAM device shown in Figure 2.1, the chip has internal clock generators as well as a built-in refresh controller. In most cases, the DRAM device itself controls the relative timing of the sequence of events for a given action. The FPM DRAM device also keeps the address of the next row that needs to be refreshed, so when the memory controller asserts a new refresh command to the DRAM device, the row address to be refreshed can be loaded from the internal refresh counter rather than having to load a separate row address from the off chip address bus. Also, pin usage has always been restrictive on DRAM devices. As a result,

[*] DDRx denotes DDR, DDR2, and variants of future DDRx SDRAM devices

modern DRAM devices move data onto and off of the device through a set of bi-directional input-output pins connected to the system. Finally, advanced DRAM devices such as ESDRAM, Direct RDRAM and RLDRAM have evolved to include more logic circuitry and functionality on chip, such as row caches or write buffers that allow for read-around-write functionality. These circuitry improve performance but add to the die cost of the DRAM device. As a result, they are not found in standard DRAM devices. However, these performance enhancing features may prove to be necessary elements in future high datarate DRAM devices.

2.3 DRAM Storage Cells

Figure 2.2 shows a circuit diagram of the basic one transistor, one capacitor (1T1C) cell structure used in modern DRAM devices as the basic storage unit. In the structure illustrated in Figure 2.2, when the access transistor is turned on by applying a voltage on the gate of the access transistor, a voltage representing the data value may be placed onto the bitline and used to charge the storage capacitor. The storage capacitor then retains the stored charge for a limited period of time after the voltage on the wordline is removed and the access transistor is turned off. However, due to leakage currents through the access transistor, the electrical charge stored in the storage capacitor gradually dissipates. As a result, before the stored charge decays to indistinguishable values, data stored in DRAM cells must be periodically read-out and written back in a process known as ***refresh***. Otherwise, the stored electrical charge will gradually leak away and the value stored in the capacitor will no longer be resolvable after some time.

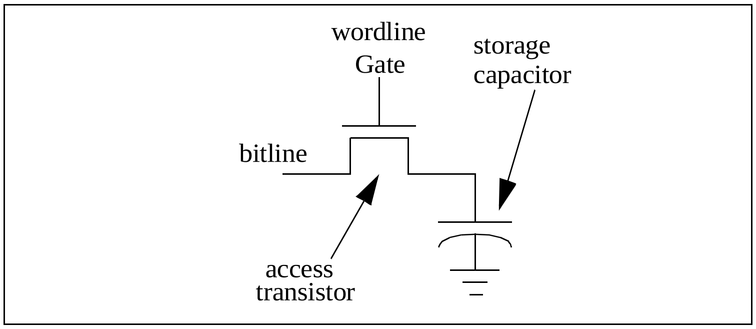


Figure 2.2: Basic 1T1C DRAM Cell Structure.

2.3.1 Cell capacitance, Leakage and Refresh

In a 90 nm process technology optimized for the manufacturing of DRAM devices, the capacitance of a DRAM storage cell in a typical DRAM device is on the order of 30 fF, and the leakage current of the DRAM access transistor is on the order of 1 fA [23]. With the cell capacitance of 30 fF and leakage current of 1 fA, a typical DRAM cell can retain the state of the stored data for hundreds of milliseconds. That is, hundreds of milliseconds after data is written, the electrical charge of a DRAM cell will still resolve to the stored digital value by the ***differential sense amplifier***. Some cells in a typical DRAM device can even hold the stored data value for upwards of several seconds. However, a reliable memory systems must be designed in such a manner that not a single bit of data would be lost due to charge leakage. The result of this requirement means that every single DRAM cell in a given device must be refreshed at least once before any single bit in the entire device would lose its stored charge due to leakage. In most modern DRAM memory systems, the storage cells in standard DRAM devices are typically refreshed once every 32 or 64 ms. In some cases where DRAM cells have low capacitance storage capacitors or high leakage currents through the access transistor, the time period between refresh intervals must be reduced to ensure reliable data retention.

2.4 DRAM Array Structures

In DRAM devices, large numbers of DRAM cells are grouped together to form DRAM array structures. Figure 2.3 illustrates a single bank of DRAM storage cells where a row address is sent to the row decoder, and the row decoder selects one row of cells. A row of cells is formed from one or more wordlines that are driven concurrently to activate one cell on each one of thousands of bitlines. There may be hundreds of cells connected to the same bitline, but only one cell will place its stored charge from its storage capacitor on the bitline at any one time. The resulting voltage on the bitline is then resolved into a digital value by a differential sense amplifier. Figure 2.3 illustrates an abstract DRAM array in a top down view, and it also abstractly illustrates the size of a cell in the array. The size of a unit cell in Figure 2.3 is 8 F^2. In the context of DRAM cell size, “F” is a process independent metric that denotes the smallest feature size in a given process technology. In a 90 nm process technology, F is literally 90 nm, and an area of 8 F^2 translates to 64800 nm^2 in the 90nm process. The cross sectional area of a DRAM storage cell is expected to scale linearly with respect to the process generation, maintaining the 6~8 F^2 cell size in each generation.

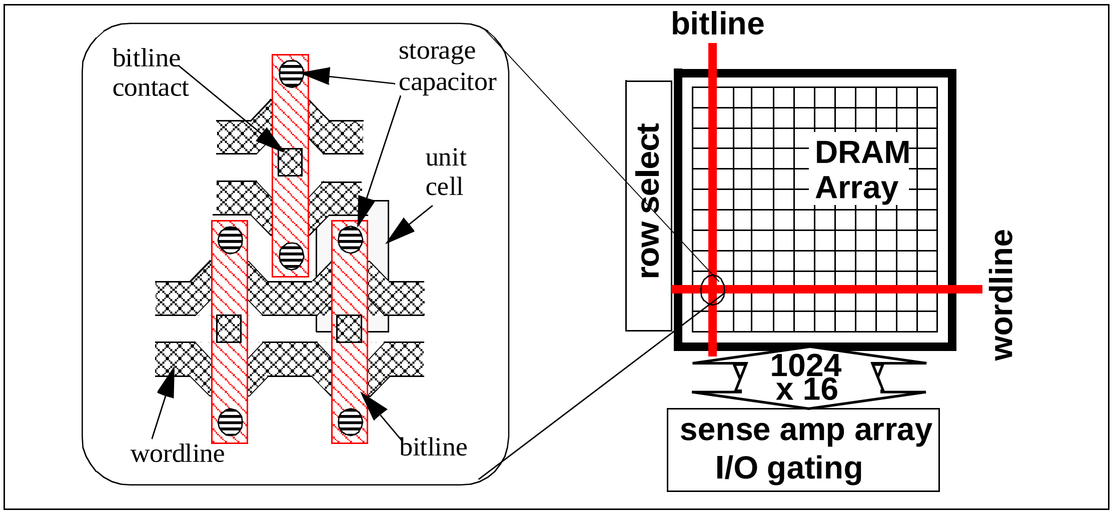


Figure 2.3: Top Down view of DRAM array.

In modern DRAM devices, the capacitance of a storage capacitor is far smaller than the capacitance of the bitline. Typically, the capacitance of a storage capacitor is one-tenth of the capacitance of the long bitline that is connected to hundreds of other cells. The relative capacitance values create the scenario that when the small charge contained in a cell is placed on the bitline, the resulting voltage on the bitline is small and difficult to measure in an absolute sense. In DRAM devices, the voltage sensing problem is resolved through the use of a differential sense amplifier that compares the voltage of the bitline to a reference voltage. In the following section, the functionality of the differential sense amplifier is examined in some detail.

2.5 Differential Sense Amplifier

In DRAM devices, the functionality of resolving small electrical charges stored in storage capacitors into digital values is performed by a differential sense amplifier. In essence, the differential sense amplifier takes the voltages from a pair of bitlines as input, senses the difference in voltage levels between the bitline pairs and amplifies the difference to one extreme or the other.

2.5.1 Functionality of Sense Amplifiers in DRAM Devices

Sense amplifiers in modern DRAM devices perform three different functions. The first function that sense amplifiers perform in a DRAM device is to sense the minute change in voltage that occurs when an access transistor is turned on and a storage capacitor places its charge on the bitline. The sense amplifier compares the voltage on that bitline against a reference voltage as provided on a separate bitline and amplifies the voltage differential to the extreme so that the storage value can be resolved as a digital one or a zero. This role of the sense amplifier is its primary role in DRAM devices, as it senses minute voltage differentials and amplifies them to represent the digital value.

The second function of sense amplifiers in a modern DRAM device is that sense amplifiers also restores the value of a cell after the voltage on the bitline is sensed and amplified. The act of turning on the access transistor allows a storage capacitor to share its stored charge with the bitline. However, the process of sharing the stored value from a storage cell discharges the storage cell. After the process of charge sharing occurs, the voltage level within the storage cell would be roughly equal to the voltage on the bitline, and this voltage level cannot be used for another read operation. As a result, after the sensing and amplification operation, the sense amplifier must also restore the amplified voltage vale to the storage cell.

The third and somewhat surprising function of sense amplifiers in a modern DRAM device is that arrays of sense amplifiers also act as temporary data storage. That is, after data values contained in storage cells are sensed and amplified, the sense amplifiers continue to drive the sensed data values until the DRAM array is precharged and readied for another access. In this manner, data in the same row of cells can be accessed by reading them from the sense amplifier without repeated row accesses to the cells themselves. In this role, the array of sense amplifiers effectively acts as a row buffer that caches an entire row of data. As a result, an array of sense amplifiers is also referred to as a row buffer. Row buffer management policies in essence control operations of the sense amplifiers. Different row buffer management policies dictate whether an array of sense amplifiers will retain the data for an indefinite period of time*, or will discharge it immediately after data has been restored to the storage cells. Active sense amplifiers consume additional current above quiescent power levels, and effective management of sense amplifier operation is an important task for systems seeking optimal trade off points between power and performance.

2.5.2 Circuit Diagram of a Basic Sense Amplifier

Figure 2.4 shows the circuit diagram of a basic sense amplifier. Complex sense amplifiers in modern DRAM devices contain the basic elements shown in Figure 2.4 as well as additional circuit elements for array isolation, careful balance of the sense amplifier structure, and faster sensing capability. In the basic sense amplifier circuit diagram shown in Figure 2.4, the **equalization (EQ)** signal line controls the voltage equalization circuit. The

*. Indefinitely long until mandatory refresh cycle kicks in.

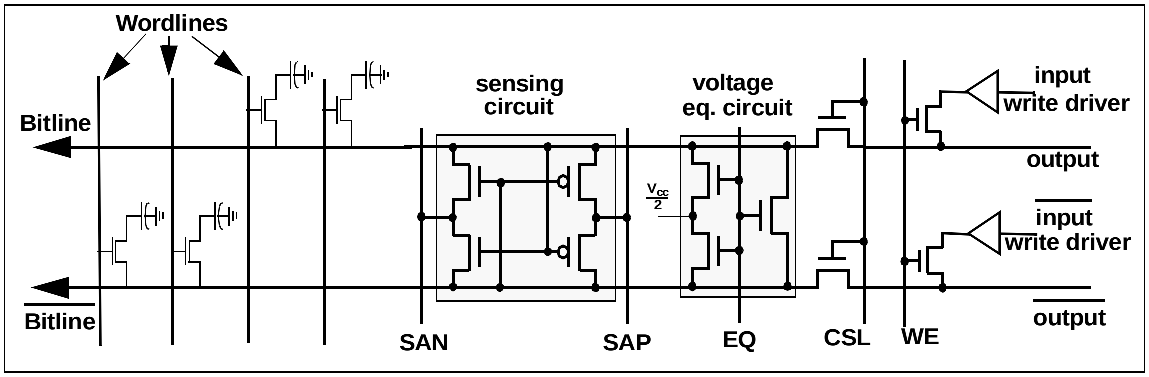


Figure 2.4: Basic sense amplifier circuit diagram.

functionality of this circuit is to ensure that the voltages on the bitline pairs are as closely matched to each other as possible. Since the differential sense amplifier is designed to amplify the voltage differential between the bitline pairs, any voltage imbalance that exists on the bitline pairs prior to the activation of the access transistors would degrade the effectiveness of the sense amplifier.

The heart of the sense amplifier is the set of 4 cross-connected transistors, labelled as the sensing circuit in Figure 2.4. The sensing circuit is essentially a bi-stable circuit, designed to drive the bitline pairs to complementary voltage extremes, depending on the respective voltages on the bitlines at the time the **SAN** and **SAP** sensing signals are activated. After the assertion of the **SAN** and **SAP**, the bitlines are driven to the full voltage levels. The column select line (**CSL**) then turns on the output transistors and allows the fully driven voltage to reach the output and be read out of the DRAM device. At the same time, the access transistor for the accessed cell remains open, and the fully driven voltage on the bitline now re-charges the storage capacitor. In case of a write operation, the input write drivers provide a larger current to overdrive the sense amplifiers and the bitline voltage. The open cell would then be overwritten by the new data values asserted by the input write drivers.

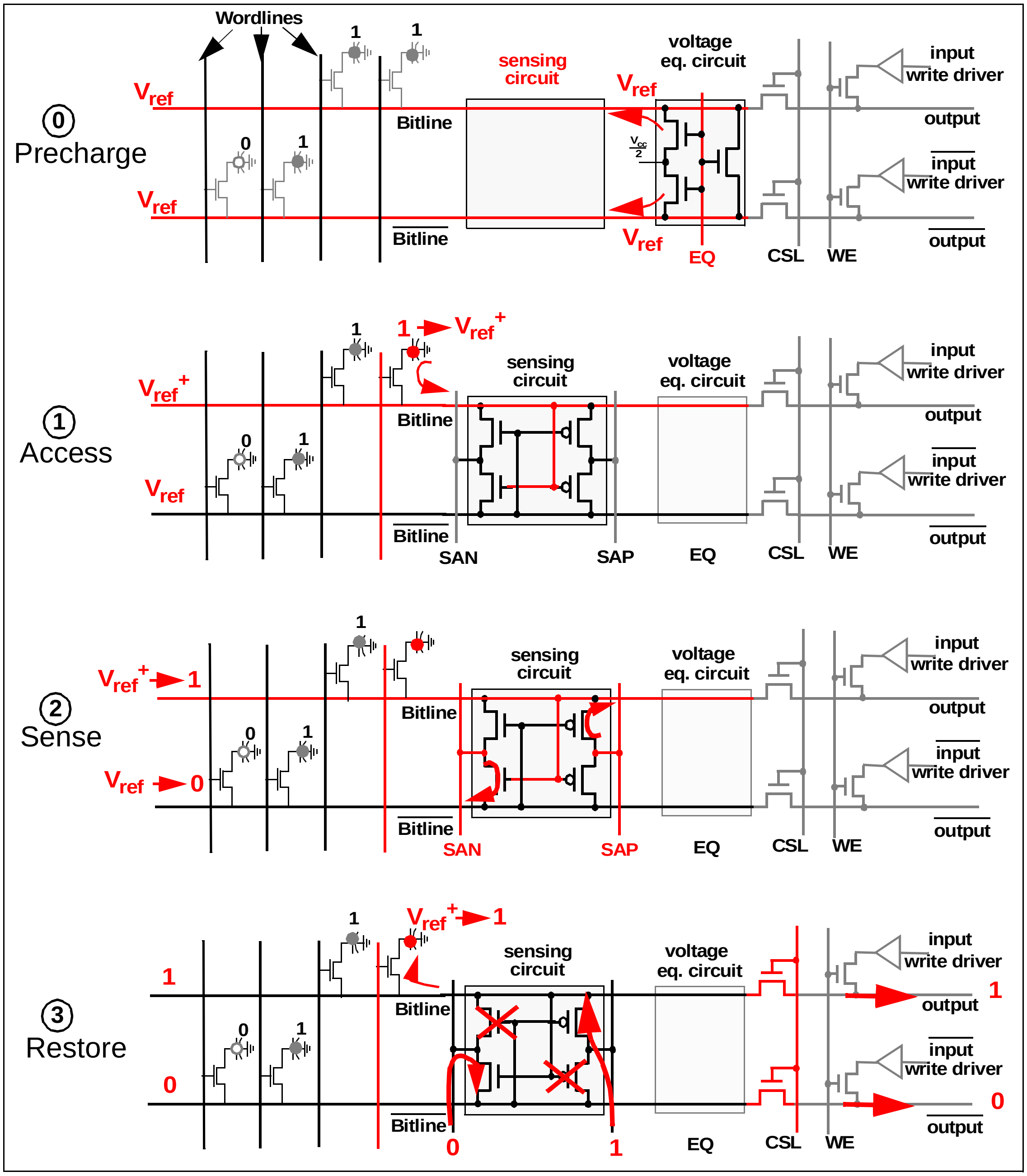


Figure 2.5: Illustrated diagrams of sense amplifier operation. Read(1) example.

2.5.3 Basic Sense Amplifier Operation

Figure 2.5 shows four different phases in the sensing operations of a differential sense amplifier. In Figure 2.5, the operations of a sense amplifiers is labelled as phases zero, one, two and three. The reason that the precharge phase is labelled as phase zero is because the precharge phase is typically considered as a separate operation from a row access operation. That is, while the ***Precharge*** phase is a prerequisite for the subsequent phases of a row access operation, it is typically performed separately from the row access. On the other hand, ***Access***, ***Sense***, and ***Restore*** are three different phases that are performed atomically in sequence for any row access operation.

Phase zero in Figure 2.5 is labelled as ***Precharge***, and it illustrates that before the process of reading data from a DRAM array can begin, the bitlines in a DRAM array is precharged to a reference voltage, V_{ref}. In many modern DRAM devices, $V_{cc}/2$, the voltage half way between the power supply voltage and ground, is used as the reference voltage. In Figure 2.5, the equalization circuit is activated, and the bitlines are precharged to V_{ref}.

Phase one in Figure 2.5 is labelled as (cell) ***Access***, and it illustrates that as a voltage is applied to the right most wordline, that wordline is overdriven to a voltage that is at least V_t above V_{cc}[*]. The voltage on the wordline activates the access transistors, and the charge in the storage cell is discharged onto the bitline. In this case, since the voltage in the storage cell was a high voltage value that represented a digital value of “1”, the voltage on the bitline increases from V_{ref} to V_{ref}^{+}. As the voltage on the bitline changes, the higher voltage on the bitline begins to affect operations of the cross connected sensing circuit. In the case illustrated in Figure 2.5, the slightly higher voltage on the bitline begins to drive the lower NFet to be more conductive than the upper NFet. Conversely, the minute voltage difference drives the lower PFet to be less conductive than the upper PFet. The bitline voltage thus biases the sensing circuit and readies it for the sensing phase.

[*]. The maximum voltage that can be placed across the access transistor is V_{gs} - V_t. (V_t is the threshold voltage of the access transistor, V_{gs} is the gate-source voltage on the access transistor) By overdriving the wordline voltage to V_{cc} + V_t, the storage capacitor could be charged to full voltage (maximum of V_{cc}) by the sense amplifier in the restore phase of the sensing operation. The higher-than-V_{cc} wordline voltage is generated by additional level-shifting voltage pumping circuitry not examined in this text.

Phase two in Figure 2.5 is labelled ***Sense***, and it illustrates that as the minute voltage differences drives a bias into the cross connected sensing circuit, SAN, the DRAM device's Nsense amplifier control signal, turns on and drives the voltage on the lower bitline down[*]. Figure 2.5 shows that as SAN turns on, the more conductive lower NFet allows SAN to drive the lower bitline down in voltage from V_{ref} to ground. Similarly, SAP, the Psense amplifier control signal drives the bitline to a fully restored voltage value that represents the digital value of "1". The SAN and SAP control signals thus collectively force the bi-stable sense amplifier circuit to be driven to the respective maximum or minimum voltages.

Finally, phase three of Figure 2.5 is labelled as ***Restore***, and it illustrates that after the bitlines are driven to the respective maximum or minimum voltage values, the overdriven wordline remains active and the fully driven bitline voltage now restores the charge in the storage capacitor through the access transistor. At the same time, the voltage value on the bitline can be driven out of the sense amplifier circuit to provide the requested data. In this manner, the contents of a DRAM row can be accessed and driven out of the DRAM device concurrently with the data restoration process.

2.5.4 Voltage Waveform of Basic Sense Amplifier Operation

Figure 2.6 shows the voltage waveforms for the bitline and selected control signals illustrated in Figure 2.5. The four phases labelled in Figure 2.6 corresponds to the four phases illustrated in Figure 2.5. Figure 2.6 shows that before a row access operation, the bitline is precharged, and the voltage on the bitline is set to the reference voltage, V_{ref}. In phase one, the wordline voltage is overdriven to at least V_t above V_{cc}, and the DRAM cell

[*]. In modern DRAM devices, the timing and shape of the SAN and SAP control signals are of great importance in defining the accuracy and latency of the sensing operation. However, for the sake of brevity, this text assumes that the timing and shape of these important signals are optimally generated by the control logic.

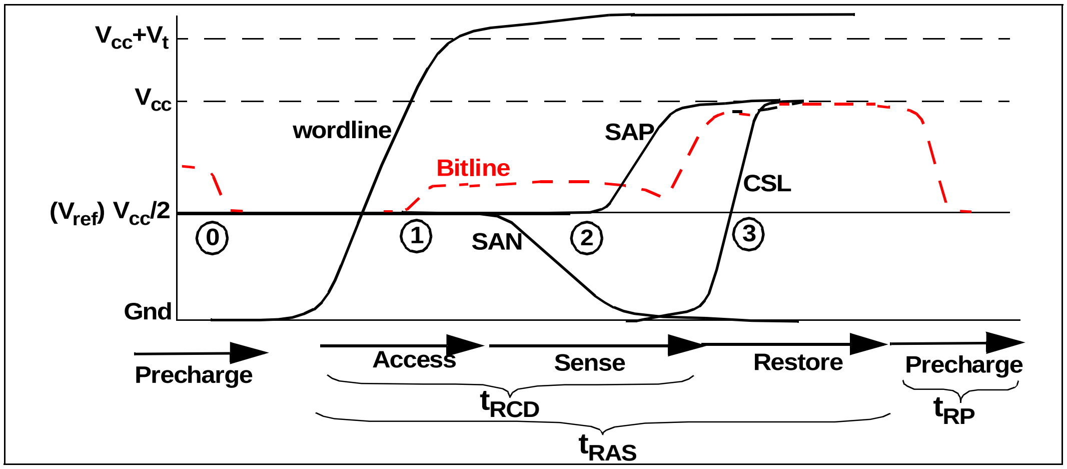


Figure 2.6: Simplified sense amplifier voltage waveform. Read(1) example.

discharges the content of the cell onto the bitline and raises the voltage from V_{ref} to V_{ref}^{+}. In phase two, the sense control signals SAN and SAP are activated in quick succession and drives the voltage on the bitline to the full voltage. The voltage on the bitline then restores the charge in the DRAM cells in phase three.

Figure 2.6 illustrates the relationship between two important timing parameters: t_{RCD} and t_{RAS}. Although the relative durations of t_{RCD} and t_{RAS} are not drawn to scale, Figure 2.6 shows that after time t_{RCD}, the sensing operation is complete, and the data can be read out through the DRAM device's data I/O after that time. However, after time period of t_{RCD} from the beginning of the activation process, data has yet to be restored to the DRAM cells. Figure 2.6 shows that after time period of t_{RAS} from the beginning of the activation process, the data restore operation is assumed to be complete, and the DRAM device is ready to accept a precharge command that will complete the entire row cycle process after time period of t_{RP}.

2.5.5 Writing into DRAM Array

Figure 2.7 shows a simplified timing characteristic for the case of a write command. As part of the row activation command, data is automatically restored from the sense amplifiers to DRAM cells. However, in the case of a write command in commodity DRAM devices, data written by the memory controller is buffered by the I/O buffer of the DRAM device and used to overwrite the sense amplifiers and DRAM cells[*]. In this case, the restore phase may be extended by the write recovery phase. Similar to the relative timing described in Figure 2.6, the addition of a column write command simply means that a precharge command cannot be issued until after the correct data values have been restored to the DRAM cells. The time period required for write data to overdrive the sense amplifiers and through written to the DRAM cells is referred to as the write recovery time, denoted as t_{WR} in Figure 2.7.

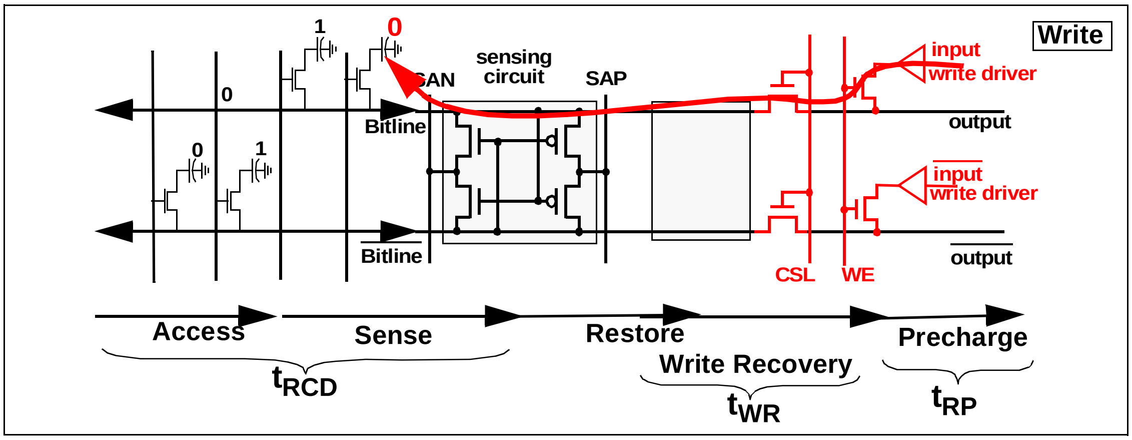


Figure 2.7: Row activation followed by column write into DRAM array.

[*] Some DRAM devices such as Direct RDRAM devices have write buffers. Data isn't driven directly into the DRAM array by the data I/O circuitry in that case, but the write mechanism into the DRAM array remains the same when the write buffer commits the data into the DRAM array prior to a precharge operation.

2.6 DRAM Device Control Logic

All DRAM devices contain some basic logic control circuitry to direct the movement of data onto, within, and off of the DRAM devices. Essentially, some control logic must exist on DRAM devices that accepts externally asserted signal and control, then orchestrates an appropriately timed sequences of internal control signals to direct the movement of data. As an example, previous discussion on sense amplifier operations hinted to the complexity of the intricate timing sequence in the assertion of the wordline voltage followed by assertion of the SAN and SAP sense amplifier control signals, followed yet again by the column select signal. The sequence of timed control signals are generated by the control logic on DRAM devices.

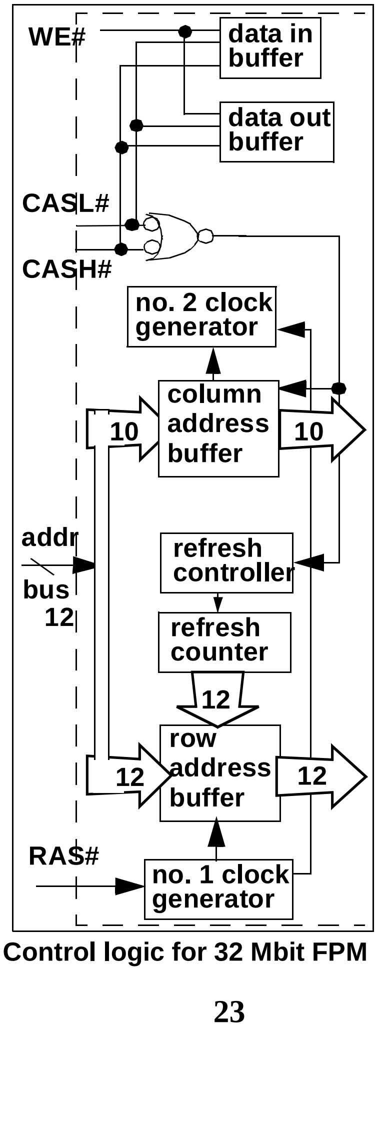


Figure 2.8: Control logic for 32 Mbit FPM DRAM device.

Figure 2.8 shows the control logic that generates and controls the timing and sequence of signals for the sensing and movement of data on the FPM DRAM device illustrated in Figure 2.1. The control logic on the FPM DRAM device asynchronously accepts external signal control and generates the sequence of internal control signals for the FPM DRAM device. The external interface to the control logic on the FPM DRAM device is simple and straightforward, consisting of essentially 3 signals: **row access strobe (RAS)**, **column access strobe (CAS)**, and **write enable (WE)**. The FPM DRAM device described in Figure 3.21 is a device with a 16 bit wide data bus, and the use of separate CASL and CASH signals allow the DRAM devices to control each half of the 16 bit wide data bus separately.

In FPM DRAM devices, the controller to FPM DRAM device interface is an asynchronous interface, and the memory controller directly controls the timing of the movement of data inside the FPM DRAM device. In early generations of DRAM devices such as FPM DRAM devices, the direct control of the internal circuitry of the DRAM device by the external memory controller and the asynchronous nature of the device interface means that the DRAM device could not be well pipelined, and new commands to the DRAM device may not be initiated until the movement of data for the previous command is completed[*]. The asynchronous nature of the interface means that system design engineers can implement different memory controller that operated at different frequencies, and designers of the memory controller are solely responsible to ensure that the controller can correctly control different DRAM devices from different DRAM device and module manufacturers, possibly with subtle timing variations.

[*] For every rule, there are exceptions to the rule. Pipeline burst EDO devices were designed to have some limited pipelining capability with an implicit clocking scheme.

2.6.1 Mode Register Based Programmability

Modern DRAM devices are controlled by synchronous statemachines whose behavior depends on the input values of the command signals as well as the values contained in the programmable mode register in the control logic. Figure 2.9 shows that in an SDRAM device, the mode register contains three fields: CAS latency, burst type, and burst length. Depending on the value of the CAS latency field in the mode register, the DRAM devices returns data two or three cycles after the assertion of the column read command. The value of the burst type determines the ordering of how the SDRAM device returns data, and the burst length field determines the number of columns that a SDRAM device will return to the memory controller with a single column read command. SDRAM devices can be programmed to return 1, 2, 4, 8 columns, or an entire row. Direct RDRAM devices and DDRx SDRAM devices contain more mode registers that control ever larger set of programmable operations including, but not limited to: different operating modes for power conservation, electrical termination calibration modes, self test modes, and write recovery duration.

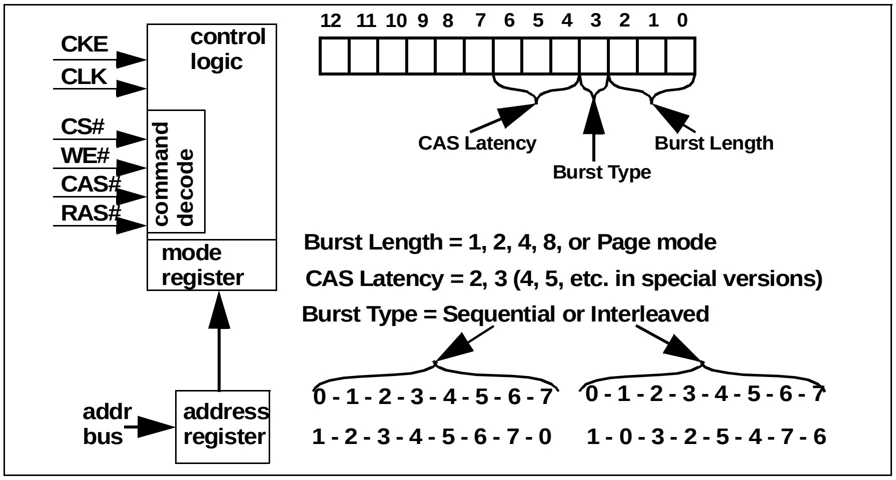


Figure 2.9: Programmable mode register in an SDRAM device.

2.7 DRAM Device Configuration

DRAM devices are classified by the number of data bits in each device, and that number typically quadruples from generation to generation. For example, 64 Kbit devices were followed by 256 Kbit devices, and those devices were in turn followed by 1 Mbit devices. Recently, half generation devices that merely double the number of data bits of previous generation devices have been used to facilitate smoother transitions between different generations. As a result, 512 Mbit devices now exist along side 256 Mbit and 1 Gbit devices.

In a given generation, a DRAM device may be configured with different data bus widths to facilitate its use in different applications. Table 2.1 shows three different configurations of a 256 Mbit device. Table 2.1 shows that a 256 Mbit SDRAM device may be configured with a 4 bit wide data bus, an 8 bit wide data bus or a 16 bit wide data bus. In the configuration with a 4 bit wide data bus, an address provided to the SDRAM device to fetch a single column of data will receive 4 bits of data, and there are 64 million separately addressable locations in the device with the 4 bit data bus. The 256 Mbit SDRAM device with the 4 bit wide data bus is thus referred to as the 64 Meg x4 device. Internally, the 64 Meg x4 device consists of 4 bits of data per column, 2048 columns of data per row, 8192 rows per bank and there are 4 banks in the device. Alternatively, a 256 Mbit SDRAM device with a 16 bit wide

Device configuration	**64 Meg x 4**	**32 Meg x 8**	**16 Meg x 16**
Number of banks	4	4	4
Number of rows	8192	8192	8192
Number of columns	2048	1024	512
Data bus width	4	8	16

TABLE 2.1: **Three different configurations of 256 Mbit SDRAM device**

data bus will have 16 bits of data per column, 512 columns per row, 8192 rows per bank, and 4 banks in the 16 Meg x16 device.

In a typical application, four 16 Meg x16 devices can be connected in parallel to form a single ***rank*** of memory with a 64 bit wide data bus and 128 MB of storage. Alternatively, sixteen 64 Meg x4 devices forms a single rank of memory with a 64 bit wide data bus and 512 MB of storage. DRAM memory system organizations are examined separately in a following chapter.

2.7.1 Device Configuration Trade-offs

In the 256 Mbit SDRAM device, the size of the row does not change in different configurations, and the number of column per row simply decreases with wider data busses specifying a larger number of bits per column. However, the constant row size between different configurations of DRAM devices within the same DRAM device generation is not a generalized trend that can be extended to different device generations. For example, table 2.2 shows different configurations of a 1 Gbit DDR2 SDRAM device where the number of bits per row differs between the x8 configuration and the x16 configuration.

Device configuration	**512 Meg x 4**	**256 Meg x 8**	**128 Meg x 16**
Number of banks	8	8	8
Number of rows	16384	16384	8192
Number of columns	2048	1024	1024
Data bus width	4	8	16

TABLE 2.2: Three different configurations of 1 Gbit DDR2 SDRAM device

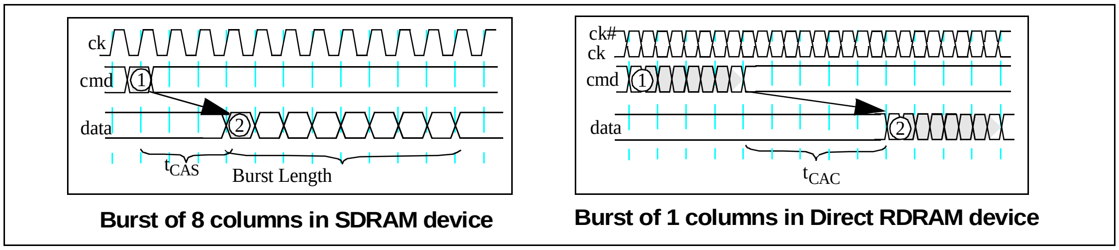
DDR2 SDRAM devices at the 1 Gbit and above densities have 8 banks of DRAM arrays per device. In the x4 and x8 configuration of the 1 Gbit DDR2 SDRAM device, there are 16384 rows per bank and each row consists of 8192 bits. In the x16 configuration, there are

8192 rows and each row consists of 16384 bits. These different configurations lead to different numbers of bits per bitline, different numbers of bits per row activation, and different number of bits per column access. In turn, differences in the number of bits moved per command lead to different power consumption and performance characteristics for different configurations of the same device generation. For example, the 1 Gbit, x16 DDR2 SDRAM device is configured with 16384 bits per row, and each time a row is activated, 16384 DRAM cells are simultaneously discharged onto respective bitlines, sensed, amplified then restored. The larger row size means that a 1 Gbit, x16 DDR2 SDRAM device with 16384 bits per row consumes significantly more current per row activation than 1 Gbit x4 and x8 configuration with 8192 bits per row. The differences in current consumption characteristics in turn leads to difference in timing parameters designed to limit peak power consumption characteristics of DRAM devices.

2.8 Data I/O

2.8.1 Burst Lengths and Burst Ordering

In SDRAM and DDRx SDRAM devices, a column read command moves a variable number of columns. As illustrated in the section on the programmable mode register, an SDRAM device can be programmed to return 1, 2, 4, or 8 columns of data as a single burst that takes 1, 2, 4 or 8 cycles to complete. In contrast, a Direct RDRAM device returns a single column of data with an 8 beat* burst. Figure 2.10 shows an 8 beat, 8 column read data burst from an SDRAM device and an 8 beat, single column read data burst from a Direct RDRAM device. The distinction between the 8 column burst of an SDRAM device and the single column data burst of the Direct RDRAM device is that each column of the SDRAM device is individually addressable, and given an a column address in the middle of an 8 column burst, the SDRAM device will re-order the burst to provide the data of the requested address first. This capability is known as critical-word forwarding. For example, in an SDRAM device programmed to provide a burst of 8 columns, a column read command with a column address of 17 will result in the data burst of 8 columns of data with the address sequence of 17-18-19-20-21-22-23-16 or 17-16-19-18-21-20-23-22, depending on the burst type as defined in the programmable register. In contrast, each column of a Direct RDRAM



Burst of 8 columns in SDRAM device

Burst of 1 columns in Direct RDRAM device

Figure 2.10: Burst lengths in DRAM devices.

*. In DDRx and Direct RDRAM devices, two beats of data are transferred per clock cycle.

device consists of 128 bits of data, and each column access command moves 128 bits of data in a burst of 8 contiguous beats in strict burst ordering. That is, differing from SDRAM and DDRx SDRAM devices, Direct RDRAM devices support neither programmable burst lengths nor different burst ordering.

2.8.2 N-bit Prefetch

In SDRAM devices, each time a column read command is issued, the control logic determines the duration and ordering of the data burst, and each column is moved separately from the sense amplifiers through the I/O latches to the external data bus. However, the separate control of each column limits the operating data rate of the DRAM device. As a result, in successive generations of SDRAM, and DDRx SDRAM devices, successively larger numbers of bits are moved in parallel from the sense amplifiers to the read latch, and the data is then pipelined through a multiplexor to the external data bus.

Figure 2.11 illustrates the data I/O structure of a DDR SDRAM device. Figure 2.11 shows that given the width of the external data bus as N, 2N bits are moved from the sense amplifiers to the read latch, the 2N bits are then pipelined through the multiplexors to the external data bus. In DDR2 SDRAM devices, and the number of bits prefetched by the

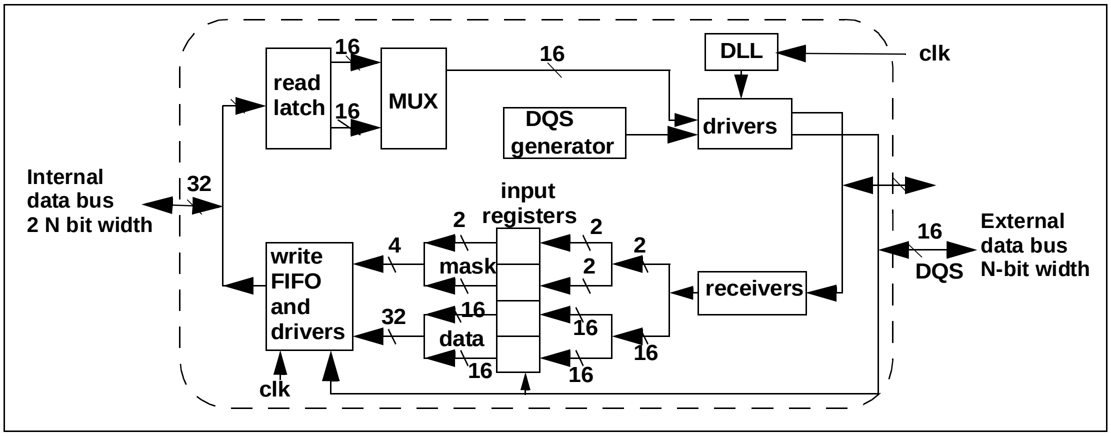


Figure 2.11: Data I/O in DDR SDRAM device illustrating 2-bit prefetch.

internal data bus is 4N. The N bit prefetch strategy in DDRx SDRAM devices means that internal DRAM circuits can remain essentially unchanged between transitions from SDRAM to DDRx SDRAM, but operating data-rate of DDRx SDRAM devices can be increased to levels not possible with SDRAM devices. However, the downside of the N bit prefetch architecture is that short bursts are no longer supported. For example, in DDR SDRAM devices, a minimum burst length of 2 columns of data are accessed per column read command, and in DDR2 SDRAM devices, a minimum burst length of 4 columns of data are accessed per column read command. This trend is likely to continue in future generations of DDR3 and DDR4 SDRAM devices, thus requiring longer data bursts for each successive generations of higher data rate DRAM devices.

2.9 DRAM Device Packaging

One difference between DRAM and logic devices is that most DRAM devices are commodity items whereas logic devices such as processors and ***application specific integrated circuits (ASIC)*** are typically specialized devices that are not commodity items. The result of the commodity status for DRAM devices is that even more so than logic device manufacturers, DRAM device manufacturers are extraordinarily sensitive to cost. One area that reflects the cost sensitivity is the packaging technology utilized by DRAM devices. Table 2.3 shows the expected pin count and relative costs from the 2002 ***International Technology Roadmap for Semiconductors (ITRS)*** for high performance logic devices as compared to memory devices. Table 2.3 shows the trend that memory chips such as DRAM will continue to be manufactured with relatively lower cost packaging with lower pin count and lower cost per pin.

	2004	**2007**	**2010**	**2013**	**2016**
Semi Generation (nm)	90	65	45	32	22
High Perf. device pin count	2263	3012	4009	5335	7100
High Perf. device cost (cents/pin)	1.88	1.61	1.68	1.44	1.22
Memory device pin count	**48-160**	**48-160**	**62-208**	**81-270**	**105-351**
Memory device pin cost (cents/pin)	0.34 -1.39	0.27 - 0.84	0.22 - 0.34	0.19 - 0.39	0.19 - 0.33

TABLE 2.3: ITRS roadmap projections for package pin count and costs

Figure 2.12 shows 4 different packages used in previous and current generations of DRAM devices. Early DRAM devices typically packaged in low pin count and low cost ***Dual Inline Packages (DIP)***. Increases in DRAM device density and wider data bus widths have required the use of the higher pincount and larger ***Small Outline J-lead (SOJ)*** packages. DRAM devices then moved to ***Thin Small Outline Package (TSOP)*** in the late 1990's. As DRAM device data rates increases to multiple hundreds of megabits per second, ***Ball Grid Array (BGA)*** packages are needed to better control signal interconnects at the package level.

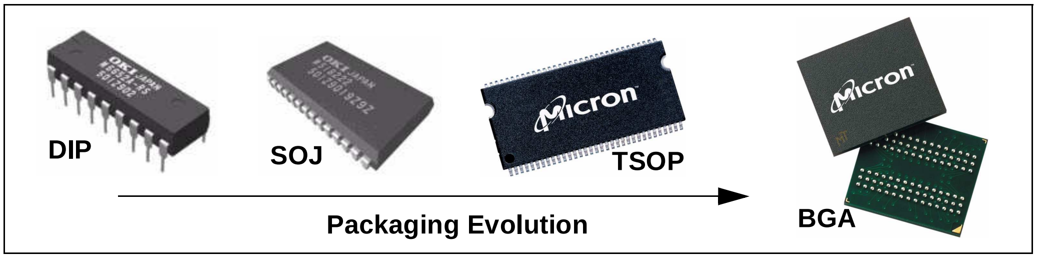


Figure 2.12: DRAM device packages.

2.10 A 256 Mbit SDRAM Device

Figure 2.13 shows the die photograph of a 256 Mbit SDRAM device. Figure 2.13 shows that much of the surface area of the silicon die is dominated by the regular structures of the DRAM arrays. In this case, roughly 70% of the silicon surface is used by the DRAM arrays, and the rest of the area is taken up by I/O pads, sense amplifiers, decoders and the minimal control logic. The SDRAM device shown in Figure 2.13 is manufactured on a DRAM optimized 0.11um process with 3 layers of metal interconnects and 6 layers of poly silicon. The SDRAM device is contained in a low cost 54 pin TSOP package. On a commodity SDRAM device, 14 pins of the 54 pin package are used for power and ground, 16 pins are used for the data bus, 15 pins are used for the address bus, 7 pins are used for the command bus, and a single pin is used for the clock signal.

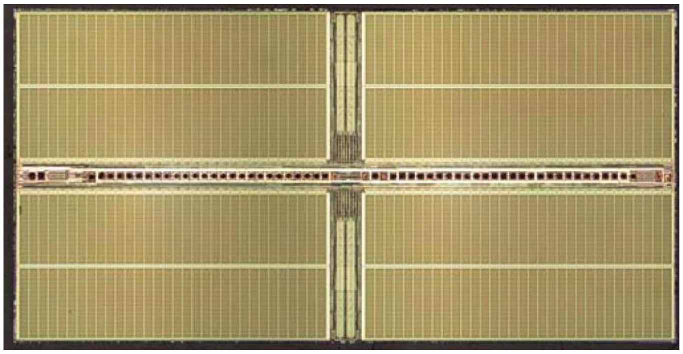


Figure 2.13: 256 Mbit SDRAM device from Micron.

2.10.1 SDRAM Device Block Diagram

Figure 2.14 shows a block diagram of the 256 Mbit SDRAM device. Figure 2.14 shows that unlike the FPM DRAM device illustrated in Figure 2.1, the 256 Mbit SDRAM device has 4 banks of DRAM arrays, each with its own array of sense amplifier, row decoders and column decoders. Similar to the FPM device, the SDRAM device contains separate address

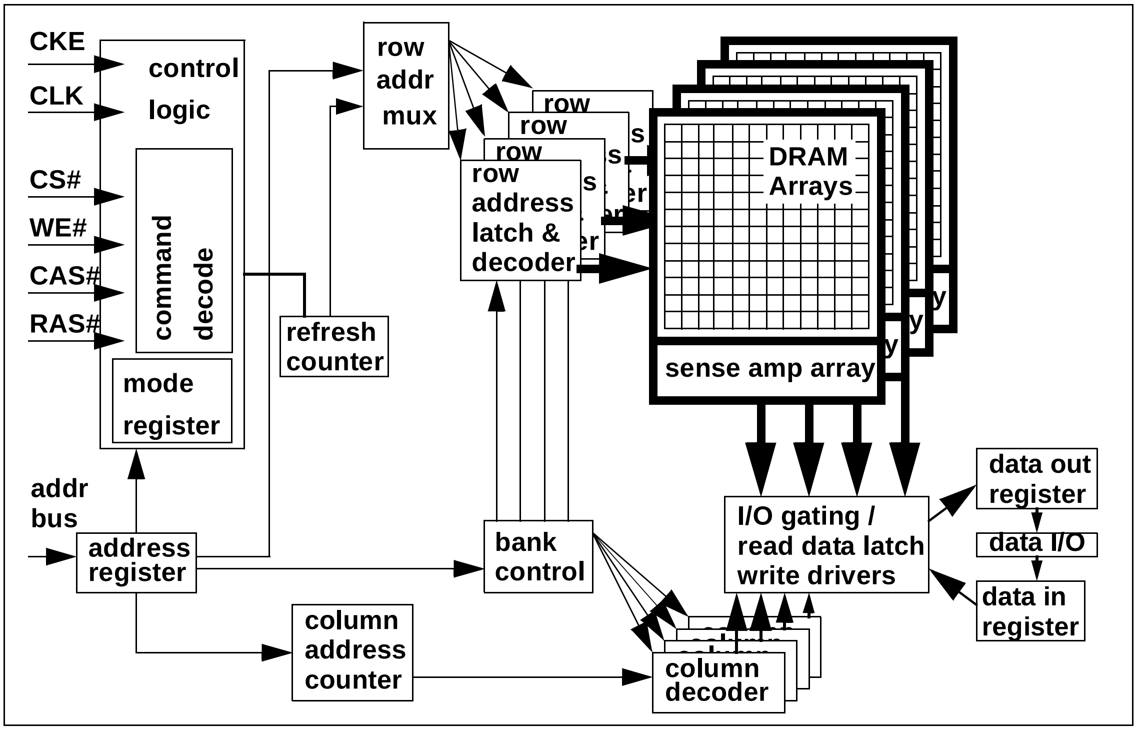


Figure 2.14: SDRAM Device Architecture with 4 Banks.

registers that are used to control dataflow on the SDRAM device. In case of a row access command, the address from the address register is forwarded to the row address latch and decoder, and that address is used to activate the selected wordline. Data is then discharged onto the bitlines and the sense amplifiers array senses, amplifies and holds the data until a subsequent column access command either reads the data through the I/O gating out to the data bus, or accepts write data from the data bus through the I/O gating, overwrites data in the sense amplifier arrays, then overwrites data in the DRAM cells to the new values.

2.10.2 Pin Assignment and Functionality

In an SDRAM device, commands are decoded on the rising edge of the clock signal (CLK) if the chip select line (CS#) is active. The command is asserted by the DRAM controller on the command bus, which consists of the write enable (WE#), column access

	CS#	RAS#	CAS#	WE#	addr
command inhibit (nop)	H	X	X	X	X
no operation (nop)	L	H	H	H	X
active (activate row - RAS)	L	L	H	H	addr
read (start read - CAS)	L	H	L	H	addr
write (start write - CAS W)	L	H	L	L	addr
burst terminate	L	H	H	L	X
precharge	L	L	H	L	**
auto refresh	L	L	L	H	X
load mode register	L	L	L	L	code

** bank address, or all banks (with a_10 assertion)

TABLE 2.4: **SDRAM commands**

(CAS#), and row access (RAS#) signal lines. Although the signal lines have function-specific names, they essentially form a command bus, allowing the SDRAM device to recognize more commands without the use of additional signal lines. Table 2.4 shows the command set of the SDRAM device and the signal combinations on the command bus. Table 2.4 shows that as long as CS# is not selected, the SDRAM device ignores the signals on the command bus. In the case that CS# is active on the rising edge of the clock, the SDRAM device then decodes the combination of control signals on the command bus. For example, the combination of an active low voltage value on RAS#, high voltage value on CAS# and high voltage value on WE#, the SDRAM device recognizes that this combination signifies a row activation command and begins the row activation process on the selected bank and row address as provided on the address bus.

Another command allows the SDRAM device to load in new values for the mode register from the address bus. That is, in case that CS#, RAS#, CAS# and WE# are all active on the rising edge of the clock signal, the SDRAM device decodes the load mode register command and loads the mode register from value presented on the address bus.

2.11 Process Technology and Scaling Considerations

The 1T1C cell structure places specialized demands on the access transistor and the storage capacitor. Specifically, the cross sectional area occupied by the 1T1C DRAM cell structure must be small, leakage through the access transistor must be low, and the capacitance of the storages capacitor must be large. The data retention time and data integrity requirements provide the boundaries in the design of a DRAM cell. Different DRAM devices can be designed to meet the demand of high performance or low cost market. Presently, DRAM devices are manufactured on DRAM-optimized process technologies whereas logic devices are typically manufactured on logic-optimized process technologies. DRAM optimized process technologies can be used to fabricate logic circuits, and logic optimized process technologies can also be used to fabricate DRAM circuits. However, DRAM optimized process technologies have diverged substantially from logic optimized process technologies in recent years. Consequently, it has become less economically feasible to fabricate DRAM circuits in logic optimized process technology and logic circuits fabricated in DRAM optimized process technology is much slower than similar circuits in a logic optimized process technology. These trends have conspired to keep logic and DRAM circuits separate in different devices.

2.11.1 Cost Considerations

Historically, manufacturing cost considerations have dominated the design of standard, commodity DRAM devices. In the spring of 2003, a single 256 megabit DRAM device, using roughly 45 mm^2 of silicon die area on a 0.11um DRAM process had a selling price of approximately $4 per chip. In contrast, a desktop Pentium 4 processor from Intel, using roughly 130 mm^2 of die area on a 0.13um logic process, had a selling price that ranged from \$80 to \$600 in the comparable time frame. Although the respective selling prices were due to the limited sources, non-commodity nature of processors and the pure commodity economics of DRAM devices, the disparity does illustrate the level of price competition on the sale of commodity DRAM devices. The result is that DRAM manufacturers are singularly focused on the low cost aspect of DRAM devices. Any proposal to add additional functionalities must then be weighed against the increase in die cost and possible increases in selling price.

2.11.2 DRAM-versus-Logic Optimized Process Technologies

One apparently inevitable trend in semiconductor manufacturing is the march toward integration. As the semiconductor manufacturing industry dutifully fulfills Moore's Law, each doubling of transistors allow design engineers to pack more logic circuitry or more DRAM storage cells onto a single piece of silicon. However, the semiconductor industry has thus far generally resisted the integration of DRAM and logic onto the same silicon device for various technical and economic reasons.

Figure 2.15 illustrates some technical issues that have prevented large scale integration of logic circuitry with DRAM storage cells. Basically, logic optimized process technologies have been designed for transistor performance while DRAM optimized process technologies have been designed for low cost, error tolerance and leakage resistance. Figure 2.15 shows a typical logic based process technology with 7 or more layers of copper interconnects while a typical DRAM optimized process technology has only 2 layers of aluminum interconnects along with perhaps an additional layer of tungsten for local

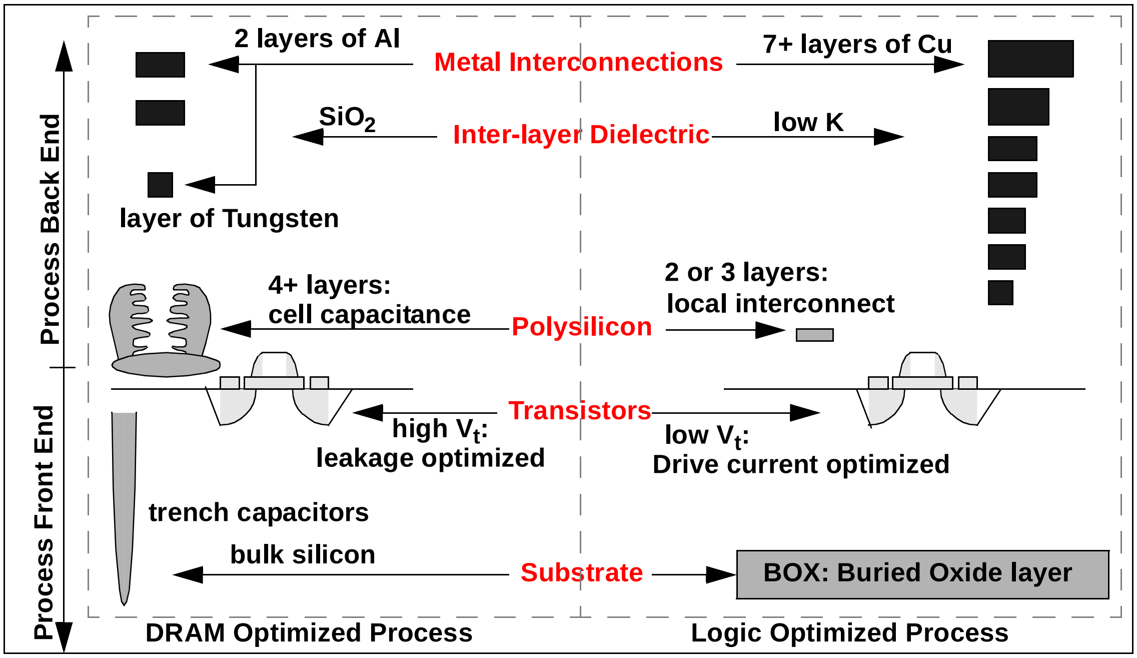


Figure 2.15: Comparison of DRAM Optimized Process versus Logic Optimized Process.

interconnects. Moreover, a logic optimized process typically uses low K material for the inter-layer dielectric while the DRAM optimized process uses the venerable SiO_2. Figure 2.15 also shows that a DRAM optimized process would use 4 or more layers of polysilicon to form the structures of a stacked capacitor (for those DRAM devices that use the stacked capacitor structure), while the logic optimized process merely uses 2 or 3 layers of polysilicon for local interconnects. Also, transistors in a logic optimized process are typically tuned for high performance while transistors in a DRAM optimized process are tuned singularly for low leakage characteristics. Finally, even the substrate of the respectively optimized process technologies are diverging as logic optimized process technologies move to depleted substrates and DRAM optimized process technologies largely stays with bulk silicon.

The respective specializations of the differently optimized process technologies have largely succeeded in preventing widespread integration of logic circuitry with DRAM storage cells. The use of a DRAM optimized process as the basis of integrating logic circuits and DRAM storage cells lead to slow transistors with low drive currents connected to few layers of metal interconnects and relatively high K SiO_2 inter-layer dielectric. That is, logic circuits implemented on a DRAM optimized process would be substantially larger as well as slower than comparable circuits on a similar generation logic optimized process[*]. Conversely, the use of a higher cost logic optimized process as the basis of integrating logic circuits and DRAM storage cells lead to high performance but leaky transistors coupled with DRAM cells with relatively lower capacitance, necessitating large DRAM cell structures and high refresh rates.

In recent years, new hybrid process technologies have emerged to solve various technical issues limiting the integration of logic circuits and DRAM storage cells. Typically, the hybrid process starts with the foundation of a logic optimized process, then additional layers are added to the process to create high capacitance DRAM storage cells. Also, different types of transistors are made available for use as low leakage access transistors as well as high drive current high performance logic transistors. However, hybrid process technology then becomes more complex than a logic optimized process. As a result, hybrid process technologies that enable seamless integration of logic and DRAM devices are typically more expensive, and their use have thus far been limited to specialty niches that require high performance processor and high performance and yet small DRAM memory systems that are limited by the die size of a single logic device. Typically, the application has been limited to high performance System-on-Chip (SOC) devices.

[*]. DRAM cell sizes in hybrid logic based process technologies are

CHAPTER 3

DRAM Memory System Organization

In this chapter, basic terminologies and basic building blocks of DRAM memory systems are described. While the previous chapter examined the operations of a single DRAM device, this chapter examines the construction, organization and operation of multiple DRAM devices in the context of a complete memory system. The goal of this chapter is to cover the definition of basic terminologies sufficient to describe DRAM memory system organizations and establish a common nomenclature for use throughout this work. The performance analysis and DRAM scheduling algorithm in subsequent chapters are described by using the basic terminology defined in this chapter.

3.1 Conventional Memory system

Historically, the number of storage bits contained in a single DRAM device has been inadequate to serve as the main memory for most computing platforms, with the exception of specialty embedded systems. In the past few decades, the growth rate of DRAM device storage capacity has roughly paralleled the growth rate of the size of memory systems for desktop computers, workstations and servers. The parallel growth rates in DRAM device storage capacity and DRAM memory system capacity have dictated system designs in that multiple DRAM devices must be interconnected together to form larger memory systems for most computing platforms. In this chapter, the organization of different multi-chip DRAM

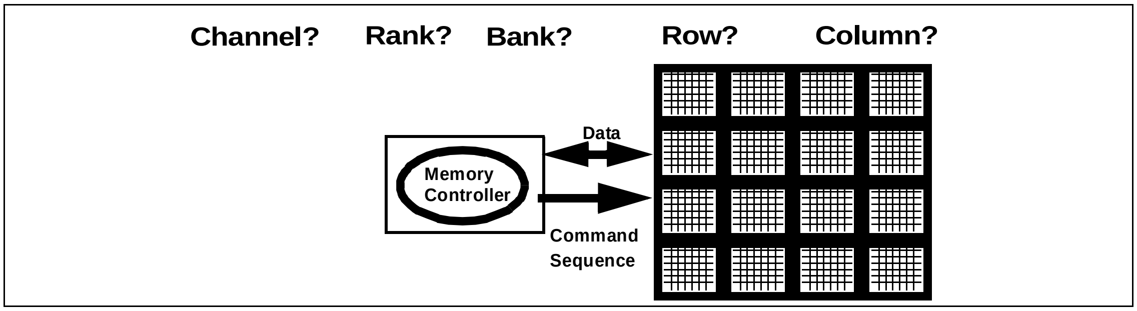


Figure 3.1: Multiple DRAM devices connected to a processor through a memory controller.

memory systems and different interconnection strategies, deployed for cost and performance concerns, are explored.

In Figure 3.1, multiple DRAM devices are interconnected together to form a single memory system that is managed by a single memory controller. In modern computer systems, one or more ***DRAM memory controllers*** (DMC) may be contained in the processor package or integrated into a system controller that resides outside of the processor package. Regardless of the location of the DRAM memory controller, the functionality of the DRAM memory controller is to accept read and write requests to a given address in memory, translate the request to one or more commands to the memory system, issue those commands to the DRAM devices in the proper sequence and proper timing, and retrieve or store data on behalf of the processor or I/O devices in the system.

3.2 Basic Nomenclature

The organization of multiple DRAM devices into a memory system can impact the performance of the memory system in terms of operating datarates, latency, and sustainable bandwidth characteristics. It is therefore of great importance that the organization of multiple DRAM devices into larger memory systems be examined in detail. However, the problem that has hindered the examination of DRAM memory system organizations is the lack of clearly defined nomenclature. Without a common basis of well defined nomenclature, technical articles and datasheets often succeed in introducing confusion rather than clarity to discussions on DRAM memory systems. In one particularly egregious example, a technical datasheet for a system controller used the word ***bank*** in two bullet items on the same page to mean two different things. In this datasheet, one bulleted item trumpeted that the system controller can support 6 ***banks*** (of DRAM devices). Then, several bulleted items later, the same datasheet states that the same system controller supports SDRAM devices with 4 ***banks***. In a second example that was nearly as egregious, an article in a well respected technical journal examined the then new Intel i875P system controller, and proceeded to discuss the performance advantage of the system controller due to the fact that it could control two ***banks*** of DRAM devices.

In these two examples, the word ***bank*** was used to mean three different things. While the meaning of the word ***bank*** can be inferred from context in each case, the overloading and repeated use of the terminology introduces unnecessary confusion into discussions about DRAM memory systems. In this section, the terminology of channel, rank, bank, row and column is defined and all subsequent discussions in this work will conform to their usage as defined in this chapter.

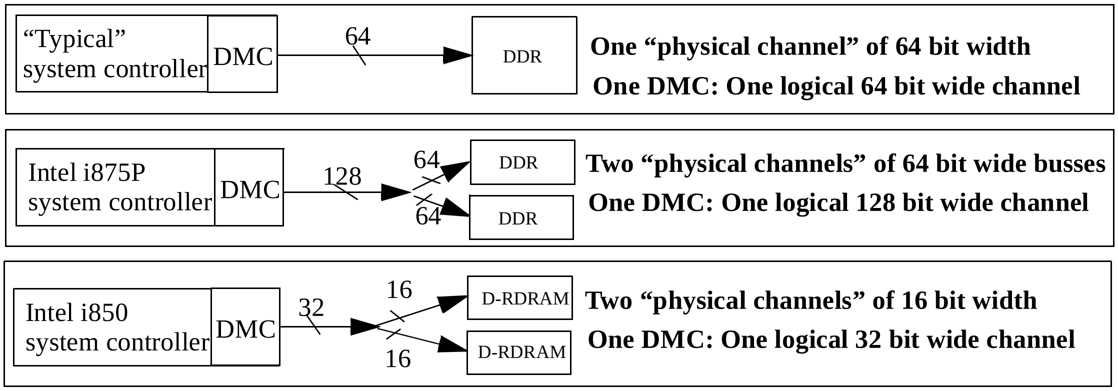
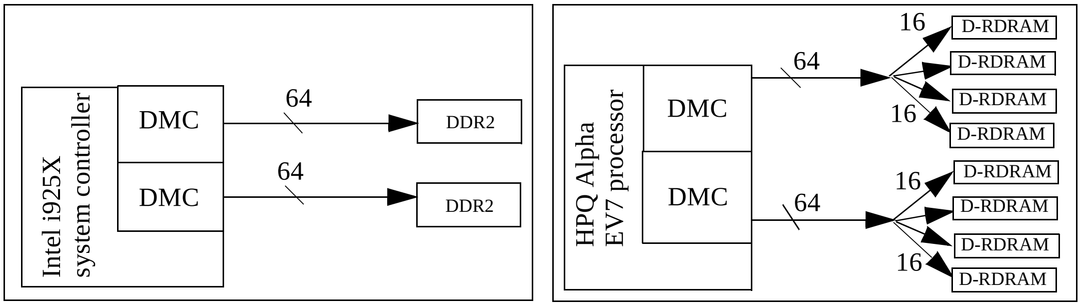

“Typical”
system controller
DMC
64
DDR
One “physical channel” of 64 bit width
One DMC: One logical 64 bit wide channel
Intel i875P
system controller
DMC
128
64
64
DDR
DDR
Two “physical channels” of 64 bit wide busses
One DMC: One logical 128 bit wide channel
Intel i850
system controller
DMC
32
16
16
D-RDRAM
D-RDRAM
Two “physical channels” of 16 bit width
One DMC: One logical 32 bit wide channel

Figure 3.2: Systems with single memory controller and different data bus widths.

3.2.1 Channel

Figure 3.2 shows three different system controllers with slightly different configurations of the DRAM memory system. In Figure 3.2, each system controller has a single DMC, and each DMC controls a single channel of memory. In the example labelled as the ***typical system controller*** in Figure 3.2, the system controller controls a single 64 bit wide channel. In modern DRAM memory systems, commodity non-ECC DRAM memory modules are standardized with 64 bit wide data busses, and the 64 bit data bus width of the memory module matches the data bus width of the typical personal computer system controller. In the exampled labelled as ***Intel i875P system controller***, the system controller connects to a single channel of DRAM with a 128 bit wide data bus. However, since commodity DRAM modules have 64 bit wide data busses, matching pairs of 64 bit wide memory modules are required for Intel’s i875P system controller to operate with the 128 bit wide data bus. The paired-memory-module configuration of Intel’s i875p system controller is often referred to as a ***dual channel*** configuration**.** However, since there is only one memory controller and both memory modules operate in parallel to store and retrieve data through the 128 bit wide data bus, the paired-memory module configuration is in fact a 128



Two Channels: 64 bit wide per channel **Two Channels: 64 bit wide per channel**

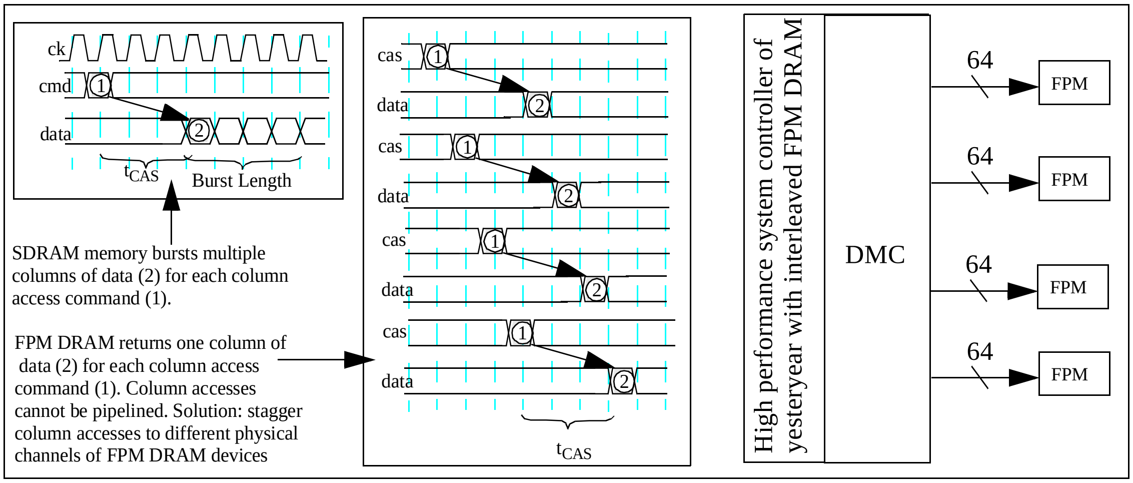
Figure 3.3: Systems with two independent memory controllers and two logical channels.

bit wide single channel memory system. Also, similar to SDRAM and DDR SDRAM memory systems, standard Direct RDRAM memory modules are designed with 16 bit wide data busses and system controllers such as the Intel i850 system controller use matched pairs of Direct RDRAM memory modules to form a single 32 bit wide logical channel of memory.

In contrast to system controllers that use a single DRAM memory controller to control the entire memory system, Figure 3.3 shows that the Alpha EV7 processor and the Intel i925x system controller each has two DRAM memory controllers that independently control 64 bit wide data busses*. The use of independent DRAM memory controllers can lead to higher sustainable bandwidth characteristics since the narrower channels lead to longer data bursts per cacheline request and the various inefficiencies dictated by DRAM access protocols can be better amortized. As a result, newer system controllers are often designed with multiple memory controllers despite the die cost of adding memory controllers.

Modern memory systems with one DRAM memory controller and multiple physical channels of DRAM devices such as those illustrated in Figures 3.2 are typically designed with the physical channels operating in lockstep with respect to each other. However, there

*. Ignoring additional bit widths used for error correction and cache directory in the case of the Alpha EV7 processor.


ck
cmd
data
1
2
tCAS
Burst Length
cas
data
1
2
tCAS
High performance system controller of yesteryear with interleaved FPM DRAM
DMC
64
64
64
64
FPM
FPM
FPM
FPM

SDRAM memory bursts multiple columns of data (2) for each column access command (1).

FPM DRAM returns one column of data (2) for each column access command (1). Column accesses cannot be pipelined. Solution: stagger column accesses to different physical channels of FPM DRAM devices

Figure 3.4: High performance DMC with 4 channels of interleaved FPM DRAM devices.

are two variations to the single-controller-multiple-physical-channel configuration. One variation of the single-controller-multiple-physical-channel configuration is that some system controllers, such as the Intel i875P system controller, allow the use of mismatched pairs of memory modules in the different physical channels. In such a case, the i875p system controller operates in an ***asymmetric*** mode and independently controls the physical channels of DRAM modules. However, since there is only one DRAM memory controller, the multiple physical channels of mismatched memory modules cannot be accessed concurrently, and only one channel of memory can be accessed at any given instance in time. In the asymmetric configuration, the maximum system bandwidth is the maximum bandwidth of a single physical channel.

A second variation of the single controller-multiple-physical-channel configuration can be found in high performance FPM DRAM memory systems that were designed prior to the emergence of DRAM devices that support consecutive-cycle data bursts. Figure 3.4 illustrates a sample timing diagram of a column access in an SDRAM memory system. Figure 3.4 shows that an SDRAM device is able to return a burst of multiple columns of data for a single column access command. However, an FPM DRAM device supported neither single-access-multiple-burst capability nor the ability to pipeline multiple column access commands. As a result, FPM DRAM devices need multiple column accesses that cannot be pipelined to retrieve the multiple columns of data for a given cacheline access.

One solution deployed to overcome the shortcomings of FPM DRAM devices is the use of multiple FPM DRAM channels in a single memory system that operates in an interleaved fashion. Figure 3.4 also shows how a sophisticated FPM DRAM memory system can send multiple column accesses to different physical channels in such a manner so that the data for the respective column accesses appear on the data bus in consecutive cycles. In this configuration, the multiple FPM DRAM channels provided the sustained throughput required in high performance workstations and servers before the appearance of modern synchronous DRAM devices that can burst through multiple columns of data in consecutive cycles.

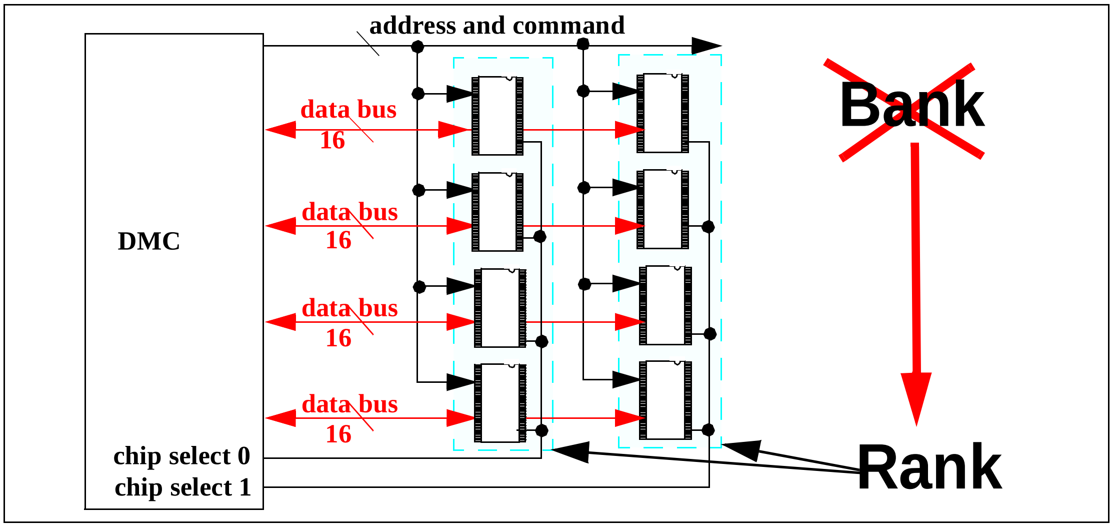


Figure 3.5: Memory System with 2 ranks of DRAM devices.

3.2.2 Rank

Figure 3.5 shows a memory system populated with 2 ranks of DRAM devices. Essentially, a ***rank*** of memory is a ***"bank"*** of one or more DRAM devices that operate in lockstep in response to a given command. However, the word ***"bank"*** is currently used by DRAM device manufacturers to describe the number of independent DRAM arrays within a DRAM device. To lessen the confusion associated with overloading the nomenclature, the word ***rank*** is now used to denote a set of DRAM devices that operate in lockstep fashion to commands in a memory system.

Figure 3.5 illustrates a configuration of two ranks of DRAM devices in a single channel in a classical DRAM memory system topology. In the classical DRAM memory system topology, the address and command busses are connected to every DRAM device in the memory system, but the wide data bus is partitioned and connected to different sets of DRAM devices within the system. The memory controller then uses chip-select signals to select the appropriate rank of DRAM devices to respond to a given command.

3.2.3 Bank

The word ***bank*** has been over used to mean a number of different things in the memory system. As described previously, the word ***bank*** had been used to describe independent memory arrays inside a DRAM device, a set of DRAM devices that collectively act in response to commands, as well as different physical channels of memory. In this text, the word ***bank*** is used strictly to denote an independent memory array inside a DRAM device.

Figure 3.6 shows an SDRAM device with 4 banks. Modern DRAM devices contain multiple banks so that multiple, independent accesses to different DRAM arrays can occur in parallel. In this design, each bank of memory is an independent array that can be in different phases of the row access cycle. Some common resources, such as the I/O gating that allows access to the data pins must be shared between different banks. However, the multi-bank architecture allows commands such as read requests to different banks to be pipelined. Certain commands, such as refresh commands, can also be engaged in multiple banks in parallel. In this manner, multiple banks can operate independently or concurrently depending on the command.

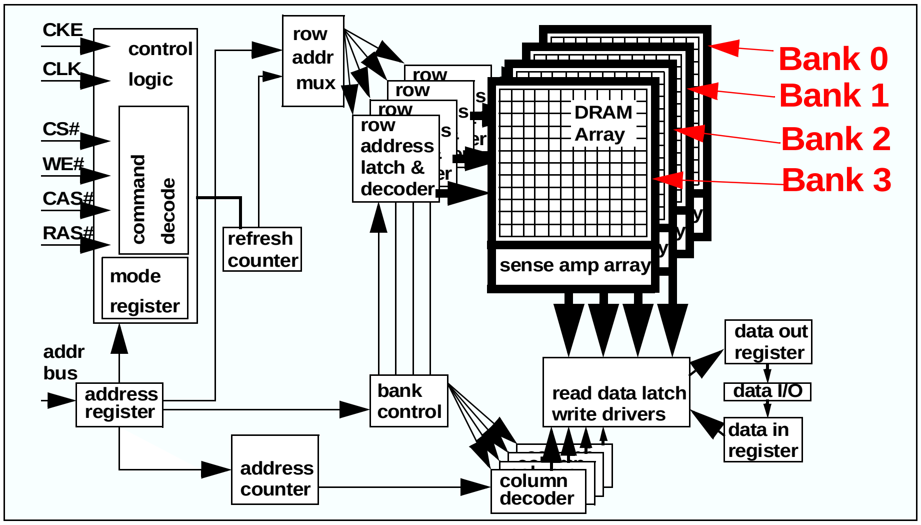


Figure 3.6: SDRAM device with 4 banks of DRAM arrays internally.

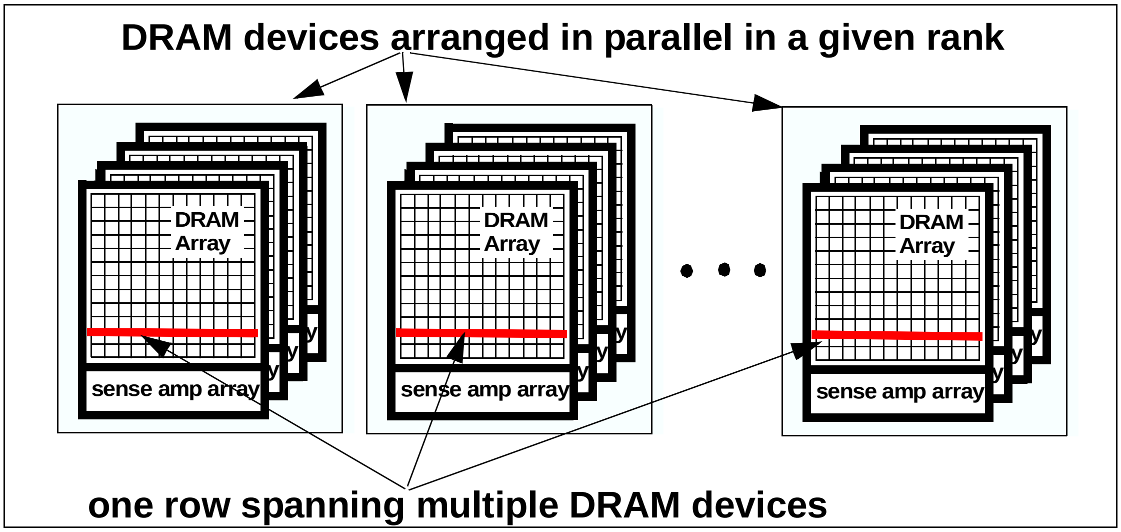

DRAM devices arranged in parallel in a given rank
DRAM Array
sense amp array
DRAM Array
sense amp array
DRAM Array
sense amp array
one row spanning multiple DRAM devices

Figure 3.7: DRAM devices with 4 banks, 8192 rows per bank, 512 columns per row, and 16 bits per column.

3.2.4 Row

In DRAM devices, a ***row*** is simply the group of storage cells that are activated in parallel in response to a row activation command. In DRAM memory systems that utilize the conventional system topology such as SDRAM, DDR SDRAM and DDR2 SDRAM memory systems, multiple DRAM devices are typically connected in parallel as ranks of memory. Figure 3.5 shows how DRAM device can be connected in parallel to form ranks of memory devices. The effect of DRAM devices connected as ranks of DRAM devices that operate in lockstep is that a row activation command will activate the addressed row in all DRAM device of a given rank of memory. This arrangement means that the size of a row is multiplied by the number of DRAM devices in a given rank, and a DRAM row spans across the multiple DRAM devices of a given rank of memory.

A ***row*** is also sometimes referred to as a DRAM page, since a row activation command in essence activates a page of memory. DRAM pages are typically several kilobytes in size, and they are cached at the sense amplifiers until a subsequent precharge command is issued

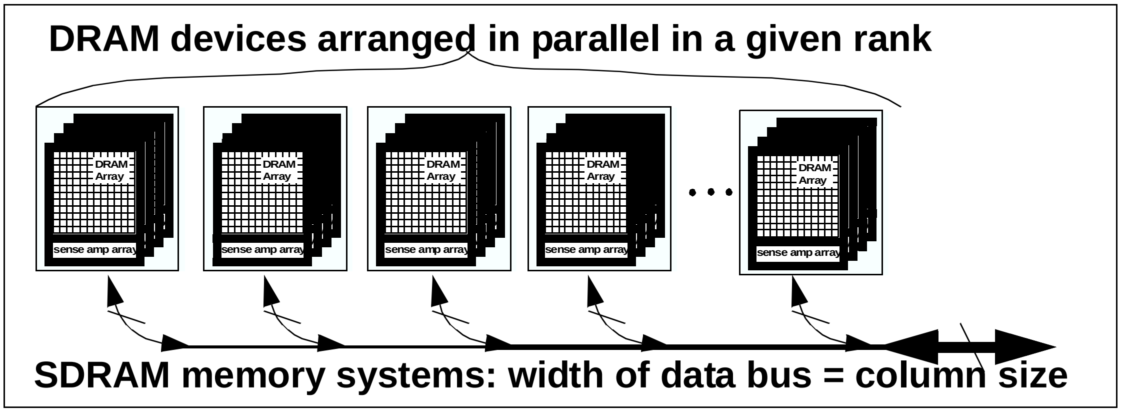

DRAM devices arranged in parallel in a given rank
DRAM Array
sense amp array
SDRAM memory systems: width of data bus = column size

Figure 3.8: Classical DRAM system topology, width of data bus equals column size.

by the DRAM memory controller. Various schemes have been proposed to take advantage of locality at the DRAM page level. However, one problem with the exploitation of locality at the DRAM page level is that the size of the DRAM page depends on the configuration of the DRAM device and the memory modules, rather than the architectural page size of the processor.

3.2.5 Column

In DRAM memory systems, a column of data is the smallest independently addressable unit of memory. Figure 3.8 illustrates that in memory systems such as SDRAM and DDRx[*] SDRAM memory systems with topology similar as to the memory system illustrated in Figure 3.5, the size of a column of data is the same as the width of the data bus. In a Direct RDRAM device, a column is defined as 16 bytes of data, and each read command accesses a single column of data 16 bytes in length from each physical channel of Direct RDRAM devices

In DDRx SDRAM memory systems, each column access command loads or stores multiple columns of data depending on the programmed burst length. For example, in a

[*]. DDRx denotes DDR SDRAM and evolutionary DDR memory systems such as DDR2 and DDR3 SDRAM memory systems, inclusively.

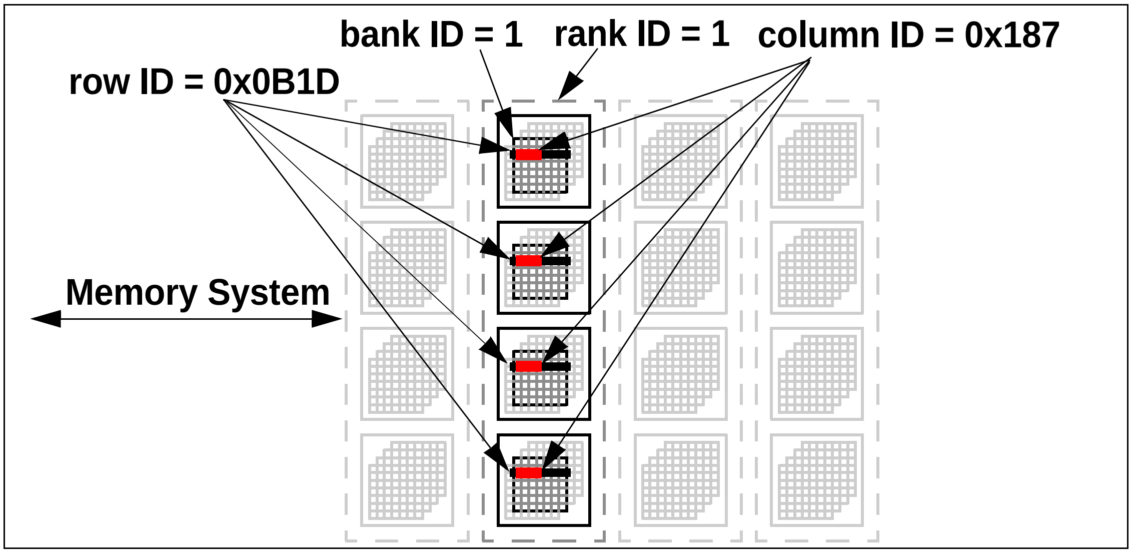


Figure 3.9: Location of data in a DRAM memory system.

DDR2 DRAM device, each memory read command returns a minimum of 4 columns of data. The distinction between a DDR2 device returning a minimum burst size of 4 columns of data and a Direct RDRAM device returning a single column of data over 8 beats* is that the DDR2 device accepts the address of a specific column, and returns the requested columns in different orders depending on the programmed behavior of the DRAM device. In this manner, each column is separately addressable. In contrast, Direct RDRAM devices do not reorder data within a given burst, and a 16 byte burst from a single channel of Direct RDRAM devices is transmitted in-order and treated as a single column of data.

3.2.6 Memory System Organization: An Example

Figure 3.9 illustrates a DRAM memory system with 4 ranks of memory, each rank of memory consists of 4 devices connected in parallel, each device has 4 banks of DRAM arrays internally, each bank has 8192 rows, and each row has 512 columns of data. In a

*. A **beat** is simply a data-transition on the data bus. In SDRAM memory systems, there is one data-transition per clock cycle, so one beat of data is transferred per clock cycle. In DDRx SDRAM memory systems, two data transfers can occur in each clock cycle, so two beats of data is transferred per clock cycle. The use of the beat terminology avoids overloading the word cycle in discussions regarding DDRx SDRAM memory systems.

DRAM based memory system, the DRAM controller accepts an address and breaks down the address into separate addresses that point to the specific channel, rank, bank, row, column where the data is contained.

Although Figure 3.9 illustrates a uniformly organized memory system, memory system organizations of many computer systems are typically non-uniform. The reason that the DRAM memory systems organizations in many computer systems are typically non-uniform is because most computer systems are designed to allow end users to upgrade the capacity of the memory system by inserting and removing commodity memory modules. To support memory capacity upgrades by the end user, DRAM controllers have to be designed to flexibly adopt to different configurations of DRAM devices and modules that the end user could place into the computer system. This support is provided for through the use of address range registers whose functionality is examined separately in the chapter on system controllers.

3.3 Memory Modules

Some earlier generations of computer systems allowed end users to increase memory capacity by providing sockets on the system board where additional DRAM devices can be inserted. Figure 3.10 illustrates a system board with sockets that allow end users to remove and insert individual DRAM devices, usually contained in dual link packages (DIP). The process of memory upgrade was cumbersome and difficult, as DRAM devices had to be individually removed and inserted into each socket. Pins on the DRAM devices may be bent and not visually detected as such. Defective DRAM chips were difficult to locate and

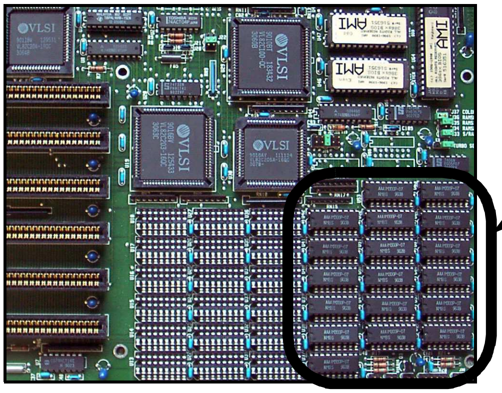

one rank of 18 DRAM devices, each providing 1 bit wide data bus.

Figure 3.10: An 80386sx system board with sockets for dual in-line package DRAM devices.

routing of sockets for a large memory system requires large surface areas on the system board. The solution to the problems associated with memory upgradability was the creation and use of memory modules.

Memory modules are essentially miniature system boards that hold a number of DRAM devices so that groups of DRAM devices can be quickly and easily inserted and removed from the system board. Memory modules provide an abstraction at the module interface so that different manufacturers can manufacture memory upgrades for a given computer system with different DRAM devices. DRAM memory modules also reduce the complexity of the memory upgrade process. Instead of the removal and insertion of individual DRAM chips, memory upgrades with modules containing multiple DRAM chips can be quickly and easily inserted into and removed from a module socket. Over the years, memory modules have themselves gradually evolved, obtained a level of sophistication, and now require exacting specifications for compatibility between different systems.

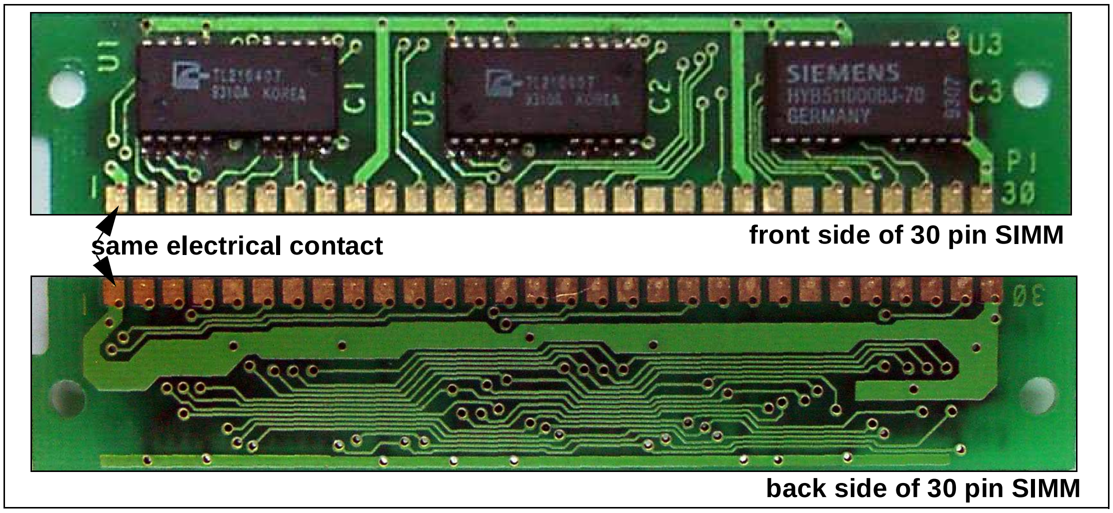


Figure 3.11: A one megabyte 30 pin SIMM.

3.3.1 Single In-line Memory Module (SIMM)

In the late 1980’s and early 1990’s, the computer industry first standardized on the use of 30 pin ***Single In-line Memory Modules*** (SIMMs), then later moved to the use of 72 pin SIMMs. SIMMs are referred to as single-inline due to the fact that the contacts on both sides of the module are electrically identical. A 30 pin SIMM provides interconnects to 8 or 9 pins on the data bus, as well as power, ground, address, command and chip select signal lines. A 72 pin SIMM increases the width of the data bus connection to 32 or 36 bits.

Figure 3.11 shows the two sides of a 30 pin SIMM. The DRAM devices on the SIMM in Figure 3.11 consists of two 4 megabit and one 1 megabit DRAM devices. Collectively, these DRAM devices provide a 9 bit wide data bus interface and 1 megabyte of parity protected memory storage capacity. Typical computer systems in the early 1990’s used sets of four matching 30 pin SIMMs similar to the one illustrated in Figure 3.11 to provide a 36 bit wide memory interface to support parity checking by the memory controller.

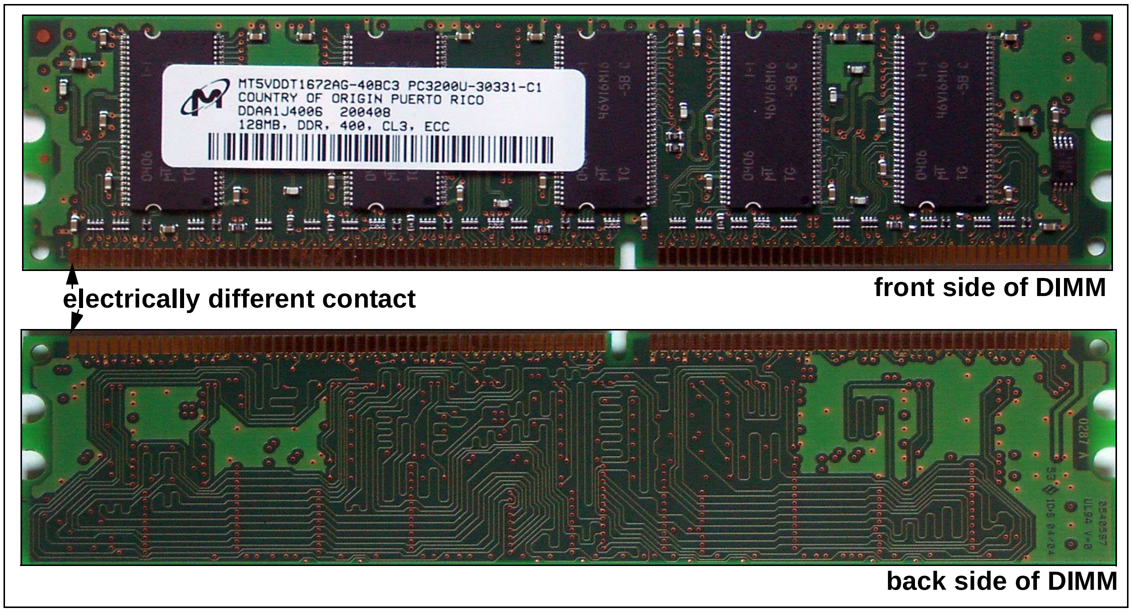

MT5VDDT1672AG-40BC3 PC3200U-30331-C1
COUNTRY OF ORIGIN PUERTO RICO
DDAA1J4006 200408
128MB, DDR, 400, CL3, ECC
electrically different contact
front side of DIMM
back side of DIMM

Figure 3.12: A 128 MB PC3200 ECC DDR SDRAM Dual In-line Memory Module.

3.3.2 Dual In-line Memory Module (DIMM)

In the late 1990's, as the personal computer industry transitioned from FPM/EDO DRAM to SDRAM, 72 pin SIMMs were phased out in favor of ***dual in-line memory modules*** (DIMMs). DIMMs are physically larger than SIMMs and provides a 64 or 72 bit wide data bus interface. The difference between a SIMM and a DIMM is that contacts on either side of the DIMM are electrically different. The electrically different contacts allow a denser routing of electrical signals as well as close pairing of power and ground signals for noise minimization.

Figure 3.12 illustrates a 128 MB PC3200 ECC DDR SDRAM DIMM with an interesting configuration. Since ECC support required that this DIMM provide a 72 bit wide data bus interface to the memory system, an odd number of DRAM devices is needed on the memory module to create such an interface. Typically, for a 128 MB ECC DIMM, four 256 megabit DRAM device with 16 bit wide interface would be utilized along with a single 128

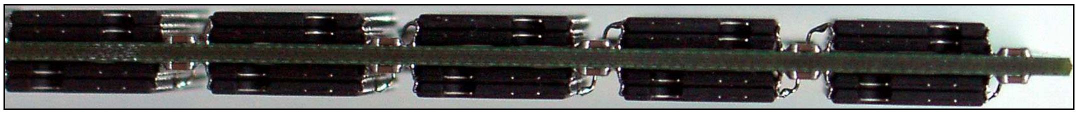


Figure 3.13: Edge view of a registered DIMM stacked with 2 ranks of DRAM devices in TSOP package.

megabit DRAM device with 8 bit wide interface. Such a configuration would minimize cost, but in this case, the manufacturer of the memory module chose to create the memory module with five identical 256 megabit DRAM devices, each with a 16 bit data bus interface. The result is an 80 bit data bus interface, but only 72 are used on the DIMM.

3.3.3 Registered Memory Module

Memory modules of varying capacity and timing characteristics are needed to suite a wide range of computers and upgrade options. High performance workstations and servers typically require large memory capacity. The problem associated with large memory capacity memory modules is that a large number of DRAM devices must be connected together to create the large capacity memory module. Figure 3.13 shows a memory module stacked with 2 ranks of 64 megabit SDRAM devices. On the memory module partially shown in edge view in Figure 3.13, each rank consists of 18 DRAM devices and a total of 36 DRAM devices are connected together on the memory module to form a memory module 256 megabytes in capacity. The large number of DRAM devices creates a problem in that the large number of DRAM devices presents a large loading factor on the various address, command and data busses.

Registered memory modules alleviate the issue of electrical loading by large numbers of DRAM devices in a large memory system through the use of registers that buffer the address

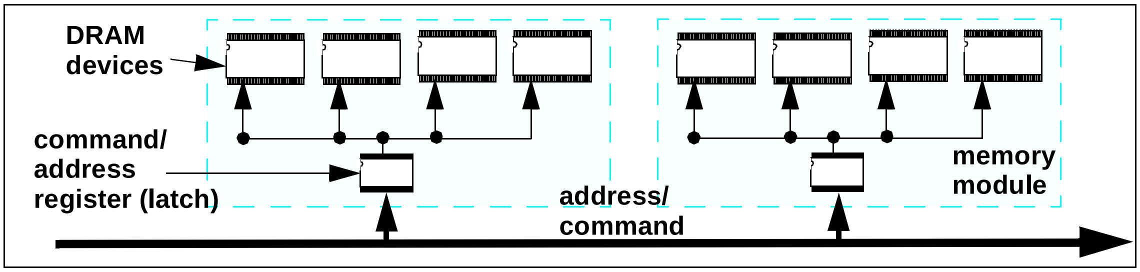


Figure 3.14: Registered latches buffer the address and command. Also introduces additional latency into the DRAM access.

and control signals at the interface of the memory module. Figure 3.14 illustrates that registered memory modules use registered latches at the interface of the memory module to buffer the address and control signals. In this manner, the registers greatly reduce the number of electrical loads that a memory controller must drive directly, and the signal interconnects in the memory system are divided into two separate segments: between the memory controller and the register, and between the register and DRAM devices. The segmentation allows timing characteristics of the memory system to be optimized by limiting the number of electrical loads as well as reducing the path lengths of the critical control signals in individual segments of the memory system. However, the drawback to the use of the registered latches on a memory module is that the buffering of the address and control signals introduce delays into the memory access latency, and the cost of ensuring of signal integrity in a large memory system is paid in terms of additional latency for all memory transactions.

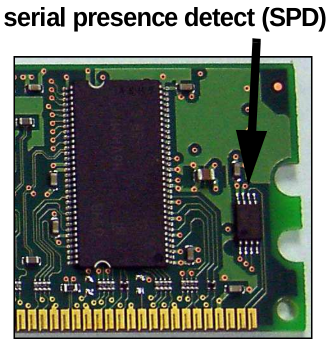
Capacity	device density	number of ranks	devices per rank	device width	number of banks	number of rows	number of columns
128 MB	64 Mbit	1	16	x4	4	4096	1024
128 MB	64 Mbit	2	8	x8	4	4096	512
128 MB	128 Mbit	1	8	x8	4	4096	1024
128 MB	256 Mbit	1	4	x16	4	8192	512

TABLE 3.1: **Four different configurations for a 128 MB SDRAM memory module**

3.3.4 Memory Module Organization

Modern DRAM memory systems often support large varieties of memory modules to give end users the flexibility of selecting and configuring to the desired memory capacity. Since the price of DRAM devices fluctuate depending on the unpredictable commodity market, one memory module organization may be less expensive to manufacture than another organization at a given instance in time, while the reverse may be true at a different instance in time. As a result, a memory system that supports different configurations of memory modules allows end users the flexibility to purchase and use the most economically organized memory module. However, one issue that memory system design engineers must account for in providing the flexibility of memory system configuration to the end user is that the flexibility translates into large combinations of memory modules that may be placed into the memory system at a given instance in time. Moreover, multiple organizations often exist for a given memory module capacity, and memory system design engineers must often account for not only different combinations of memory modules of different capacities, but also different modules of different organizations for a given capacity.

Table 3.1 shows that a 128 MB memory module can be constructed from a combination of sixteen 64 Mbit DRAM devices, eight 128 Mbit DRAM devices, or four 256 Mbit DRAM devices. Table 3.1 shows that the different memory module organizations not only



Configuration	Value (interpreted)
DRAM type	DDR SDRAM
No. of row addresses	16384
No. of column addresses	1024
No. of banks	4
Data rate	400
Module type	ECC
Cas latency	3

TABLE 3.2: **Sample parameter values stored in SPD**

Figure 3.15: The SPD stores memory module configuration information.

use different number of DRAM devices, but also presents different numbers of rows and columns to the memory controller. To access the memory on the memory module, the DRAM controller must recognize and support the organization of the memory module inserted by the end user into the memory system. In some cases, new generations of DRAM devices can enable memory module organizations that a memory controller was not designed to support, and incompatibility follows naturally.

3.3.5 Serial Presence Detect (SPD)

Memory modules have gradually evolved as each generation of new memory modules gains additional levels of sophistication and complexity. Table 3.1 showed that a DRAM memory module can be organized as multiple ranks of DRAM devices on the same memory module, each rank consisting of multiple DRAM devices and the memory module can have differing numbers of rows and columns. What isn't shown in table 3.1 is that each DRAM memory module may in fact have different timing parameters, and the variability of the DRAM modules in turn increases system level complexity that a memory system design engineer must deal with.

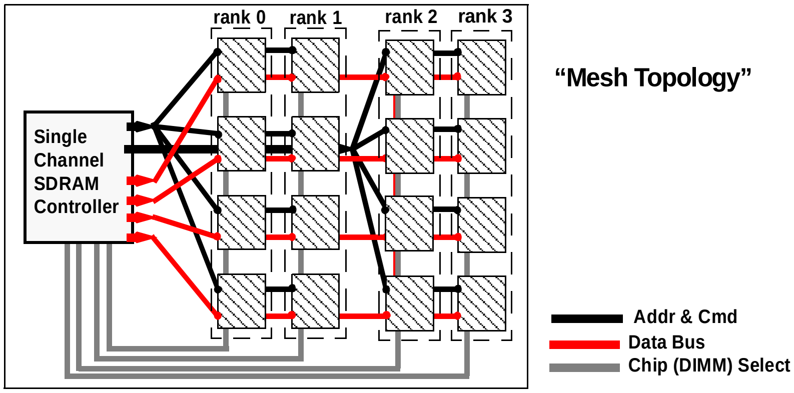


Figure 3.16: Topology of a generic DRAM memory system.

To reduce the complexity and eliminate confusion involved in the memory upgrading process, the solution adopted by the computer industry is to store the configuration information of the memory module on a flash memory device whose content can be retrieved by the memory controller as part of the system initialization process. In this manner, the memory controller can obtain the configuration and timing parameters required to optimally access data from DRAM devices on the memory module. Figure 3.15 shows the image of a small flash memory device on a DIMM. The small flash memory device is known as a ***Serial Presence Detect*** (SPD) device, and it stores parameter values that defines the configuration and timing characteristics of the memory module. Table 3.2 also shows some parameter values that are stored in the SPD.

3.4 Memory System Topology

In Figure 3.16, a memory system where 16 DRAM devices are connected to a single DRAM controller is shown. In Figure 3.16, the 16 DRAM devices are organized into four separate **ranks** of memory. Although all 16 DRAM devices are connected to the same

DRAM controller, different numbers of DRAM devices are connected to different networks for the uni-directional address and command bus, the bi-directional data bus, and the uni-directional chip select lines. In this topology, when a command is issued, electrical signals on the address and command busses are sent to all 16 DRAM devices in the memory system, but the separate chip-select signal selects a set of 4 DRAM devices in a single rank to provide the data for a read command or receive the data for a write command. In this topology, each DRAM device in a given rank of memory is also connected to a subset of the width of the data bus along with three other DRAM devices in different ranks of memory

Memory system topology determines the signal path lengths and electrical loading characteristics in the memory system. As a result, designers of modern high performance DRAM memory systems must pay close attention to the topology and organizations of the DRAM memory system. However, due to the evolutionary nature of the memory system, the classic memory system topology described above and shown in Figure 3.16 has remained essentially unchanged for Fast Page Mode DRAM (FPM), Synchronous DRAM (SDRAM) and Dual Data Rate SDRAM (DDR) memory systems. Furthermore, variants of the classical topology with fewer ranks are expected to be used for DDR2 and DDR3 memory systems.

3.4.1 Direct RDRAM System Topology

One memory system with a system topology dramatically different from the classical DRAM memory system topology is the Direct RDRAM memory system. In Figure 3.17, 4 Direct RDRAM devices are shown connected to a single Direct RDRAM memory controller. Figure 3.17 shows that in a Direct RDRAM memory system, the DRAM devices

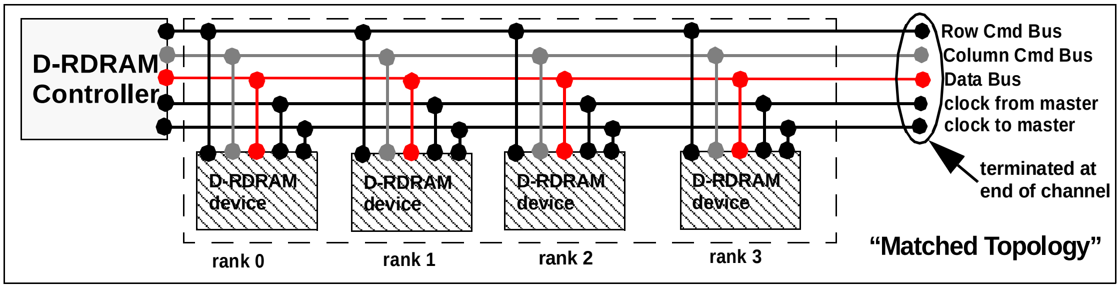


Figure 3.17: Topology of a generic Direct RDRAM memory system.

are connected to a well matched network of interconnects where the clocking network, the data bus and the command busses are all path length matched by design. The benefit of the well matched interconnection network is that signal skew is minimal by design and electrical signalling rates in the Direct RDRAM memory system can be increased to higher frequencies than a memory system with the classic memory system topology. Modern DRAM systems with conventional multi-rank topology can also match the raw signalling rates of a Direct RDRAM memory system. However, the drawback for these DRAM systems is that idle cycles must be designed into the access protocol and devoted to system level synchronization. As a result, even when pushed to comparable data rates, multi-rank DRAM memory systems with classical system topologies are less efficient in terms of data transported per cycle per pin.

The Direct RDRAM memory system can achieve higher efficiency in terms of data transport per cycle per pin. However, in order to take advantage of the system topology and enjoy the benefits of higher pin data rates and higher data transport efficiency, Direct RDRAM memory devices are more complex than comparable DRAM memory devices that use the classic memory system topology. In DRAM devices, complexity translates directly to increased costs. As a result, the higher data transport efficiency of Direct RDRAM memory systems has to be traded off against relatively higher DRAM device costs.

CHAPTER 4 *DRAM Memory Access Protocol*

The basic structures of DRAM devices and memory system organizations are described in some detail in previous chapters. In this chapter, the DRAM memory access protocol is examined in similar detail. A memory access protocol defines commands that a DRAM memory controller uses to manage the movement of data on DRAM devices in the memory system, and each memory system has a slightly different access protocol. The DRAM memory access protocol described in this chapter can be broadly applied to modern memory access protocols such as SDRAM, and DDRx SDRAM memory access protocols.

This chapter examines a generic DRAM memory access protocol by focusing on basic DRAM commands common to all commodity DRAM devices. Modern DRAM devices with additional logic circuitry, write buffers or cache require the use of additional commands to manage data flow and device operations, on a given DRAM device, and those commands are not covered in the examination of the generic DRAM access protocol.

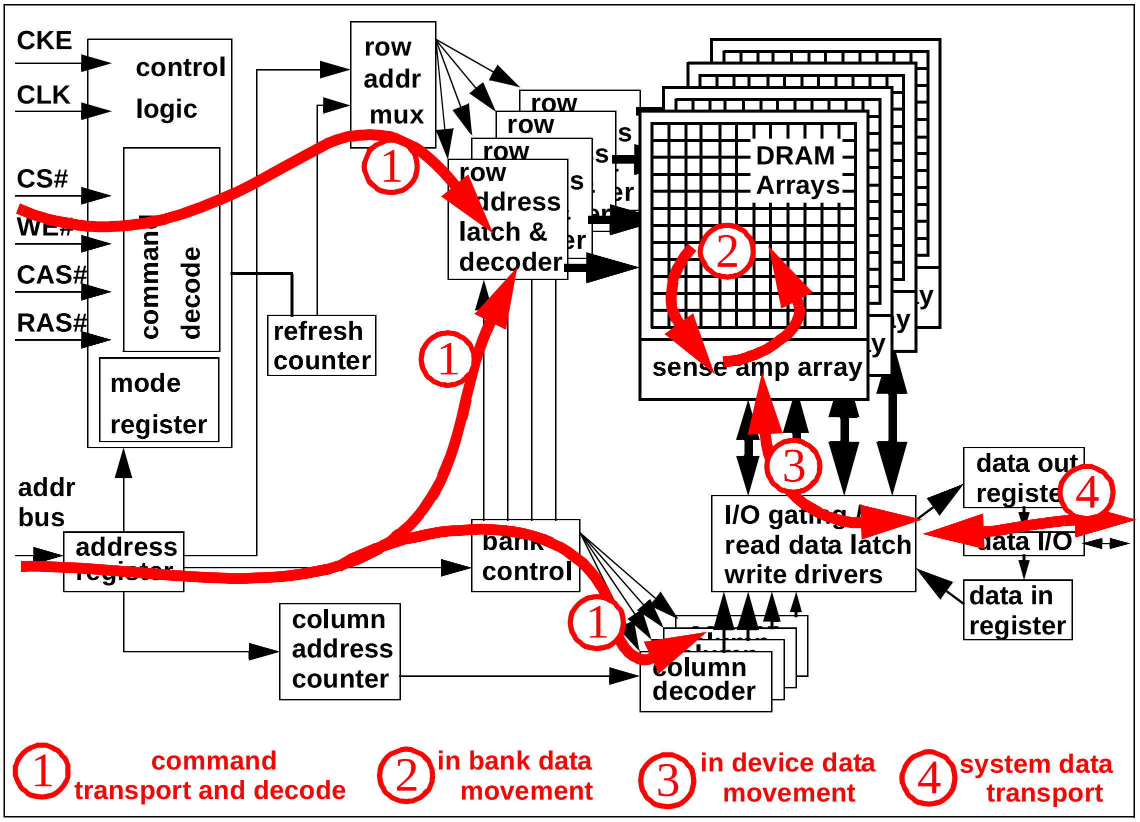


Figure 4.1: Command and data movement on generic SDRAM device.

4.1 Basic DRAM Commands:

A detailed examination of any DRAM memory access protocol is a difficult and complex task. The complexity of the task arises from the number of combinations of commands in modern DRAM memory systems. Fortunately, a basic memory access protocol can be modeled by accounting for a limited number of basic DRAM commands*. In this section, five basic DRAM commands are described. The descriptions of the basic

*. Modern DRAM devices such as Direct RDRAM and DDR2 SDRAM devices support larger sets of commands. However, most are used to manage the electrical characteristics of the DRAM devices, and only indirectly impacts access latency and sustainable bandwidth characteristics of a DRAM memory system at a given operating frequency.

commands form the foundation of the DRAM memory access protocol examined in this chapter. The interaction of the basic DRAM commands are then used to determine the latency response and sustainable bandwidth characteristics of DRAM memory systems in this text.

Throughout this chapter, the SDRAM device illustrated in Figure 4.1 is used as a generic DRAM device for the purposes of defining the basic memory access protocol. Figure 4.1 illustrates that different phases of operations occurs on the DRAM devices to facilitate the movement of data for each command. The generic DRAM access protocol described in this chapter is based on a resource usage model. That is, the generic DRAM access protocol assumes that two different commands can be fully pipelined on a given DRAM device as long as they do not require the use of a shared resource at the same time, and that there are no additional constraints beyond the immediate resource sharing constraint[*]. Figure 4.1 illustrates four overlapped phases of operation for an abstract DRAM command. In phase one, the command is transported through the address and command busses and decoded by the DRAM device. In phase two, data is moved within a bank, either from the cells to the sense amplifiers or from the sense amplifiers back into the DRAM arrays. In phase three, the data is moved through the shared I/O gating, read latches and write drivers. In phase four, read data is placed onto the data bus by the DRAM device or the memory controller. Since the data bus may be connected to multiple ranks of memory, no two commands to different ranks of memory can use the shared data bus at the same instance in time.

[*]. Additional constraint on the scheduling of DRAM commands may be the presence of timing parameters such as t_{RRD} and t_{FAW}. These timing parameters are used to limit the maximum current draw of DRAM devices, so while the resource on the DRAM device is not used, these timing parameters specify that the resource cannot be used until sometime later to limit peak power consumption characteristics.

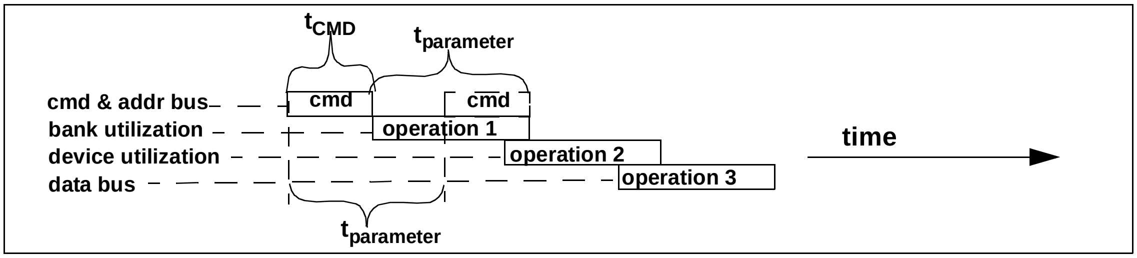


Figure 4.2: Different phase of an abstract DRAM commands in a generic DRAM device.

A DRAM access protocol indirectly defines the minimum timing constraints between consecutive DRAM commands. In this chapter, the description of the DRAM memory access protocol begins with the examination of individual DRAM commands and progresses with the examination of combinations of DRAM commands. Power limitation constraints are then described in some detail. The chapter concludes by correlating the generic DRAM memory access protocol to a specific DRAM memory access protocol, the DDR2 SDRAM memory access protocol.

4.1.1 Generic DRAM Command Format

Figure 4.2 abstractly illustrates the progression of a generic DRAM command. In Figure 4.2, the time period that it takes to transport the command from the DRAM controller to the DRAM device is illustrated and labelled as t_{CMD}. Figure 4.2 also illustrates $t_{parameter}$, a generic timing parameter that measures the duration of “operation 1”. In this text, the timing of operations is measured from the end of the command transport stage until the end of the operation itself[*]. In cases where the duration of an operation limits the timing of command issuance, $t_{parameter}$ then defines the minimum time that commands may be placed onto the

[*] CAS commands excepted. t_{CAS} denotes the beginning of the CAS command to the beginning of the data transport phase. It is defined in this manner due to historical usage of the t_{CAS} timing parameter. Also, the definition of the t_{CAS} command enables t_{CWD} to be defined as zero rather than as a negative value in memory systems without a write delay.

command and address bus. As a result, $t_{parameter}$ also denotes the minimum time that must pass between the start of two commands whose relative timing is limited by the duration of an operation measured by $t_{parameter}$.

DRAM commands are abstractly defined in this text, and the abstraction separates the actions of each command from the timing specific nature of each action in specific DRAM access protocols. That is, the abstraction enables the same set of DRAM command interactions to be applied to different DRAM memory systems with different timing parameter values. For example, the command transport time requires 1 clock cycle on SDRAM and DDRx SDRAM memory systems and 4 clock cycles in Direct RDRAM memory systems. By abstracting out protocol specifics timing characteristics, DRAM commands can be described in abstract terms. The generic DRAM memory access protocol in turn enables abstract performance analysis of DRAM memory systems, and the results of the abstract analysis is then equally applicable to many different memory systems.

Parameter	Description	Illustration
t_{Burst}	Data **Burst** duration. The time period that data burst occupies on the data bus. Typically 4 or 8 beats of data. In DDR SDRAM, 4 beats of data occupies 2 cycles	Figure 4.4
t_{CAS}	**C**olumn **A**ccess **S**trobe latency. Time interval between column access command and data return by DRAM device(s). Also known as t_{CL}.	Figure 4.4
t_{CMD}	**C**o**m**man**d** transport duration. Time period that a command occupies on the command bus as it is transported from the DRAM controller to the DRAM devices	Figure 4.3
t_{CWD}	**C**olumn **W**rite **D**elay. Time interval between issuance of column write command and placement of data on data bus by the DRAM controller.	Figure 4.5
t_{DQS}	**D**ata **S**trobe turnaround. Used in DDR and DDR2 SDRAM memory systems. Not used in SDRAM or Direct RDRAM memory systems. 1 full cycle in DDR SDRAM	Figure 4.17
t_{FAW}	**F**our (row) bank **A**ctivation **W**indow. A rolling time frame in which a maximum of four bank activation may be engaged. Limits peak current profile.	Figure 4.29
t_{RAS}	**R**ow **A**ccess **S**trobe. Time interval between row access command and data restoration in DRAM array. After t_{RAS}, DRAM bank could be precharged.	Figure 4.3
t_{RC}	**R**ow **C**ycle. Time interval between accesses to different rows in a given bank $t_{RC} = t_{RAS} + t_{RP}$	Figure 4.6
t_{RCD}	**R**ow to **C**olumn command **D**elay. Time interval between row access command and data ready at sense amplifiers.	Figure 4.3
t_{RFC}	**R**e**f**resh **C**ycle Time. Time interval between Refresh and Activation command	Figure 4.7
t_{RRD}	**R**ow activation to **R**ow activation **D**elay. Minimum time interval between two row activation commands to same DRAM device. Limits peak current profile.	Figure 4.29
t_{RP}	**R**ow **P**recharge. Time interval that it takes for a DRAM array to be precharged and readied for another row access.	Figure 4.6
t_{WR}	**W**rite **R**ecovery time. Minimum time interval between end of write data burst and the start of a precharge command. Allows sense amplifiers to restore data to cells	Figure 4.5

TABLE 4.1: Summary of timing parameters used in generic DRAM access protocol

4.1.2 Summary of Timing Parameters

The examination of the DRAM access protocol begins by careful definition of the timing parameters. Table 4.1 summarizes the timing parameters used in the description of the DRAM memory access protocol in this chapter. The timing parameters summarized in table 4.1 is far from a complete set of timing parameters used in the description of a modern memory access protocol. Nevertheless, the timing parameters describe here is a minimum set of timing parameters whose use is sufficient to characterize and illustrate important interactions in modern DRAM memory systems.

4.1.3 Row Access Command

Figure 4.3 abstractly illustrates the progression of a generic row access command, also known as a row activation command. The purpose of a row access command is to move data from the DRAM arrays to the sense amplifiers. Two timing parameters are associated with a row access command: t_{RCD} and t_{RAS}. The time it takes for the row access command to move data from the DRAM cell arrays to the sense amplifiers is known as the **R**ow **C**olumn (Command) **D**elay, t_{RCD}. After t_{RCD}, an entire row of data is held in the sense amplifiers. At that time, a column read or write access commands can be engaged to move data between the sense amplifiers and the memory controller through the data bus.

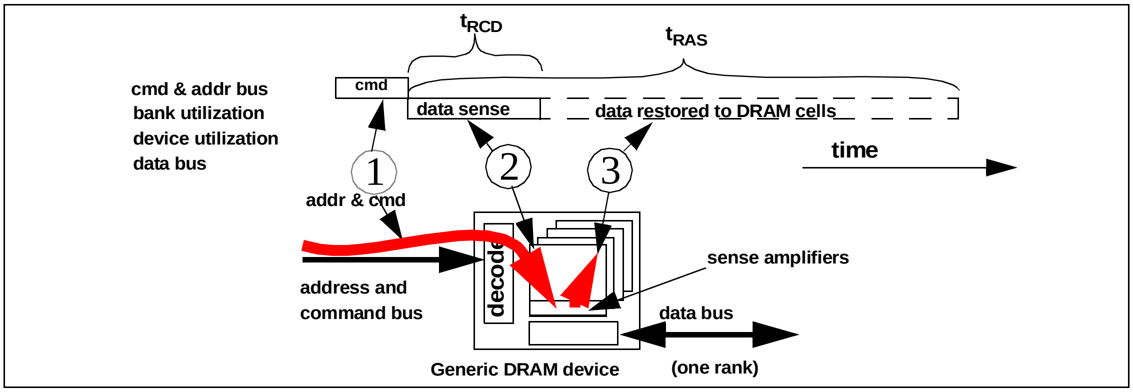


Figure 4.3: Row Access command and timing.

After t_{RCD} time, data is available at the sense amplifiers, but not yet restored to the DRAM cells. A row access command discharges the DRAM cells of the accessed row. As a result, the row of data must be restored from the sense amplifiers back into the DRAM cells before the sense amplifiers can be used to sense the data in a different row. The time it takes for a row access command to discharge and restore data from the row of DRAM cells is known as the Row Access Strobe latency or t_{RAS}. After t_{RAS}, the sense amplifiers are assumed to have completed the data restoration process, and the DRAM array can be precharged for another row access to the same bank.

4.1.4 Column Read Command

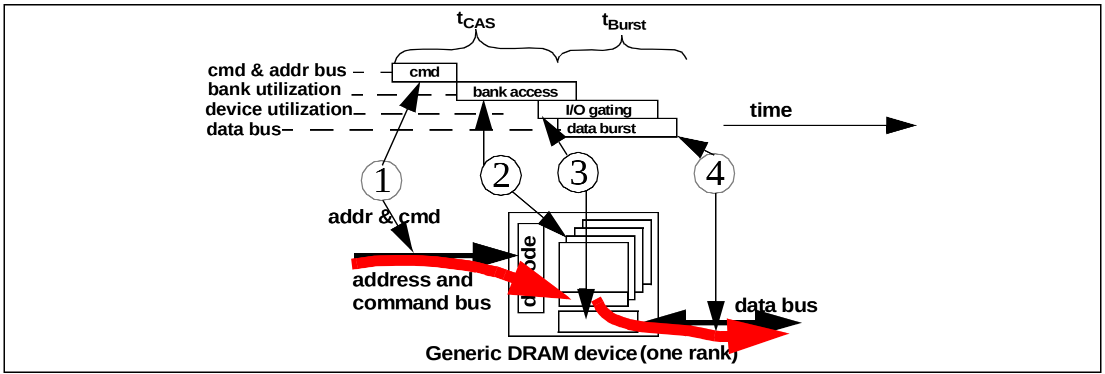


Figure 4.4: Column Read command and timing.

Figure 4.4 abstractly illustrates the progression of a column read command. A column read command moves data from the array of sense amplifiers of a given bank to the memory controller. There are two timing parameters associated with a column read command: t_{CAS} and t_{Burst}. The time it takes for the DRAM device to place the requested data onto the data bus after issuance of the column read command is known as the Column Access Strobe Latency (t_{CAS}, or t_{CL}). After t_{CAS}, the requested data is moved from the sense amplifiers onto the data bus, then into the memory controller. Modern memory systems move data in relatively short bursts, typically occupying 2, 4 or 8 beats on the data bus. To maintain consistency in the description of the access protocol, the duration of the data burst is described in terms of a time duration rather than the number of clock cycles. The data burst duration is labelled in Figure 4.4 as t_{Burst}.

Figure 4.4 shows that the column read command goes through 4 separate overlapping phases. In phase one, the command is transported on the address and command bus then decoded by the DRAM device. In phase two, the appropriate columns of data is retrieved from the sense amplifier array of the selected bank and moved to the I/O gating. In phase three, the data flows through the I/O gating and out to the data bus. In phase four, the data occupies the data bus for time duration of t_{Burst}.

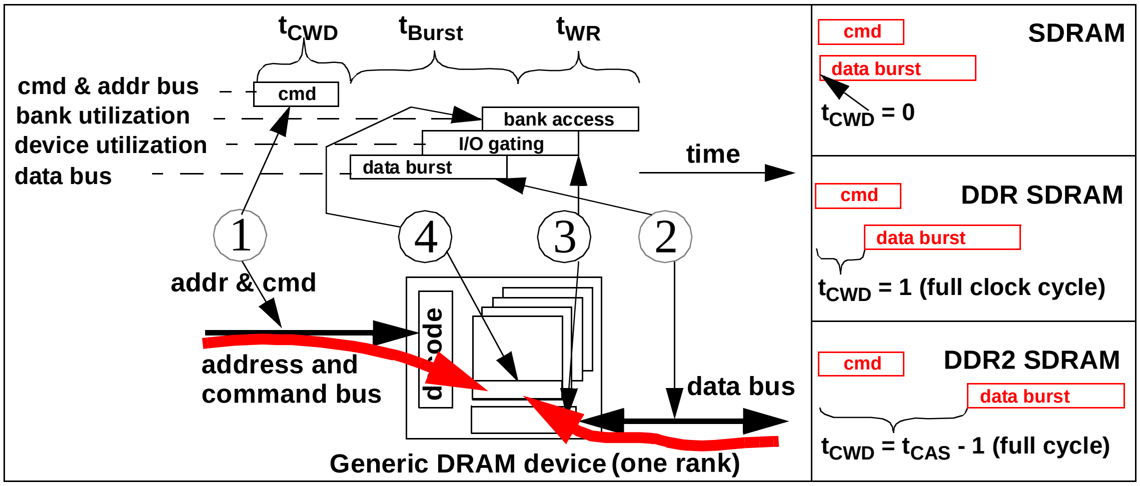


Figure 4.5: Column Write command and timing for SDRAM, DDR SDRAM and DDR2 SDRAM

4.1.5 Column Write Command

Figure 4.5 abstractly illustrates the progression of a column write command. A column write command moves data from the memory controller to the sense amplifiers of a given bank. The column write command goes through a similar set of overlapped phases as the column read command. However, due to the fact that the direction of the data movement differs between a read command and a write command, the ordering of the phases is reversed. In Figure 4.5, phase one shows that the column address and column write command is placed on the address and command bus. In phase two, the data is placed on the data bus by the memory controller. Then in phase three, the data flows through the I/O gating, and in phase four, the data reaches the sense amplifiers in the appropriate bank. One timing parameter associated with a column write command is t_{CWD}, command write delay. Column write delay is the delay between the time when the column write command is issued and the write data moved onto the data bus by the memory controller. Different memory access protocols have different settings for t_{CWD}. Figure 4.5 shows that in SDRAM and earlier DRAM devices, data for the write command is placed onto the data bus at the same

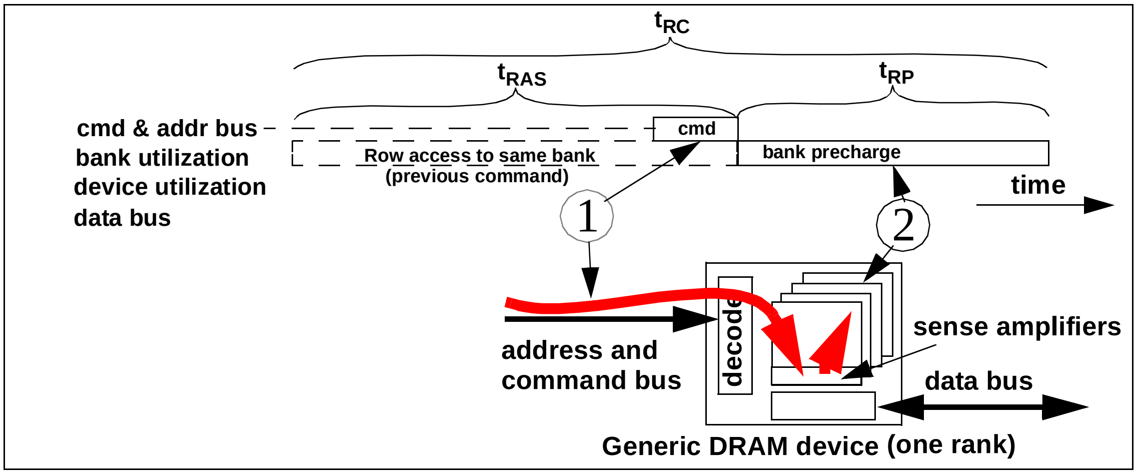


Figure 4.6: Row precharge command and timing.

time as the issuance of the column write command. In DDR SDRAM, write data is delayed one full clock cycle, and in DDR2, the write delay is one cycle less than t_{CAS}. Figure 4.5 also illustrates t_{WR}, the write recovery time. The write recovery time denotes the time between the end of the data burst and the completion of the movement of data into the DRAM arrays.

4.1.6 Precharge Command

Accessing data on a DRAM device data access is a two step process. A row access command moves data from the DRAM cells to the array of sense amplifiers. The data remains in the array of sense amplifiers for one or more column access commands to move data to and from the DRAM devices to the DRAM controller. In this context, a precharge command completes the sequence as it resets the array of sense amplifiers and the bitlines and prepares the sense amplifiers for another row access command. Figure 4.6 illustrates the progression of a precharge command. Figure 4.6 shows that in the first phase, the precharge command is sent to the DRAM device, and in phase two, the selected bank is precharged.

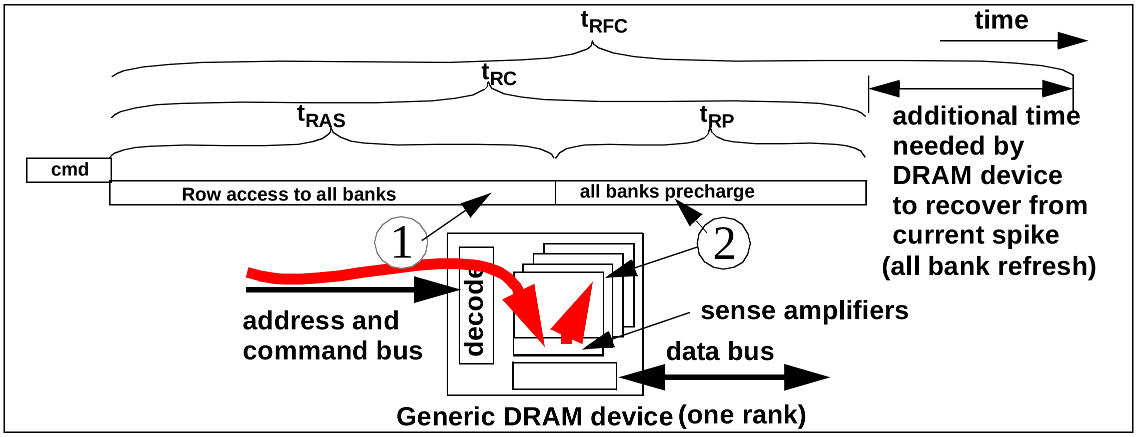


Figure 4.7: Row refresh timing.

The timing parameter associated with the (row) precharge command is t_{RP}. The two row-access related timing parameters, t_{RP} and t_{RAS}, can be combined to form t_{RC}, the row cycle time. The row cycle time of a given DRAM device denotes the speed at which the DRAM device can bring data from the DRAM cell arrays into the sense amplifiers, restore the data to the DRAM cells, then precharge the bitlines to the reference voltage level and made ready for another row access command. The row cycle time is the fundamental limitation to the speed at which data may be retrieved from different rows within the same DRAM bank.

4.1.7 Refresh Command

The word DRAM is an acronym that stands for Dynamic Random Access Memory. The reason that the memory is referred as “dynamic” is that the electrical charge retained by the storage capacitor gradually leaks out with the passage of time, and data stored in DRAM cells must be occasionally read out and restored to full value. A DRAM refresh command accomplishes the task of data readout and restoration to full value into the DRAM cells. As long as the time interval between refresh commands is shorter than the worst case time period in which data in storage cells deteriorate to indistinguishable values, DRAM refresh commands can be used to overcome leaky DRAM cells and maintain functionality of the DRAM storage system. The drawback to the refresh mechanism is that refresh commands consume bank bandwidth and power. As a result, there are a number of different refresh mechanisms used by different systems, some are designed to minimize controller complexity while others are designed to minimize bandwidth impact.

Figure 4.7 illustrates a basic refresh command that allows the DRAM controller to send a single refresh command to refresh one row in all banks. When a basic refresh command is issued, the DRAM device takes a row address from an internal register, then sends the same row address to all banks to be refreshed concurrently. As illustrated in Figure 4.7, the single refresh command to all banks take one refresh cycle time to complete. Figure 4.7 also illustrates that the refresh cycle time, t_{RFC}, is at least equal to the row cycle time t_{RC}, and in many cases, much longer than t_{RC}.

The refresh command illustrates one weakness of the resource usage model in that according to the strict interpretation of the resource usage model, a DRAM controller should be able to issue a refresh command to a DRAM device every row cycle time. However, Figure 4.7 shows that the DRAM device can issue the basic refresh command only once every refresh cycle time, and that refresh cycle time is longer than the row cycle time. The reason that the resource usage model fails in this case is because the basic bank-concurrent refresh cycle is power-limited, and the DRAM device needs more time for the current spike induced by the concurrent refresh of all banks in a given DRAM device to settle before another refresh or row activation command can be engaged.

Density	Bank Count	Row Count	Row Size (bits)	Row cycle time (ns)	Refresh Cycle time (ns)
256 Mbit (x8)	4	8192	8192	55	75
512 Mbit (x8)	4	16384	8192	55	105
1 Gbit (x8)	8	16384	8192	55	127.5

TABLE 4.2: Refresh cycle times of DDR2 SDRAM devices

In modern DRAM memory systems, depending on the refresh requirement of the DRAM devices, the memory controller typically injects one row refresh command once every 32 or 64 milliseconds for each row in a bank. That is, in a DRAM device with 8192 rows per bank and 64 ms refresh cycle requirement, 8192 refresh commands are issued every 64 ms to a DRAM device to refresh one row in all banks concurrently. Depending on the design of the memory controller, the 8192 refresh commands may be issued consecutively or evenly distributed throughout the 64 ms time period.

Table 4.2 shows the general trend of refresh cycle times in DDR2 SDRAM devices. The general trend illustrated in table 4.2 shows that with increasing DRAM device density, and in combination with the desire to retain the per bit charge capacitance of DRAM storage cells, it takes an increasing amount of current draw to refresh one row in all banks concurrently, and the refresh cycle time increases in each successive generation of DRAM devices. Fortunately, DRAM device design engineers and DRAM memory system design engineers are actively exploring alternatives to the bank-concurrent refresh command. Some advanced memory systems are designed in such a manner that the controller manually injects row cycle reads to individual banks. The per-bank refresh scheme decreases the bandwidth impact of refresh commands at the cost of increased complexity in the memory controller.

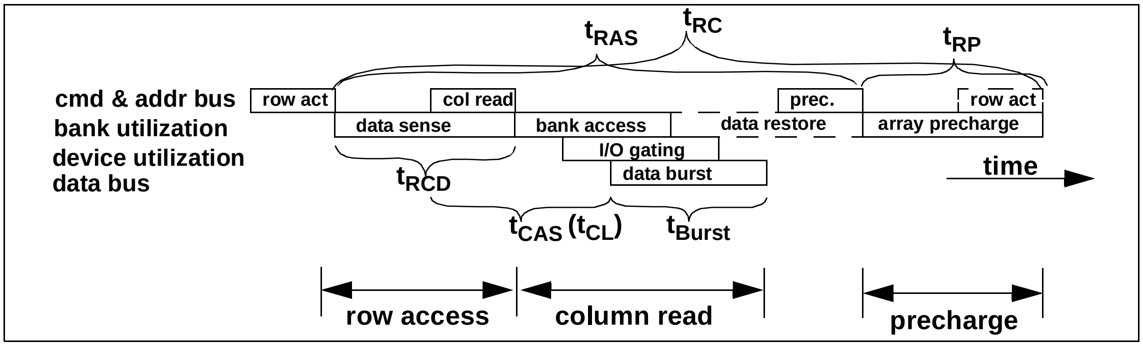


Figure 4.8: One read cycle in a “close-page” memory system.

4.1.8 A Read Cycle

Figure 4.8 illustrates a read cycle in a generic DRAM memory system. In modern DRAM devices, each row access command brings thousands of bits of data in parallel to the array of sense amplifiers in a given bank. A subsequent column read command then brings tens or hundreds of those bits of data through the data bus into the memory controller. For applications that are likely to stream through memory, keeping thousands of bits of a given row of data active at the sense amplifiers means that subsequent memory reads from the same row do not have to incur the latency or energy cost of another row access. In contrast, applications that are not likely to access data in adjacent locations favor memory systems that immediately precharges the DRAM array and prepares the DRAM bank for another access to a different row. In Figure 4.8, a sequence of commands in an abstract memory system designed for applications that do not benefit from keeping rows of data in the sense amplifiers for subsequent accesses is illustrated. In Figure 4.8, data is brought in from the DRAM cells to the sense amplifiers by the row access command. After t_{RCD}, data from the requested row has been resolved by the sense amplifiers, and a subsequent column read command can then be issued by the memory controller. After t_{CAS}, the DRAM chip begins

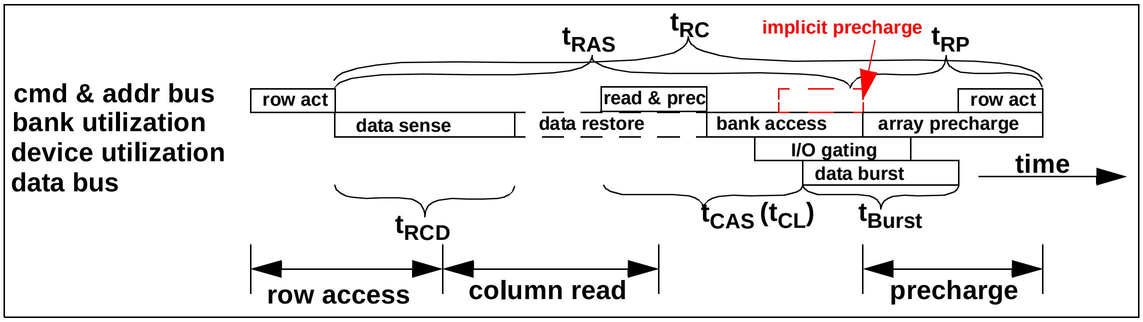


Figure 4.9: One "read cycle" with single column read and precharge command.

to return data on the data bus. Concurrent with the issuance of the column read command, the memory device actively restores data from the sense amplifiers to the DRAM cells, and after t_{RAS} from the initial issuance of the row access command, the DRAM cells would be ready for another row access. Collectively, memory systems that immediately precharges a bank to prepare it for another access to a different row are known as ***close-page*** memory systems. Memory systems that keep rows active at the sense-amplifiers are known as ***open-page*** memory systems.

4.1.9 Complex Commands

In the previous section, Figure 4.8 illustrated a read cycle in a generic DRAM memory system by issuing three separate commands. As part of the evolution of DRAM devices and architecture, some DRAM devices support commands that perform more complex series of actions. Figure 4.9 shows the same sequence of DRAM commands that cycles through a row and issues a single column read command as presented in Figure 4.8. However, the simple column read command in Figure 4.8 was replaced with a compound ***column read and precharge*** command. As the name implies, the column read and precharge command combines a column read command and a precharge command into a single command. The advantage of a column read and precharge command is that for memory systems that

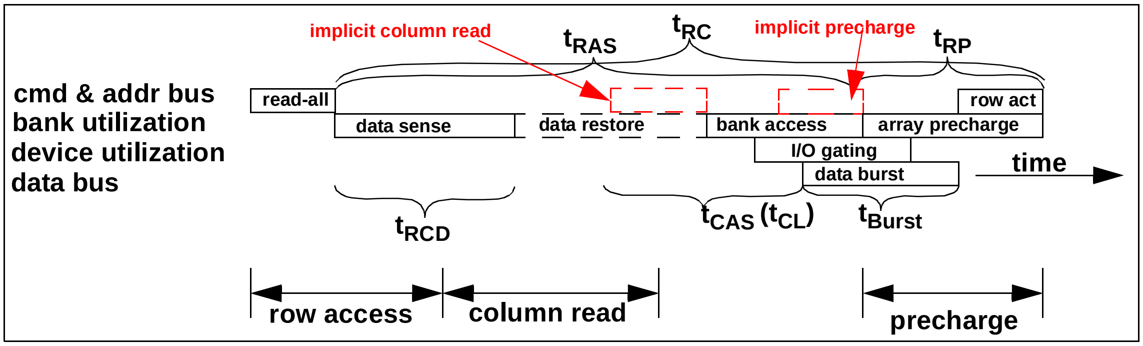


Figure 4.10: One "read cycle" with single do-it-all read command.

precharge the DRAM bank immediately after a read command, the complex command frees the DRAM controller from having to keep track of the timing of the precharge command. The DRAM controller in a system that utilizes the compound column read and precharge command also gains from the fact that the controller can now place a different command on the address and command bus that the separate precharge command would have otherwise occupied. The complex command thus reduces the address and command bus bandwidth requirement for a read cycle. Although the column read and precharge command illustrated in Figure 4.9 delays the timing of the column read command to satisfy t_{RAS}, the flexibility and reduced address and command bus bandwidth can result in a net win for certain bandwidth bound close-page DRAM memory systems despite the small increase in additional hardware on the DRAM device.

Some DRAM devices support even more complex compound commands, such as a command that performs all of the actions normally required in a DRAM read cycle. Figure 4.10 illustrates a read cycle in a specialized DRAM device that performs all of the actions of a read cycle with a single read command. The specialized single read command further simplifies controller design and allows the DRAM device to operate as an SRAM-like

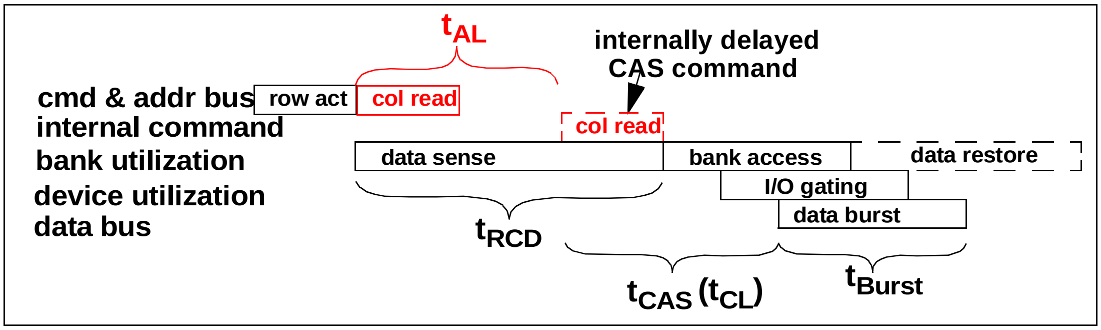


Figure 4.11: Delayed column read command with posted CAS.

device. Memory systems that support the do-it-all single read commands are typically found in high performance embedded systems.

One complex command supported by the DDR2 SDRAM memory system is the posted-CAS command. The posted-CAS command is simply a delayed column access command. Figure 4.11 abstractly illustrates a posted-CAS command. The posted CAS command is simply an ordinary column access (read or write) command that can be issued to the DRAM device before t_{RCD} for the row activation command has been satisfied. The DRAM device internally delays the action of the CAS command. The number of delay cycles for the posted-CAS command is pre-programmed into the DRAM device. The advantage of the posted-CAS command is that it allows a DRAM memory controller to issue the column access command immediately after the row access command.

Aside from the complex read and delayed read commands, some DRAM devices also support additional complex commands that are needed to manage specialized hardware structures such as write buffers in ESDRAM and Direct RDRAM devices or channel buffers in Virtual Channel DRAM devices.

4.2 DRAM Command Interactions

In the previous section, basic DRAM commands were described in some detail. In this section, the interactions between these previously described basic DRAM commands are examined in similar detail. In this text, a resource usage model is used to model DRAM command interactions. In the resource usage model, DRAM commands can be scheduled consecutively subject to availability of shared on-chip resources such as sense amplifiers, I/O gating buffers, and the availability of off-chip resources such as the command, address and data busses. However, even with the availability of shared resources, secondary considerations such as power limitation can prohibit commands from being scheduled consecutively[*].

This section examines read and write commands in a memory system with simplistic open-page and close-page row buffer management policies. In a memory system that implements the open-page row buffer management policy, once a row is opened for access, the array of sense amplifiers continues to hold an entire row of data for subsequent column read and write accesses to the same row. Open-page memory systems rely on workloads that access memory with some degree of spatial locality so that multiple column accesses can be performed to the same row and minimizes the number of DRAM row cycles. In an open-page memory system, DRAM command sequence for a given request depends on the state of the memory system, and the dynamic nature of the command sequences in open-page memory systems means there are larger numbers of possible DRAM command interactions and memory system state combinations in an open-page memory system than the number of DRAM command interactions in a close-page memory system. The larger number of

[*] i.e. t_{RRD} and t_{FAW}, examined separately.

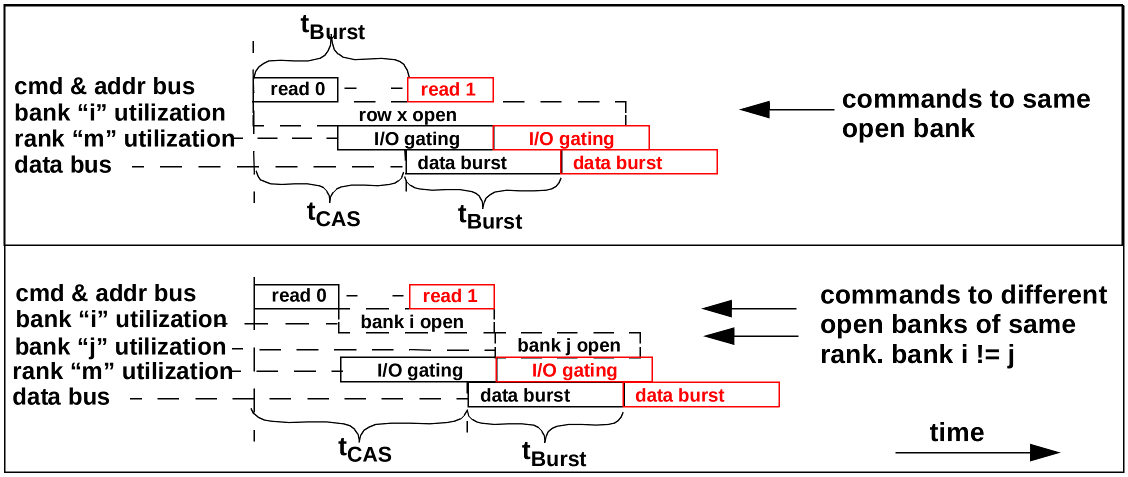


Figure 4.12: Consecutive column read commands to same bank, rank and channel.

command interactions and the dynamic nature of DRAM command sequences result in more complex command interactions and a higher degree of difficulty in scheduling command sequences in open-page memory systems. In the following sections, the large number of possible DRAM command interactions for open-page memory systems are examined in detail. The detailed examination of DRAM command combinations enables the creation of a table that summarizes the minimum scheduling distances between DRAM commands. The summary of minimum scheduling distances in turn enables performance analysis of DRAM memory systems in this text.

4.2.1 Consecutive Reads to Same Rank

In modern DRAM memory systems such as SDRAM, DDR SDRAM and Direct RDRAM memory systems, read commands to the same open row of memory in the same bank, rank and channel can be pipelined and scheduled consecutively subject to the availability of the data bus. Figure 4.12 shows two read commands, labelled as read 0 and read 1, pipelined consecutively. As illustrated in Figure 4.4, t_{CAS} after a read command is placed on to the command and address bus, the DRAM device begins to return data on the data bus. Since column read commands to the same open bank of the same rank can be pipelined consecutively, and the limitation on the scheduling of these commands is the duration of the data burst on the data bus, it follows that consecutive DRAM read commands to the same row of the same bank of memory can be scheduled every t_{Burst} time period.

Modern DRAM devices contain multiple banks inside a single rank of memory. In modern memory systems such as SDRAM, Direct RDRAM and DDR SDRAM, read commands to open rows in different banks within the same rank of memory can also be pipelined consecutively. Similar to consecutive column read commands to the same bank of the same rank of memory, DRAM column read commands can be scheduled to different open banks within the same rank of memory once every t_{Burst} time period. Figure 4.12 also shows the scheduling of column read consecutive accesses to different open banks within the same rank.

4.2.2 Consecutive Reads to Different Rows of Same Bank

Modern DRAM devices are designed to hold an entire row of data in the array of sense amplifiers for multiple column read or write accesses until a precharge command is issued independently or as part of a column read and precharge command. As a result, consecutive read commands to the same open bank can be issued and pipeline consecutively. However, read commands to different rows within the same bank would incur the cost of an entire row cycle time as the current DRAM array must be precharged and a different row activated by the array of sense amplifiers.

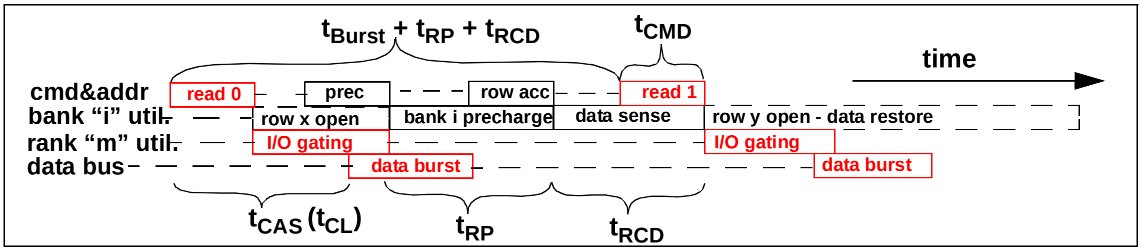


Figure 4.13: Consecutive column read commands to different rows of same bank: best case scenario.

Best Case Scenario:

Figure 4.13 illustrates the timing and command sequence of two consecutive read requests to different rows within the same bank of memory array. In this sequence, the first read command, labelled as read 0 is issued, the array of sense amplifiers must be precharged before a different row to the same bank can be accessed. After a time period t_{RP} from the issuance of a precharge command, a different row access command can then be issued, and time period t_{RCD} after the row access command, the second read command labelled as read 1 can then proceed. Figure 4.13 illustrates that consecutive column read accesses to different rows within the same bank could at best be scheduled with minimum timing of t_{Burst} + t_{RP} + t_{RCD}[*].

Worst Case Scenario:

Figure 4.13 illustrates the best case timing of two consecutive read commands to different rows of the same bank. However, in the case that data from the current row had not yet been restored to the DRAM cells, a precharge command cannot be issued until t_{RAS} time period after the previous row access command to the same bank. In contrast to the best case

[*]. The best case timing of a read command to a different row also depends on the internal prefetch data path of the DRAM device. In a DDR2 SDRAM device, a burst of 8 is fetched with two separate fetches of burst of 4. In that case, the precharge command cannot begin until at best $t_{Burst}/2$ + t_{RP} + t_{RCD} time period has passed from the previous column read command.

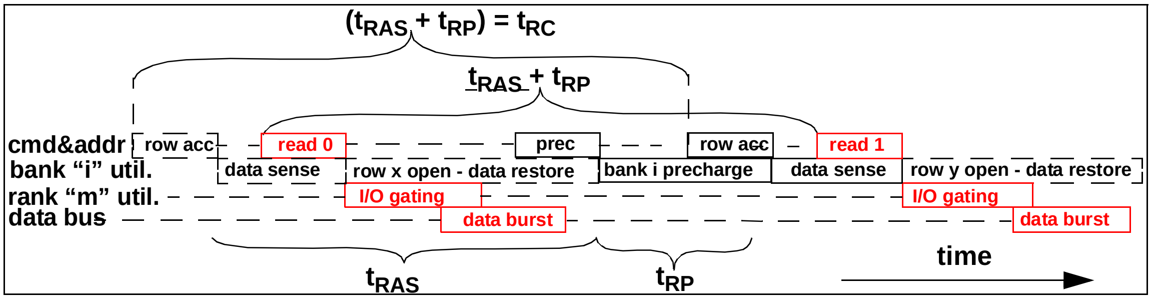


Figure 4.14: Consecutive column read commands to different rows of same bank: worst case scenario.

timing shown in Figure 4.13, Figure 4.14 shows the worst case timing for two consecutive read commands to different rows of the same bank where the first column command was issued immediately after a row access command. In this case, the precharge command cannot be issued immediately after the first column read command, but must wait until t_{RAS} time period after the previous row access command has elapsed. Then, t_{RP} time period after the precharge command, the second row access command could be issued, and t_{RCD} time period after that row access command, the second column read command completes this sequence of commands.

Figure 4.13 illustrates the best case timing of two consecutive read commands to different rows of the same bank and Figure 4.14 illustrates the worst case timing between two column read command to different rows of the same bank. The difference between the two different scenarios means that a DRAM memory controller must keep track of the timing of a row access command and delay any row precharge command until the row restoration requirement has been satisfied.

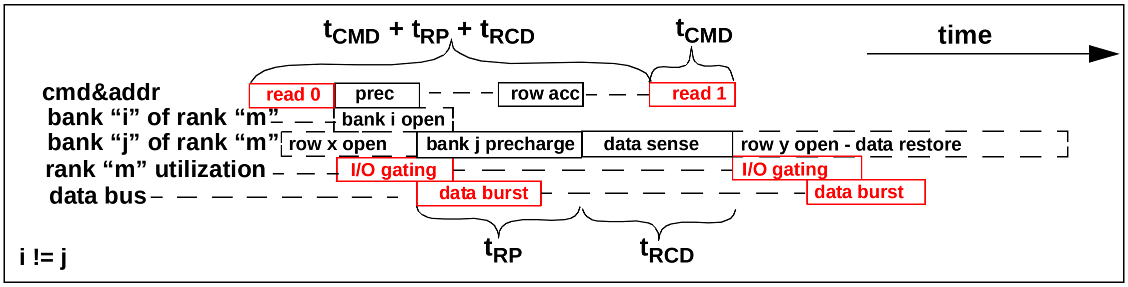


Figure 4.15: Consecutive DRAM read commands to different banks, bank conflict, no command re-ordering.

4.2.3 Consecutive Reads to Different Banks: Bank Conflict

The case of consecutive read commands to different rows of the same bank has been examined in the previous section. This section examines the case of consecutive read requests to different banks with the second request hitting a bank conflict against an active row in that bank. The consecutive read request scenario with the second read request hitting a bank conflict to a different bank has several different combinations of possible minimum scheduling distances that depend on the state of the bank as well as the capability of the DRAM controller to re-order commands between different transaction requests.

Without Command Re-Ordering

Figure 4.15 illustrates the timing and command sequence of two consecutive read requests to different banks of the same rank, and the second read request is made to a row that is different than the active row in the array of sense amplifiers of that bank. Figure 4.15 makes three implicit assumptions. The first assumption made in Figure 4.15 is that both banks i and j are open, where bank i is different from bank j. The second read request is made to bank j, but to different row than the row of data presently held in the array of sense amplifiers of bank j. In this case, the precharge command to bank j can proceed concurrently with the column read access to a bank i. The second assumption made in Figure 4.15 is that the t_{RAS} requirement had been satisfied in bank j, and bank j can be immediately precharged. The third and final assumption made in Figure 4.15 is that the DRAM controller does not support command or transaction re-ordering between different transaction requests. That is, all of the DRAM commands associated with the first request must be scheduled before any DRAM commands associated with the second request can be scheduled.

Figure 4.15 shows that due to the bank conflict, the read request to bank j is translated into a sequence of three DRAM commands. The first command in the sequence precharges the sense amplifiers to bank j, the second command brings the selected row to the sense amplifiers, and the last command in the sequence performs the actual read request and returns data from the DRAM devices to the DRAM controller. Figure 4.15 illustrates that consecutive read requests to different rows, with the second row hitting a bank conflict, given that the DRAM command sequences cannot be dynamically re-ordered, then the two requests can at best be scheduled with minimum timing distance of $t_{CMD} + t_{RP} + t_{RCD}$.

With Command Re-Ordering

Figure 4.15 illustrates the timing of two requests to different banks with the second request hitting a bank conflict; and the DRAM controller does not support command or transaction re-ordering. In contrast, Figure 4.16 shows that the DRAM memory system can obtain bandwidth utilization if the DRAM controller can interleave or re-order DRAM commands from different transactions requests. Figure 4.16 shows the case where the DRAM controller allows the precharge command for bank j to proceed ahead of the column read command for the transaction request to bank i. In this case, the column read command

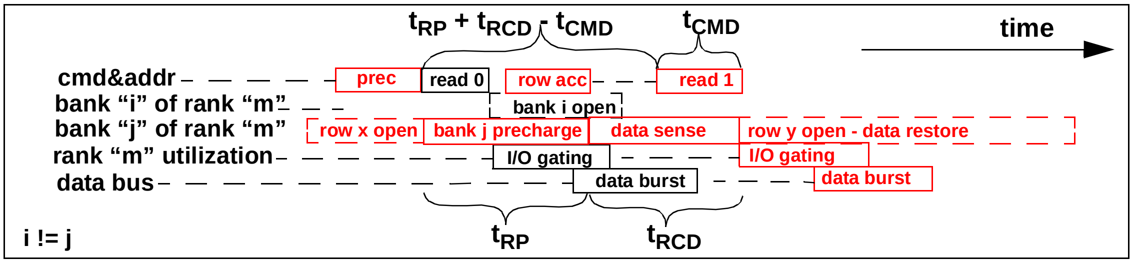


Figure 4.16: Consecutive DRAM read commands to different banks, bank conflict, with command re-ordering.

to bank i can proceed in parallel with the precharge command to bank j, since these two commands utilize different resources in different banks. To obtain the better utilization of the DRAM memory system, the DRAM controller must be designed with the capability to re-order and interleave commands from different transaction requests. Figure 4.16 shows that in the case the DRAM memory system can interleave and re-order DRAM commands from different transaction requests, the two column read commands can be scheduled with the timing of t_{RP} + t_{RCD} - t_{CMD}. Figure 4.16 thus illustrates one way that a DRAM memory systems can obtain better bandwidth utilization with advanced DRAM controller designs.

4.2.4 Consecutive Read Requests to Different Ranks

Consecutive read commands to the open banks of the same rank of DRAM device can be issued and pipelined consecutively. However, consecutive read commands to different ranks of memory may not be issued and pipelined back to back depending on the system level synchronization mechanism and the operating data rate of the memory system. In some memory systems, consecutive read commands to different ranks of memory relies on system level synchronization mechanisms that are non-trivial for multi-rank, high data rate memory systems. In these systems, the data bus must idle for some period of time between data bursts from different ranks on the shared data bus. Figure 4.17 illustrates the timing and

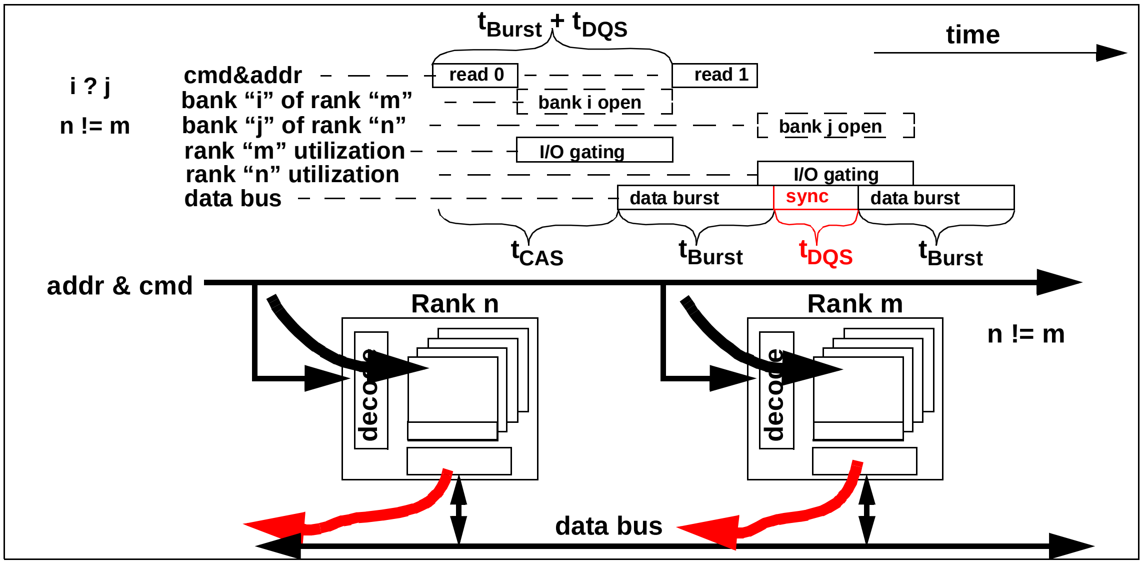


Figure 4.17: Back-to-back column read commands to different ranks.

command sequence of two consecutive read commands to different ranks. In Figure 4.17, the read-write data strobe re-synchronization time is labelled as t_{DQS}. For relatively low frequency SDRAM memory systems, data synchronization strobes are not used, and t_{DQS} is zero. For Direct RDRAM memory system, the use of the topology matched source synchronous clocking scheme obviates the need for a separate strobe signal, and t_{DQS} is also zero. However, for DDR SDRAM, DDR2 and DDR3 SDRAM memory systems, the use of a system level data strobe signal shared by all of the ranks means that the t_{DQS} data strobe re-synchronization penalty is non-zero.

4.2.5 Consecutive Write Requests: Open Banks

Differing from the case of consecutive column read commands to different ranks of DRAM devices, consecutive column write commands to different ranks of DRAM devices can be pipelined consecutively in modern DRAM memory systems. The difference between consecutive column write commands to different ranks of DRAM devices and consecutive column read commands to different ranks of DRAM devices is that in case of consecutive

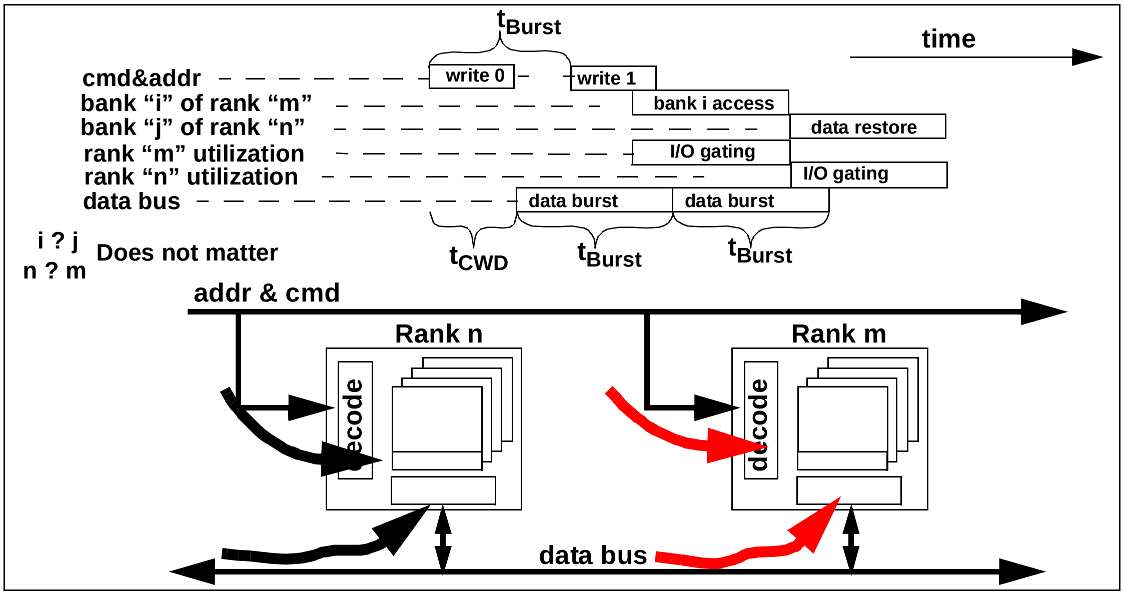


Figure 4.18: Consecutive write commands to different ranks.

column read commands to different ranks of DRAM devices, one rank of DRAM devices must first send data on the shared data bus, give up control of the shared data bus, then the other rank of DRAM devices must then gain control of the shared data bus and send its data to the DRAM controller. In the case of the consecutive column write commands to different ranks of memory, the DRAM memory controller sends the data to both ranks of DRAM devices without needing to give up control of the shared data bus to another bus master. Figure 4.18 shows two write commands to different ranks, labelled as write 0 and write 1, pipelined consecutively, and consecutive column write commands to open banks of memory can occur every t_{Burst} cycles without needing any idle time on the data bus.

4.2.6 Consecutive Write Requests: Bank Conflicts

Similar to the case of the consecutive read requests to different rows of the same bank, consecutive write requests to different rows of the same bank must also respect the timing requirements of t_{RAS} and t_{RP}. Additionally, column write commands must also respective the timing requirements of the write recovery time t_{WR}. In case of write commands to

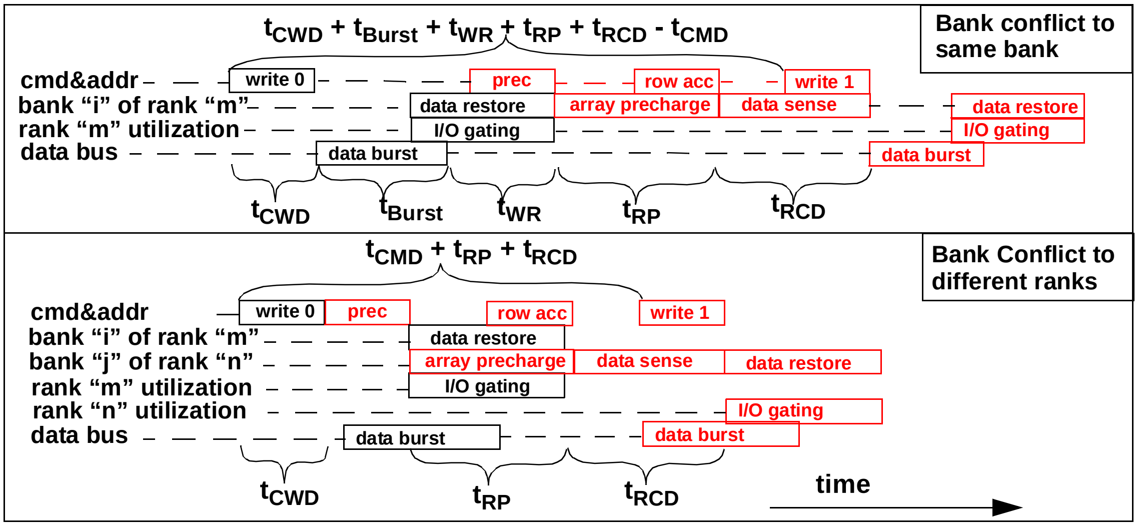


Figure 4.19: Consecutive write commands, bank conflict best cases.

different rows of the same bank, the write recovery time means that the precharge cannot begin until the write recovery time has allowed data to move from the interface of the DRAM devices through the sense amplifiers into the DRAM cells. Figure 4.18 shows two of the best case timing of two consecutive write requests made to different rows in the same bank. The minimum scheduling distance between two write commands to different rows of the same bank is t_{CWD} + t_{Burst} + t_{WR} + t_{RP} + t_{RCD} - t_{CMD}.

Figure 4.18 also shows the case where consecutive write requests are issued to different ranks of DRAM devices with the second write request results in a bank conflict. In this case, the first write command proceeds, and assuming that bank j for rank n had previously satisfied the t_{RAS} timing requirement, the precharge command for a different bank or different rank can be issued immediately. Similar to the case of the consecutive read requests with bank conflicts to different banks, bank conflicts to different banks and different ranks for consecutive write requests can also benefit from command re-ordering.

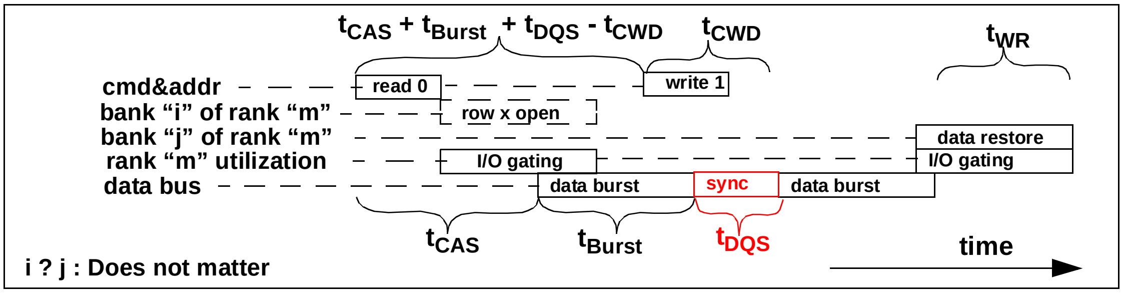


Figure 4.20: Write command following read command to open banks.

4.2.7 Write Request Following Read Request: Open Banks

Similar to consecutive read commands and consecutive write commands, the combination of a write command that immediately follows a read command can be scheduled consecutively subject to the timing of the respective data bursts on the shared data bus. Figure 4.20 illustrates a write command that follows a read command and shows that the internal data movement of the write command does not conflict with the internal data movement of the read command. As a result, a column write command can be issued into the DRAM memory system after a column read command as long as the timing of data burst returned by the DRAM device for the column read command does not conflict with the timing of the data burst sent by the DRAM controller to the DRAM device. Figure 4.20 shows that the minimum scheduling distance between a read command that follows a read command is $t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$.

The minimum scheduling distance between a write request that follows a read request is different for different memory access protocols. For example, in the SDRAM memory system, t_{DQS} and t_{CWD} are both zero, and the minimum scheduling distance between a write request that follows a read request is simply $t_{CAS} + t_{Burst}$.

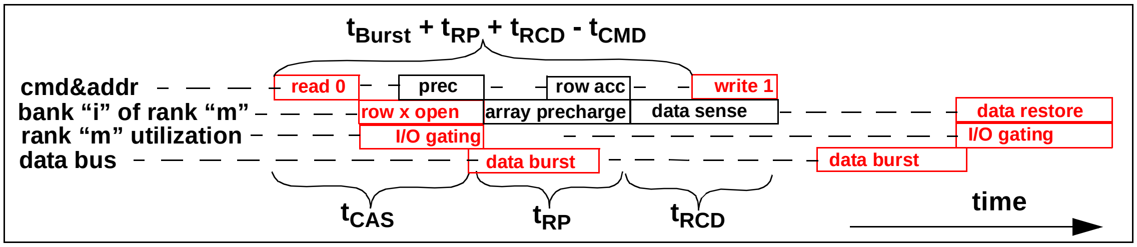


Figure 4.21: Write command following read command to same bank: bank conflict, best case.

4.2.8 Write Following Read: Same Bank, Conflict, Best Case

Figure 4.21 illustrates the best case scenario for a write request that follows a read request to the same bank, but to different rows. In the best case scenario presented in Figure 4.21, data in the row accessed by the read request has already been restored to the DRAM cells. That is, the t_{RAS} timing requirement has already been satisfied for the row held by bank "i" before the read command illustrated in Figure 4.21 was issued into the DRAM memory system. Figure 4.21 shows that under this condition, the precharge command can be issued consecutively to the column read command. The row access command to the different row in bank i can then be issued into the DRAM memory system after the DRAM array in bank i is precharged. The column write command can then proceed after time t_{RCD} following the row access command. Figure 4.21 thus shows that a read request that follows a read request to different rows of the same bank can at best occur with the minimum scheduling distance of $t_{Burst} + t_{RP} + t_{RCD} - t_{CMD}$.

Figure 4.21 show the best case timing of the scenario where a read request that follows a read request to different rows of the same bank. The best case scenario assumes that the t_{RAS} timing requirement has been satisfied for bank i. In worst case that the read command was in fact issued immediately after the preceding row access command, the t_{RAS} timing requirement must be satisfied before the precharge command can be issued. In the worst

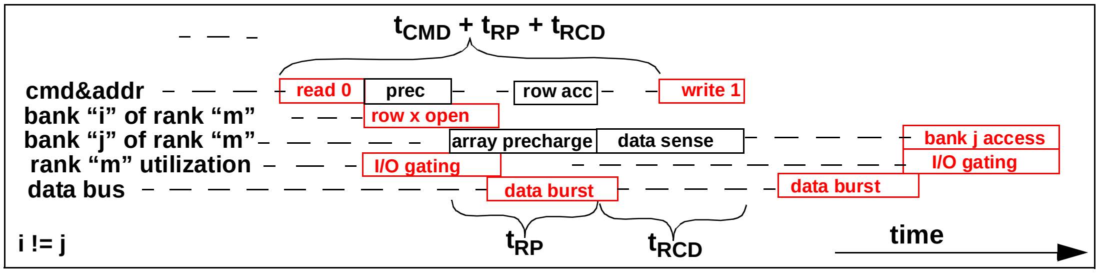


Figure 4.22: Write command following read command to different banks: bank conflict, best case.

case scenario, the minimum scheduling distance between the column read command and the column write command that follows it increases to a entire row cycle, t_{RC}.

4.2.9 Write Following Read: Different Banks, Conflict, Best Case

Figure 4.22 illustrates the case where a write request follows a read request to different banks. Figure 4.22 shows that the column read command is issued to bank i, the column write command is issued to bank j, and i is different from j. In the common case, the two commands can be pipelined consecutively with the minimum scheduling distance shown in Figure 4.20. However, the assumption given in Figure 4.22 is that the write command is a write command to a different row than the row currently held in bank j. As a result, the DRAM controller must first precharge bank j and issue a new row access command to bank j before the column write command can be issued. In the best case scenario presented, the row accessed by the write command in bank j had already been restored to the DRAM cells, and more than t_{RAS} time period had elapsed since row was initially accessed. Figure 4.22 shows that under this condition, the read command and the write command that follows it to a different bank can best scheduled with the minimum scheduling distance of $t_{CMD} + t_{RP} + t_{RCD}$.

Figure 4.22 shows the case where the ordering between DRAM commands from different requests is strictly observed. In this case, the precharge command sent to bank j is

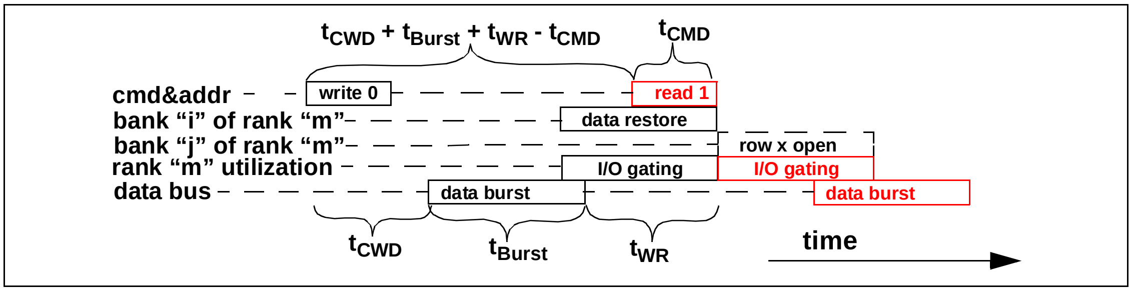


Figure 4.23: Read following write to same rank of DRAM devices.

not constrained by the column read command to bank i. In a memory system with DRAM controllers that support command re-ordering and interleaving DRAM commands from different transaction requests, the efficiency of the DRAM memory system in scheduling a write request with a bank conflict that follows a read request can be increased in the same manner as illustrated for consecutive read requests in Figures 4.15 and 4.16.

4.2.10 Read Following Write to Same Rank, Open Banks

Figure 4.23 shows the case for a column read command that follows a column write command to open banks in the same rank of DRAM devices. The difference between a column read command and a column write command is that the direction of data flow within the selected DRAM devices is reversed with respect to each other. The importance in the direction of data flow can be observed when a read command is scheduled after a write command to the same rank of DRAM devices. Figure 4.23 shows that the difference in the direction of data flow limits the minimum scheduling distance between the column write command and the column read command that follows to the same rank of devices. Figure 4.23 shows that after the DRAM controller places the data onto the data bus, the DRAM device must make use of the shared I/O gating resource in the DRAM device to move the write data through the buffers into the proper columns of the selected bank. Since the I/O

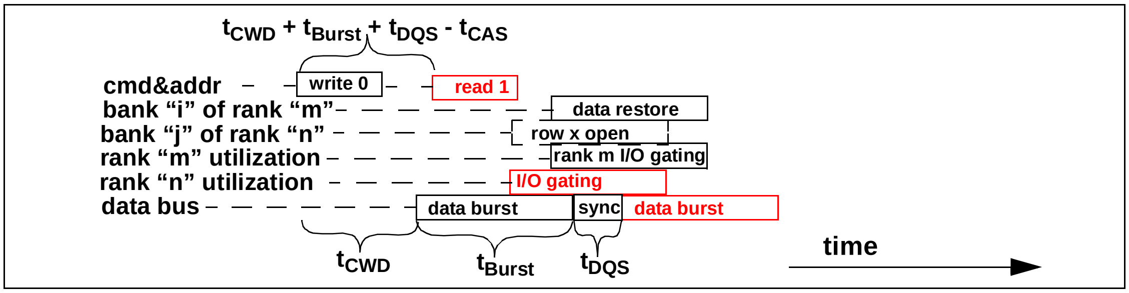


Figure 4.24: Read following write to different ranks of DRAM devices.

gating resource is shares between all banks within a rank of DRAM devices, the sharing of the I/O gating device means that a read command that follows a write command to the same rank of DRAM devices must wait until the write command has been completed before the read command can make use of the shared I/O gating resources regardless of the target or destination bank ID’s of the respective column access commands. Figure 4.23 shows that the minimum scheduling distance between a write command and a subsequent read command to the same rank of memory is t_{CWD} + t_{Burst} + t_{WR} - t_{CMD}.

In order to alleviate the write-read turnaround time illustrated in Figure 4.23, some high performance DRAM devices have been designed with write buffers so that as soon as data have been written into the write buffers, the I/O gating resource can be used by another command such as a column read command.

4.2.11 Read Following Write to Different Ranks, Open Banks

Figure 4.24 shows a slightly different case for a column read command that follows a column write command than the case illustrated in Figure 4.23. The combination of column read command issued after a column write command illustrated in Figure 4.24 differs from the combination of column read command issued after a column write command illustrated in Figure 4.23 in that the column write command and the column read command are issued

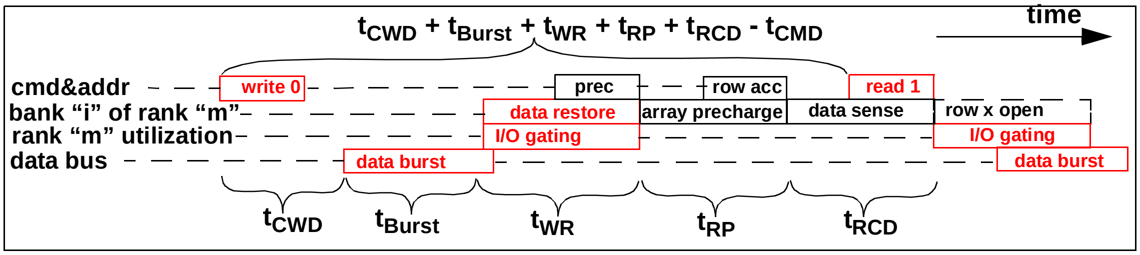


Figure 4.25: Read following write to different rows of the same bank: best case.

to different ranks of memory. Since the data movements are to different ranks of memory, the conflict in the directions of data movement inside of each rank of memory is irrelevant. The timing constraint between the issuance of a read command after a write command to different ranks is then reduced to the data bus synchronization overhead of t_{DQS}, the burst duration t_{Burst}, and the relative timing differences between read and write command latencies. The minimum time period between a write command and a read command to different ranks of memory is thus $t_{CWD} + t_{Burst} + t_{DQS} - t_{CAS}$.

In an SDRAM memory system, t_{CWD} and t_{DQS} are both zero, and the minimum scheduling distance between a column write command and a column read command that follows it to a different rank of memory is $t_{Burst} - t_{CAS}$. In contrast, t_{CWD} is one full cycle less than t_{CAS} in a DDR2 SDRAM memory system. If t_{DQS} can be minimized to one full cycle, $t_{CWD} + t_{DQS} - t_{CAS}$ would cancel to zero, and the minimum scheduling distance between a read command that follows a write command to a different rank in DDR2 SDRAM memory system is simply t_{Burst}.

4.2.12 Read Following Write to Same Bank, Bank Conflict

Figure 4.25 illustrates the case where a read request follows a write request to different rows of the same bank. In the best case scenario presented, the t_{RAS} row restoration time requirement for the previous row has already been satisfied. Figure 4.25 shows that under

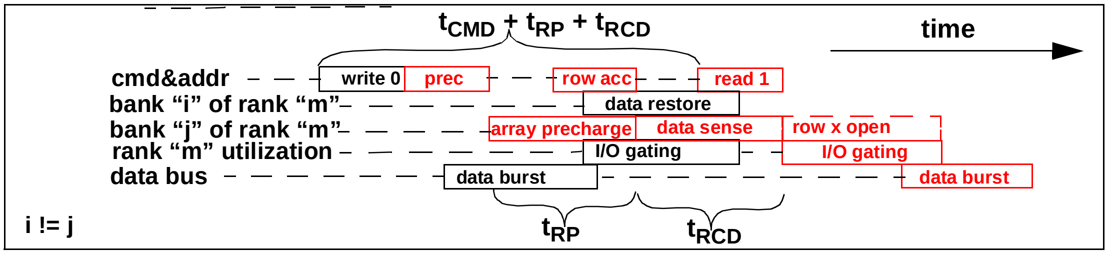


Figure 4.26: Read following write to different banks, bank conflict, best case.

this condition, the precharge command can be issued as soon as the data from the column write command has been written into the DRAM cells. That is, the write recovery time t_{WR} must be respected before the precharge command can proceed to precharge the DRAM array. Figure 4.25 shows that the best case minimum scheduling distance between a read request that follows a write request to different rows of the same bank is t_{CWD} + t_{Burst} + t_{WR} + t_{RP} + t_{RCD} - t_{CMD}.

Figure 4.25 shows the command interaction of a read request that follows a write request to different rows of the same bank on a DRAM device that does not have a write buffer. In DRAM devices with write buffers, the data for the column write command is temporarily stored in the write buffer. In case that a read request arrives after a write request to retrieve data from a different row of the same bank, a separate commit-data command may have to be issued by the DRAM controller to the DRAM devices and force the write buffer to commit the data stored in the write buffer into the DRAM cells before the array can be precharged for another row access.

4.2.13 Read Following Write: Different Banks Same Rank, Conflict: Best Case

Finally, Figure 4.26 illustrates the case where a read request follows a write request to different banks of the same rank of DRAM devices. However, the read request is sent to bank j, and a different row is presently active in bank j than the row needed by the read request. Figure 4.26 assumes that the t_{RAS} timing requirement has already been satisfied for bank j, and the DRAM memory system does not support DRAM command re-ordering between different memory transactions. Figure 4.26 shows that in this case, the precharge command for the read request command is can be issued as soon as the write command is issued. Figure 4.26 thus shows that the minimum scheduling distance in this case is t_{CMD} + t_{RP} + t_{RCD}.

Figure 4.26 also reveals several points of note. One obvious point is that the DRAM command sequence illustrated in Figure 4.26 likely benefits from command re-ordering between different memory transactions. A second, less obvious point illustrated in Figure 4.26 is that the computed minimum schduling distance depends on the relative duration of the various timing parameters. That is, Figure 4.26 assumes that the precharge command can be issued immediately after the write command and that t_{CMD} + t_{RP} + t_{RCD}. is greater than t_{CWD} + t_{Burst} + t_{WR}. In case that t_{CMD} + t_{RP} + t_{RCD}. is in fact less than t_{CWD} + t_{Burst} + t_{WR}, the use of the shared I/O gating resource becomes the bottleneck and the column read command must wait until the write recovery phase of the column write command has completed before the column read command can proceed. That is, the minimum schduling distance between a write request and a read request to a different bank with a bank conflict is in fact the larger of t_{CMD} + t_{RP} + t_{RCD}. and t_{CWD} + t_{Burst} + t_{WR}.

4.3 Minimum Scheduling Distances

In previous sections, the resource usage model for DRAM devices was applied to basic DRAM commands and minimum scheduling distances between different combinations of DRAM commands were examined in detail. Table 4.3 summarizes the minimum scheduling distances of read and write requests in an open-page DRAM memory system to a combination of channels, ranks, banks and rows. Table 4.3 summarizes the minimum scheduling distances between read and write requests rather than between row access, column read, column write and precharge commands. In table 4.3, the letter “R” represents a read request, the letter “W” represents a write request, the letter “s” means that the consecutive requests are made to same channel, rank, bank or row, and “d” means that the requests are made different channel, rank, bank or row. For example, the first row of the table shows that consecutive DRAM read commands to open banks in the same channel, rank, bank and row can be issued with a minimum timing of t_{Burst}.

In case of a bank conflict between two consecutive requests to a DRAM memory system, some degree of uncertainty exists as to the minimum scheduling distance between those commands since the timing of the second request depends on the progress of the data restoration phase of the previous row access. Table 4.3 shows both the best case and worst case minimum scheduling distances for consecutive requests to an open-page DRAM memory system that does not support command re-ordering. The best case scenario shows the minimum scheduling distance given that the t_{RAS} timing requirement of the row access command had already been satisfied, and the worse case scenarios shows minimum scheduling distance given that the t_{RAS} timing requirement of the row access command had not been satisfied*.

Legend

R = Read
W = Write
s = same;
d = different
o = open
c = conflict;

prev	next	rank	bank	row	Minimum scheduling distance between DRAM commands open-page No Command Re-Ordering Best Case	Minimum scheduling distance between DRAM commands Worst Case
R	R	s	s	o	t_{Burst}	-
R	R	s	s	c	$t_{Burst} + t_{RP} + t_{RCD}$	t_{RC}
R	R	s	d	o	t_{Burst}	-
R	R	s	d	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
R	R	d	-	o	$t_{DQS} + t_{Burst}$	-
R	R	d	-	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
R	W	s	s	o	$t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$	-
R	W	s	s	c	$t_{Burst} + t_{RP} + t_{RCD} - t_{CWD}$	t_{RC}
R	W	s	d	o	$t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$	-
R	W	s	d	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
R	W	d	-	o	$t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$	-
R	W	d	-	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
W	R	s	s	o	$t_{CWD} + t_{Burst} + t_{WR} - t_{CMD}$	-
W	R	s	s	c	$t_{CWD} + t_{Burst} + t_{WR} + t_{RP} + t_{RCD} - t_{CMD}$	t_{RC}
W	R	s	d	o	$t_{CWD} + t_{Burst} + t_{WR} - t_{CMD}$	-
W	R	s	d	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
W	R	d	-	o	$t_{CWD} + t_{Burst} + t_{DQS} - t_{CAS}$	-
W	R	d	-	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
W	W	s	s	o	t_{Burst}	-
W	W	s	s	c	$t_{CWD} + t_{Burst} + t_{WR} + t_{RP} + t_{RCD} - t_{CMD}$	t_{RC}
W	W	s	d	o	t_{Burst}	-
W	W	s	d	c	$t_{CWD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
W	W	d	-	o	t_{Burst}	-
W	W	d	-	c	$t_{CMD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$

TABLE 4.3: **Minimum timing for consecutive read and write transactions: open-page**

*. Some request combinations list t_{RC} as the worst case minimum scheduling distance while other request combinations list $t_{RC} - t_{Burst}$ as the worst case minimum scheduling distance. The assumption used in table 4.3 is that a row access is only issued in combination with a column access command. As a result, there must be at least one column access command to an open row before another column access command to a different row arrives at the same bank, and the minimum scheduling distance between two requests to different banks is $t_{RC} - t_{Burst}$ rather than t_{RC}. Table 4.3 in essence shows the second and third request in a sequence of requests where the first request that conflicts with the third request is inferred by the status of the open row.

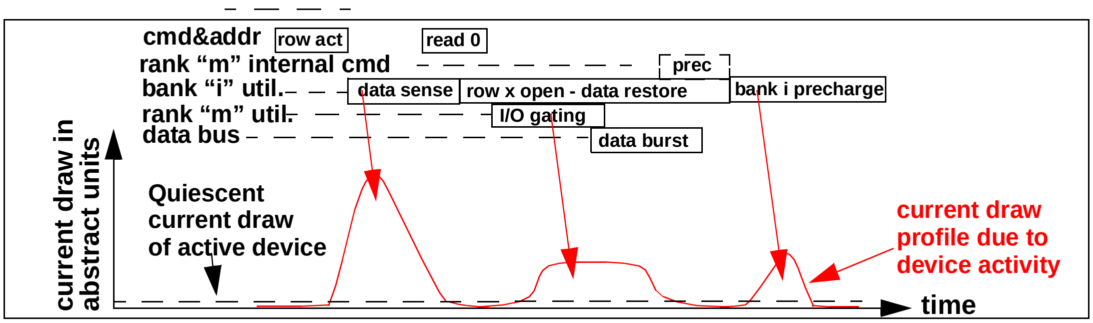

cmd&addr
row act
read 0
rank "m" internal cmd
prec
bank "i" util.
data sense
row x open - data restore
bank i precharge
rank "m" util.
I/O gating
data bus
data burst
current draw in abstract units
Quiescent current draw of active device
current draw profile due to device activity
time

Figure 4.27: : Current Profile of a DRAM Read Cycle.

4.4 Additional Constraints: Power

In the previous sections, the resource contention model was used to construct the table of minimum scheduling distances between DRAM commands. Unfortunately, constraints in addition to the resource contention issue exist in modern DRAM memory systems and limits bandwidth utilization of modern DRAM based memory systems. One such constraint is related to the power consumption of DRAM devices. With continuing emphasis placed on memory system performance, DRAM manufacturers are expected to push for ever higher data transfer rates in each successive generation of DRAM devices. However, just as increasing operating frequencies lead to higher activity rates and higher power consumption in modern processors, increasing data rates for DRAM devices also increase the potential for higher activity rates and higher power consumptions on DRAM devices. One solution deployed to limit the power consumption of DRAM devices is to constrain the activity rate of DRAM devices. Constraints on the activity rate of DRAM devices in turn limit the capability of DRAM devices to move data, and limits the performance of DRAM memory systems.

In modern DRAM devices, each time a row is activated, thousands of bits are discharged, sensed, then restored to the DRAM cells in parallel. As a result, the row activation command is a relatively energy intensive operation. Figure 4.27 shows the abstract current profile of a DRAM read cycle. Figure 4.27 shows that an active DRAM device draws a relatively low and constant quiescent current level. The DRAM device then draws additional current for each activity on the DRAM device. The total current draw of the DRAM device is simply the summation of the quiescent current draw and the current draw of each activity on the DRAM device.

The current profile shown in Figures 4.27 and 4.28 are described in terms of abstract units rather than concrete values. The reason that the current profiles are shown in abstract units in Figures 4.27 and 4.28 is that the magnitude of the current draw for the row activation command depends on the number of bits in a row that are activated in parallel, and the magnitude of the current draw for the data burst on the data bus depends on the data bus width of the DRAM device. As a result, the current profile of each command on each respective device depends not only on the type of the command, but also on the internal organization and external configurations of the DRAM device.

All modern DRAM devices contain multiple banks of DRAM arrays that can be pipelined to achieve high performance. Unfortunately, since the current profile of an DRAM device is proportional to its activity rate, a high performance, highly pipelined DRAM device can also draw a large amount of current. Figure 4.28 shows the individual contributions to the current profile of two pipelined DRAM read cycles on the same device. The total current profile of the pipelined DRAM device is not shown in Figure 4.28, but can be computed by the summation of the quiescent current profile and the current profiles of

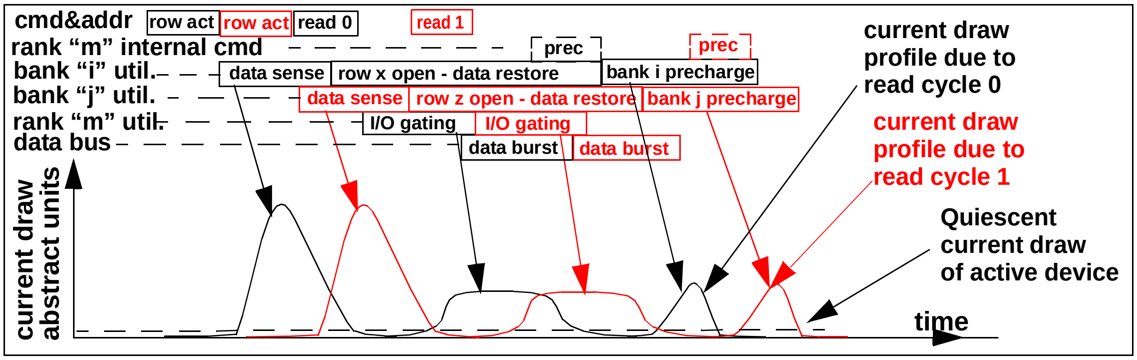


Figure 4.28: : Current Profile of Two Pipelined DRAM Read Cycles.

the two respective read cycles. The problem of power consumption for a high performance DRAM device is that instead of only two pipelined read or write cycles, multiple read or write cycles can be pipelined, and as many as t_{RC}/t_{Burst} number of read or write cycles can be theoretically pipelined and in different phases in a single DRAM device. To limit the maximum current draw of a given DRAM device and avoid the addition of heat removal mechanisms such as heat spreaders and heatsinks, new timing parameters have been defined in DDR2 and DDR3 devices to limit the activity rate and power consumption of DRAM devices.

4.4.1 t_{RRD}: Row to Row (activation) Delay

In DDR2 SDRAM devices, the timing parameter t_{RRD} has been defined to specify the minimum time period between row activations on the same DRAM device. In the present context, the acronym RRD stands for **r**ow-to-**r**ow activation **d**elay. The timing parameter t_{RRD} is specified in terms of nanoseconds, and table 4.4 shows that by specifying t_{RDD} in terms of nanoseconds instead of number of cycles, a minimum spacing between row activation is maintained regardless of operating datarates. For memory systems that implement the close-page row buffer management policy, t_{RRD} effectively limits the

Device configuration	512 Mbit x 4	256 Mbit x 8	128 Mbit x 16
Data bus width	4	8	16
Number of banks	8	8	8
Number of rows	16384	16384	8192
Number of columns	2048	1024	1024
Row size (bits)	8192	8192	16384
t_{RRD} (ns)	7.5	7.5	10
t_{FAW} (ns)	37.5	37.5	50

TABLE 4.4: t_{RRD} and t_{FAW} for 1 Gbit DDR2 SDRAM device from Micron

maximum sustainable bandwidth of a memory system with a single rank of memory. In memory systems with 2 or more ranks of memory, consecutive row activation commands can be directed to different ranks to avoid the t_{RRD} constraint.

Table 4.4 shows t_{RRD} for different configurations of a 1 Gbit DDR2 SDRAM device from Micron. Table 4.4 shows that the 1 Gbit DDR2 SDRAM device with the 16 bit wide data bus is internally arranged as 8 banks of 8192 rows per bank and 16384 bits per row. Comparatively, the 512 Mbit x 4 and 256 Mbit x 8 configuration of the 1 Gbit DDR2 SDRAM device are arranged internally as 8 banks of 16384 rows per bank and 8192 bits per row. Table 4.4 thus shows that with the larger row size, each row activation on the 128 Mbit x 16 configuration draws more current than a row activation on the 256 Mbit x 8 or 512 Mbit x 4 configuration, and the data sheet requires that the row activations must be spaced farther apart in time.

4.4.2 t_{FAW}: Four Bank Activation Window

In DDR2 SDRAM devices, the timing parameter t_{FAW} has been defined to specify a rolling time frame in which a maximum of four row activations on the same DRAM device may be engaged concurrently. The acronym FAW stands for Four bank Activation Window.

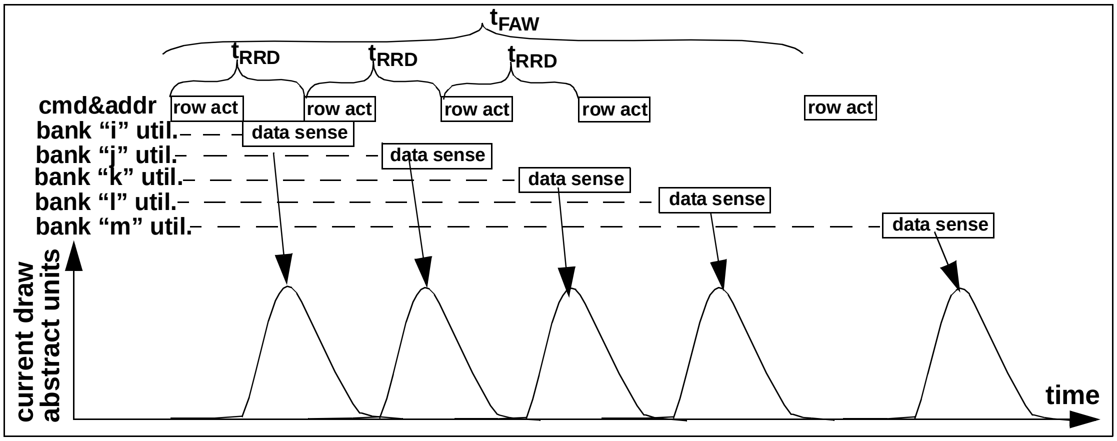


Figure 4.29: : Maximum of Four Row Activations in any t_{FAW} time frame.

Figure 4.29 shows a sequence of row activation requests to different banks on the same DDR2 SDRAM device that respects both t_{RRD} as well as t_{FAW}. Figure 4.29 shows that the row activation requests are spaced at least t_{RRD} apart from each other, and that the fifth row activation to a different bank is deferred until at least t_{FAW} time period has passed since the first row activation was initiated. For memory systems that implement the close-page row buffer management system, t_{FAW} places additional constraint on the maximum sustainable bandwidth of a memory system with a single rank of memory regardless of operating datarates.

The timing parameters t_{RRD} and t_{FAW} have been defined for DDR2 SDRAM devices that are 1 Gbit or larger. These timing parameters will carry over to DDR3 and future DDRx devices, and they are expected to increase in importance as future DRAM devices are introduced with larger row sizes.

4.5 DDR2 SDRAM Protocol

In previous sections, a generic DRAM access protocol was examined in detail. In this section, the DDR2 SDRAM memory access protocol is described in detail. The goal of this section is to illustrate by example of how the generic DRAM access protocol applies to a specific DRAM memory system.

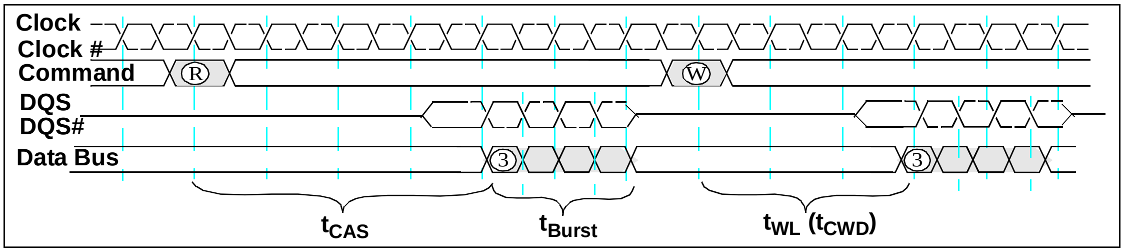


Figure 4.30: A column read command and a column write command in DDR2 SDRAM syste

4.5.1 DDR2 SDRAM Memory System Basics

Figure 4.30 illustrates the progression of a column read command and a column write command in a DDR2 SDRAM memory system. Figure 4.30 illustrates some important aspects of the DDR2 SDRAM memory system:

- DRAM commands asserted on the command bus occupy a full clock cycle and straddle clock boundaries.
- DDR, DDR2 and future DDRx SDRAM memory systems transport two beats of data in each clock cycle, each beat equates to one column of data.
- Data for column commands are transported with respect to the timing of the source synchronous differential data strobe signals, DQS and DQS#.
- Data for column read commands are sent by the DRAM devices and edge aligned to the data strobe signal.
- Data for column write commands are sent by the DRAM controller and center aligned to the data strobe signal.

		# of cycles @		
parameter	value	400 Mbps	533 Mbps	667 Mbps
t_{Cycle}		5 ns	3.75 ns	3 ns
t_{Burst}	-	2, 4	2, 4	2, 4
t_{CAS}	-	3, 4	3, 4, 5	4, 5
t_{CMD}	-	1	1	1
$t_{CWD}(t_{WL})$	-	2, 3, 4	3, 4	3, 4
t_{DQS}	-	1	1	1, 2
t_{RAS}	40 ns	8	11	14
t_{RC}	55 ns	11	15	19
t_{RCD}	15 ns	3	4	5
t_{RP}	15 ns	3	4	5
t_{WR}	15 ns	3	4	5

TABLE 4.5: Typical timing parameter values of DDR2 SDRAM Devices

4.5.2 Typical Parameter Values

The timing parameters used to examine the DRAM access protocol were previously described only in abstraction. Table 4.5 shows typical values for those timing parameters in three different speed grades of DDR2 SDRAM devices. Table 4.5 shows that the timing parameters t_{RAS}, t_{RC}, t_{RCD}, t_{RP} and t_{WR} are naturally specified in terms of nanoseconds. Table4.5 also lists the values of the various timing parameters in terms of the number of cycles in specific DDR2 SDRAM devices. For example, in a DDR2 SDRAM device that operates at 400 Mbps, each cycle is 5 ns in duration. To meet the t_{RAS} timing requirement of 40 ns as illustrated, there must be at least 8 clock cycles between the time when a row is opened for access and the time when a precharge command can be issued to the 400 Mbps DDR2 SDRAM device.

DRAM Memory modules are now commonly offered for sale directly to consumers. To reduce the complexity of the myriad of timing parameters, DRAM memory modules are sold on the basis of the latency values of t_{CAS}, t_{RCD} and t_{RP}. That is, DRAM modules are now offered for sale as PC2-4300 (3-4-4) where the CAS latency is 3 cycles, t_{RCD} is 4 cycles and t_{RP} is 4 cycles or PC2-4300 (4-5-5) where the CAS latency is 4 cycles, t_{RCD} is 5 cycles and t_{RP} is 5 cycles.

4.6 Summary

In this chapter, a generic DRAM access protocol is described in some detail. The generic DRAM access protocol is created from a basic resource usage model. That is, two DRAM commands can be pipeline consecutively if they do not require the use of a shared resource at the same instance in time. Additional constraints, such as power consumption limitations, can further limit the issue rate of DRAM commands.

One popular question that is often asked, but not addressed by the description of the generic DRAM access protocol, is in regards to what happens if the timing parameters are not fully respected. For example, what happens when a precharge command is issued before the t_{RAS} data restoration timing parameter has been fully satisfied? Does the DRAM device contain enough intelligence to delay the precharge command until t_{RAS} has been satisfied?

The answer to questions such as these is that in general, DRAM devices contain very little intelligence. The DRAM device manufacturers provide datasheets to specify the minimum timing constraints for individual DRAM commands. To ensure that the DRAM devices operate correctly, the DRAM controller must respect the minimum and maximum timing parameters as defined in the datasheet. In the specific case of a precharge command issued to a DRAM device before t_{RAS} has been satisfied, the DRAM device may still operate correctly, since the electrical charge in the DRAM cells may have already been mostly restored by the time the precharge command was engaged. The issue of early command issuance is thus analogous to that of the practice of processor overclocking. That is, a processor manufacturer can specify that a processor will operate correctly within a given frequency and supply voltage range, but an end user may increase the supply voltage and operating frequency of the processor in hopes of obtaining better performance from the processor. In such a case, the processor may well operate correctly from a functional perspective. However, the parameters of operation is then outside of the bounds specified by the processor manufacturer, and the functional correctness of the processor is no longer guaranteed by the processor manufacturer. Similarly, in case that a DRAM command is issued at a timing that is more aggressive than that specified by the DRAM device datasheet, the DRAM device may still operate correctly, but the functional correctness of the DRAM device is no longer guaranteed by the manufacturer of the DRAM device or memory module.

Finally, the work in this chapter points out that modern DRAM memory systems such as SDRAM, DDR SDRAM, DDR2 SDRAM, DDR3 SDRAM and even Direct RDRAM have memory access protocols that are more similar to each other than different. The similarity of the memory access protocols enables the definition of the generic DRAM access protocol, and the performance analysis of the generic DRAM memory system can be broadly applied to different memory systems. The work presented in this chapter creates a baseline DRAM memory access protocol that can be used in a framework that abstractly analyzes DRAM based memory systems. Specifically, the table summarized as table 4.3 contains the minimum scheduling distance for any pair of DRAM commands to a single channel of a DRAM memory system. The table thus enables the computation of DRAM memory system bandwidth and latency characteristics in support of abstract analysis of DRAM memory system performance characteristics.

CHAPTER 5

DRAM Memory Controller

5.1 Primary Functions

In modern computer systems, the system controller acts as the glue logic that connects processors, high speed input-output devices and the memory system to each other. The system controller can exist as a separate device or as part of the processor and integrated into the processor package; the function of the system controller remains essentially same in either case. The primary function of the system controller is to manage the flow of data between the processors, input-output devices and the memory system, correctly and efficiently. Within the system controller, the function of the DRAM memory controller is to manage the flow of data into and out of the DRAM devices. However, due to the complexity of DRAM memory access protocols, the large number of timing parameters, the innumerable combinations of memory system organizations, different workload characteristics and different design goals, the design space of a DRAM memory controller has as much freedom as the design space of a processor that implements a specific instruction set architecture. In that sense, just as the instruction set architecture defines the programming model of a processor, the DRAM access protocol and timing parameters define the interface protocol of a DRAM memory controller. In both cases, the performance characteristics of the respective devices depend on the implementation specifics of the microarchitecture, rather than the superficial description of a programing model or the interface protocol.

DRAM memory controllers can be designed to minimize die size, minimize power consumption, maximize system performance, or simply a reasonably optimum combination of various conflicting design goals. The goal in this chapter is to examine the various issues important to the design and implementation of modern DRAM memory controllers. Specifically, the following items are particularly important to the design and implementation of a DRAM memory controller:

- Row-buffer Management Policy
- Address Mapping Scheme
- Memory Transaction and DRAM Command Ordering Scheme

Due to the increasing disparity in the operating frequency of modern processors and the access latency to main memory, there is a large body of active and ongoing research in the architectural community devoted to the optimization of the DRAM memory controller. Specifically, ***Address Mapping Scheme*** designed to minimize bank address conflicts have been studied by Lin et. al. and Zhang et. al[24,25,27]. ***DRAM Command and Memory Transaction Ordering Schemes*** have been studied by Briggs et. al., Cuppu et. al., Hur et. al., McKee et. al., and Rixner et. al[28,32,33,34,37,42,51]. Due to the sheer volume of research into optimal DRAM controller designs for different types of DRAM memory systems and workload characteristics, this chapter is not intended as a comprehensive summary of all prior work. Rather, the text in this chapter proceeds to describe the basic concepts of DRAM memory controller design in abstraction. Relevant research on specific topics are then referenced as needed.

5.2 Row-buffer Management Policy

Modern memory controllers typically use one of two policies to manage the operations of sense amplifiers in the DRAM devices. In modern DRAM devices, arrays of sense amplifiers also act as buffers that can provide temporary storage for an entire row of data. As a result, policies that manage the operation of sense amplifiers are better known as ***row-buffer management policies***. The two primary row-buffer management policies are the ***open-page*** policy and the ***close-page*** policy. Different row-buffer management policies exist, including ***dynamic*** row-buffer management policies that use timers to keep a row open for a limited period of time before closing it, or use the access history of the memory access sequence to dynamically determine whether the controller should implement the open-page or close-page policy. However, dynamic row-buffer management policies are typically based on either the close-page policy or the open-page policy, and the examination in this text is limited to open-page and closed-page policies.

5.2.1 Open-Page Row-buffer Management Policy

In modern, commodity DRAM devices, data access to and from the data storage cells is a two step process that requires a separate row access command and a column access command. In cases where the memory access sequence has a high degree of spatial locality, it makes sense to direct the memory access sequences to the same row of memory. The ***Open-page*** row-buffer management policy is designed to favor memory accesses to the same row of memory by keeping sense amplifiers open and holding an entire row of data for ready access. In the ***Open-Page*** row-buffer management policy, the primary assumption is that once a row of data is brought to the array of sense amplifiers, different columns of the same row may be accessed again in the near future. Under this assumption, after a row is activated, the sense amplifiers are kept active to await another memory access to the same row. In the case another memory read access is made to the same row, that memory access could occur with the minimal latency of t_{CAS}, since the row is already active in the sense amplifier and only a column access command is needed to move the data from the sense amplifiers to the memory controller. However, in the case that the access is to a different row of the same bank, the memory controller would have to first precharge the DRAM array, perform another row access, then perform the column access. The minimum latency to access data in the bank conflict case is $t_{RP} + t_{RCD} + t_{CAS}$.

5.2.2 Close-Page Row-Buffer Management Policy

In contrast to the open-page policy, the ***Close-page*** row-buffer management policy is designed to favor random accesses to different rows of memory. The open-page row-buffer management policy and closely related variant policies are typically deployed in memory systems designed for low processor count general purpose computers. In contrast, the close-page row-buffer management policy and closely related variants are typically deployed in memory systems designed for large processor count multiprocessor systems or specialty embedded systems. The reason that open-page row-buffer management policies are deployed in memory systems of low processor count platforms while close-page row-buffer management policies are deployed in the memory systems of larger processor count platforms is that in a memory system that services memory requests from multiple processors or multiple threaded contexts concurrently, the intermixing of memory request sequences reduces the available spatial locality of the resulting memory access sequence*. Moreover, each memory request in an open-page memory system translates to different combinations of DRAM commands with different timings, and the resulting sequence is more difficult to schedule consecutively as compared to the same memory access sequence in a close-page memory system. The bottom line is that as the bandwidth demand to the memory system increases with increasing processor count, the efficiency of the DRAM system to schedule the resulting combinations of DRAM commands decreases, and the throughput of the close-page system can exceed that of a comparable open-page system.

The definition of the row-buffer management policy forms the foundation for the design of a DRAM memory controller. The row-buffer management policy directly or indirectly impacts the selection of the address mapping scheme, the memory command re-ordering mechanism, and the transaction re-ordering mechanism for DRAM memory controllers. In the following sections, the address mapping scheme, the memory command re-ordering mechanism, and the transaction re-ordering mechanism are explored in the context of the row-buffer management policy used.

[*] Some memory systems designed for large processor count platforms, such as Alpha EV7's Direct RDRAM memory system, use open-page policy to manage the sense amplifiers. A fully loaded Direct RDRAM memory system has 32 banks per rank and 32 ranks per channel. The large number of banks in the Direct RDRAM memory system means that memory requests from different processors can be directed to different banks. More conventional memory systems that use SDRAM and variants of DDRx SDRAM memory devices are limited to far fewer banks of DRAM arrays per rank and fewer ranks as well. The result is that the number of bank conflicts grows rapidly for a large processor count application.

5.3 Address Mapping Scheme

Many factors collectively contribute to impact the latency and sustainable bandwidth characteristics of a DRAM memory system. One factor that impacts the performance of a memory system is the address mapping scheme. In a memory system with a poorly devised scheme to suite the workload, multiple, consecutive memory accesses may be mapped to different rows of the same bank, resulting in bank conflicts that impacts performance. On the other hand, a well devised address mapping scheme could map the same memory access sequence to different rows of different banks, where accesses to different banks can occur with some degree of concurrency. The task of an address mapping scheme is to minimize bank conflicts and maximize parallelism in the memory system. The process of devising and examining address mapping schemes begins by examining the property of DRAM memory channels, ranks, banks, rows and columns.

5.3.1 System Organization Variable Definition

To facilitate the examination of address mapping schemes, variables are defined in this section to abstractly denote the organization of memory systems. For the sake of simplicity, a uniform memory system is assumed throughout this text. Specifically, the memory system under examination is assumed to have K independent channels of memory, and each channel consists of L ranks per channel, B banks per rank, R rows per bank, C columns per row, and V bytes per column[*], and the total size of physical memory in the system is simply K * L * B * R * C * V. Furthermore, it is assumed that each memory access loads and stores memory at the granularity of a cacheline. The length of a cacheline is defined as Z bytes, and the

[*] The number of bytes per column may be provided by multiple devices in parallel.

number of cachelines per row is denoted as N. The number of cachelines per row is a dependent variable and it can be computed by multiplying the number of columns per row by the number of bytes per column and divided through by the number of bytes per cacheline. That is, N = C * V / Z. The organization variables are summarized in table 5.1.

Symbol	Variable Dependence	Description
K	Independent	Number of channels in system
L	Independent	Number of ranks per channel
B	Independent	Number of banks per rank
R	Independent	Number of rows per bank
C	Independent	Number of columns per row
V	Independent	Number of bytes per column
Z	Independent	Number of bytes per cacheline
N	Dependent	Number of cachelines per row

TABLE 5.1: Summary of System Configuration Variables

In general, the value of a given system configuration parameter can be any positive integer. For example, a memory system can have 3 channels of memory with 6 ranks of memory per channel. However, for the sake of simplicity, the values of parameters defined for the system under study are assumed to be integer powers of 2, and the lower case letter of the respective parameters are used to denote that power of two. For example, there are 2^b = B banks in each rank, and 2^l = L ranks in each channel of memory. A memory system with the size of K * L * B * R * C * V can then be indexed with k + l + b + r + c + v number of address bits.

5.3.2 Available Parallelism in DRAM System Organization

channel:

Independent channels possess the highest degrees of parallelism in the memory system. There are no restrictions on requests to different channels controlled with independent memory controllers. For a performance optimized design, consecutive cacheline accesses should be mapped to different channels.[*]

rank:

Most DRAM accesses can proceed in parallel in different ranks, subject to the availability of the shared address, command and data busses. However, rank-to-rank switching penalties in high frequency, globally synchronous DRAM memory systems such as DDRx SDRAM memory systems limit the desirability of sending consecutive DRAM requests to different ranks.

bank:

Similar to the case of multiple DRAM accesses to multiple ranks, multiple DRAM accesses can proceed in parallel in different banks of a given rank subject to the availability of the shared address, command and data busses. Scheduling consecutive DRAM read accesses to different banks within a given rank is more efficient than scheduling consecutive read accesses to different ranks since idle cycles are not needed to re-synchronize the data bus. However, consecutive DRAM read and writes are more efficiently performed to different ranks of memory instead of different banks of the same rank. In modern systems, read requests tend to have higher spatial locality than write requests due to the existence of write back caches. The result is that in a high performance design, bank addresses should be mapped lower than rank addresses to favor the extraction of spatial locality from consecutive memory read accesses.

[*]. The exploration of parallelism in the memory system is an attempt to extract maximum performance. For low-power targeted systems, different criteria may be needed to optimize the address mapping scheme.

row:

In a DRAM memory system, only one row per bank can be active in any time period, provided that additional ESDRAM-like row-buffers are not present in the DRAM device. The result of the forced serialization of accesses to different rows of the same bank means that row addresses are typically mapped to higher order memory address ranges to lessen the likelihood that spatially-close consecutive accesses would be made to different rows of the same bank.

column:

In a memory system that implements open-page row-buffer management policy, consecutive cachelines are mapped to the same row of memory, and a memory access sequence that streams through memory would produce memory accesses to adjacent locations of the same DRAM row. As a result, for a memory system that utilize the open-page row-buffer management policy, adjacent columns should be mapped to the low address range. In contrast, for a memory system that utilize the close-page row-buffer management policy, consecutive cachelines should be mapped to different banks, then to different rows of memory. Such a mapping scheme scatters consecutive memory access streams across different banks, different ranks and different channels. The result is that in a memory system that utilize the close-page row-buffer management policy, the low range of the column address that denote the column offset within a cacheline is optimally mapped to the lowest range of the address, but the remainder of the column addresses are best mapped to the high address ranges comparable to the row addresses.

5.3.3 Baseline Address Mapping Schemes

In a DRAM memory system that utilize the open-page row-buffer management policy, consecutive read requests to the same row, bank, rank and channel can be pipelined consecutively, whereas a similar combination of read requests to the same row, bank, rank and channel in a memory system that utilize the close-page row-buffer management policy incurs the same latency penalty as read requests to different rows of the same bank, rank and channel. That is, in a memory system that utilize the close-page row-buffer management policy, there is no difference between accesses to the same row or different rows of the same bank, rank and channel. The difference in the access preferences means that optimal address mapping schemes are different for memory systems that utilize open-page and close-page row-buffer management policies.

In the previous section, the available parallelism of memory channels, ranks, banks, rows and columns were examined in abstraction. In this section, two baseline address mapping schemes are established. In the abstract memory system, the total size of memory is simply K * L * B * R * C * V. The convention adopted in this work is that the colon, ":" is used to denote separation in the address ranges. As a result, k:l:b:r:c:v not only denotes the size of the memory, but also the order of the respective address ranges in the address mapping scheme. Finally, for the sake of simplicity, C * V can be replaced with N * Z in the performance analysis. That is, instead of the number of bytes per column multiplied by the number of columns per row, the number of bytes per cacheline multiplied by the number of cachelines per row can be used equivalently. The size of the memory system is thus K * L * B * R * N * Z, and the address mapping scheme can be denoted by k:l:b:r:n:z.

Open-page Baseline Address Mapping Scheme

In this section, a simple baseline address mapping scheme that is favorable for a memory system that utilize the open-page row-buffer management policy is described. In a system that utilize the open-page row-buffer management policy, consecutive cacheline addresses should be placed into different channels, then adjacent cachelines should be mapped into the same row, same bank, and same rank. The baseline address ordering is thus row, rank, bank, cachelines per row, channel, and cacheline offset. Utilizing the previously described convention, this baseline address mapping scheme for the open-page row-buffer management policy is r:l:b:n:k:z.

Close-page Baseline Address Mapping Scheme

Similar to the baseline address mapping policy for a memory system that utilize the open-page row-buffer management policy, consecutive cacheline addresses should be mapped to different channels in the address mapping policy for a memory system that utilize the close-page row-buffer management system. However, the central belief of the close-page row-buffer management policy is that there is little spatial locality between temporally adjacent memory accesses, so DRAM rows are precharged as soon as possible. In this case, mapping consecutive cacheline addresses to the same bank, same rank and same channel of memory results in a bank conflict and greatly reduces available memory bandwidth. In order to minimize the chances of bank conflict, adjacent lines are mapped to different channels, then to different banks, then to different ranks. The baseline ordering is thus row, cachelines per row, rank, bank, channel and cacheline offset, and denoted as r:n:l:b:k:z.

5.3.4 Parallelism versus Expansion Capability

In modern computing systems, one capability that system designers often must provide to end users is to permit the end users to conFigure the memory capacity of the memory system by adding or removing memory modules. In the context of the discussion of address mapping schemes, adjustable memory expansion capability means that respective channel, row, column, rank and bank address ranges must be flexibly adjusted depending on the configuration of the DRAM modules inserted into the memory system by the end user. In order to minimize the complexity of DRAM memory controllers, memory system organization parameters that can be varied are typically mapped to the highest address range. In this manner, the lower order address bits can remain unchanged regardless of the number, capacity and configuration of the memory modules in the system. As an example, in contemporary desktop personal computer systems, system memory capacity can be adjusted by adding or removing ranks of DRAM devices. In these systems, rank indices are mapped to the highest address range in the DRAM memory system. The result of such a mapping scheme means that an application that utilizes only a subset of the memory address space would typically make use of fewer ranks of memory than is available in the system. The address mapping scheme optimized for expansion capability would thus present less rank parallelism to memory accesses, or in the case where channel indices are mapped to the high address ranges, parallelism presented by multiple channels may not be available to individual applications.

In the respective baseline address mapping schemes described previously, the channel and rank address ranges are mapped to low end of the address range. For a flexible, user configurable memory system, the channel and rank indices may be moved to the highest address ranges. The result is that the k:l:r:b:n:z address mapping scheme would be used in a memory controller with an expandable memory system that implements an open-page row-buffer management policy, and the k:l:r:n:b:z address mapping scheme would be used in a memory controller with an expandable memory system that implements the close-page row-buffer management policy. In these address mapping schemes geared toward memory system expendability, some degrees of channel and rank parallelism are often lost to workloads that use only a subset of the contiguous physical address space.

The loss of parallelism for single threaded workloads in memory systems designed for configuration flexibility is less of a concern for memory systems designed for large multi-processor systems. In such a system, concurrent memory accesses from different memory access streams to different regions of the physical address space would make use of the parallelism offered by multiple channel and multiple ranks. Moreover, the constraint on the limited parallelism available to single threaded workloads can be alleviated in cases where the virtual address mapping mechanism randomizes the address translation from the virtual address space to the physical address space. Finally, some address mapping schemes have been devised to make use of the available parallelism offered by multiple banks and ranks to alleviate the issue of address aliasing in concurrent array accesses.

5.3.5 Bank Address Aliasing (stride collision)

One additional problem in the consideration of an address mapping scheme is the problem of bank address aliasing. Bank address aliasing occurs when workloads access arrays whose respective sizes are powers-of-two concurrently. Concurrent accesses to arrays that are powers-of-two can result in bank conflicts when the arrays are aligned to a given address boundary and the paired accesses are made to different rows of the same bank.

To obtain the highest performance, an address mapping scheme should be devised so that consecutive memory accesses to different DRAM rows are mapped to different banks, different ranks or different channels of memory. The problem in a simple address mapping scheme is that arrays whose sizes are power-of-two may have each element of the respective array coincidentally mapped to different rows of the exact same channel, bank and rank. In such a case, concurrent array accesses would result in bank conflicts for each pair of memory accesses. The task then is to devise a scheme that avoids bank conflicts for concurrent array accesses to arrays aligned on power-of-two address boundaries.

Proposed Solution to Alleviate Address Aliasing

The bank address aliasing problem has been respectively investigated by Lin et. al. [27] and Zhang et. al.[24,25]. In Lin et. al., the proposed solution was applied to a Direct RDRAM memory system with 32 banks per rank, 32 ranks per channel and 4 channels. In this configuration, the rank address and 4 out of 5 bits of the bank address are bitwise XOR'ed with the 9 bit row address. The resulting bank address was then placed in reverse ordering in the address mapping scheme. Lin et. al. illustrated that this configuration effectively rotated the bank and rank mapping so that there is no address boundary where every pair of concurrent array accesses would result in bank conflicts. The address mapping scheme proposed by Lin et. al. is shown as Figure 5.1.

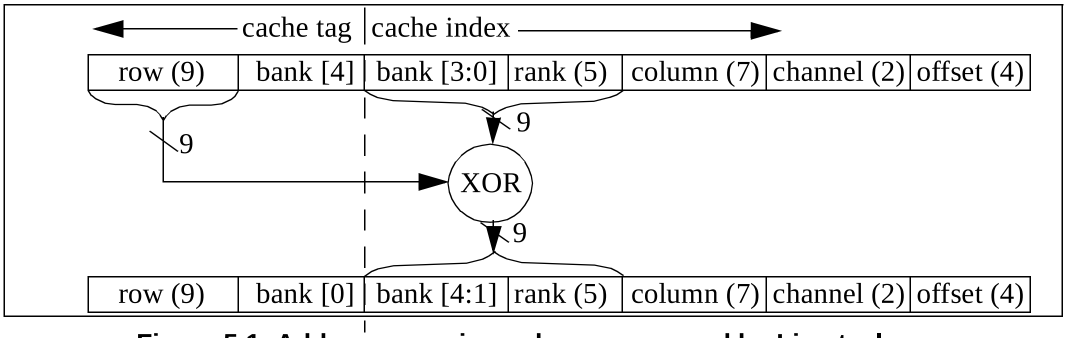


Figure 5.1: Address mapping scheme proposed by Lin et. al.

In the work by Zhang et. al., a similar approach in rotating bank addresses through the use of bitwise XOR function is retained. In their work, Zhang et. al. applied the bank rotation function to a more conventional DRAM memory system, simulating up to 32 banks, and this scheme also showed varying degrees of improvement. The mapping scheme described by Zhang et. al. is shown as Figure 5.2.

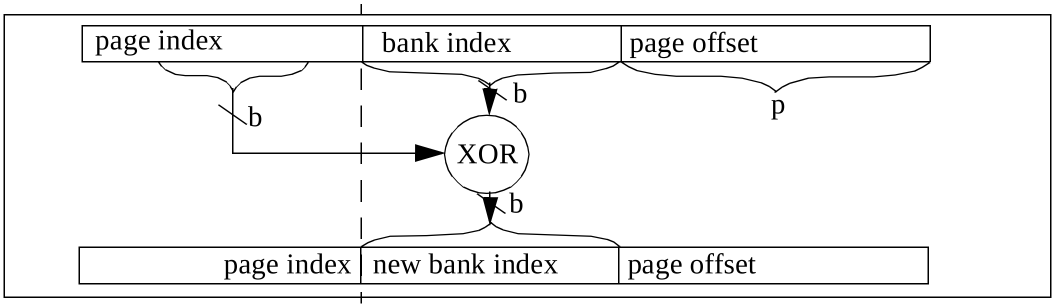


Figure 5.2: Address mapping scheme proposed by Zhang et. al.

Problems with Proposed Solutions

The schemes proposed by Lin et. al. and Zhang et. al. are similar schemes applied to different memory systems. The use of the Direct RDRAM memory system allowed Lin et. al. a higher degree of bank parallelism in the form of 1024 banks DRAM arrays. The generous level of bank parallelism allowed Lin et. al. to create a 1:1 mapping that permutes the available number of banks through the address space in the system configuration examined. In contrast, Zhang et. al. illustrated a more modest memory system where the page index was larger than the bank index. The problem is that there are few banks in contemporary SDRAM and DDR SDRAM variant memory systems, and for a DRAM memory system with 2^b banks, there are only 2^b possible permutations in mapping the physical address to memory address. In implementing the bank address permutation scheme, the address aliasing problem is simply shifted to a larger granularity. That is, without bank permutation, arrays aligned on address boundaries of $2^{(b+p)}$ would cause a bank conflict on every pair of concurrent array accesses. The act of permuting the bank index means that arrays aligned on address boundaries of $2^{(b+p+b)}$ would cause a bank conflict on every pair of concurrent array accesses. Essentially, there are not enough banks to rotate through the address space in a contemporary memory systems to completely avoid the memory address aliasing problem, but the presence of more banks does defer the address aliasing problem to larger arrays aligned on an exact power-of-two address boundaries.

An Address Aliasing Example: STREAM on a Desktop Personal Computer

The STREAM benchmark is designed to measure the maximum bandwidth available on a given computer system[40]. The benchmark contains array access sequences that create bank address aliasing problems in that the STREAM benchmark is specifically designed to march through large arrays whose sizes may be statically defined as powers-of-two number of bytes. As a result, the address boundaries of the arrays used in the benchmark are naturally aligned to cause concurrent read and write streams of the accessed arrays to map to the same DRAM bank. Fortunately, the addition of a simple offset to increase the size of the respective arrays means that the statically allocated arrays are no longer proper powers-of-two in size, and the address boundaries of the respective arrays are not aligned to the same bank addresses.

As an example, the address mapping scheme utilized by the memory controller in Intel's 875P system controller places the bank address range on physical address bits 14 and 15 when 256 Mbit DDR SDRAM devices are used to conFigure the memory system[41]. In this configuration, arrays that are powers-of-two and larger than 2^{16} bytes in size could have all array indices mapped to the same bank if the static arrays are allocated consecutively. The address aliasing problem can be alleviated in this specific system by increasing the size of the respective arrays by 2^{14} bytes. Table 5.2 summarizes the change in measured

	Copy	Scale	Add	Triad
No Offset (MB/s)	2331	2473	2584	2496
With Offset (MB/s)	2448	2474	3164	3157
Difference (MB/s)	117	1	580	661

TABLE 5.2: : Measured STREAM Results: With and Without OFFSET

bandwidth for the STREAM benchmark in a Dell PowerEdge 400SC computer system that uses the Intel 875P system controller. The results show that the insertion of the 2^{14} byte array offset can alleviate bank conflicts on aligned address boundaries and improve effective DRAM bandwidth by as much as 25%.

5.4 Memory Transaction and DRAM Command Ordering Schemes

The design space of a modern DRAM controller is incredibly large. A DRAM controller can be fully protocol compliant for a given DRAM memory system, but implements the most simplistic controller possible to minimize complexity and die size of the memory controller. Alternatively, a highly complex, high performance memory controller can be implemented to extract the maximum performance from a given memory system. The performance characteristic of the DRAM memory controller depends on the DRAM command and memory transaction ordering schemes have been studied by Briggs et. al., Cuppu et. al., Hur et. al., McKee et. al., Lin et. al, and Rixner et. al[27,28,32,33,34,37,42]. In studies performed by Briggs et. al., Cuppu et. al., McKee et. al., Lin et. al., and Rixner et. al, various DRAM-centric scheduling schemes are examined. In the study performed by Hur et. al., the observation is noted that the ideal DRAM scheduling algorithm depends not only only the optimality of scheduling to the DRAM memory system, but also depends on the requirement of the application. In particular, the integration of DRAM memory controllers with the processor core onto the same silicon die means that the processor core can interact directly with the memory controller and provide direct feedback to select the optimal DRAM scheduling algorithm.

The design of a high performance DRAM memory controller is further complicated by the emergence of modern high performance multi-threaded processors and multi-core processors. While the use of multi-threading have been promoted as a way to hide the effects of memory access latency in modern computer systems[44,45], the net effect of multi-threaded and multi-core processors on a DRAM memory system is that the intermixed memory request stream from the multiple threaded contexts to the DRAM memory system disrupts the row-locality of the request pattern and increases bank conflicts[50]. As a result, an optimal DRAM controller design not only have to account for the idiosyncrasies of specific DRAM memory systems, application specific requirements, but also the type and number of processing elements in the system.

The large number of design factors that a design engineer must consider further increases the complexity of a high-performance DRAM memory controller. Fortunately, some basic strategies exist in common for the design of a high performance DRAM memory controller. Specifically, the strategies of bank-centric organization, write caching, seniors-first are common to many high-performance DRAM controllers, and specific adaptive arbitration algorithms are unique specific DRAM controllers.

5.4.1 Write Caching

One strategy deployed in many modern DRAM controllers is the strategy of write caching. The basic idea for write caching is that write requests are typically non-critical in terms of performance, but read requests may be critical. As a result, it is typically desirable to cache write requests and allow read requests to proceed ahead. Furthermore, DRAM devices are typically poorly designed to support back-to-back read and write requests. Figure 5.3 repeats the illustration of a column read command that follows a write command

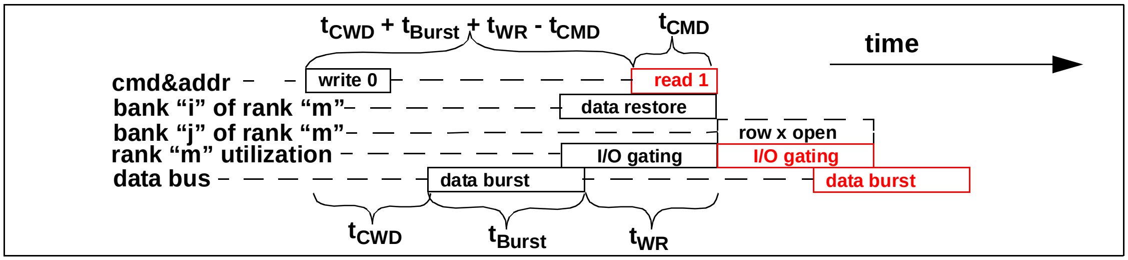


Figure 5.3: Write command following read command to open banks.

and shows that due to the differences in the direction of data flow between read and write commands, significant overheads exist when column read and write commands are pipelined back-to-back. The strategy of write caching allows read requests that may be critical to application performance to proceed ahead of write requests, and the write caching strategy can also reduce read-write overheads when it is combined with a strategy to send multiple write requests to the memory system consecutively. One memory controller that utilizes the write caching strategy is Intel's 870 system controller which can buffer upwards of 8 KB of write data to prioritize read requests over write requests[49].

5.4.2 DRAM-Bank-Centric Request Queuing Organization

In modern DRAM controllers, before a data for a given memory transaction is stored or retrieved from the DRAM devices, the transaction must be translated into a sequence of DRAM commands. To facilitate the pipelined execution of commands in a DRAM memory system, DRAM commands can be placed into a single queue or multiple queues. One organization that can facilitate the pipelined execution of commands in a high performance DRAM memory controller is a set of queues organized on a per bank basis*. In this manner, DRAM commands from different transactions are directed to the same queue in the case that they access the same bank. The per bank organization allows a memory controller to quickly recognize requests that are directed to the same row or different rows of the same bank. In the case that multiple pending requests are directed toward the same row in the same bank, but interleaved with requests directed toward different rows of the same bank, the per-bank queuing mechanism can easily facilitate transaction re-ordering to minimize the number of

*. The bank-centric request queuing construct is a conceptual construct. Memory controllers can utilize a unified queue with sophisticated hardware to perform the transaction re-ordering and bank rotation described in this text, albeit with greater difficulty.

bank conflicts. Moreover, the bank-centric organization also facilitates a bank-rotation mechanism that can process concurrent requests to different banks and increase the utilization of the memory system in between bank conflicts to a given same bank.

Figure 5.4 shows one organization of a set of request queues organized on a per-bank basis. In the organization illustrated in Figure 5.4, memory transaction requests are translated into memory addresses and directed into different request queues based on their respective bank addresses. In this organization, multiple column commands can be issued from a given request queue to a given bank if they are directed to the same open row. In the case that a given request queue has exhausted all pending requests to the same open row and the next ordered request in the queue is addressed to a row, the request queue can then issue a precharge command and allow the next bank to issue commands into the memory system. Figure 5.4 shows that the scheduling priority is passed from bank to bank in a round robin fashion. In the round-robin bank-rotation command scheduling scheme, DRAM bank conflict overhead to a given bank can be hidden by accesses to different banks if there are sufficient numbers of pending requests to other banks that can be processed before the scheduling priority rotates back to the same bank.

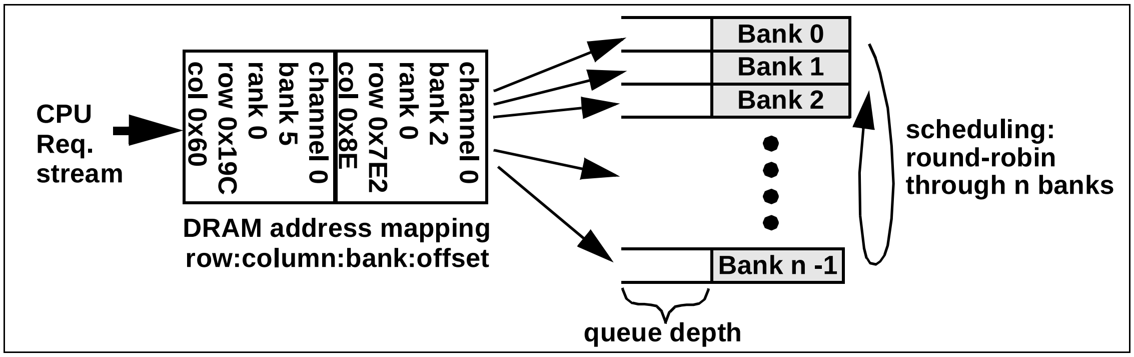


Figure 5.4: Per Bank Organization of DRAM Request Queues.

5.4.3 Feedback Directed Scheduling

In modern computer systems, memory access is performed by the memory controller on behalf of processors or intelligent I/O devices. Memory access requests are typically encapsulated in the form of transaction requests that contains the type, address, and data for the request in the case of write requests. However, in the majority of systems, the transaction requests typically do not contain information that allows a memory controller to prioritize the transactions. Instead, memory controllers typically rely on the type, access history, and memory system state to schedule the memory transactions. In one recent study performed by Hur and Lin, the use of a history based arbiter that selects among different scheduling policies is examined in detail[51]. In this study, the memory access request history is used to select from different arbitration policies dynamically, and speedups between 5 and 60 percent are observed on some benchmarks.

The exploration of a history-based DRAM transaction and command scheduling algorithm is enabled by the fact that the Hur and Lin based the study on a POWER5 processor, a processors with an integrated DRAM controller. As more processors are designed with integrated DRAM memory controllers, these processors can communicate directly with the DRAM memory controllers and schedule DRAM commands based not only on the availability of resources within the DRAM memory system, but also on the DRAM command access history. In particular, as multi-threaded and multi-core processors are integrated with DRAM memory controllers, these DRAM memory controller not only have to be aware of the availability of resources within the DRAM memory system, but they must also be aware of the state and access history of the respective threaded contexts on the processor in order to achieve the highest performance possible.

CHAPTER 6

Performance Analysis Methodology: Request Access Distances:

6.1 Motivation

In recent years, the importance of memory system performance as a limiter of computer system performance has been widely recognized[52,53]. However, DRAM devices specifically and memory systems as a whole are still designed by engineers whose predominant concerns are those of cost minimization and functional correctness. Moreover, the commodity nature of main stream SDRAM, DDR SDRAM and DDR2 SDRAM devices means that DRAM design engineers are reluctant to add functionalities or to restructure DRAM devices in such a way that would increase the die size overhead of these devices. As a result, the topic of memory system performance analysis is important not only to system architects, but it is also needed by DRAM design engineers to evaluate design trade-off points between the die cost of various features against potential performance benefits of those features.

6.1.1 DRAM Device Scaling Considerations

Figure 6.1 shows general DRAM scaling trends from 1998 to 2004. Figure 6.1 shows that while DRAM device datarate have doubled every three years in between 1998 and 2004, row cycle times have decreased by roughly 7% per year during the same period of time. The difference in the scaling trends means that each generation of DRAM devices has a different combination of datarate and row cycle time. Moreover, as each generation of DRAM devices scale in size, the physical organization of the DRAM device directly impacts timing parameters such as t_{RCD}, t_{RAS}, t_{FAW}, t_{RRD} and t_{RFC}. Specifically, as DRAM device density doubles with each generation, DRAM device design engineers can choose to double the number of cells per row, double the number of rows in each bank, or double the number of banks within a given DRAM device.

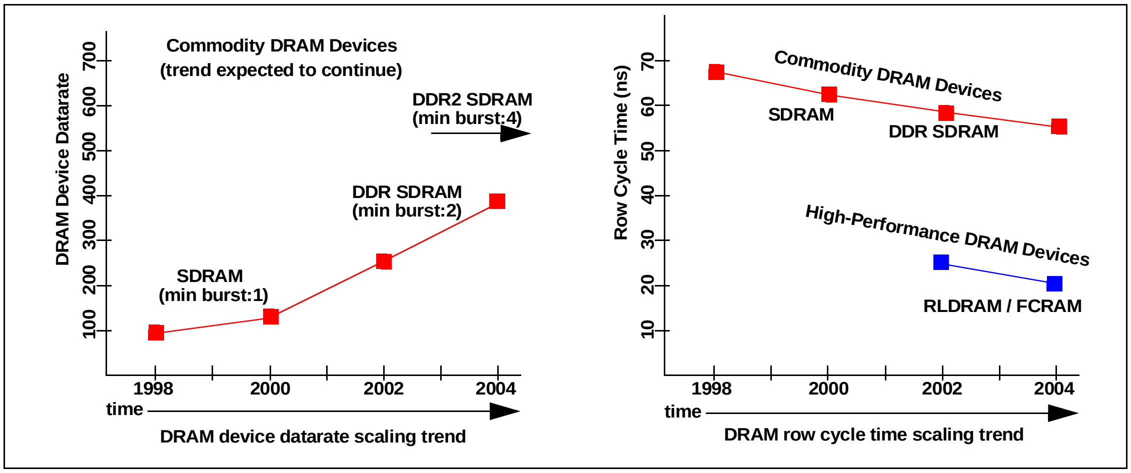


Figure 6.1: DRAM datarate and row cycle time scaling trends.

In the case that the number of storage cells are doubled for each row, the desire to keep the electrical charge of the storage cell at the same level means that the energy consumed for each row access roughly doubles. The increase in power consumption in turn means that t_{FAW}, t_{RRD} and t_{RFC} must be increased when the number of storage cells are doubled. In the case that the number of rows are doubled, the number of storage cells per bitline segment or the number of segments must increase appropriately. The increasing number of DRAM storage cells per bitline then impacts t_{RCD}, t_{RAS} and t_{RC}. Furthermore, the doubling of the number of rows means that twice the number of refresh cycles are needed per unit time, and memory system performance may be further degraded. Finally, the doubling of the number of banks has the smallest impact on DRAM device timing parameters, but the increase in bank count increases the complexity of the control logic, and the larger number of logic transistors increases die size.

The different combinations of device datarates, row cycle times, and device organization impact for each generation of DRAM devices lead to the situation that each generation of DRAM devices must be re-examined in terms of performance characteristics in the context of the larger memory system. Moreover, a suitable analytical framework must be used to examine the performance characteristics so that a wide range of system configurations can be considered while DRAM device timing parameters are varied.

6.1.2 Execution Based Analytical Framework

Two types of analytical frameworks are typically used to evaluate the performance of DRAM memory systems. In general, the two types of analytical frameworks can be described as execution based and trace based analytical frameworks, respectively[1]. Typically, memory system studies are based on closed-loop, execution based simulations[28,32,33,34,37,42,46,51]. The use of execution based simulations means that the performance of the memory system is impacted by the request rate of the processor or

[1] Alternatively, closed-loop and open-loop systems. The open-loop system can be driven by address trace inputs or random number generators. The fundamental concept remains the same.

processors, and the performance of the memory system is tightly coupled to the performance of the processor or processors. For example, one study with a specific set of DRAM device timing parameters, workloads and system configuration can reach one conclusion in regards to the performance sensitivity of specific DRAM timing parameters, while a second study with the same set of DRAM device timing parameters, workloads and system configuration but different processor frequency or cache sizes can reach a different conclusion in regards to performance sensitivity of specific DRAM timing parameters. In this manner, execution based simulation frameworks can accurately measure system performance sensitivity to DRAM device parameters for specific system configurations.

The accuracy of execution based simulations in measuring system performance sensitivity to DRAM device parameters for specific system configurations ironically presents a problem in that the overall system performance depends on both processor performance and memory system performance. However, the individual contributions of processor performance and memory system performance are difficult to separate out from each other. Figure 6.2 abstractly illustrates the point that the idle times in the data bus of the DRAM system are non-linear functions of processor frequency, protocol overhead and DRAM row cycle times. The result is that the performance characteristic of a DRAM memory system is a non-linear function of each individual parameter, even when other parameters are held as constants.

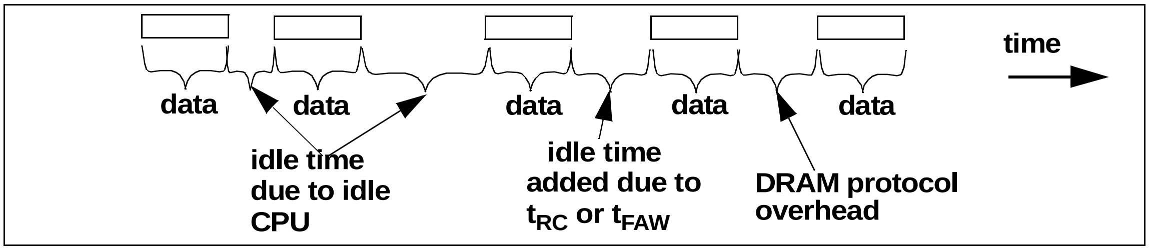

time
data
data
data
data
data
idle time due to idle CPU
idle time added due to t_{RC} or t_{FAW}
DRAM protocol overhead

Figure 6.2: Abstract illustration of DRAM system data bus activity.

The complexity of system level interactions means that while execution based simulation frameworks are highly accurate in reflecting the sensitivity of system level performance to DRAM system configurations and timing parameters, the intermixing of processor performance in the equation means that a different framework that can separate out processor performance from DRAM memory system performance is required to analyze DRAM memory system performance in isolation.

6.1.3 Trace Based Analytical Framework

Trace based analytical frameworks differ from execution based analytical frameworks in that trace-based analytical frameworks are open-loop systems, and memory system performance can be separated from processor performance. In general, trace based analytical frameworks are less suitable for use in the analysis of system level performance characteristics. However, the use of an open-loop trace-based analytical framework means that the input request rate can be independently controlled. In the extreme case, a trace-based analytical framework can assume an infinitely fast processor, and memory requests can be issued into the memory system at saturation rates. In the case that the trace input is driven at saturation rates, all idle times in the DRAM memory system attributable to the processor are eliminated. In this manner, a trace based analytical framework can measure the limits of performance sensitivity to individual DRAM system configuration and timing parameters, assuming ideal processors.

6.1.4 Trace Based versus Execution Based Analytical Framework

The trace based methodology is deficient in some ways while it is advantageous in other ways when it is compared to an execution driven analytical framework. The trace based analytical framework is deficient in that memory address traces do not contain information in regards to dependencies in the memory request stream. The result is that even inherently dependent memory references can be collapsed entirely and the analytical framework can compute a higher bandwidth efficiency than the theoretical bandwidth efficiency of the workload running on an infinitely fast processor. In this manner, a trace based analytical framework computes a bandwidth efficiency that represents the upper-bound of bandwidth efficiency for any given single threaded workload, and the amount of deviation between the efficiency computed by a trace based analytical framework and the efficiency obtained from an execution based simulation is dependent on the number of dependent memory requests in a given workload relative to the total number of memory requests in the workload.

The issue of trace based methodologies not respecting the dependency of memory references can be resolved by using an execution based methodology that uses highly accurate models of the processors and the memory system. However, an execution based methodology is also problematic in that the processor state machine is dramatically more complex than the memory system statemachine, and the vast majority of the simulation cycles are used for processor state simulation. Moreover, that problem is exacerbated when the goal of the simulation is to examine fundamental limitations of the DRAM memory system. In the case that an infinitely fast processor is approximated by using an extreme ratio of processor to memory system operating frequency, an execution based methodology would be many orders of magnitude slower than a trace based methodology. In this work, a trace based methodology is deployed so that a large design space of DRAM memory system configuration and timing characteristics can be examined in detail. The trade off is the loss of accuracy in that memory dependency in single threaded workloads is not respected.

6.2 The Request Access Distance Framework

In this study, a trace-based analytical framework is used to evaluate the impact of DRAM timing parameters and memory system configuration to memory system performance. The analytical framework is based on the computation of memory system bandwidth efficiency subjected to different combinations of configuration and timing parameters including, but not limited to, DRAM device datarates, burst lengths, t_{RC}, t_{FAW}, and t_{DQS}. The maximum bandwidth efficiency of the DRAM memory system can be computed by separately computing, then adding up the DRAM protocol overhead and DRAM row cycle time constraints for each request. The trace-based analytical framework described in this work is referred to as the ***Request Access Distance*** methodology for the computation of DRAM memory system efficiency.

The process of computing DRAM memory system bandwidth efficiency begins with a re-examination of memory system activity once the memory requests are driven at saturation rates. Figure 6.3 illustrates that once DRAM memory system idle times due to non-memory-intensive processor or processors are removed, all remaining idle times in the DRAM memory system must be directly contributed by the overhead of the DRAM protocol, DRAM row cycle times, or DRAM power constraints such as t_{FAW} or t_{RRD}.

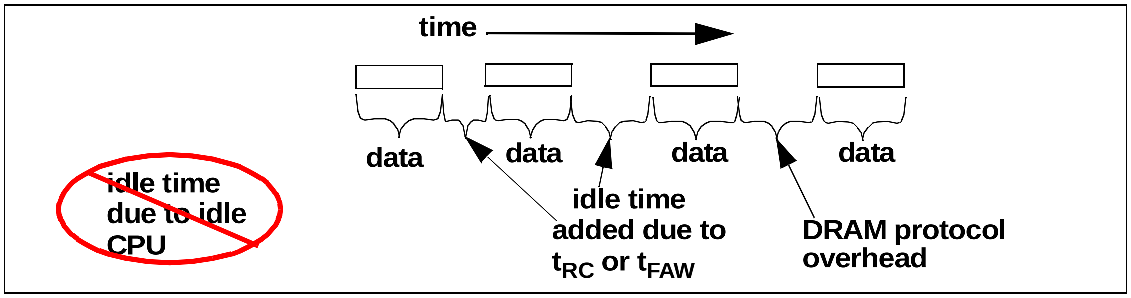

time
data
data
data
data
idle time due to idle CPU
idle time added due to t_{RC} or t_{FAW}
DRAM protocol overhead

Figure 6.3: Abstract illustration of DRAM system data bus activity.

6.2.1 Computing DRAM Protocol Overhead

One source of DRAM bandwidth inefficiency is attributable to the protocol inefficiencies. In this regard, the previous work that formalized the description of the abstract DRAM access protocol now enables the computation of DRAM protocol overhead for any pair of memory references. Table 6.1 contains the entries of minimum DRAM command scheduling distances converted to request access distance overhead. Essentially, the DRAM protocol overhead between any pair of memory references can be computed by subtracting out the data burst duration, t_{Burst}, from the minimum scheduling distance, and the protocol overhead is converted to units of t_{Burst} by dividing through by t_{Burst}. In this work, the protocol overhead between a memory request j and the request that immediately precedes it is denoted by $D_o(j)$.

previous	next	channel	rank	bank	Minimum scheduling distance between commands Open-page No command re-ordering Best case	DRAM protocol overhead between DRAM command pairs. (unit: t_{Burst})
R	R	s	s	s	t_{Burst}	0
R	R	s	s	d	t_{Burst}	0
R	R	s	d	-	$t_{DQS} + t_{Burst}$	t_{DQS} / t_{Burst}
R	W	s	s	s	$t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$	$(t_{CAS} + t_{DQS} - t_{CWD}) / t_{Burst}$
R	W	s	s	d	$t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$	$(t_{CAS} + t_{DQS} - t_{CWD}) / t_{Burst}$
R	W	s	d	-	$t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$	$(t_{CAS} + t_{DQS} - t_{CWD}) / t_{Burst}$
W	R	s	s	s	$t_{CWD} + t_{Burst} + t_{WR} - t_{CMD}$	$(t_{CWD} + t_{WR} - t_{CMD}) / t_{Burst}$
W	R	s	s	d	$t_{CWD} + t_{Burst} + t_{WR} - t_{CMD}$	$(t_{CWD} + t_{WR} - t_{CMD}) / t_{Burst}$
W	R	s	d	-	$t_{CWD} + t_{Burst} + t_{DQS} - t_{CAS}$	$(t_{CWD} + t_{DQS} - t_{CAS}) / t_{Burst}$
W	W	s	s	s	t_{Burst}	0
W	W	s	s	d	t_{Burst}	0
W	W	s	d	-	t_{Burst}	0

Legend

R = read; W = write;
s = same; d = different;
o = open; c = conflict;

TABLE 6.1: **Table of Request Access Distance Overhead**

6.2.2 Computing Row Cycle Time Constraints

In this work, a request j is defined to have a request access distance, $D_r(j)$, to a prior request made to a different rows of the same bank as request j. The request access distance $D_r(j)$ describes the timing distance between it and the previous request to a different row of the same bank. In the case that two request are made to different rows of the same bank and the minimum constraints for row cycle times are not met, some idle time must be inserted into the command and data busses of the DRAM memory system. The minimum request access distance D_m is the number of requests that must be made to an open row of a given bank, to different banks, or to different ranks, between two requests to the same bank that requires a row cycle for that bank. The minimum access distance ensures that minimum DRAM row cycle time requirements are satisfied for all requests in a request stream. In the case that $D_r(j)$ is less than D_m, some amount of idle time, $D_i(j)$, must be added so that the total access distance for request j, $D_t(j)$, is greater than or equal to D_m. The basic unit for the minimum access distance statistic is the data bus utilization time for a single transaction request, t_{Burst} time period. In a close-page memory system with row cycle time of t_{RC} and access burst duration of t_{Burst}, D_m is simply $(t_{RC} - t_{Burst}) / t_{Burst}$.

The request access distance for request j in a close-page memory system is simply defined as $D_r(j)$. However, two different request distances are defined for each request j in an open-page memory system: $D_{r-ff}(j)$ and $D_{r-lf}(j)$. $D_{r-ff}(j)$ and $D_{r-lf}(j)$ are need for request j if and only if request j is the first column access of a given row access. If request j is not the first column access of a row access to a given bank, then the respective row activation and precharge time constraints do not apply, and $D_{r-ff}(j)$ and $D_{r-lf}(j)$ are not needed. The request access distance $D_{r-ff}(j)$ is the request access distance between request j and the first column access of the previous row access to the same bank and the request access distance $D_{r-lf}(j)$ is the request access distance between request j and the last column access of the previous row access to the same bank.

In an open-page memory system, a row is kept active at the sense amplifiers once it is activated so subsequent column accesses to the same row can be issued without additional row cycles. In case of a bank conflict in an open-page memory system between two requests to different rows of the same bank, the second request may not need to wait for the entire row cycle time before it can be issued. Figure 6.4 shows that in the best case, bank conflicts between two different column access requests can be scheduled with the timing of t_{Burst} + t_{RP} + t_{RCD} if the row restoration time t_{RAS} has already been satisfied for the previous row access. In the best case scenario, the minimum scheduling distance between two column commands in an open-page system to different rows of the same bank is $(t_{RP} + t_{RCD}) / t_{Burst}$. The best case scenario illustrated in Figure 6.4 shows that D_m is by itself insufficient to describe the minimum access distances for an open-page system. In this work, two different access distance distances, D_{m-ff} and D_{m-lf}, are separately defined for open-page memory systems to represent the worst case and best case timing between column accesses to different rows of the same bank, respectively. The variable D_{m-ff} denotes the minimum request access distance between the first column access of a row access and the first column

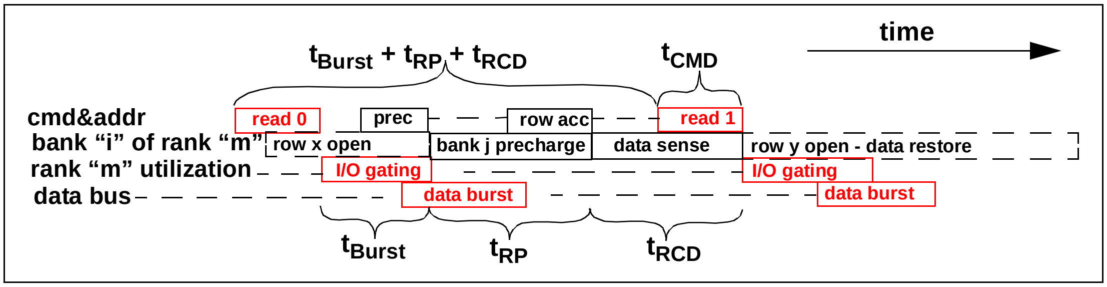


Figure 6.4: Consecutive Read Commands to Same Bank: Bank Conflict.

access to the previous row access of the same bank. The variable D_{m-lf} denotes the minimum request access distance between the last column access to a row and the first column access to the previous row access of the same bank. In a close-page memory system, D_{m-ff} is same as D_{m-lf} and D_{m-ff} can be simplified as D_m. In an open-page memory system, multiple column access commands can be issued to the same row while that row is active, and both D_{m-ff} and D_{m-lf} are needed to account for the different timing conditions that exist between different column accesses of different rows in an open-page memory system.

The total access distance for request j in a close-page memory system is defined as $D_t(j)$. Since two sets of request access distances are needed to ensure that minimum row cycle timings are met in a DRAM memory system, two different total access distances are also needed for each request in an open page memory system. The two total access distances are $D_{t-ff}(j)$ and $D_{t-lf}(j)$. The total request distances $D_{t-ff}(j)$ and $D_{t-lf}(j)$ must be greater than D_{m-ff} and D_{m-lf} for every request j that is the first column access of a row access to a given bank, respectively. In cases where either $D_{r-ff}(j)$ is less than D_{m-ff} or $D_{r-lf}(j)$ is less than D_{m-lf}, additional idling distance must be added. The required idling distance $D_{i-ff}(j)$ is needed to satisfy the minimum request access distance of D_{m-ff} and $D_{i-lf}(j)$ is needed to satisfy the minimum request access distance of D_{m-lf}. In a close-page memory system, $D_{i-ff}(j)$ is by itself sufficient to account for the minimum idling distance needed by request j, and $D_i(j)$ equals $D_{i-ff}(j)$. In an open-page memory system, $D_i(j)$ is equal to the larger value of $D_{i-ff}(j)$ and $D_{i-lf}(j)$ for request j that is the first column access of a given row. In the case that a given request j is not the first column access of a given row, $D_i(j)$ is zero.

The request access distance statistic is a first order model that examines DRAM performance characteristics with varying ratios of t_{RC} and device data rate. The use of t_{RP}

and t_{RCD} in the formal definition introduces additional variables into the first order model. For the purposes of simplicity, $(t_{RC} - t_{Burst}) / 2$ can be used as an approximation for $t_{RP} + t_{RCD}$. The various request access distance definitions are summarized in table 6.2.

	Notation	Description	Formula
	$D_o(j)$	DRAM Protocol overhead for request j	table 6.1
close-page system	D_m	Minimum access distance required for each request j	$(t_{RC} - t_{Burst}) / t_{Burst}$
	$D_r(j)$	Access distance for request j	-
	$D_i(j)$	Idling distance needed for request j to satisfy t_{RC}	Figure 6.5
	$D_t(j)$	Total access distance for request j	-
open-page system	$D_{m\text{-}ff}$	Minimum distance needed between first column commands of different row accesses	$(t_{RC} - t_{Burst}) / t_{Burst}$
	$D_{m\text{-}lf}$	Between last column and first column of different rows	$D_{m\text{-}lf} = (t_{RP} + t_{RCD}) / t_{Burst} \sim= (t_{RC} - t_{Burst}) / (2 * t_{Burst})$
	$D_{r\text{-}ff}(j)$	Access distance for request j to first column of last row	-
	$D_{r\text{-}lf}(j)$	Access distance for request j to last column of last row	-
	$D_{i\text{-}ff}(j)$	Idling distance needed by request j to satisfy t_{RC}	Figure 6.5
	$D_{i\text{-}ff}(j)$	Idling distance needed by request j to satisfy $t_{RP}+ t_{RCD}$	Figure 6.5
	$D_i(j)$	Idling distance needed by request j that is first column access of a row access	$D_i(j) = \max (D_{i\text{-}ff}(j), D_{i\text{-}lf}(j))$
	$D_i(j)$	Idling distance needed by request j that is not the first column access of a row access	0
	$D_{t\text{-}ff}(j)$	Total access distance for request j to satisfy t_{RC}	-
	$D_{t\text{-}ff}(j)$	Total access distance for request j to satisfy $t_{RP}+ t_{RCD}$	-

TABLE 6.2: Request Access Distance Terminologies Defined

The key element in the Request Access Distance statistic for the computation of DRAM memory system bandwidth efficiency is the set of formulas used to compute the necessary idling distances for each request in a request stream. The fundamental insight that enables the creation of the Request Access Distance statistic is that idling distances added for $D_r(j)$ requests previous to request j must be counted toward the total access distance needed by request j since these previous idling distances increases the effective access distance of request j. The formula for computing the additional idling distances needed by request j for both open-page and close-page memory systems are illustrated in Figure 6.5.

close-page system

$$\left\{ D_i(j) = MAX\left(D_o(j), D_m - \left(D_r(j) + \sum_{n = \lfloor j - D_r(j) \rfloor}^{j-1} D_i(n) \right) \right) \right.$$

open-page system

$$\left\{ \begin{array}{l} D_{i-ff}(j) = MAX\left(D_o(j), D_{m-ff} - \left(D_{r-ff}(j) + \sum_{n = \lfloor j - (D_{r-ff}(j)) \rfloor}^{j-1} D_i(n) \right) \right) \\ D_{i-lf}(j) = MAX\left(D_o(j), D_{m-lf} - \left(D_{r-lf}(j) + \sum_{n = \lfloor j - D_{r-lf}(j) \rfloor}^{j-1} D_i(n) \right) \right) \end{array} \right.$$

Figure 6.5: Definition of Idling Distances for Request j.

The request access distance analysis computes the idling distances that are needed for every request in a request stream to satisfy. The sum of the idling distances for the entire sequence may be used to compute the bandwidth efficiency for a specific system configuration and a given workload. The addition of the idling distance for request j enables the total request distances $D_{t\text{-}lf}(j)$ and $D_{t\text{-}ff}(j)$ to satisfy $D_{m\text{-}lf}$ and $D_{m\text{-}ff}$, respectively.

6.2.3 Computing t_{FAW} Constraints

In DDR2 and future generations of DRAM devices, operational restrictions on the DRAM device are under active consideration by DRAM design engineers to ensure that a given DRAM device does not exceed the defined limit for instantaneous power draw. The restriction on peak power draw of DDRx DRAM devices is implemented in terms of the number of banks that can be activated in a given period of time. Specifically, the timing parameter t_{FAW} has been defined as the rolling window of time within which no more than 4 banks can be activated. In a close-page memory system, the rolling four bank activation window equates to a rolling timing window within which a maximum of 4 column commands can be issued. In an open-page memory system, the rolling four bank activation window equates to a rolling timing window within which a maximum of four column accesses that are the first column accesses of a row access can be issued.

The t_{FAW} banking restriction can be incorporated into the Request Access Distance statistic by limiting the number of column accesses that are the first request accesses for a row access. The process of incorporating the t_{FAW} banking restriction into the Request Access Distance statistic begins by computing the number of row activations allowed in a rolling t_{RC} window. The equivalent number of banks that can be active in a rolling t_{RC} window is denoted as A_{max} in this study, and it can be obtained by taking the 4 banks that may be active in a rolling t_{FAW} window, multiplying by t_{RC} and dividing through by t_{FAW}. The formula for computing A_{max} is shown in Figure 6.6.

$$\textbf{Maximum Row Activation (per rank, per } \mathbf{t_{RC}}\textbf{) : } \mathbf{A_{max}} \textbf{ = } 4 \times \frac{t_{RC}}{t_{FAW}}$$

Figure 6.6: Formula for Maximum Number of Bank Activations per t_{RC} window.

The maximum number of row activations per rolling t_{RC} window further constrains the efficiency of a DRAM memory system in that there can at best be A_{max} number of column accesses that are the first column accesses of each respective row activation in any rolling t_{RC} time frame. In case that there are more than A_{max} number of column accesses that are the first column accesses of each respective row activation in any rolling t_{RC} time frame, additional idling distances needs to be added to the idling distance of request j. The additional idling distances needed to satisfy the t_{FAW} constraint is denoted as $D_{i\text{-}xtra}(j)$.

The computation of $D_{i\text{-}xtra}(j)$ requires the definition of a new variable, $D_{iv}(j,m)$, where m is the rank id of request j, and $D_{iv}(j,m)$ represents the idling value of request j. The basic idea of the idling value of a given request is that in a multi-rank memory system, requests made to different ranks means that a given rank can idle for that period of time. As a result, a request j made to rank m means that request j has an idling value of 1 to all ranks other than rank m, and it has an idling value of 0 to rank m. Figure 6.7 illustrates the formula for the computation of additional idling distances required to satisfy the t_{FAW} constraint.

$$D_{i-xtra}(j) = MAX\left(0, D_{m-ff} - \left(A_{max} - 1 + \sum_{n = \lfloor j - (A_{max} - 1) \rfloor}^{j} D_i(n) + D_{iv}(n, m)\right)\right)$$

$D_{iv}(j, m) = 0$ **for request n that is the first column access of a row activation to a bank that is in the same rank as request j.**

$D_{iv}(j, m) = 1$ **for request n that is not the first column access of a row activation or if request n is not made to the same rank as request j.**

Figure 6.7: Formula for Additional Idling Distance $D_{i\text{-}xtra}$ (j) for Request j.

Finally, Figure 6.8 shows the formula for $D_{i\text{-}total}(j)$, for the total number of idling distance for request j that is the first column accesses of a row activation. The process of computing bandwidth efficiency in a DRAM memory system constrained by t_{FAW}, is then as simple as replacing $D_i(j)$ with $D_{i\text{-}total}(j)$ in the formulas for computation of additional idling distances in Figure 6.5.

$$D_{i-total}(j) = D_i(j) + D_{i-xtra}(j)$$

Figure 6.8: Definition of Idling Distance $D_{i\text{-}total}$(j) for Request j - Constrained by t_{FAW}.

6.2.4 DRAM Memory System Bandwidth Efficiency Computation

Finally, Figure 6.9 illustrates that the bandwidth utilization efficiency of a DRAM memory system can be obtained by dividing the number of requests in the request stream by the sum of the number of requests in the request stream and the total number of idling distances needed by the stream to satisfy the DRAM protocol overhead, the row cycle time constraints and the t_{FAW} bank activation constraint needed by the requests in the request stream.

$$Efficiency = \frac{r}{r + \sum_{n=1}^{r} D_{i-total}(n)}$$

r = number of requests in request stream

Figure 6.9: Bandwidth Efficiency of Request Stream.

6.2.5 System Configuration

The Request Access Distance methodology can be used to compute the bandwidth efficiency of a given memory system. However, the bandwidth efficiency computed with the Request Access Distance mythology is workload specific and sensitive to the organization of the DRAM memory system. Figure 6.10 shows one system organization of a close-page

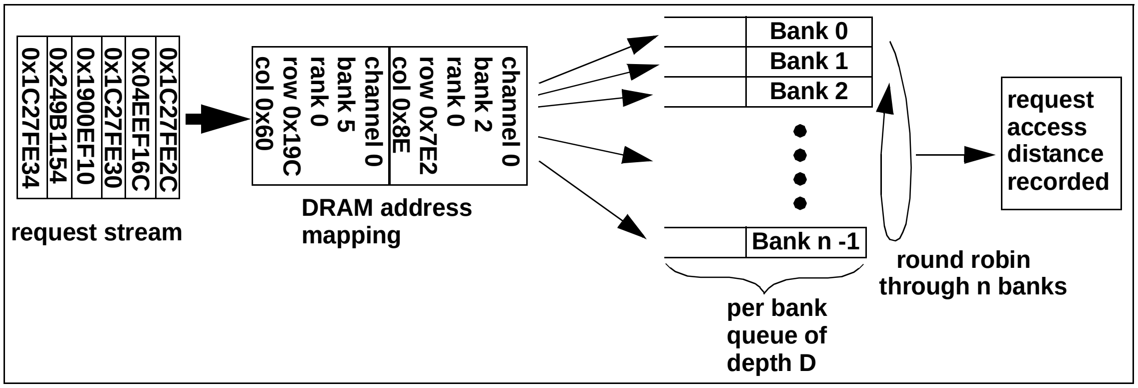


Figure 6.10: Request Access Distance in a close-page system with per-bank queues and round robin bank rotation.

memory system where the request stream is subjected to an address mapping policy and translated into DRAM channel, rank, bank, row and column addresses. The request stream is then sent to each DRAM bank, and each DRAM bank simply records the number in between each pair of accesses to that bank.

The request access distance statistic computes the number of idle distances that must be inserted into a given request stream in order to meet all of the specified constraints. However, a well designed DRAM memory system can substantially increase maximum bandwidth efficiency by re-ordering the request stream to minimize the number of bank conflicts. In Figure 6.10, requests from the request stream are placed into n separate queues, one queue per bank in the memory system. Each queue has the depth of D, and the memory system rotates through the n banks in a round robin fashion. In an idealized memory system with infinitely deep queues, each queue will have some requests in queue waiting for access to a given bank in the memory system. In the idealized case, the request access distance for all requests will be n - 1, since the distance between accesses to any bank is n - 1. However, in the case where the depth of the queues is shallow, or if the bank address distribution of the access sequence is not sufficiently random, then at a given instance in time some queues may be empty while other queues are filled to the maximum depth. In such a case, many idle scheduling slots may need to be inserted to meet minimum DRAM row cycle time requirements. Finally, In the extreme case where the depth D of the queue is 0, the memory system does not re-order the request stream, and the memory references are executed in strict order.

6.3 Impact of Refresh

The Request Access Distance methodology for the computation of maximum DRAM memory system bandwidth efficiency described in this chapter have thus far omitted any mention of the impact of DRAM refresh on the bandwidth efficiency. The reason for the omission is that the impact of refresh is somewhat difficult to formalize in the same manner as the Request Access Distance methodology. In general, DRAM refresh commands are injected into the memory system once for each row in the memory system every 64 milliseconds, and each refresh command refreshes one row in all banks concurrently. The refresh action takes t_{RFC} time, and the overhead of stopping and restarting the DRAM command pipeline is one row cycle time.

In a close-page memory system, the impact of refresh is relatively easy to compute. The impact of refresh commands can be presented as a bandwidth overhead of fixed percentage value. The percentage value can be computed by multiplying the number of rows per bank by the sum of t_{RFC} and t_{RC}, then dividing through by the overall refresh cycle time period, typically 64 milliseconds.

In an open-page memory system, the impact of refresh can be approximated by the same method described for the close-page memory system. However, the subtle difference between the impact of a refresh command in an open-page memory system and the impact of a refresh command in a close-page memory system is that the refresh command not only presents itself as a bandwidth overhead, but it also closes all open rows in all banks. In this manner, the impact of a refresh command in an open page memory system presents a second order impact on DRAM memory system performance that is not captured by the simple overhead computation. However, since the cost of re-opening a row is relatively less than the refresh overhead of t_{RFC} + t_{RC} per refresh command, the impact of the second order effect may be ignored in the overall performance computation. Moreover, in cases where DRAM memory system performance characteristics are directly compared to each other using the same set of workloads, the impact of the refresh overhead may be factored out entirely, since refresh would have identical impact on systems with the same workload and the same number of rows per bank. As a result, it is believed that the impact of refresh can be ignored altogether in cases where systems of identical configurations are compared directly to each other, and in cases where absolute values are desired, the impact of refresh can be factored in by computing the overhead as a fixed percentage, using the formula describe above.

6.4 Applied Examples

Two simplified examples are shown in this section to illustrate how the Request Access Distance analytical framework can be applied to a sample request stream to compute the maximum bandwidth efficiency for a workload.

6.4.1 Close-Page System Example

Figure 6.11 shows how maximum bandwidth efficiency can be computed for a request stream in a close-page memory system. For the purpose of simplifying the example, Figure 6.11 assumes an idealized DRAM access protocol where the DRAM protocol overhead is zero for all requests, the t_{FAW} bank activation constraint does not apply, and only the row cycle time determines the bandwidth efficiency of the request stream. In Figure 6.11, the request stream has been simplified down to the sequence of bank ID’s, The access distances for each request are then computed from the sequence of bank ID’s. The example illustrated as Figure 6.11 specifies that a minimum of 8 requests need to be active at any given instance in time in order to achieve full bandwidth utilization. In terms of access distances, there must be 7 accesses to different banks in between memory accesses to a given bank. At the beginning of the sequence in Figure 6.11, two requests are made to bank 0 with only 4 other

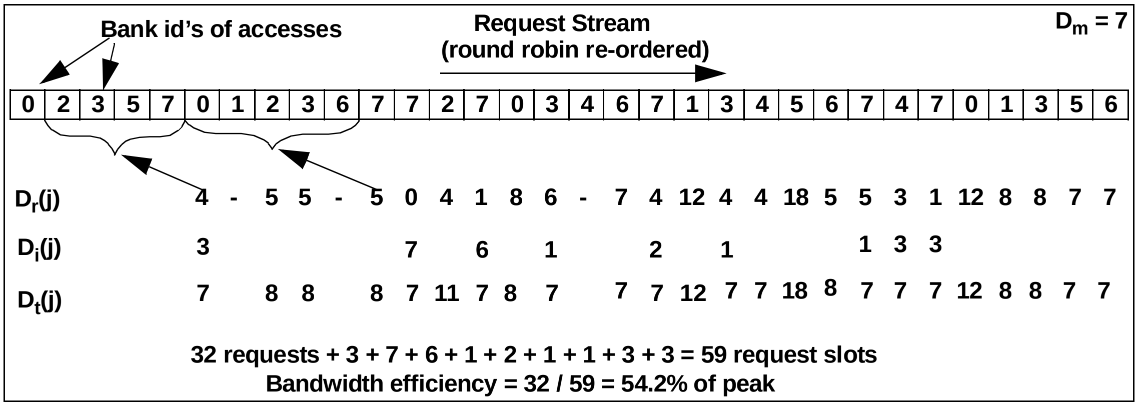
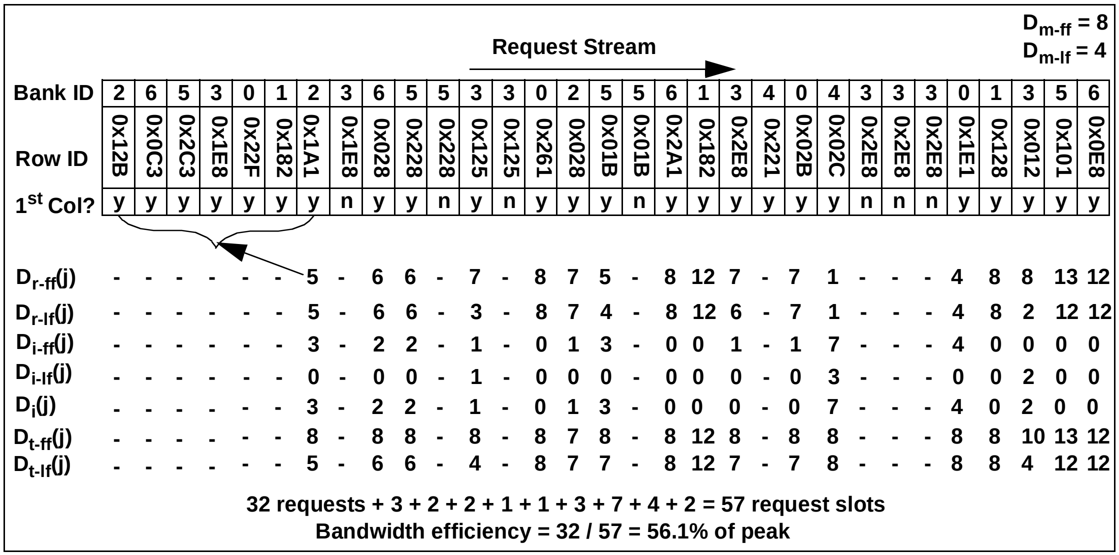

Bank id’s of accesses
Request Stream
(round robin re-ordered)
$D_m = 7$
0 2 3 5 7 0 1 2 3 6 7 7 2 7 0 3 4 6 7 1 3 4 5 6 7 4 7 0 1 3 5 6
$D_r(j)$ 4 - 5 5 - 5 0 4 1 8 6 - 7 4 12 4 4 18 5 5 3 1 12 8 8 7 7
$D_i(j)$ 3 7 6 1 2 1 1 3 3
$D_t(j)$ 7 8 8 8 7 11 7 8 7 7 7 12 7 7 18 8 7 7 7 12 8 8 7 7
32 requests + 3 + 7 + 6 + 1 + 2 + 1 + 1 + 3 + 3 = 59 request slots
Bandwidth efficiency = 32 / 59 = 54.2% of peak

Figure 6.11: Efficiency Computation Example: Close-Page D_m = 7.

requests to different banks in between them. As a result, an idling distance of 3 must be added to the access sequence before the second request to bank 0 can proceed. Two requests later, the request to bank 2 has an access distance of 5. However, idling distances added to requests in between accesses to bank 2 also counts toward its effective total access distance. The result is that the total access distance for the second access to bank 2 is 8, and no additional idling distances ahead of the access to bank 2 are needed. Finally, after all idling distances have been computed, the maximum bandwidth efficiency of the access sequence can be computed by dividing the total number of requests by the sum of the total number of requests and all of the idling distances. In the example shown as Figure 6.11, the maximum sustained bandwidth efficiency is 54.2%.

6.4.2 Open-Page System Example

In this section, an example is used to illustrate the process for obtaining maximum sustainable bandwidth for different DRAM row cycle times and device data rates in an open-page memory system. Figure 6.12 shows a request stream that has been simplified


Request Stream
$D_{m-ff} = 8$
$D_{m-lf} = 4$

Bank ID	2	6	5	3	0	1	2	3	6	5	5	3	3	0	2	5	5	6	1	3	4	0	4	3	3	3	0	1	3	5	6
Row ID	0x12B	0x0C3	0x2C3	0x1E8	0x22F	0x182	0x1A1	0x1E8	0x028	0x228	0x228	0x125	0x125	0x261	0x028	0x01B	0x01B	0x2A1	0x182	0x2E8	0x221	0x02B	0x02C	0x2E8	0x2E8	0x2E8	0x1E1	0x128	0x012	0x101	0x0E8
1st Col?	y	y	y	y	y	y	y	n	y	y	n	y	n	y	y	y	n	y	y	y	y	y	y	n	n	n	y	y	y	y	y
$D_{r-ff}(j)$	-	-	-	-	-	-	5	-	6	6	-	7	-	8	7	5	-	8	12	7	-	7	1	-	-	-	4	8	8	13	12
$D_{r-lf}(j)$	-	-	-	-	-	-	5	-	6	6	-	3	-	8	7	4	-	8	12	6	-	7	1	-	-	-	4	8	2	12	12
$D_{i-ff}(j)$	-	-	-	-	-	-	3	-	2	2	-	1	-	0	1	3	-	0	0	1	-	1	7	-	-	-	4	0	0	0	0
$D_{i-lf}(j)$	-	-	-	-	-	-	0	-	0	0	-	1	-	0	0	0	-	0	0	0	-	0	3	-	-	-	0	0	2	0	0
$D_i(j)$	-	-	-	-	-	-	3	-	2	2	-	1	-	0	1	3	-	0	0	0	-	0	7	-	-	-	4	0	2	0	0
$D_{t-ff}(j)$	-	-	-	-	-	-	8	-	8	8	-	8	-	8	7	8	-	8	12	8	-	8	8	-	-	-	8	8	10	13	12
$D_{t-lf}(j)$	-	-	-	-	-	-	5	-	6	6	-	4	-	8	7	7	-	8	12	7	-	7	8	-	-	-	8	8	4	12	12

32 requests + 3 + 2 + 2 + 1 + 1 + 3 + 7 + 4 + 2 = 57 request slots
Bandwidth efficiency = 32 / 57 = 56.1% of peak

Figure 6.12: Efficiency Computation Example: open-page, $D_{m-ff} = 8$, $D_{m-lf} = 4$.

down to a sequence of the bank ID's and row ID's of the individual requests, and the access distances are then computed from the sequence of bank ID's and row ID's. The example illustrated as Figure 6.12 specifies that a minimum of 9 requests need to be active at any given instance in time to achieve full bandwidth utilization, and there must be 8 requests between row activations as well as 4 requests between bank conflicts. Figure 6.12 shows that $D_{i\text{-}ff}(j)$ and $D_{i\text{-}lf}(j)$ are separately computed but the idling distance $D_i(j)$ is simply the maximum of $D_{i\text{-}ff}(j)$ and $D_{i\text{-}lf}(j)$. After all idling distances have been computed, the maximum bandwidth efficiency of the request sequence can be computed by dividing the total number of requests by the sum of the total number of requests and all of the idling distances. In the example shown as Figure 6.12, the maximum sustained bandwidth efficiency is 56.1%.

CHAPTER 7

DRAM Memory System Performance Analysis: Results

7.1 Introduction

In this chapter, the trace based ***Request Access Distance*** analytical framework is used to examine the performance sensitivity of DRAM memory systems to different device and system organizations, timing parameters, and workloads. To demonstrate the utility and flexibility of the Request Access Distance analytical framework, systems with differing organizations and timing parameters are used to study the impact of different row cycle times, device datarates, data burst lengths, t_{FAW} power constraints, t_{DQS} rank-to-rank data bus switching time, the number of banks and the number of ranks in the memory system.

The Request Access Distance analytical framework formalizes the methodology for the computation of maximum sustainable DRAM memory system bandwidth, subjected to different configurations, timing parameters, and address traces. However, the use of the Request Access Distance analytical framework does not reduce the complexity of analysis in that the analytical framework does not reduce the number of independent variables that impact the performance of a DRAM memory system, it simply identifies them. To reduce the complexity of the task of analyzing the performance of DRAM memory systems, two sets of studies are performed with slightly differing system assumptions in this work to examine varying aspects of DRAM memory system performance sensitivity to different configurations and timing parameters.

7.1.1 Workloads

The performance characteristics of DRAM memory systems depend on workload-specific characteristics of access rates and access patterns. In the Request Access Distance analytical framework, input traces are driven at saturation rates so that the effects of processor performance can be factored out from memory system performance. Despite the fact that the workload traces are driven at saturation rate of the respective memory systems, the workload-specific request access patterns remain important in the analysis of DRAM memory system performance. To examine the range of workload-specific variances, a large set of application traces are used in this study. From the SPEC CPU 2000 benchmark suite, address traces from 164.gzip, 176.gcc, 197.parser, 255.vortex, 172.mgrid, 178.galgel, 179.art, 183.equake, and 188.ammp are used. The address traces from the SPEC CPU 2000 benchmark suite were captured through the of the MASE simulation framework with the L2 cache size of the simulated processor set to 256 KB[29]. In addition, processor bus traces captured from a desktop personal computer system running various benchmarks and applications such as JMark 2.0 CPU, JMark 2.0 Complex Mathematics, 3DWinbench CPU, SETI@Home and Quake3, are added to the mix. The SPEC workload traces and desktop computer application traces collectively form a rich set of diverse workloads that enable generalized observations to be made about DRAM memory system performance characteristics in this study, and detailed information on traces used in this study can be found in appendix A.

7.2 Close-page System Performance Analysis

To limit the number of independent variables that affect DRAM memory system performance to a manageable subset, a single rank memory system that uses a close-page row buffer management policy is used in the first set of studies to examine the performance trade-off of a DRAM device with 8 banks versus 16 banks, t_{FAW} limitations with t_{FAW} set equal to t_{RC} and $t_{RC}/2$, memory controller sophistication in terms of transaction re-ordering capability, and data burst duration.

7.2.1 System Configuration Assumptions

Figure 7.1 shows the system configuration used in the close-page memory system study.

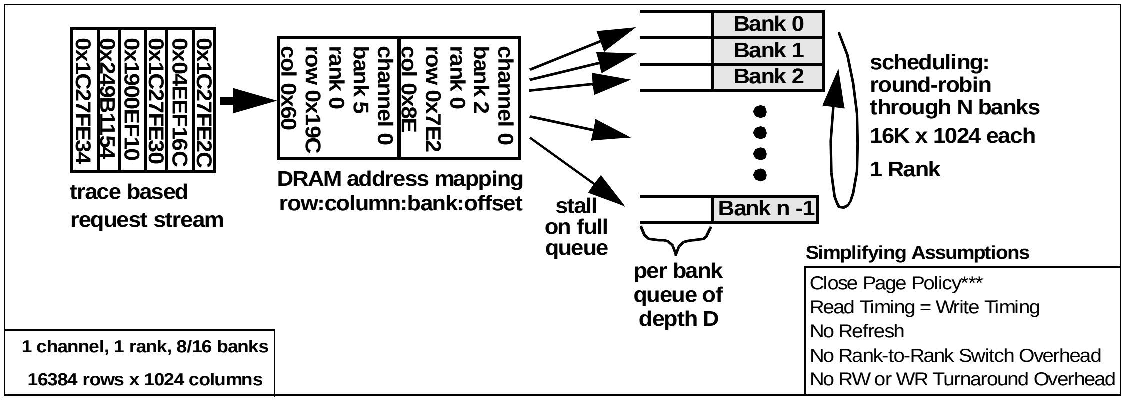


Figure 7.1: Close-page studies system configuration .

Figure 7.1 shows that the analytical framework accepts transaction requests from a trace input file, maps the request into the memory system, then places the request into one of N queues in the system. The system is configured with a single channel and a single rank of memory, with 16384 rows per bank and 1024 columns per row. The number of banks can be set to 8 or 16 banks depending on the configuration, and there is one queue for each bank in the memory system. In the analytical framework, one request per queue is selected from the memory system on a round-robin basis and sent into the Request Access Distance computation engine. Finally, in the analytical framework, each request moves data in the granularity of a cacheline, and the contiguous movement data for a given request occupies t_{Burst} duration on the data bus.

Re-Ordering Queue Depth

In this study, the depth of the per-bank request queues used for memory request re-ordering can be varied depending on configuration, and this depth is used to represent the level of sophistication of the memory controller. Pending requests are selected from the requests queues to maximize the temporal distance between any two requests from the same bank. In the case that the queue depth D is very deep, the round-robin request selection mechanism will likely have at least one pending entry in each queue each time it goes to the queue to obtain a request. However, in the case that the queue depth is shallow, the round-robin request selection mechanism will likely have to skip over many queues that do not have any pending requests, and idling time may have to be added so that row cycle time requirements can be met for any two requests to the same bank. In the extreme case where the queue depth D is zero, transaction requests are sent into the Request Access Distance computation engine in strict request order.

Finally, the memory request re-ordering performed in the analytical framework is based on the address request stream from the captured traces, and this re-ordering may further violate the natural memory ordering of a given workload. In this sense, the maximum achievable bandwidth of a given workload computed by the trace based analytical framework further deviates from the maximum achievable bandwidth of that specific workload in a real system. However, the goal of this work is to examine the limits of DRAM memory system configuration and timing parameter variations, and in that context, the specific memory dependencies of a single threaded workload is not a critical consideration. In particular, future high performance DRAM memory systems are expected to support multi-threaded and multi-core processors. For these high performance DRAM memory systems, the higher memory request rate from the multi-threaded and multi-core processors means that more pending requests would be available in the request queue for re-ordering, and more requests can be re-ordered for performance. In this sense, the workloads used in this study represent realistic address request patterns found in various single-threaded workloads, but the memory dependency assumptions are more applicable to current and near-future high performance DRAM memory systems that support multi-threaded and multi-core processors.

DRAM protocol overhead

For the close-page memory system study, a single rank memory system is used along with a memory controller that performs various degrees of sophisticated memory request re-ordering. As a result, the read-write turnaround overheads can be minimized and the rank-to-rank switching overhead does not apply. As part of the effort to limit the number of independent variables examined in each set of the overall study, the DRAM protocol is assumed to be ideal and protocol overheads are set to zero for the close-page system study.

7.2.2 Workload Characteristics: 164.gzip

In this work, memory address traces from various workloads are sent through the Request Access Distance analytical framework so that the maximum bandwidth efficiency of the address request pattern in the trace can be computed, given differing system configurations and timing parameters. Figure 7.2 shows the bandwidth efficiency of the memory system as address trace from 164.gzip is used to drive the analytical framework. Essentially, the address trace from 164.gzip is subjected to varying conditions of the re-ordering queue depth, number of banks in the memory system, t_{FAW} constraints and t_{RC}/t_{Burst} ratios. For each set of condition, a sustained bandwidth efficiency is computed and plotted in Figure 7.2. Figure 7.2 shows that in cases where the address trace is not re-ordered, the memory system is limited by the row cycle time and differences in the number of banks or t_{FAW} does not matter. However, when the 164.gzip address trace is subjected to some amount of re-ordering, the efficiency of the DRAM memory system increases dramatically. As the efficiency increases, the performance of the DRAM memory system becomes more sensitive to the number of available banks and to the bandwidth limitations imposed by t_{FAW} at higher t_{RC}/t_{Burst} ratios.

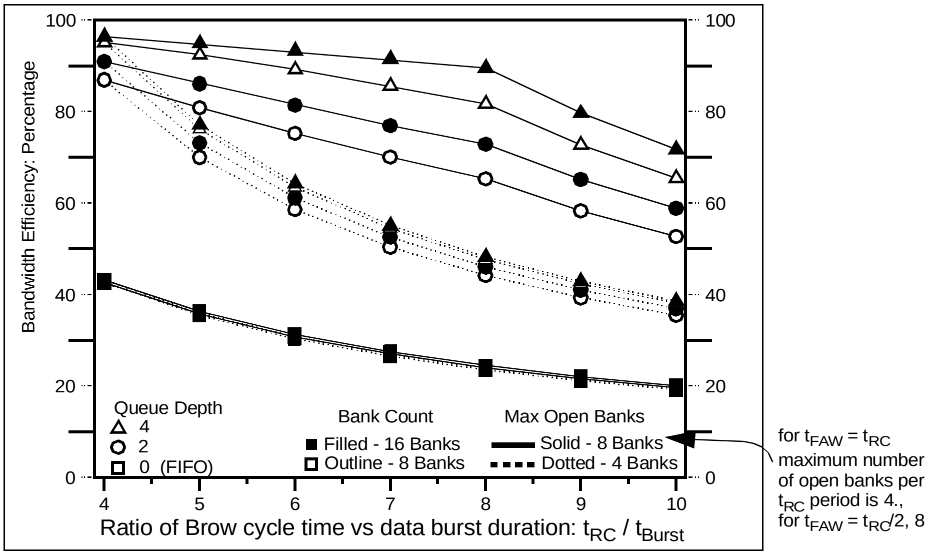


Figure 7.2: 164.gzip bandwidth efficiency graph.

Maximum Sustainable Bandwidth of 164.gzip Address Trace in Close-page Systems

Figure 7.2 shows the efficiency of the DRAM memory system subjected to the memory request pattern of 164.gzip. However, the efficiency graph does not provide certain insight into workload performance characteristics that can be observed when the data is plotted as a function of sustained bandwidth. Figure 7.3 shows the same dataset as Figure 7.2, but in place of the abstract t_{RC}/t_{Burst} ratio, Figure 7.3 assumes a specific system configuration where t_{RC} is 60ns, the data burst duration is 8 beats of data, and the width of the data bus is 8 bytes. Given specific DRAM device datarate and system configuration, the efficiency graph shown in Figure 7.2 can be converted to a graph that shows the maximum sustainable bandwidth of a given memory request sequence. In the bandwidth view, Figure 7.3 shows that without transaction re-ordering, the maximum sustainable bandwidth of the DRAM memory system increases very slowly with respect to increasing datarate of the memory system. Figure 7.3 shows that for t_{FAW} = t_{RC}, the maximum number of open banks per rolling t_{RC} window is 4, and DRAM memory system can only sustain approximately 4 GB/s of bandwidth for the 164.gzip address trace. However, in the case that t_{FAW} = $t_{RC}/2$, the maximum number of open banks is 8, and the sustained bandwidth for 164.gzip continues to increase until the datarate of the DRAM memory system reaches 1.07 Gbps. At the datarate of 1.07 Gbps, the ratio of t_{RC}/t_{Burst} equals the maximum number of concurrently open banks, and the maximum sustained bandwidth reaches a plateau for all test configuration.

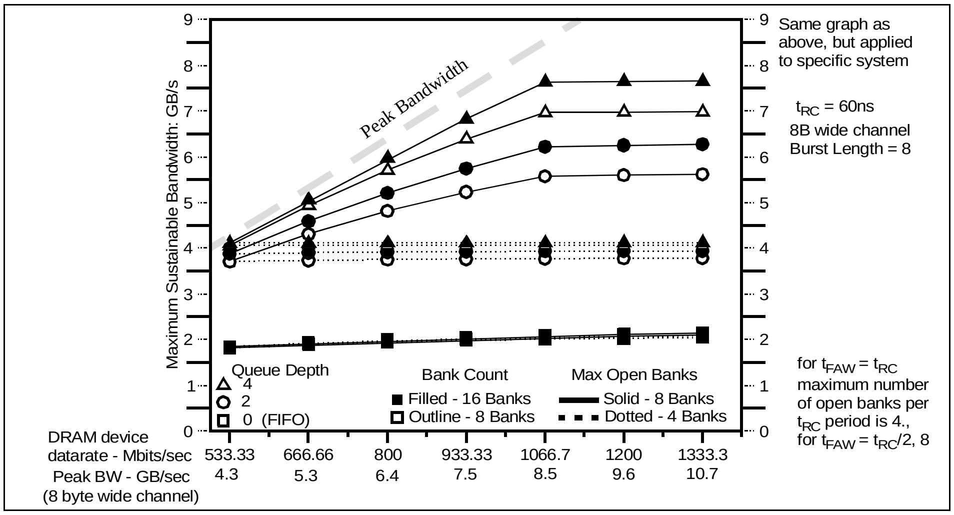


Figure 7.3: 164.gzip maximum sustainable bandwidth: close-page.

Finally, Figures 7.2 and 7.3 both show the performance benefit from having 16 banks compared to 8 banks in the DRAM memory system. Both Figures 7.2 and 7.3 show that at low datarates, the performance benefit of having 16 banks is relatively small. However, the performance benefit of 16 banks increases with increasing t_{RC}/t_{Burst} ratio.

7.2.3 t_{FAW} Limitations in Close-page Systems: All Workloads

In a close page memory system with a single rank of memory devices, t_{FAW} limits the number of banks that can be utilized in any rolling t_{RC} time frame. The net effect is that t_{FAW} limits available bank bandwidth, and that impact is particularly damaging for close-page memory systems operating with high ratios of t_{RC}/t_{Burst}. Figure 7.4 summarizes the impact of t_{FAW} by showing the respective average curves of the maximum sustainable bandwidth for all address traces used in this study. The average curve of the respective maximum sustainable bandwidth for all address traces with t_{FAW} set equal to t_{RC} is shown as a solid black line, and the average curve for the case where t_{FAW} set equal to half of t_{RC} is shown as a dashed red line. Figure 7.4 shows that larger t_{FAW} limits DRAM bandwidth at all

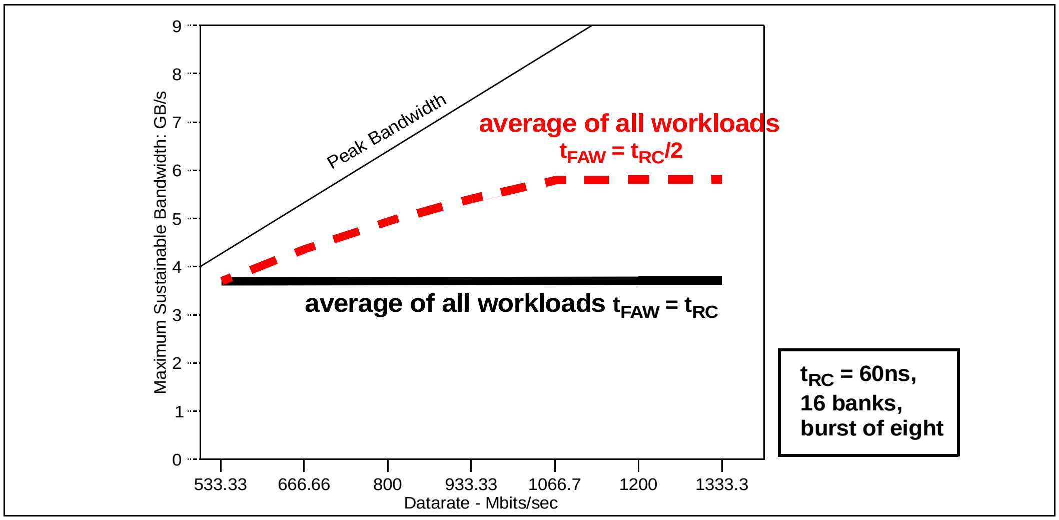


Figure 7.4: t_{FAW} impact: comparing $t_{FAW} = t_{RC}$ versus $t_{FAW} = t_{RC}/2$.

illustrated datarates, but the impact of t_{FAW} can be deferred until higher datarates by reducing t_{FAW}. Figure 7.4 can thus be use as an illustration for memory system design engineers to seek design alternatives to a close-page DRAM memory system with only one rank of DRAM devices and high t_{FAW} value.

7.2.4 Bank Comparison: 8 versus 16: All Workloads

In all DRAM devices, a bank of DRAM arrays cannot be accessed while it is in the process of being opened with a row activation command or closed with a precharge command. In this sense, the row activation and precharge actions on a DRAM bank represent the access overhead to that bank. This access overhead is particularly important in a close-page memory system, since each column access incurs the access cost of the row cycle time for a single bank. In modern DRAM devices, DRAM arrays are organized into multiple banks to hide the row access overhead, and accesses to different banks can be pipelined as long as there are enough banks in a DRAM system to service pending requests to those banks.

Figure 7.5 shows the mean bandwidth improvement curves for the 8 banks to 16 banks comparison for all address traces examined in this study with different system configurations. Figure 7.5 illustrates two intriguing points about the importance of having more banks in a high performance DRAM memory system. One intriguing point illustrated by Figure 7.5 is that a system that supports moderate amounts of re-ordering shows a higher degree of bandwidth improvement when compared to a system that supports a high degree of memory transaction re-ordering in the range of relatively lower t_{RC}/t_{Burst} ratios. The respective bandwidth improvement curves crossover and systems that support a high degree of memory transaction re-ordering performs better when the t_{RC}/t_{Burst} ratio increases. One

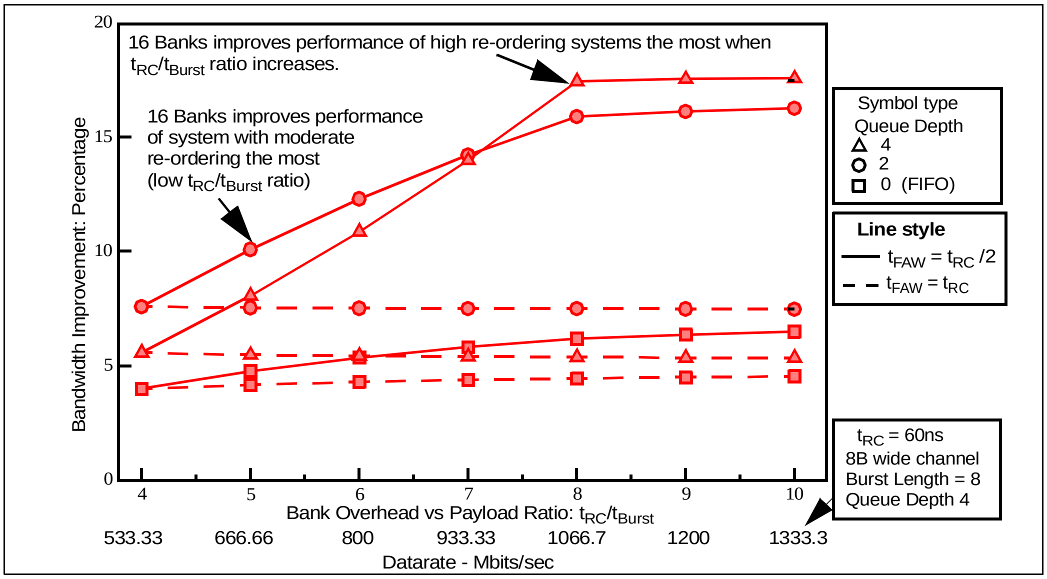


Figure 7.5: Mean and median bandwidth improvements: 8 banks versus 16 banks.

explanation of the crossover phenomenon is that a sophisticated memory system with deep re-ordering queues can readily extract available DRAM bandwidth, and as long as the t_{RC}/t_{Burst} ratio is far less than the number of available banks, such a system can readily extract near maximum bandwidth from a given DRAM memory system. In this case, the highly sophisticated memory system would not benefit much from additional banks as long as the ratio of t_{RC}/t_{Burst} is relatively low. In contrast, a memory system that performs only moderate amounts of transaction re-ordering can readily make use of the additional banks at lower t_{RC}/t_{Burst} ratios, thus showing a relatively higher degree of bandwidth improvement as compared to the highly sophisticated memory system. However, as the t_{RC}/t_{Burst} ratio continues to increase, the pressure to fully utilize available DRAM banks also increases. The result is that at higher t_{RC}/t_{Burst} ratios, a highly intelligent memory system can benefit the most from having additional banks.

A second intriguing point illustrated by Figure 7.5 is that as long as the DRAM device is not constrained by t_{FAW}, the benefit in the maximum sustainable bandwidth of having 16 banks over 8 banks continues to increase with increasing t_{RC}/t_{Burst} ratio. In modern DRAM devices with relatively constant t_{RC} values and higher data rates in successive generations of DRAM devices, the point illustrated by Figure 7.5 shows that as data rates increase, the value of having more banks also increases. However, Figure 7.5 also illustrates that the benefit of having more banks no longer increases with increasing t_{RC}/t_{Burst} ratio once t_{FAW} limits kick in to limit bank bandwidth.

Finally, Figure 7.5 shows that on average, maximum sustainable bandwidth for all workloads increases by 2 to 6% in memory systems that do not re-order memory transactions, and having 16 banks only minimally improves the sustainable bandwidth characteristics for these systems. However, in memory systems that do re-order memory transactions, having 16 banks improves the average bandwidth from 4% to 18%, depending on specific system configurations and t_{RC}/t_{Burst} ratios.

7.2.5 Burst Length Impact: SPEC Workloads

Aside from the examination of bank count and the impact of t_{FAW}, the Request Access Distance analytical framework can also be used to investigate interesting system architecture issues. One issue that can be examined with the Request Access Distance analytical framework is the cacheline size issue. In recent years, one strategy that has been deployed by DRAM device design engineers to increase DRAM device datarate without fundamentally changing DRAM circuits is to increase the internal prefetch bit width of the DRAM device for each column access command. That is, in each successive generation of DRAM devices, more and more bits are fetched in parallel internally in the DRAM device. The wide parallel word is moved to the interface of the DRAM device and sent across a narrow data bus at higher datarates. The internal N-bit fetch mechanism is commonly referred to as the prefetch-n architecture, where N bits of data are fetched internally per bit of external DRAM device data bus width. In these cases, the data bus is designed to operate at N times the datarate of the DRAM core. For SDRAM devices, the prefetch width is 1. For DDR SDRAM devices, the prefetch width is 2. For DDR2 SDRAM devices, the prefetch width is 4. For DDR3 SDRAM devices, the prefetch width will be increased to 8.

The prefetch width of DRAM devices impacts system architecture in that it defines the minimum data burst length in the DRAM memory system. The specification of a minimum burst length in turn means that a minimum amount of data must be transferred for each request. In the case that the length of the cacheline is smaller than the minimum amount of data moved per column access command, the system design engineer may have no choice but to increase the length of the cacheline. Fortunately, in the case where a given workload makes use of a high percentage of data in each accessed cacheline, the longer cacheline and longer data bursts from the DRAM memory system is more efficient in transporting data in the memory system. However, in the case where a given workload only uses a small fraction of each accessed cacheline, the transfer of longer cachelines becomes counter productive and detrimental to the overall performance characteristic of the memory system. Unfortunately, the complexity of the picture in regards to the burst length issue increases as the ratio of t_{RC}/t_{Burst} changes, and t_{FAW} limits the number of row activations per unit time. That is, as the ratio of t_{RC}/t_{Burst} increases, the cost associated with moving a single cacheline increases dramatically. In this case, the transport of long cachelines becomes more economical, particularly in a close-page system.

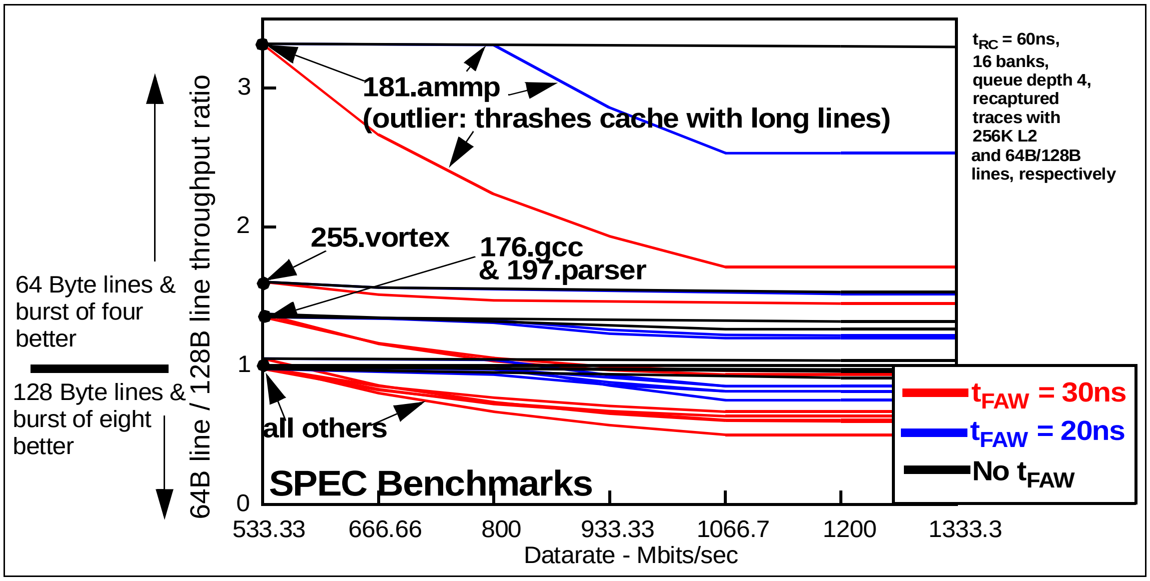


Figure 7.6: Ratio of request throughput: burst of four versus burst of eight.

Figure 7.6 shows the ratio of bandwidth throughput for two sets of address traces from nine different workloads from the SPEC CPU 2000 benchmark suite. The first set of address traces were captured with the cacheline size set to 128 bytes. The second set of address traces were captured with the same set of workloads and the cacheline size set to 64 bytes.

In this study, the set of address traces captured with the cacheline size set to 128 bytes is coupled with a DRAM memory system with a 16 byte wide data bus that transfers data in bursts of eight beats, and the set of address traces captured with the cacheline size set to 64 bytes is coupled with the same DRAM memory system that provides data in bursts of four beats. Then, the two sets of address traces are subjected to the Request Access Distance bandwidth efficiency analysis. The respective bandwidth efficiencies are then converted into runtimes by multiplying through the computed efficiency against the number of requests in each trace. The ratios of the runtimes are then compares to each other in Figure 7.6. Figure 7.6 aims to illustrate the trade-off points between systems with different cacheline sizes given different t_{RC}/t_{Burst} ratios and t_{FAW} constraints of in a close-page memory system.

In Figure 7.6, the request throughput ratio of one means that the workload performs equally well with a cacheline length of 128 bytes as it does with a cacheline length of 64 bytes. In the case that the request throughput ratio is greater than one, the system with the 64 byte cacheline performed better. Figure 7.6 shows that if the DRAM device is not constrained by t_{FAW}, then a processor with 64 byte cachelines and coupled to a DRAM memory system with minimum burst duration of four beats of data has equal performance to the processor with 128 byte cachelines and coupled to a DRAM memory system with minimum burst duration of eight beats of data in five out of nine SPEC workloads. Specifically, 164.gzip, 172.mgrid, 178.galgel, 179.art, and 183.equake, collectively labelled as ‘all others” in Figure 7.6, show no performance advantage from shorter cachelines and shorter DRAM transfers. In contrast, Figure 7.6 shows that the address trace from 181.ammp performed much better with shorter cachelines due to the fact that with longer cachelines, fewer sets of cachelines are available to the application, and 181.ammp with 128 byte cacheline thrashed the small 256 KB L2 cache. In this scenario, the memory system with 64 byte cachelines is unquestionably better. In contrast, address traces from 176,gcc, 197.parser and 255.vortex also show significant performance advantage for the system with shorter cachelines and shorter DRAM transfers.

Finally, for the close-page memory system with 64 byte cachelines and 4-beat DRAM data burst, t_{FAW} limitations begin to constrain the top end bandwidth available in the DRAM memory system. As a result, t_{FAW} limitations unambiguously swing the performance advantage to systems with longer cachelines and longer DRAM data transfers as the data rate of the DRAM memory system increases.

7.2.6 Queue Depth Analysis

Modern high performance DRAM memory systems utilize a range of techniques for performance optimization. In this study of a close-page memory system, a unique queuing structure was used to facilitate the implementation of a round-robin request re-ordering mechanism and extract parallelism from available DRAM banks. The intent of the re-ordering mechanism and the construction of the queuing structure in this study is not to examine the benefit of the specific re-ordering mechanism described. Rather, the intent of the re-ordering mechanism is to provide a basis for comparing the performance of DRAM memory systems with differing levels of sophistication in the memory controller.

Figure 7.7 summarizes the performance characteristics of all workloads examined in this study when subjected to differing re-ordering assumptions. The average curve of all workloads subjected to a DRAM memory system that does not re-order memory requests are set to black in color, and the average curve of workloads subjected to a DRAM memory system that aggressively re-orders memory transactions are set to red in color.

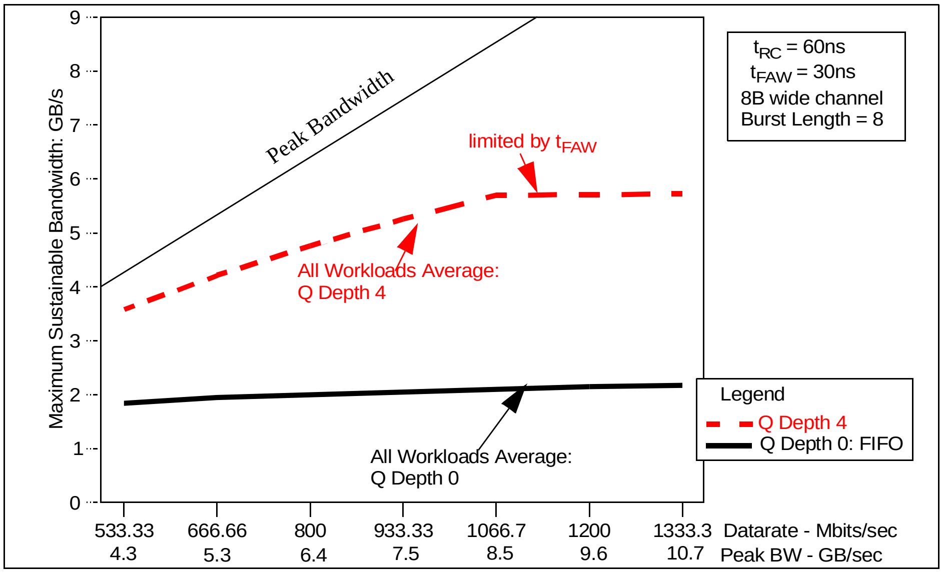


Figure 7.7: Sustainable bandwidth: re-ordering versus FIFO.

Figure 7.7 illustrates that a close-page DRAM memory system that does not re-order memory requests is limited by t_{RC} row cycle times. In such a case, the performance of the DRAM memory system increases only minutely with increasing datarate. Figure 7.7 also illustrates the point that with aggressive re-ordering, the t_{RC} row cycle time constraint can be alleviated, and such a DRAM memory system can sustain much higher DRAM bandwidth as a DRAM memory system that does not re-order memory transactions. Finally, as the datarate of the DRAM device climbs, t_{FAW} begins to constrain available bandwidth, and once t_{FAW} limitation kicks in, no further bandwidth improvement can be observed with increasing datarate.

7.3 Open-page System Performance Analysis

In this section, a study that compares slightly different sets of system configuration and timing parameters than the close-page memory system study is used to examine their respective impact on an open-page memory system. Specifically, the performance trade-off of a DRAM device with 8 banks versus 16 banks, a system configured with 1 rank or two ranks of DRAM devices, t_{FAW} limitations with t_{FAW} set equal to t_{RC} and $t_{RC}/2$, and different rank-to-rank switching time overheads are compared in this study.

7.3.1 System Configuration Assumptions

Figure 7.8 shows the system configuration used in the open-page memory system study.

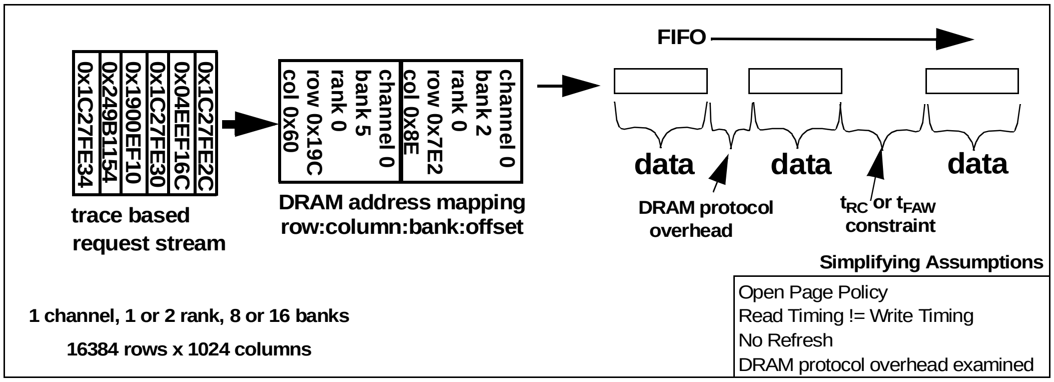


Figure 7.8: Open-page studies system configuration .

Figure 7.8 shows that the analytical framework accepts transaction requests from a trace input file, maps the request into the memory system, then sends the request to the Request Access Distance computation engine in strict ordering. The system is configured with a single channel and a 1 or 2 ranks of memory, with 16384 rows per bank and 1024 columns per row. The number of banks can be set to 8 or 16 depending on the configuration.

DRAM protocol overhead

For the open-page memory system study, a one rank memory system is compared to a two rank memory system. Also, the memory controller is assumed to perform no transaction request re-ordering. As a result, the rank-to-rank switching overhead may be significant, and read-write turnaround overheads may also impact the maximum sustainable bandwidth to a non-trivial degree. As a result, the open-page memory systems study is designed to fully accounts for the effects of the DRAM protocol, and the rank-to-rank turn around time, t_{DQS}, can be varied from 0 to 3 clock cycles (0~6 beats). The other timing parameters used, but not varied independently in this study are: t_{CWD}, set to 3 clock cycles, t_{CMD}, set to 1 clock cycle, t_{WR}, set to 4 clock cycles, t_{Burst}, set to 4 clock cycles, and t_{CAS}, set to 4 clock cycles. Finally, the performance impact of the refresh overhead, although non-trivial, is also ignored in the open-page memory systems study due to the belief that the effects of effect can be factored out in the comparison of two system configurations driven with the same workload, with the same refresh cycle times and the same number of rows per bank to be refreshed.

7.3.2 Address Mapping

In the open-page study, the address mapping scheme is designed so that consecutive cacheline requests are directed to the same row of the same bank, and the rank ID is mapped immediately above the range of the bank ID to take advantage of parallelism between different ranks. Figure 7.9 shows the address mapping scheme used in the open-page study.

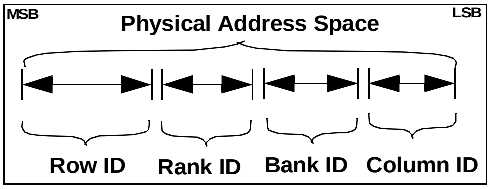


Figure 7.9: Open-page address mapping scheme.

7.3.3 Average of All Workloads

Figure 7.10 shows the maximum sustainable bandwidth averaged across all workloads used in the study. Figure 7.10 compares 4 different system configurations: a 1 rank 8 bank (1R8B) memory system, a 1 rank 16 bank (1R16B) memory system. a 2 rank 8 bank (2R8B) memory system, and a 2 rank 16 bank memory system. The system configuration is varied with the timing parameter t_{FAW} set equal to t_{RC} and $t_{RC}/2$, and t_{DQS} varied between 0 clock cycles and 3 clock cycles.

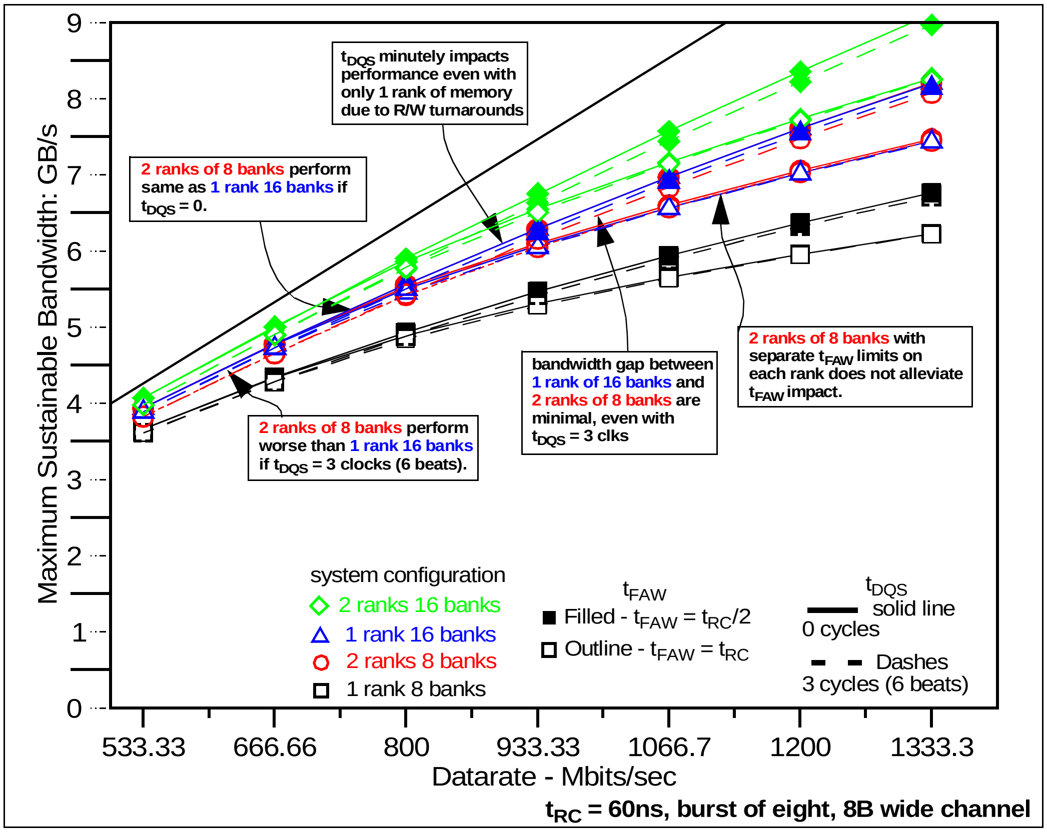


Figure 7.10: Maximum bandwidth averages: all workloads.

Figure 7.10 shows that for the open-page memory system, the high-degree of access locality provided by the single threaded workloads enable it to achieve high bandwidth efficiency without the benefit of sophisticated transaction request re-ordering mechanism.

Figure 7.10 also shows the following characteristics for the sustainable bandwidth graph of the average of all workloads.

- The address mapping scheme effectively utilizes parallelism afforded by the multiple ranks, and the performance of a 2R8B memory system is equal to that of a 1R16B memory system.
- The bandwidth degradation suffered by the 2R8B memory system compared to the 1R16B memory system is relatively small, even with the rank-to-rank switching overhead of t_{DQS} set to 3 clock cycles (6 beats). The reason for this minimal impact is that the access locality of the single-threaded workloads tended to keep accesses to within a given rank, and rank-to-rank switching time penalties are relatively minor or largely hidden by row-cycle time impacts.
- Interestingly, the difference of t_{DQS} set to 3 clock cycles also impacted the sustainable bandwidth of single-rank memory systems due to the fact that t_{DQS} impacts read-write turnaround times.
- The four bank activation window constraint, t_{FAW}, negatively impacts the sustainable bandwidth characteristic of a 2 rank memory system just as it does for a 1 rank memory system. This surprising result can be explained with the observation that the address mapping scheme, optimized to obtain bank parallelism for the open-page row buffer management policy, tended to direct accesses to the same bank and the same rank. The result is that bank conflicts are also directed onto the same rank, and multiple row cycles tended to congregate in a given rank of memory, rather than distributed across two different ranks of memory.
- The impact of t_{DQS} is relatively constant across different datarates for systems that are not impacted by t_{FAW}. A close examination of the bandwidth curves for the 2R16B system reveals that in systems impacted by t_{FAW} limitations, the impact of t_{DQS} is mitigated to some extent. That is, idle cycles inserted into the memory system due to rank-to-rank switching times also contributes to the idle time needed by the DRAM device to recover between consecutive row-accesses. In that sense, the same idle cycles can be used for multiple purposes, and the impact of these respective constraints are not additive.

7.3.4 Workload Characteristics: 164.gzip

Figure 7.11 shows the maximum sustainable bandwidth characteristic of 164.gzip in open-page memory systems. Figure 7.11 shows that 164.gzip is an outlier in the sense that the workload has high degree of access locality, and most request are kept within the same rank of memory systems. In the case that DRAM accesses are made to a different rank, a bank conflict also follows. As a result, the impact of t_{DQS} is not readily observable in any system configuration, and a 2 rank, 8 bank system configuration performance identically to a 1R16B memory system. Also, the number bank conflicts are relatively few and the impact of t_{FAW} is minimal and not observable until datarates reach significantly above 1 Gbps. Finally, the maximum sustainable bandwidth for 164.gzip scales nicely with the total number of banks in the memory system, and the bandwidth advantage of a 2R16B system over that of a 1 rank 16 bank system is nearly as great as the bandwidth advantage of the 1R16B system over that of a 1R8B system.

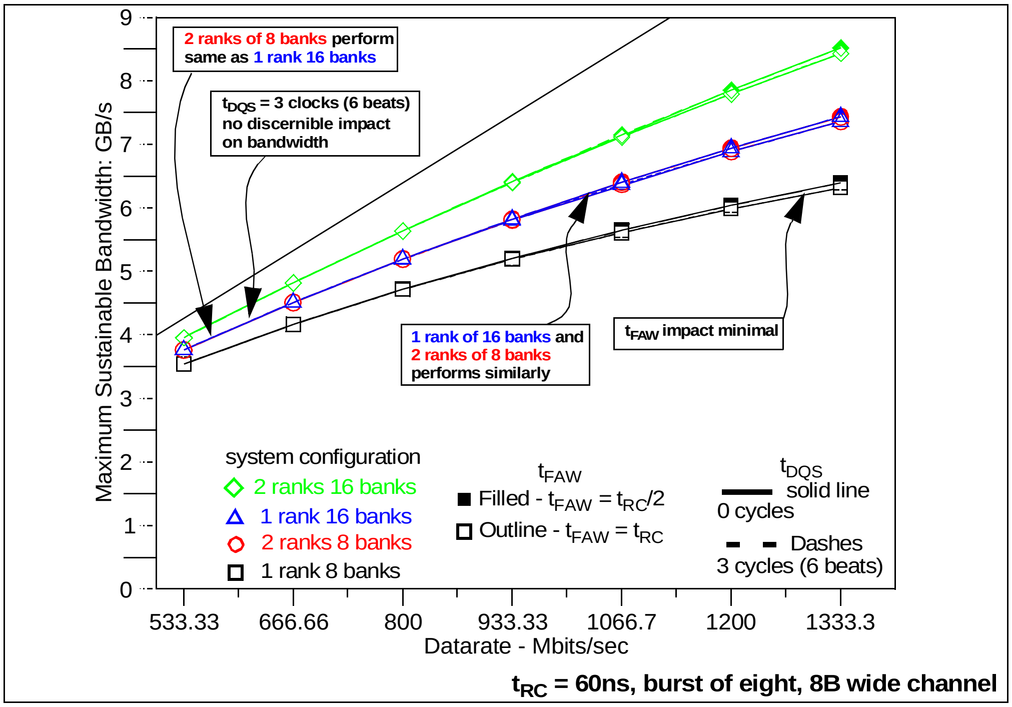


Figure 7.11: 164.gzip maximum sustainable bandwidth: open-page.

7.3.5 Workload Characteristics: 255.vortex

Figure 7.12 shows the maximum sustainable bandwidth characteristic of 255.vortex in open-page memory systems. Figure 7.12 shows that similar to 164.gzip, 255.vortex is an outlier from the set of average curves for all workloads shown in Figure 7.10. However, Figure 7.12 shows that in contrast to 164.gzip, 255.vortex is an outlier that is not only sensitive to the system configuration in terms of the number of ranks and banks, but it is also extremely sensitive to the impacts of t_{FAW} and t_{DQS}. Figure 7.12 also shows that 255.vortex has a relatively lower degree of access locality, and fewer column accesses are made to the same row than other workloads, resulting in a relatively higher rate of bank conflicts. The bank conflicts also tended to be clustered to the same rank of DRAM devices, even in 2 rank system configurations. The result is that the t_{FAW} bank activation constraint greatly impacted the maximum bandwidth of the DRAM memory system in all system configurations.

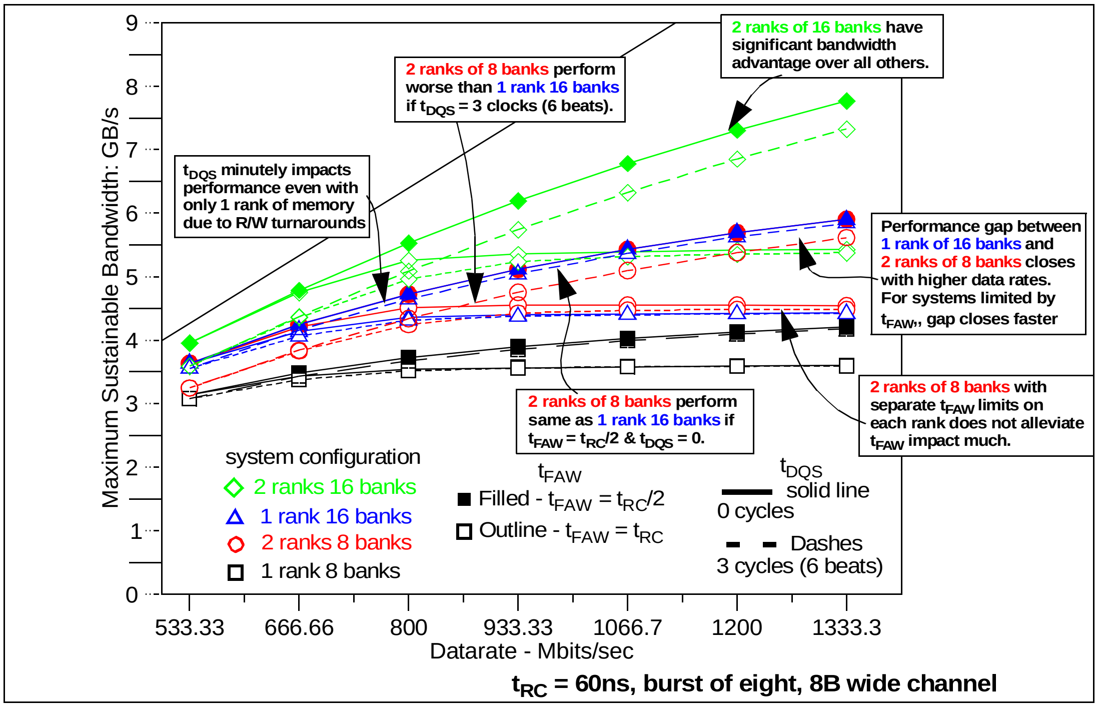


Figure 7.12: 255.vortex maximum sustainable bandwidth: open-page.

To summarize, Figure 7.12 shows the following characteristics for the sustainable bandwidth characteristics of 255.vortex.

- 255.vortex is greatly impacted by the rank-to-rank switching overhead, t_{DQS}, in system configurations with 2 ranks of memory. The overhead is somewhat alleviated at higher data rates, as other overheads become more significant. At higher datarates, the bandwidth impact of t_{DQS} remains, but the effects become less discernible as an independent source of hindrance to data transport in a DRAM memory system.
- The rank-to-rank switching overhead, t_{DQS}, also impacts the maximum sustainable bandwidth characteristic of single rank DRAM memory systems due to its impact on the read-write turnaround time.
- Like 164.gzip, 255.vortex also benefits greatly from a system configuration with 2 ranks of devices, each with 16 banks internally.
- Unlike 164.gzip, 255.vortex is extremely sensitive to t_{FAW}, and the two rank memory system organization only minimally alleviate the impact of t_{FAW}. Figure 7.12 shows that at high datarates, a t_{FAW} limited 2R8B memory system does achieve higher bandwidth utilization compared to a similar t_{FAW} limited 1R16B memory system despite the impact of a 3 cycle t_{DQS} rank-to-rank switching overhead.
- Finally, the impact of t_{FAW} on 255.vortex is reminiscent to that of all benchmarks in a close-page memory system where the simulation assumption of $t_{FAW} = t_{RC}/2$ completely limits sustainable bandwidth for all system configurations beyond a certain datarate. In this case, all t_{FAW} limited memory systems reach a plateau in terms of the maximum sustainable bandwidth at roughly 800 Mbps. At datarates higher than 800 Mbps, no further improvements in maximum sustainable bandwidth can be observed for 255.vortex in all t_{FAW} limited memory systems.

7.3.6 t_{FAW} Limitations in Open-page System: All Workloads

Figure 7.4 shows that t_{FAW} is particularly detrimental to the sustainable bandwidth characteristic for close-page memory systems operating with high ratios of t_{RC}/t_{Burst}. Figure 7.13 shows that t_{FAW} can similarly impact an open-page memory system, although to a lesser degree. Figure 7.13 shows the impact of t_{FAW} in a 1 rank 16 bank memory system in terms of the percentage of bandwidth differential between the case where $t_{FAW} = t_{RC}$ and the case where $t_{FAW} = t_{RC}/2$. The bandwidth advantage curves for different workloads used in the simulation are drawn separate lines in Figure 7.13, illustrating the wide variance in workload sensitivity to the limitation presented by a restrictive t_{FAW} parameter. One workload worthy of note is the previously examined 255.vortex, where bandwidth impact for the case where $t_{FAW} = t_{RC}$ can impact bandwidth by upwards of 40~50%. However, on average, a workload running on a memory system where $t_{FAW} = t_{RC}$ suffers a bandwidth loss on the order of 0~12% compared to the same system with a more restrictive t_{FAW} value where $t_{FAW} = t_{RC}/2$.

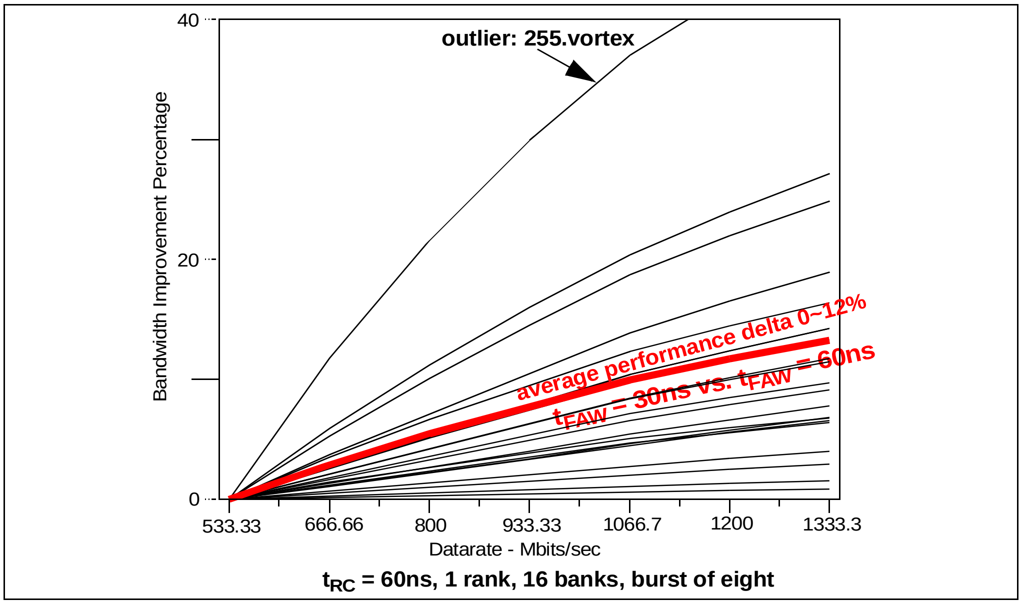


Figure 7.13: Comparing $t_{FAW} = t_{RC}/2$ versus $t_{FAW} = t_{RC}$ in open-page system.

7.3.7 Configuration Comparison: 1R8B vs. 2R8B vs. 1R16B vs. 2R16B

Figure 7.14 shows three sets of cross comparisons for the sustainable bandwidth characteristics of four different system configurations: 1R16B versus 1R8B, 2R8B versus 1R8B, and 2R16B versus 2R8B. Figure 7.14 shows that the bandwidth advantage of a 2R8B memory system over that of a 1R8B memory system is roughly comparable as the bandwidth advantage of a 1R16B memory system over that of a 1R8B memory system. Figure 7.14 also illustrates the point that in case of a memory system configuration with relatively lower datarates and high rank-to-rank switching time penalty, the 1R16B memory system has some advantage over that of a 2R8B memory system. However, as data rate increases, the impact of t_{FAW} becomes more important. In such a case, the 2R8B configuration begins to outperform the 1R16B configuration, albeit minutely. Finally, the bandwidth advantage of a 2R16B memory system over that of a 1R16B memory system is roughly half of the bandwidth advantage presented by the 1R16B memory system over that of the 1R8B memory system.

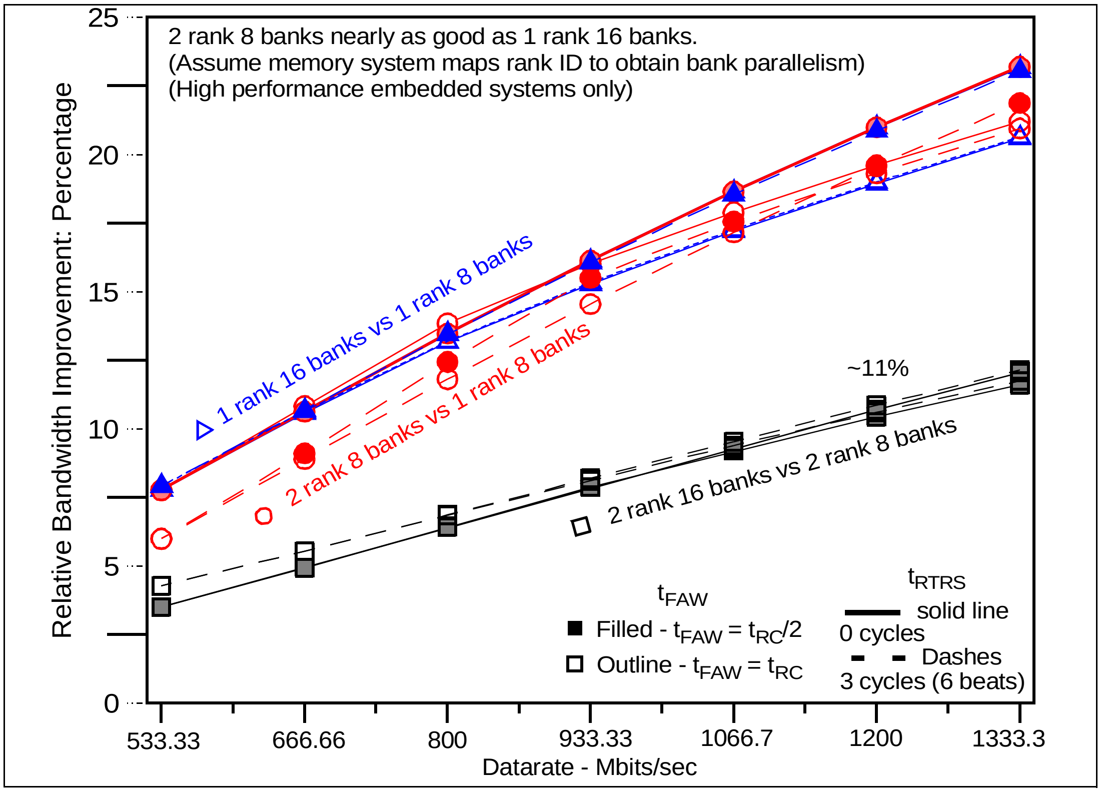


Figure 7.14: All workloads mean sustainable bandwidth: cross comparisons.

Mean Bandwidth Improvements: Open-page and Close-page.

Figure 7.15 shows the mean bandwidth improvement curves for the 1R16B to 1R8B comparison for the open-page memory system and the close-page memory system with per-bank re-ordering queue depth of 4. Figure 7.15 shows that despite the differences in the row buffer management policy and the differences in the re-ordering mechanism, the bandwidth advantage of a 1R16B memory system over that of a 1R8B memory system correlates nicely between the open-page memory system and the close-page memory system. In both cases, the bandwidth advantage of having more banks in the DRAM device scales at roughly the same rate with respect to increasing datarate and constant row cycle time. In both memory systems, the bandwidth advantage of the 1R16B memory system over that of the 1R8B memory system reaches approximately 18% at 1.067 Gbps.

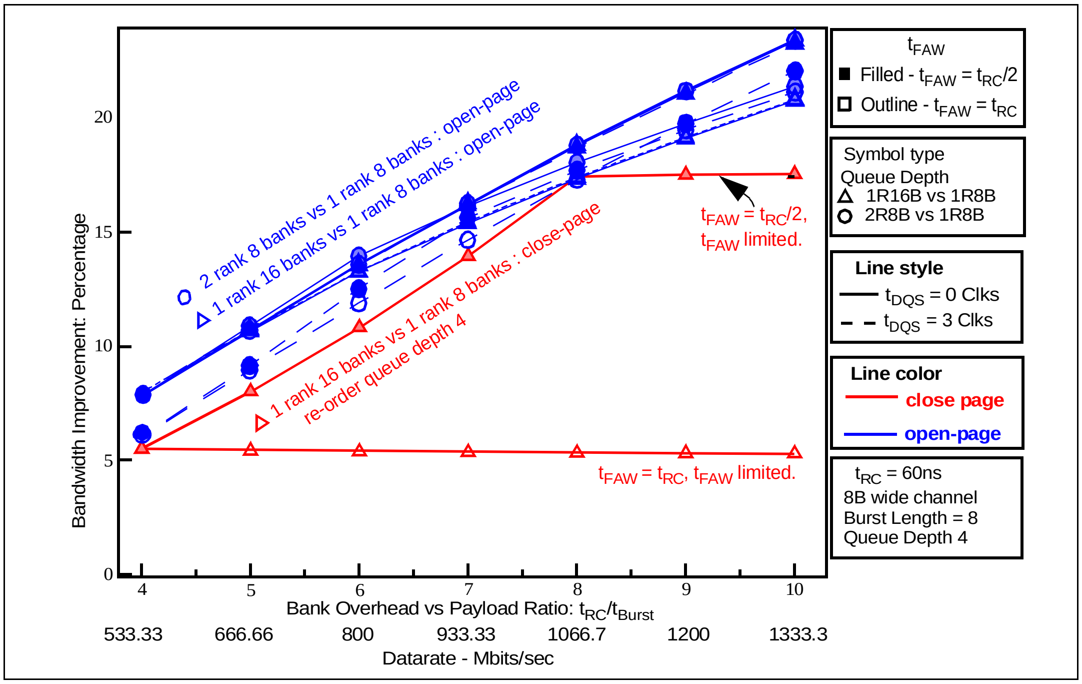


Figure 7.15: Mean bandwidth improvements: close-page and open-page.

7.4 DRAM Performance Analysis Summary

An extensive study of DRAM memory system performance characteristics is performed in this chapter. As examined in this work, the performance of DRAM memory systems depends on workload-specific characteristics. However, some observations about the performance of DRAM memory systems can be made in general:

- The benefit of having a 16 bank device over an 8 bank device in a one rank memory system configuration increases with datarate. The performance benefit increases to approximately 18% at 1 Gbps for both open-page as well as close-page memory systems. While some workloads may only see minimal benefits, others will benefit greatly. Embedded systems that are limited in the number of workloads should examine the bank count issue carefully.
- Single threaded workloads have high degrees of access locality, and sustainable bandwidth characteristics of an open-page memory system for a single threaded workload is similar to that of a close-page memory system that performs relatively sophisticated transaction re-ordering.
- The increase in DRAM internal bit-prefetch depth means a loss of randomness in memory access and an increase in minimum burst length for each access. The increase in minimum burst length may dictate the design of longer cachelines in some systems, depending on system configuration. For some embedded processors with relatively small cache sizes, the increase in data burst length may have a significant performance impact, particularly if the application does not use the additional data moved with longer burst lengths, and the longer burst lengths lead to cache thrashing.
- The t_{FAW} activation window constraint will greatly limit close-page memory systems without sophisticated re-ordering mechanisms. The impact of t_{FAW} is relatively less in open-page memory systems, but some workloads, such as 255.vortex, exhibit relatively less spatial locality, and their performance characteristics are similar to that for all workloads in close-page memory systems. In this study, even a two-rank memory system did not alleviate the impact of t_{FAW} on the memory system. Consequently, a DRAM scheduling algorithm that takes the impact of t_{FAW} into consideration is needed for the next generation DRAM controller.

CHAPTER 8

Power-Constrained DDRx Scheduling Algorithm

8.1 Introduction

The primary goal in the design of high performance memory systems is to obtain a design that can obtain maximum bandwidth with low request access latencies. However, constraints such as data bus synchronization overhead in DDRx[1] SDRAM memory systems and mechanisms that limit peak power in DDR2 and DDR3 memory systems will significantly impact sustainable bandwidth in high performance DDRx SDRAM memory systems. Moreover, while DRAM device datarate increases with each new generation of DDRx SDRAM devices at the rate of 100% every three years, DRAM row cycle times are only decreasing at a rate of 7% per year[22]. Collectively, these trends increase the difficulty of achieving maximum sustainable bandwidth from each successive generation of higher datarate DDRx SDRAM devices by increasing the ratio of DRAM row cycle time to data transport time. Previous studies have recognized and examined the importance of DRAM access scheduling but do not address the issue of data bus synchronization and power limiting constraints in DDRx SDRAM memory systems[34,37,42,43,48]. Recent work by Rixner examines the impact of data bus synchronization overhead and row-to-row activation time, but does not address the four-bank-activation window limitation of t_{FAW}, nor specific algorithms to deal with the conflicting requirements of these different overheads.[46] To design a high performance DDRx SDRAM memory controller, the issue of memory access

[1] DDRx denotes DDR, DDR2, DDR3 and future DDR SDRAM variant memory systems.

scheduling is re-visited in this chapter to address the constraints imposed on DDR2 and DDR3 SDRAM memory systems by the data bus synchronization overhead of t_{DQS} and peak power limiting timing parameters t_{FAW} and t_{RRD}. In this work, we propose a memory transaction and DRAM command scheduling algorithm that enables a two rank DDRx SDRAM memory system to achieve optimal bandwidth utilization while fully respecting the timing constraints imposed on the DDRx SDRAM memory system by t_{FAW}, t_{RRD} and t_{DQS}. The proposed DRAM transaction and command ordering algorithm selects pending memory transactions based on DRAM bank and rank addresses, then sequences the DRAM row activation and column access commands in a specific ordering to minimize the bandwidth impact imposed on the DRAM memory system.

In a 1 Gbit DDR3 SDRAM memory system examined in this study, the proposed DRAM transaction and command ordering algorithm increases the maximum sustainable bandwidth by 41% above a moderately intelligent memory system that implements a round robin bank rotation scheduling algorithm. Simulations show that the aggressive DRAM transaction and command ordering algorithm increases the performance of bandwidth intensive workloads by roughly 40% when compared against a round-robin bank-rotation scheduling algorithm that does not account for the bandwidth impact of t_{FAW}, t_{RRD} and t_{DQS}. In this chapter, the proposed DRAM transaction and command scheduling algorithm is described, and the maximum sustainable bandwidth of the proposed algorithm is illustrated. The simulation framework and the workloads used in this study as well as implementation requirements that enable the unique rank hopping scheduling algorithm is also described in detail. Finally, the results and analysis and impact of the proposed scheduling algorithm on scaling trends is discussed in the concluding section.

8.2 Background Information

8.2.1 Row Buffer Management Policy

Previous trace based studies have shown that single threaded-workloads benefit well from a open-page row buffer management policy and that t_{FAW} impacts are relatively minor compared to that of a close-page memory system. However, the impact of t_{FAW} grows worse with relatively constant row cycle times and increasing datarates in both open-page and close-page systems and t_{FAW} greatly limits the performance of close-page memory systems. In this study, the goal is to examine a scheduling algorithm that facilitates the extraction of maximum bandwidth in t_{FAW} limited, close-page memory systems. The rationale for the focus on close-page memory systems in this work is that the impact of t_{FAW} on close-page memory systems is immediate and extreme. As a result, the scheduling algorithm examined in this work is specifically targeted for close-page memory systems to alleviate the impact of the t_{FAW} bank activation window in DDRx SDRAM memory systems. The extension of the algorithm and study to the less-affected open-page memory systems is deferred to a future study.

8.2.2 Timing Parameters

The timing parameters used in this part of the study and the projected values for a 1 Gbps (500 Mhz) DDR3 SDRAM memory system are summarized in table 8.1[1].

Parameter	Description	Value
t_{Burst}	Data **Burst** duration. Time period that data burst occupies on the data bus. Typically 4 or 8 beats of data. In DDR SDRAM, 4 beats of data occupies 2 full cycles. Also known as t_{BL}.	8 ns
t_{CAS}	**C**olumn **A**ccess **S**trobe latency. Time interval between column access command and data return by DRAM device(s). Also known as t_{CL}.	10 ns
t_{CMD}	**C**o**m**man**d** transport duration. Time period that a command occupies on the command bus as it is transported from the DRAM controller to the DRAM devices.	2 ns
t_{CWD}	**C**olumn **W**rite **D**elay. Time interval between issuance of column write command and placement of data on data bus by the DRAM controller.	8 ns
t_{DQS}	**D**ata **S**trobe turnaround. Used in DDR and DDR2 SDRAM memory systems. Not used in SDRAM or Direct RDRAM memory systems. 1 full cycle in DDR SDRAM systems.	4 ns
t_{FAW}	**F**our bank **A**ctivation **W**indow. A rolling time frame in which a maximum of four bank activation may be engaged. Limits peak current profile.	48 ns
t_{RAS}	**R**ow **A**ccess **S**trobe. Time interval between row access command and data restoration in DRAM array. After t_{RAS}, DRAM bank could be precharged.	40 ns
t_{RC}	**R**ow **C**ycle. Time interval between accesses to different rows in same bank $t_{RC} = t_{RAS} + t_{RP}$	50 ns
t_{RCD}	**R**ow to **C**olumn command **D**elay. Time interval between row access command and data ready at sense amplifiers.	10 ns
t_{RRD}	**R**ow activation to **R**ow activation **D**elay. Minimum time interval between two row activation commands to same DRAM device. Limits peak current profile.	10 ns
t_{RP}	**R**ow **P**recharge. Time interval that it takes for a DRAM array to be precharged and readied for another row access.	10 ns
t_{WR}	**W**rite **R**ecovery time. Minimum time interval between end of write data burst and the start of a precharge command. Allows sense amplifiers to restore data to cells	10 ns

TABLE 8.1: Summary of timing parameters

8.2.3 Bank Activation Window Limited Memory System

To ensure that a commodity DDRx SDRAM device does not exceed a specified maximum power draw, timing parameters have been introduced to limit the power consumption characteristics. In DDRx SDRAM devices, t_{RRD} and t_{FAW} have been defined to specify the minimum time periods for row (bank) activations on a given DRAM device. The acronym RRD stands for ***r**ow-to-**r**ow activation **d**elay*, and FAW stands for ***f**our bank*

[1] 1 Gbps DDR3 SDRAM devices are currently under development at the time of this study. The timing parameters illustrated in table 8.1 are projected from 667 Mbps (333 MHz, dual data rate) DDR2 SDRAM devices subjected to current scaling trends in DRAM devices.

activation window. The timing parameters t_{RRD} and t_{FAW} are specified in terms of nanoseconds, and Figure 8.1 shows that by specifying t_{RDD} and t_{FAW} in terms of nanoseconds instead of the number of cycles, the minimum spacing between row activation is maintained regardless of operating datarates: on a given DRAM device, row activations must be scheduled at least t_{RRD} apart from each other, and within any t_{FAW} time period, at most four row activations can be engaged[1]. For close-page memory systems, t_{RRD} and t_{FAW} effectively limit the maximum sustainable bandwidth to each rank of memory, irrespective of the device datarate. In this case, the maximum bandwidth efficiency of a single rank, t_{FAW} limited close-page DRAM memory system is (4 * t_{Burst}) / t_{FAW}.

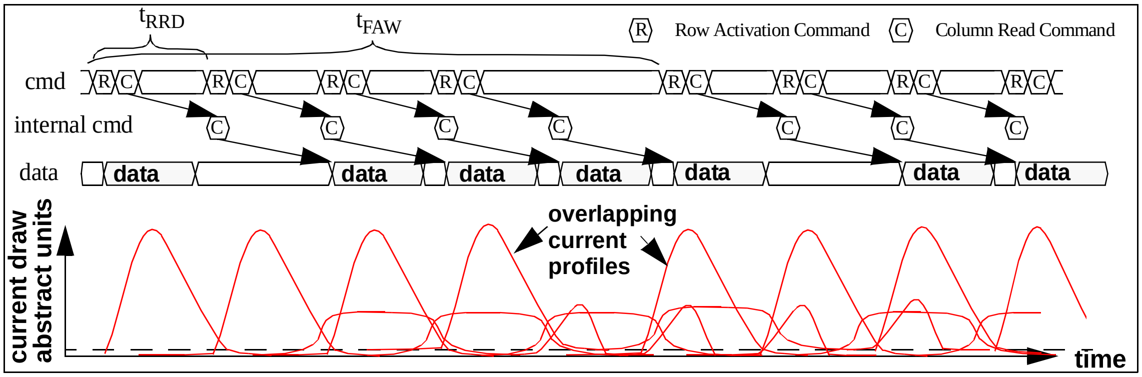


Figure 8.1: Maximum of Four Row Activations in any t_{FAW} time frame.

8.2.4 Consecutive Commands to Different Ranks: Data Bus Synchronization

In all modern DRAM memory systems, consecutive column-read commands to the same open row of the same bank or to different open rows of different banks of the same rank can be issued and pipelined consecutively. However, consecutive column-read commands to different ranks of memory cannot be pipelined consecutively in DDR, DDR2

[1] Precharge commands are not shown in the heavily pipelined timing diagrams of Figures 8.1, 8.2, and 8.3 in order to simplify the timing diagrams. In these Figures, the precharge command is assumed to be issued separately or via a column-read/write and precharge command. Since command bandwidth is not a constraint for the memory system examined in this study, leaving the illustration of the precharge command out of the timing diagrams does not impact statements made in the study. The timing and usage of the precharge command is accurately simulated in the simulation framework.

and DDR3 SDRAM memory systems due to insertion of idle cycles on the data bus to ensure proper transfer of control of the source synchronous data strobe signals from one rank of DRAM devices to another[1]. In this study, a 2 cycle, 4 ns switching time is specified for a DDR3 SDRAM memory system that operates at 1 Gbps (500 MHz).

Figure 8.2 illustrates the timing and command sequence of consecutive close-page read cycles to alternate ranks of DRAM devices. In Figure 8.2, each DRAM access is translated to a row-activation command and a column-access command. Figure 8.2 illustrates that the minimum spacing of t_{DQS}, the read-write data-strobe re-synchronization time, is needed in between each pair of column-read commands to allow one rank of DRAM devices to release control of data strobe synchronization signals and for a different rank of DRAM devices to gain control of them. In this case, each column-read access incurs the rank switching overhead of t_{DQS}, and the maximum sustainable bandwidth efficiency of a close-page memory system that alternates memory requests between two different ranks is t_{Burst} / (t_{Burst} + t_{DQS}). The compound effect of t_{DQS} and t_{FAW} is that neither a one-rank-at-a-time nor a simple alternate-rank hopping algorithm can sustain high bandwidth with ideally pipelined DRAM commands. In these cases, either the peak power limiting timing parameters or the rank-to-rank switching time will significantly impact maximum sustainable bandwidth in traditionally designed DDRx SDRAM memory systems.

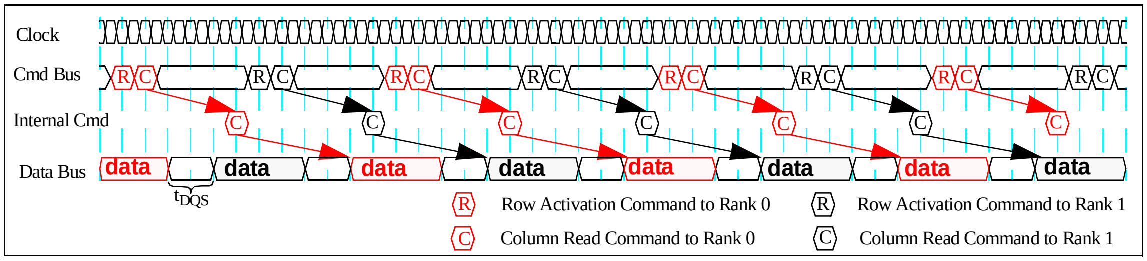

Clock
Cmd Bus
Internal Cmd
Data Bus
R
C
data
t_{DQS}
R Row Activation Command to Rank 0
C Column Read Command to Rank 0
R Row Activation Command to Rank 1
C Column Read Command to Rank 1

Figure 8.2: Consecutive Read Command to Alternate Ranks in DDR3 SDRAM (@ 1 Gbps).

[1] Future high frequency memory systems will be limited to at most two ranks of memory on a multidrop bus.

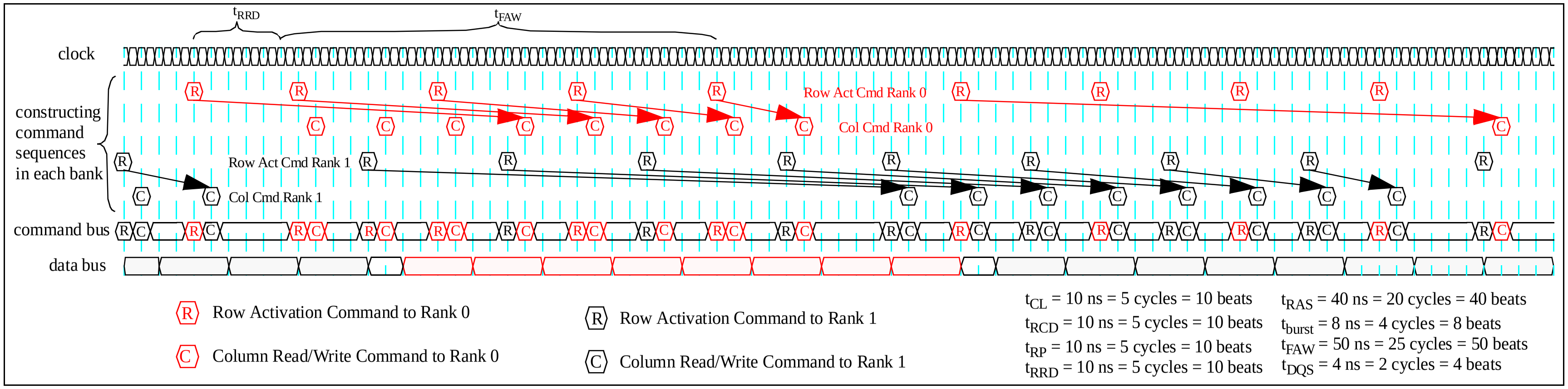

t_{RRD}
t_{FAW}
clock
constructing command sequences in each bank
Row Act Cmd Rank 0
Col Cmd Rank 0
Row Act Cmd Rank 1
Col Cmd Rank 1
command bus
data bus
R
C
R Row Activation Command to Rank 0
C Column Read/Write Command to Rank 0
R Row Activation Command to Rank 1
C Column Read/Write Command to Rank 1
t_{CL} = 10 ns = 5 cycles = 10 beats
t_{RCD} = 10 ns = 5 cycles = 10 beats
t_{RP} = 10 ns = 5 cycles = 10 beats
t_{RRD} = 10 ns = 5 cycles = 10 beats
t_{RAS} = 40 ns = 20 cycles = 40 beats
t_{burst} = 8 ns = 4 cycles = 8 beats
t_{FAW} = 50 ns = 25 cycles = 50 beats
t_{DQS} = 4 ns = 2 cycles = 4 beats

Figure 8.3: : Paired Row and Column Command Scheduling Algorithm @ 1 Gbps.

8.3 Proposed Rank Hopping Scheduling Algorithm

In the previous section, respective maximum sustainable bandwidth efficiencies for a single rank DDRx memory system and a dual rank DDRx memory system that alternates memory accesses between the two ranks of memory were computed as (4 * t_{Burst}) / t_{FAW} and t_{Burst} / (t_{Burst} + t_{DQS}), respectively. Substituting in the projected values for timing parameters for the 1 Gbps DDR3 SDRAM device specified in table 8.1, the maximum bandwidth efficiencies is 66.7% for both cases[1]. In contrast, the proposed DRAM transaction and command scheduling algorithm amortizes the rank switching overhead and increases the maximum bandwidth efficiency for a dual rank memory system to B * t_{Burst} / (B * t_{Burst} + t_{DQS}), where B denotes the number of banks in a given rank of DRAM devices in this study[2]. Substituting in the projected values for timing parameters as specified in table 8.1, the proposed scheduling algorithm increases the maximum sustainable bandwidth efficiency from 66.7% to 94%. The maximum bandwidth efficiency of 94% represents increases of 41% of additional bandwidth over the maximum bandwidth efficiencies of the baseline memory systems.

The key to increasing the bandwidth efficiency of a two-rank DDRx SDRAM memory system can be found through a fundamental examination of the causes of the respective constraints imposed on a DDRx SDRAM memory system by t_{DQS}, t_{RRD} and t_{FAW}. In a high frequency DDRx SDRAM memory system with a single rank of memory, row activations cannot be scheduled closely to each another, and a dual rank DDRx SDRAM memory

[1] Without accounting for refresh. DRAM refresh cycles are assumed as a constant bandwidth overhead for all systems compared in this study. As a constant overhead in all systems, it can be factored out and safely ignored as a simplifying assumption.

[2] DDR2 devices larger than 1 Gbit and all DDR3 devices have 8 banks internally. B is equal to 8 for these devices. The bank count may be further increased in future DDRx devices

system that alternates read cycles between different ranks incurs the rank switching overhead of t_{DQS} for each access. To minimize the bandwidth impact of t_{DQS}, t_{RRD} and t_{FAW}, a high performance DDRx SDRAM memory system must schedule row accesses to alternate ranks of memory to avoid the constraints of t_{RRD} and t_{FAW}. In contrast, to minimize the bandwidth impact of t_{DQS}, a high performance DDRx SDRAM memory system must group schedule column-read commands to the same rank of memory for as long as possible. The solution to the bandwidth constraints imposed by t_{DQS}, t_{RRD} and t_{FAW} in a high datarate DDRx SDRAM memory system is then a scheduling algorithm that de-couples row access commands from column access commands, distributes row-access commands to different ranks of memory to avoid incurring the constraints of t_{RRD} and t_{FAW}, and group schedules column-read commands to a given rank of memory for as long as possible, thus amortizing the rank switching overhead of t_{DQS}.

In this work, the ***command-pair rank hopping*** (CPRH) memory transaction re-ordering and DRAM command scheduling algorithm is described that alleviates the impacts of t_{FAW} and t_{DQS} simultaneously. The CPRH memory scheduling approach rely on the basic principle of round robin access rotation through all of the banks in a two rank memory system. The CPRH algorithm superficially resembles the simpler alternating rank scheduling illustrated in Figure 8.2 in that each row activation command is followed immediately by a column access command. However, unlike the alternating rank scheduling where each column command is a posted CAS command that immediately follows the row activation command to the same bank, the column command issued in the command pair algorithm is issued to a different bank of DRAM arrays. In essence, the command pair algorithm further de-couples the row-activation command and the column-access commands to create the regular scheduling pair of row and column commands that mimic the command pairs found in basic DDRx SDRAM command scheduling algorithms.

The command ordering sequence for the CPRH scheduling algorithm can be constructed from the basis of a round robin rotation through the banks. That is, DRAM column accesses are scheduled to bank 0, bank 1, bank 2, and rotated through sequentially to the (B-1)th bank of a given rank of memory. The algorithm then switches to bank 0 of the alternate rank of memory and the process repeats itself in rotating through all banks in a two rank memory system. Then, working backwards from the respective column access commands, the row access commands are scheduled to each rank in alternate ordering. Figure 8.3 illustrates the construction and timing of the CPRH algorithm for a memory system with two ranks of 1 Gbps DDR3 SDRAM devices. Figure 8.3 shows that the CPRH algorithm achieves high bandwidth in a two rank DDR3 SDRAM memory system despite the constraints imposed on the memory system by t_{RRD}, t_{FAW} and t_{DQS}.

The DRAM command sequence for the command pair scheduling algorithm is summarizes as Figure 8.4. Figure 8.4 shows that while the column access commands are

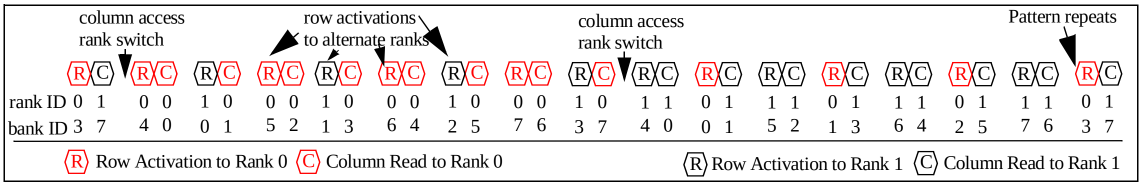

column access rank switch
row activations to alternate ranks
column access rank switch
Pattern repeats
rank ID 0 1 | 0 0 | 1 0 | 0 0 | 1 0 | 0 0 | 1 0 | 0 0 | 1 0 | 1 1 | 0 1 | 1 1 | 0 1 | 1 1 | 0 1 | 1 1 | 0 1
bank ID 3 7 | 4 0 | 0 1 | 5 2 | 1 3 | 6 4 | 2 5 | 7 6 | 3 7 | 4 0 | 0 1 | 5 2 | 1 3 | 6 4 | 2 5 | 7 6 | 3 7
R Row Activation to Rank 0
C Column Read to Rank 0
R Row Activation to Rank 1
C Column Read to Rank 1

Figure 8.4: Row and Column Command Sequences in Rank Hopping Algorithm.

group-scheduled successively to each bank in a given rank of memory, the row-activation commands are alternately scheduled to different ranks of memory. In the DRAM command sequence shown in Figures 8.3 and 8.4, the command pair algorithm amortizes the rank switching cost of t_{DQS} and achieves the theoretical maximum sustainable bandwidth.

Finally, Figures 8.3 and 8.4 also reveal a subtle optimization to the command pair algorithm in that row activations need not strictly alternate between different ranks. Figure 8.4 shows a sequence that begins with a column access rank switch overlapped with two row activations to the same rank. In this case, the rank-switching overhead of t_{DQS} increases the minimum scheduling distance between two row activation commands, and the t_{RRD} row activation constraint does not expose additional latency in the scheduling of DRAM commands in the memory system.

8.4 Experimental Methodology

8.4.1 Simulation Framework

MASE, the ***m**icro **a**rchitecture **s**imulation **e**nvironment*, is part of SimpleScalar version 4.0, and it is used as the simulation framework for this study[22]. MASE is used as the basis of the simulation framework due to the fact that it has been designed to interact with an event-driven variable-memory-access-latency DRAM memory system. That is, memory access latencies are not pre-computed at the instance in time when memory requests are initiated. Rather, the simulator uses the event-driven memory system to simulate each memory access independently, then returns the status of the memory access to the functional unit that initiated the memory reference when the memory transaction is serviced and marked as *COMPLETED* by the DRAM memory system simulator. In this simulation framework, multiple memory transactions exist in the memory system concurrently, and the event driven simulation allows the latency of a memory request to be affected by a request that happens later in time. This framework allows memory write requests to be deferred in favor of memory read requests, and it also allows for the implementation of prioritization of read transactions, instruction fetch transactions and prefetch transactions. The transaction request ordering mechanism is a necessary element that enables the study of memory access ordering algorithms examined in this study.

In Figure 8.5a, we show that the processor core of MASE consists of three basic blocks, each block representing a different portion of a high performance out of order processor: the instruction fetch and decode front end, the out-of-order execution engine, and the retirement unit. These three basic blocks of the processor are then simulated by three sets of simulation code: mase-fe, mase-exec, and mase-commit, respectively. In MASE, each portion of the processor can independently access memory through the cache hierarchy. In the case of a memory access that misses the on chip cache hierarchy, a memory transaction request is created and sent to the bus interface unit (BIU)[1]. Mase-fe, the front end of the processor, would stall completely in the case of an instruction cache miss, but mase-exec could generate multiple outstanding memory references concurrently and continue simulation of the out of order execution core as long as it has instructions not dependent on the data from an outstanding memory request. Similarly, mase-commit, the in-order backend of the

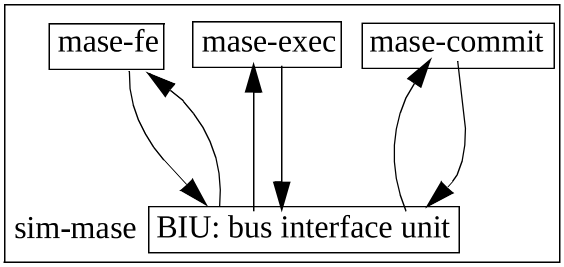


Figure 8.5: MASE Simulator Structure.

status	rid	start_time	address	access_type
Valid	0	54	0xXXXX	I Fetch
Invalid	-1	——	——	——
Valid	-1	14	0xXXXX	D Write
Valid	0	36	0xXXXX	D Read
Invalid	-1	——	——	——
Invalid	-1	——	——	——

Figure 8.5b: Bus Interface Unit Data Structure

[1] In this role, the BIU is functionally equivalent to a data structure of miss status handling registers (MSHRs)

processor could generate memory write requests independently from mase-fe and mase-exec. In this manner, MASE provides the framework for a workload to generate and sustain multiple, concurrent memory transactions to the memory system.

In MASE, pending memory references are tracked through the BIU that keeps track of the state of all active memory transactions in the system. An abbreviated view of the internal data structure of the BIU is shown as Figure 8.5b. For each transaction, the bus interface data structure keeps track of the requesting functional unit, the requesting processor (in case of multiple processors), the request time, the address of the request, and the type of the request. When a memory transaction is generated, the processor places the request into the bus interface data structure. The DRAM memory system is assumed to exist in a separate timing domain but operates concurrently with the processor. The DRAM memory system simulator then selects transactions from the BIU and simulate the transaction through the DRAM memory system on a cycle by cycle basis. The DRAM memory system simulates the progress of every memory transaction, then updates the status of the memory transaction request in the BIU as it completes the simulation of the given transaction. In this manner, the processor simulation is completely de-coupled from the DRAM memory system simulation.

The MASE simulation code described here has been enhanced to include a realistic, cycle accurate, and user configurable DRAM memory system. The DRAM memory system simulator simulates the timing of memory transactions subject to the type, speed, configuration, and the state of the memory system. The DRAM memory system in our simulation framework consists of a bus interface unit (BIU), one or more transaction-driven memory controllers, and one or more command-driven memory systems. In Figure 8.6, we

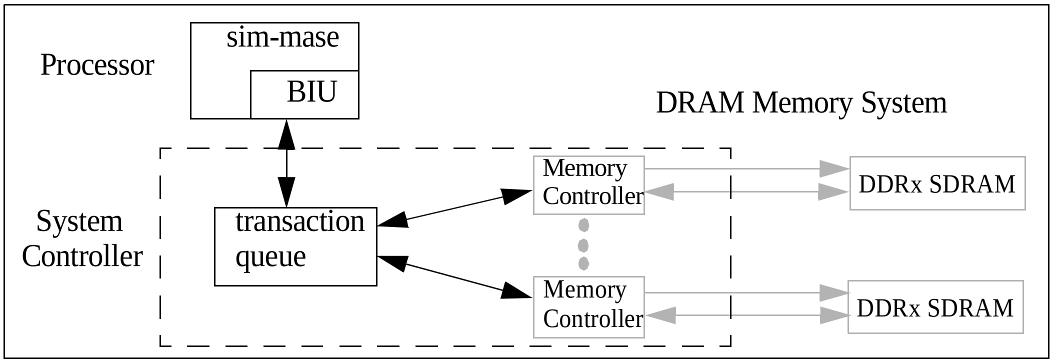


Figure 8.6: : Memory System Enhancement to MASE.

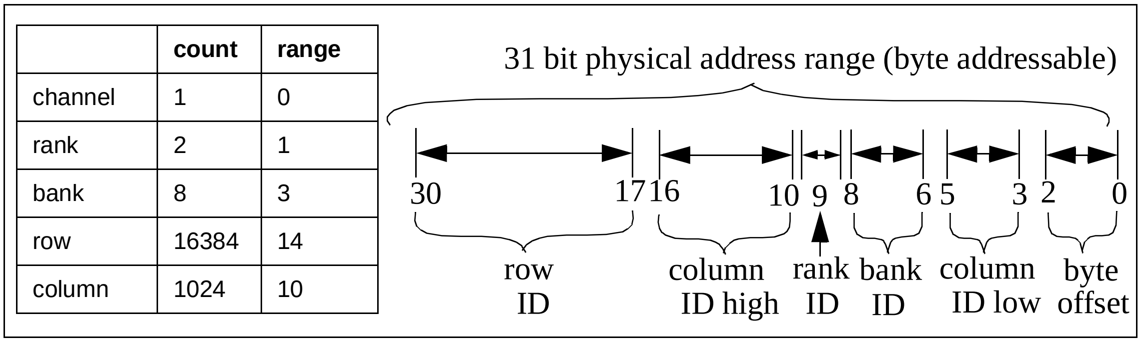
illustrate a symbolic representation of the simulation framework in this study. In this simulation framework, the processor or processors generate memory references that are then held in the BIU as transactions. The memory system then selects pending memory transactions for processing based on a transaction ordering policy. Each memory transaction is then converted to a sequence of DRAM commands and simulated based on the configuration, state, and timing of the DRAM memory system. The DRAM simulator models SDRAM, DDR SDRAM DDR2 and DDR3 SDRAM memory systems. In this simulation framework, all DRAM timing parameters summarized in table 8.1including t_{CAS}, t_{RAS}, t_{RP}, t_{DQS} and t_{FAW} are accurately simulated according to the DRAM access protocols specified in respective DRAM device datasheets[38]. Furthermore, the DRAM timing parameters can be easily adjusted through the use of a configuration file. The same configuration file also specifies the column, row, bank, rank and channel configuration of the memory system that the simulator need to generate the proper DRAM command sequences and accurately simulates timing of the DRAM commands.

8.4.2 System Configuration

In this study, processor and memory system configurations are kept as constants. The only two variables that differentiates between the various systems are the DRAM transaction ordering policy and the DRAM-command-scheduling algorithm. In all configurations, the L2 cache size of the processor is configured as 256 KB, and the processor frequency is set to 5 GHz. The memory system consists of a single 8 byte wide channel of 1 Gbps DDR3 SDRAM devices, and the peak bandwidth of this memory system is 8 GB/s. The DRAM memory system consists of 2 ranks of memory devices, each rank consists of 8 separate 1 Gbit DDR3 SDRAM devices connected in parallel, and there are 8 banks per rank, 16384 rows per bank and 1024 columns per row. Collectively, the two ranks of DDR3 SDRAM devices form a memory system with 2 GB of memory that is accessible by a 31 bit physical address space.

8.4.3 Address Mapping and Row Buffer Management Policy

The importance of address mapping has been examined some detail in previous literature [24,25,27,47,48]. In a memory system that implements the open-page row buffer management policy, the role of the address mapping scheme is to optimize the temporal and spatial locality of the address request stream and direct memory accesses to an open DRAM row (bank) and minimize DRAM bank conflicts. However, in a memory system that implements the close-page row-buffer management policy, the goal of the address mapping scheme is to minimize temporal and spatial locality to any given bank and instead distribute memory accesses throughout different banks in the memory system. In this manner, the DRAM memory system can avoid memory accesses to the same bank of memory and instead focus on transaction and DRAM command ordering algorithms that rotates through all available DRAM banks to achieve maximum DRAM bandwidth. In this study, a simple address mapping scheme optimized for close-page memory systems is used, and the address mapping scheme is summarized in Figure 8.7. In the address mapping scheme illustrated in Figure 8.7, the three lowest bits in a physical address is translated as the byte offset in the 8 byte wide channel, and the next three lowest address bits denote the three lowest bits of the column address. In essence, these three bits also represents the cacheline offset, since the size of the cacheline used in the system is 64 bytes. Address bits 6~8 then denote the bank ID of the memory address, and the 9th address bit then denotes the rank ID. In this manner, an application that streams through a large array will have consecutive cachelines that reside in different banks, and the memory system can optimally pipeline the consecutive memory accesses to different banks with maximum bandwidth efficiency.



	count	range
channel	1	0
rank	2	1
bank	8	3
row	16384	14
column	1024	10

Figure 8.7: Close-page-optimal address mapping for 2 GB DDR3 SDRAM memory system.

8.4.4 Structural Enhancement to Bus Interface Unit

The DRAM memory system enhancement in MASE has been previously described to contain the BIU, one or more transaction driven memory controllers and one or more command driven memory systems. In this simulation framework, a memory access that misses the cache is sent to the BIU, and the BIU can hold up to 256 memory references to the memory system. The pending memory references are then sent to the transaction queue according to the transaction ordering policy specified. However, in a traditional system topology, the BIU is a separate structure that resides separately from the memory controller and the address mapping stage that converts physical addresses to DRAM addresses occurs after the memory transaction is moved into the transaction queue in the memory controller. As a result, the BIU in a traditional system topology is not aware of the memory addresses of pending transactions, and it cannot prioritize memory references based on the rank or bank address of pending transactions. In this work, we assume that the memory controller is integrated with the processor (a common practice in modern processors), and the memory address mapping stage can be moved into the BIU. In this scheme, the BIU contains a copy of the address range registers and it is aware of the configuration of the DRAM memory system. In this scheme, as a memory transaction is placed into the BIU, the physical address is immediately translated into respective memory addresses. With the respective memory addresses, the BIU can then prioritize memory transactions based on the type as well as the memory addresses of pending transactions in the system.

8.4.5 Write Sweeping

In this work, considerable effort is devoted toward the optimization of DRAM bandwidth utilization in a DDRx SDRAM memory system. Various schemes have been devised and timing diagrams have been shown that maximize DRAM memory bandwidth. Implicit in the various schemes and timing diagrams is the assumption that the illustrated access sequences consist purely of memory read transactions. The reason that the various schemes and timing diagrams consist of purely memory read transactions is that in DDRx SDRAM memory systems, the bandwidth overheads for read-write turnarounds are even greater than the overhead imposed on DRAM memory system by t_{DQS}, t_{RRD} and t_{FAW}[1].

Figure 8.8 illustrates the case where a column-read command follows a column write command to the same rank of DRAM devices and shows that this combination of DRAM commands can at best be scheduled with the minimum timing of $t_{CWD} + t_{Burst} + t_{WR} - t_{CMD}$. In this case, the data bus can utilized for t_{Burst} time period by the write command, so the overhead for a read command that follows a write command to the same rank of memory is $t_{CWD} + t_{WR} - t_{CMD}$. Using timing parameters values specified in table 8.1, the bandwidth overhead for write-read turnaround is 16 ns. The value of 16 ns for the write-read turnaround is greater than other bandwidth overheads examined in this study. To alleviate the bandwidth overhead of read-write and write-read turnarounds in the DRAM memory system, some transaction ordering policies in this study utilize the technique of write sweeping. That is, the BIU acts as a write buffer for pending memory write transactions, and as much as possible, DRAM write requests are scheduled consecutively to the memory system. In this manner, pending write transactions are occasionally swept as a group into the memory system, and the number of read-write turnarounds are minimized.

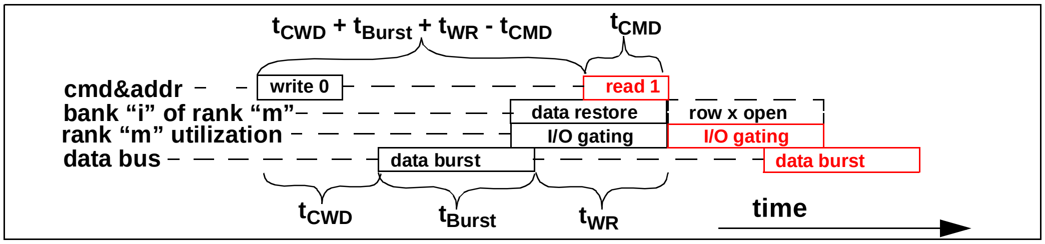


Figure 8.8: : Read Command following Write Command to Same Rank.

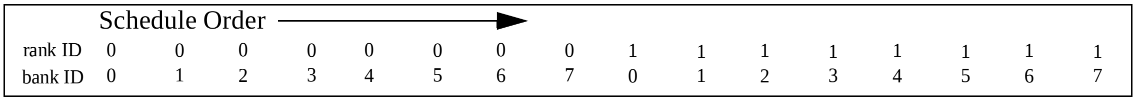
[1] Direct RDRAM devices have write buffers designed to alleviate the read-write turnaround overhead in high datarate memory systems. Data for column write commands are temporarily stored in the write buffer then retired into the DRAM array via a separate command. The write buffer minimizes but does not completely eliminate the read-write turnaround overhead.

8.4.6 Transaction Ordering Policy

To keep the DRAM memory system optimally saturated with DRAM commands to the appropriate banks and ranks, a transaction ordering policy must select the appropriate transactions with the proper rank and bank addresses from the BIU to send to the transaction queue. As a result, the transaction ordering policies should match the DRAM command scheduling algorithm to keep the memory system operating at maximum efficiency. Three different DRAM transaction ordering policies and the associated DRAM command scheduling algorithm are compared against each other in this study: *First Come First Serve, Bank Round Robin, and Command Pair Rank Hopping*.

In the *First Come First Serve* transaction ordering policy, hereafter referred to as the FCFS transaction ordering policy, the BIU buffers all memory transactions, but sends transactions to the DRAM memory system in the order of arrival regardless of the address or type of the memory transaction request. The DRAM memory system simulator allows different DRAM commands from different transactions to be pipelined, but DRAM commands of the same type cannot be re-ordered in the transaction queue. That is, while the column-read command from a prior memory transaction is still waiting in the transaction queue for a row to be opened, a separate row activation command from a subsequent transaction can be engaged, subject to the constraints of t_{RRD} and t_{FAW}. However, the column access command of the subsequent transaction cannot be scheduled ahead of the column access command of the prior transaction in this scheduling algorithm.

In the *Bank Round Robin* transaction ordering policy, hereafter referred to as the BRR transaction ordering policy, the BIU buffers all transactions, and memory read transactions are given priority and scheduled to the DRAM memory system according to the type and bank address of the pending request. Figure 8.9 shows the address sequence for the BRR



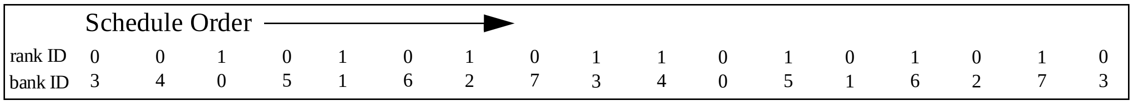
Schedule Order →

rank ID	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
bank ID	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7

Figure 8.9: : Address Sequence for Bank Round Robin Transaction Ordering Policy.

transaction ordering policy where the bus interface unit schedules transaction to the memory system by selecting the oldest memory transactions with specific rank and bank addresses. In case that there are no pending memory read transactions in the BIU or in case that nearly all 256 entries of the BIU are filled with pending transactions, write sweeping is triggered to move pending write transactions to the DRAM devices. In the BRR transaction ordering policy, DRAM commands are also scheduled in strict ordering, and DRAM commands of the same type from different transactions are not re-ordered in the transaction queue.

In the *Command Pair Rank Hopping* transaction ordering policy, hereafter referred to as the CPRH transaction ordering policy, the BIU also buffers all transactions, and memory read transactions are given priority and scheduled to the DRAM memory system according to the bank address of the pending request. However, in order to enable the DRAM command sequence and timing diagram as illustrated in Figure 8.3 and Figure 8.4, the CPRH transaction ordering policy must send a stream of memory transactions whose address sequence is not readily intuitive from the BIU to the transaction queue. Figure 8.10



Schedule Order →

rank ID	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
bank ID	3	4	0	5	1	6	2	7	3	4	0	5	1	6	2	7	3

Figure 8.10: : Address Sequence for Rank Hopping Transaction Ordering Policy.

shows the address sequence for the CPRH transaction ordering policy where the BIU schedules transaction to the transaction queue by selecting the earliest pending memory transaction of the appropriate type with specific rank and bank addresses. Similar to the BRR transaction ordering policies, in case that there are no pending memory read transactions in the BIU or in case that nearly all 256 entries of the BIU are filled with pending transactions, a burst of write sweeping is triggered. In this study, the CPRH transaction ordering policy is paired with the CPRH DRAM scheduling algorithm, and the DRAM command sequence is as previously described in Figure 8.4.

8.4.7 Workloads

In this study, a random sampling of workloads from the SPEC CPU 2000 benchmark suite are used to validate the proposed design. A range of different applications that exhibit different memory bandwidth utilization rates are used to characterize the impact of the transaction reordering algorithm on different memory request patterns and request rates.

Table 8.2 summarizes the workloads used in this study.

Workload Name	Description
164.gzip	popular data compression program written by Jean-Loup Gailly for the project. CPU 2000 INT
176.gcc	C compiler program. CPU 2000 INT
255.vortex	single user object oriented database transaction benchmark. CPU 2000 INT
256.bzip2	another compression program. CPU 2000 INT
172.mgrid	multigrid solver of 3D potential field CPU 2000 FP
173.applu	Partiall differntial equation algorithm CPU 2000 FP
177.mesa	graphic routine, creating a 3D object from a 2D scalar field. CPU 2000 FP
178.galgel	computational fluid dynamics. CPU 2000 FP
179.art	neural networks-object recognition CPU 2000 FP
183.equake	quake simulation algorithm CPU 2000 FP
188.ammp	computational chemistry. CPU 2000 FP
300.twolf	VLSI placement and routing algorithm CPU 2000 INT

TABLE 8.2: **SPEC CPU 2000 workloads used in study**

8.5 Simulation Results

In this work, all of the workloads are simulated through the execution of two billion instructions. Each workload is simulated with three different sets of memory systems that are identical in terms of the paging policy, address mapping scheme, the DRAM memory system configuration. The only difference that exists between the different memory systems is the memory transaction and DRAM command scheduling algorithms described in previous sections.

8.5.1 Improvement in Sustained Bandwidth

In this work, different algorithms that impact the maximum sustainable bandwidth of the DRAM memory system are examined. We expect that the performance impact of the respective algorithms on each workload will depend on the bandwidth utilization of the workload, so we present the average sustained bandwidth of the each workload through the simulated execution of 2 billion instructions in Figure 8.11. The average sustained

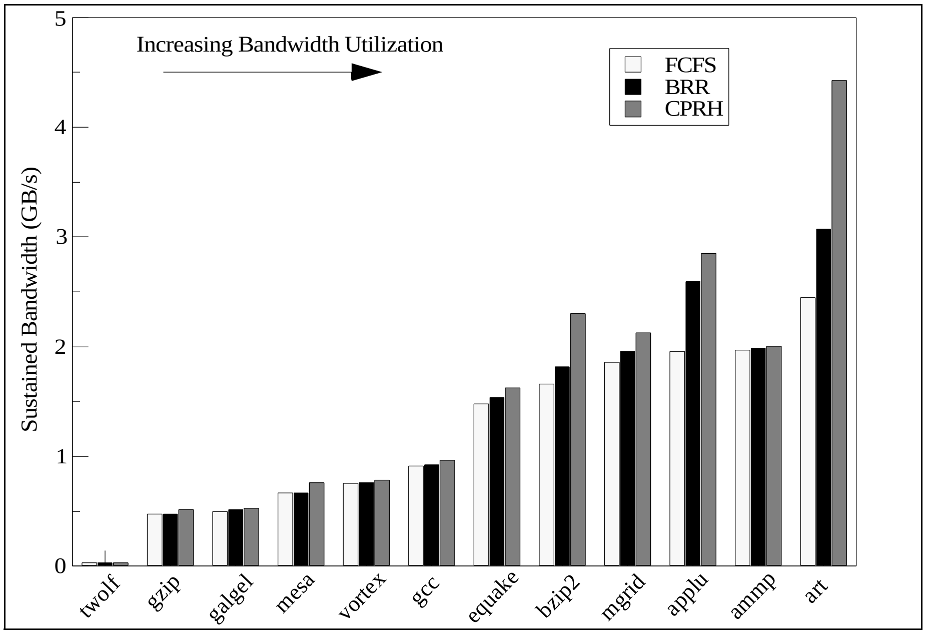


Figure 8.11: : Average Sustained Bandwidth through 2 Billion Instructions.

bandwidth for each workload is obtained by counting the total number of memory transactions processed through the simulation of 2 billion instructions, multiplying through by the number of bytes per transaction and the simulated processor frequency, then dividing through by the number of simulated processor cycles. Figure 8.11 shows that the average sustained bandwidth for workloads used in this study ranges from 0.5 GB/s to 4.5 GB/s. Figure 8.11 also shows that three workloads with the greatest improvements in bandwidth utilization from the CPRH scheduling algorithm are among the five workloads that already utilize high memory bandwidth. Finally, Figure 8.11 also shows that not all bandwidth intensive workloads benefit greatly from the CPRH scheduling algorithm.

8.5.2 Workload Speedups

Figure 8.12 shows the IPC speedups of the BRR and CPRH scheduling algorithms with respect to the in-order FCFS scheduling algorithm. The workloads are arranged by the order of their respective bandwidth utilization. In three workloads, 300.twolf, 164.gzip and 177.mesa, the BRR scheduling algorithm showed no appreciable speedup relative to the

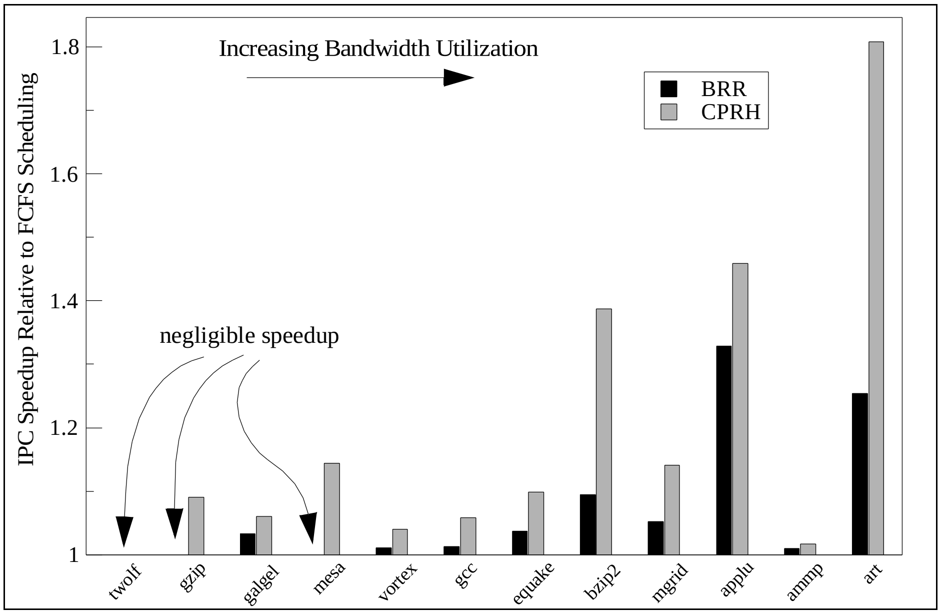


Figure 8.12: : IPC Speedup of BRR and CPRH Scheduling Relative to FCFS Scheduling .

FCFS scheduling algorithm. On the other hand, 179.art showed an 80% speedup for the CPRH scheduling algorithm relative to the FCFS scheduling algorithm.

Figure 8.13 shows the IPC speedup of the CPRH scheduling algorithm compared to the moderately intelligent BRR scheduling algorithm. The maximum bandwidth advantage of the CPRH scheduling algorithm compared to the BRR scheduling algorithm has been previously computed to be 47%. In Figure 8.13, 179.art shows a 44% speedup in IPC in using the CPRH algorithm compared to the BRR algorithm, and 188.ammp as well as 300.twolf report negligible speedups for the CPRH scheduling algorithm over the BRR algorithm.

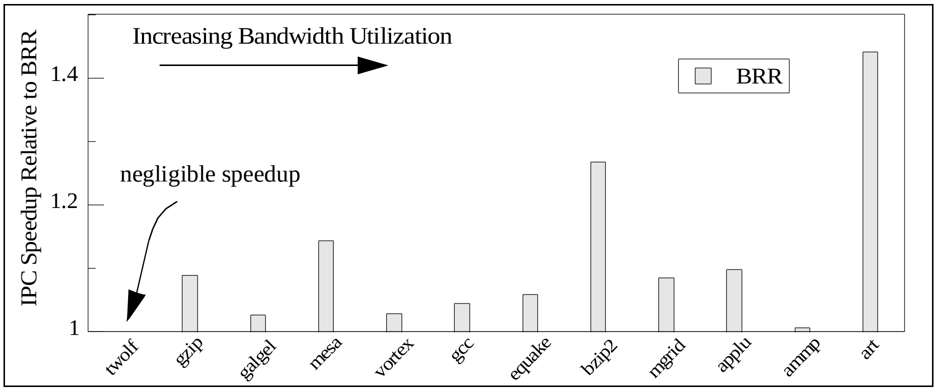


Figure 8.13: : IPC Speedup of CPRH Scheduling Relative to BRR Scheduling .

8.5.3 Memory Access Latency Distribution

In modern uni-processor and multi-processor systems, multiple memory transactions may be sent to the memory system concurrently. In case that the memory system is not immediately available to service a memory transaction, or if a memory transaction is deferred to allow a later transaction to proceed ahead of it, the latency of the later transaction will decrease at the expense of the increased latency of the prior memory transaction. However, if the transaction or DRAM command re-ordering algorithm results in a more efficient utilization of the memory system, then the average memory access latency for all memory transactions will decrease. Figure 8.14 shows the impact of the CPRH scheduling algorithm on the memory access latency distribution for the 179.art through 2 billion instructions. The memory access latency distribution illustrated in Figure 8.14 is obtained by a mechanism that records the access latency for each memory transaction in the BIU. In the simulation framework, each time a memory transaction is sent to the BIU, the start time of the transaction is recorded by the BIU. Upon completion of the memory transaction, the BIU simply computes the latency and keeps track of the number of transactions for each a specific latency value.

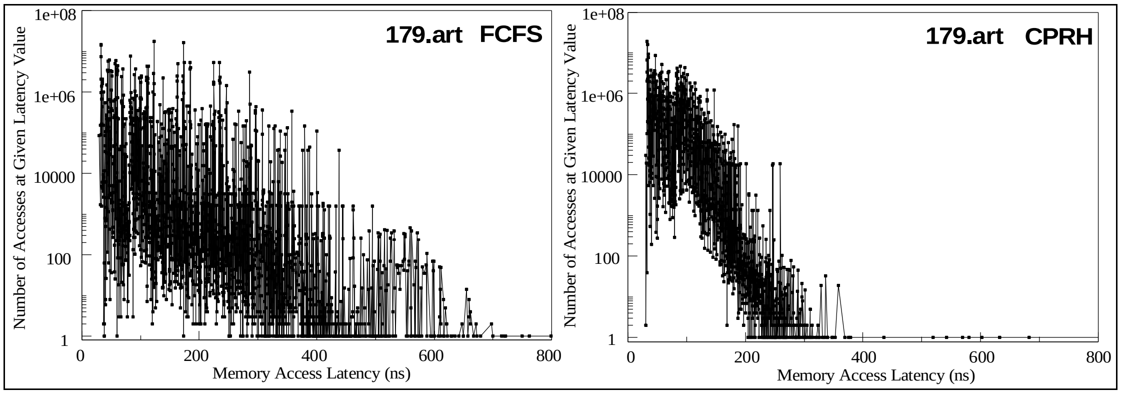


Figure 8.14: : Impact of Scheduling Policy on Memory Access Latency Distribution: 179.art.

In the simulated memory system, the minimum latency of a memory transaction is simply the delay through the BIU added to the delay of the memory controller and the minimum DRAM latencies of t_{RCD} + t_{CAS}. In the simulated memory system, the delays through the BIU and memory controller is set to 10 ns, and the minimum access latency is approximately 30 ns for the set of timing values used in this study and illustrated in Figure 8.14. Figure 8.14 shows that the CPRH scheduling algorithm greatly decreases the queueing delay for many pending memory transactions in 179.art, and the number of transactions with memory access latency greater than 400 ns is significantly less than the same workload operating with the FCFS scheduling algorithm.

In Figure 8.14, the memory access latency distribution curve graphically illustrates the benefits of the CPRH algorithm for 179.art. However, just as the memory access latency distribution curve can be used to illustrate the benefit of the CPRH scheduling algorithm, it can also be used to illustrate possible problems with the CPRH scheduling algorithm for other workloads. Figure 8.15 shows the latency distribution curve for 188.ammp, and 188.ammp was one workload that points to possible issues withe the CPRH algorithm. That is, Figure 8.11 shows that 188.ammp was the second most bandwidth intensive workload with the FCFS scheduling algorithm, but aside from the non-bandwidth intensive 300.twolf, 188.ammp also saw the smallest speedup for all workloads with the CPRH scheduling algorithm. Figure 8.15 shows that the CPRH scheduling algorithm resulted in longer latencies for a number of transactions, and the number of transactions with memory access latency greater than 400 ns actually increased. Figure 8.15 also shows that the increase of the small number of transactions with memory access latency greater than 400 ns is offset by the reduction of the number of transactions with memory transaction latency around 200 ns and the increase of the number of transactions with memory access latency less than 100 ns.

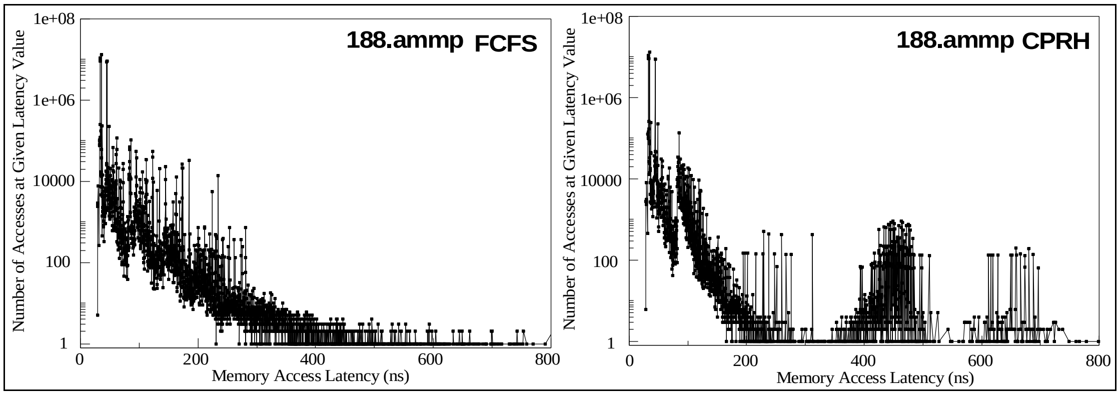


Figure 8.15: Impact of Scheduling Policy on Memory Access Latency Distribution: 188.ammp.

In other words, the CPRH scheduling algorithm redistributed the memory access latency curve so that most memory transactions received a modest reduction in access latency, but a few memory transaction suffered a substantial increase in access latency. The net result is that the changes in access latency cancelled each other out, and 188.ammp only shows a minor speedup for the CPRH algorithm over the FCFS or the BRR algorithm.

8.6 Quick Summary of the Rank Hopping Algorithm

Power consumption and heat dissipation considerations are constraining high performance DRAM memory systems just as they are constraining high performance processors. The combination of power limitation and data bus synchronization constraints limits available memory bandwidth in DDR2 and future DDR3 SDRAM memory systems that do not adequately account for these bandwidth constraints. This work presented a unique memory transaction ordering policy and DRAM command-scheduling algorithm that maximizes sustainable bandwidth of the memory system while operating within power and system synchronization constraints of DDR2 and DDR3 SDRAM devices. In a 1 Gbit DDR3 SDRAM memory system used as the baseline in this study, the resulting DRAM command ordering algorithm increases the maximum sustainable bandwidth by more than 41% compared to a moderately intelligent memory system. Simulations with the projected timing parameters show that the proposed algorithm increases IPC by 40~80% on a bandwidth intensive workload, 179.art, over a baseline of an unintelligent memory system, and over 40% with the baseline of the moderately intelligent memory system. Moreover, two different scaling trends means that the scheduling algorithm proposed in this work will become even more important as process scaling continues in the future. The first trend that favors the proposed scaling algorithm is that as processor frequencies and DRAM device data rates increase, the power limitation constraints will likely remain in place or increase at a much lower rate. The result is that row activations must be scheduled farther apart from each other in terms of number of cycles, and the proposed scheduling algorithm allows the row activation commands to be scheduled farther apart in a given rank without impacting the scalability of maximum bandwidth efficiency as long as t_{RRD} does not exceed 2 * t_{Burst}, or t_{FAW} does not exceed 8 * t_{Burst}. The second trend that favors the proposed scheduling algorithm is that as transistor budgets continue to grow, the trend toward multi-threaded cores and chip-level multiprocessors appears to be inevitable. Memory request streams from these processors will have higher access rates and less spatial locality compared to memory request streams from traditional uniprocessors. The higher access rate will require more bandwidth per pin from the memory system, and the decreased spatial locality property means an increase in the number of row cycles per transaction, even in open-page DRAM memory systems. Both effects of the multi-threaded and multi-processor system increase the importance of a close-page, bandwidth optimized DRAM transaction and command scheduling algorithms such as the one proposed in this work.

CHAPTER 9 *Concluding Remarks*

The work performed in this dissertation is devoted to answering questions that are fundamental to understanding the performance of modern DRAM memory systems. In particular, the following questions were asked and answered in this work:

- How do changes in system configuration impact the performance of DRAM memory systems?
- How would the performance characteristic of the DRAM memory system change in relation to scaling trends of DRAM device datarates and row cycle times?
- What can be done to alleviate constraints that limit the efficiency of modern DRAM memory systems, such as t_{FAW} and t_{DQS}?

Throughout the text of this dissertation, answers to each of these questions have been gradually revealed. In this final chapter, the answers to these questions are summarized, the significance of the work is highlighted, related work is addressed, and directions of future research are sketched out.

9.1 Summary and Contributions

In this dissertation, the foundation to facilitate future research into DRAM memory systems performance analysis is laid. The definition of the abstract and generic DRAM access protocol that is common to SDRAM, DDR SDRAM, DDR2 SDRAM, and future

DDRx SDRAM variants means that DRAM performance analysis performed for each generation of DRAM devices can be compared directly to each other, subjected to the scaling considerations of changing datarates and row cycle times.

The definition of the generic DRAM access protocol in turn enables the description of high level analytical frameworks that can be used for performance analysis of DRAM memory systems. In this work, the Request Access Distance methodology for the computation of sustainable memory bandwidth is described in detail. The Request Access Distance methodology is then used to extensively analyze the performance characteristics of modern DRAM memory systems that will operate at datarates of 1 Gbps and above. This work advances the state of the art in terms of an analytical method that can be applied to a DRAM memory system to understand its performance characteristics, and it supports that work with extensive studies that aid in the understanding of modern DRAM memory systems. In particular, we show that the addition of more banks in DRAM memory systems can improve performance by an average of 18% for both close-page and open-page memory systems. We also show that the t_{FAW} bank activation constraint is critical for close-page memory systems, and less so for open-page systems, although its impact does grow with increasing datarate.

Finally, this work presents a novel DRAM command scheduling algorithm that amortizes the overhead costs of rank-switching time and schedules around the t_{FAW} bank activation constraint. We show that the scheduling algorithm can improve maximum bandwidth by 41%, and it can improve performance by as much as 40% for some specific workloads.

9.2 Limitations

The ultimate metric of DRAM memory system performance is related to how fast it can service critical requests from processors. However, the focus of this work is to characterize and improve sustainable bandwidth of a DRAM memory system, not to identify and service critical requests. The rationale used to justify the focus of this work is that by improving the sustainable bandwidth of the DRAM memory system, the average request service time can be reduced. The study performed in chapter 8 shows that while this strategy does benefit most workloads, it does not benefit all workloads. The transaction re-ordering policy used by the rank-hopping algorithm, designed to improve sustainable bandwidth of the DRAM memory system, unintentionally deferred some critically needed requests at the expense of non-critical requests. The result is that the net performance improvement was negligible despite the improvement of available bandwidth from the DRAM memory system.

The study of the rank-hopping algorithm points out that an ideal DRAM memory scheduling algorithm must be coupled with the identification of critical loads within request sequences. In the case that the DRAM memory system is given that information, it can then dynamically adjust the scheduling algorithm to further improve system performance depending on the request pattern. In a recent study, an adaptive feedback memory system that dynamically adjusted DRAM memory scheduling algorithm based on processor request pattern is described[51]. However, that study utilizes a relatively simplified model of a multi-rank DDR SDRAM memory system. We believe that future DRAM memory systems study should proceed along parallel paths, so that the needs of the processor is accounted for while scheduling around DRAM hazards such as row cycle times, t_{FAW} bank activation constraints, and t_{DQS} rank-switching overheads.

9.3 Related Work

The traced-based Request Access Distance analytical framework is a unique methodology in analyzing DRAM memory system performance in terms of citations that can be traced in previously published literatures. However, a comparable framework may exist in work performed independently by Pawlowski and presented in a tutorial session at HPCA in 2005[54]. In the tutorial session, simulation results were presented that compared the bandwidth efficiency of various existing memory systems subjected to randomly generated sequences of read and write access patterns. It can be speculated from the presented results that the fundamental methodology shares fundamental similarities with the Request Access Distance methodology described in this work. However, due to the fact that the underlying framework for the results presented in the tutorial was not disclosed, a direct comparison against the Request Access Distance analytical framework was not possible.

This work also presented a unique DRAM scheduling algorithm that performs memory access sequences in specific ordering to maximize available DRAM bandwidth. Although the scheduling algorithm is itself unique, much work already exist in the field of DRAM memory scheduling algorithm. In particular, works by Briggs et. al., Cuppu et. al., Hur et. al., McKee et. al., and Rixner et. al are particularly instrumental in advancing the state of the art in DRAM memory scheduling algorithms[28,32,33,34,37,42,51]. This work advances the state of the art by proposing a scheduling algorithm that is unique to high performance, high datarate, short channel DDRx SDRAM memory systems.

9.4 Future Work

The studies performed in this work can be extended in many different directions. In particular, the following directions are particularly interesting and they are currently under consideration for specific future research directions.

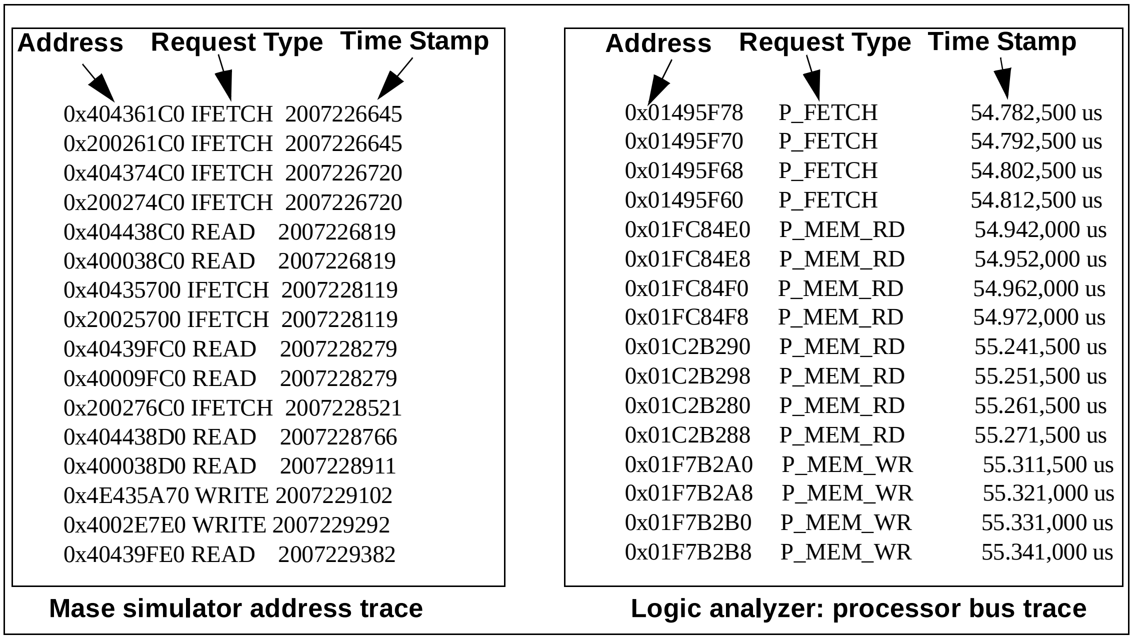
- Extend the abstract protocol to cover high performance DRAM devices such as FCRAM, RLDRAM and XDR DRAM. The extension of the abstract protocol in turn will enable the computation of the table of DRAM protocol overheads. The creation of the table in turn enables the extension of the Request Access Distance analytical framework to those memory systems.
- Explore adaptive DRAM memory scheduling algorithms that simultaneously account for the effects of DRAM system inefficiencies as well as processor request priority. In particular, a study of adaptive scheduling algorithms for a processor that can send along priority information to aid the DRAM controller in selecting amongst transaction re-ordering algorithms that are not based solely on recent request access histories.
- The Request Access Distance method reports the sustainable DRAM bandwidth in terms of in-efficiency. As described in this text, the inefficiencies are computed in terms of the idle times that must be added to fully account for DRAM protocol overhead, DRAM row cycle time constraints and DRAM device power constraints. A future version of the simulation framework may be developed to keep track of the each set of causes of bandwidth loss separately so that the dominant causes can be identified and analyzed specifically.
- In this work, DRAM memory system performance is studied based on use of traces from single-threaded workloads. While the study does not lose relevance with the onset of multi-threaded (MT) and chip multiprocessors (CMP), the explicit examination of DRAM memory system performance characteristics subjected to MT and CMP processors will gain increasing relevance as these processors become mainstream.
- Finally, the development of the rank-hopping algorithm for MT and CMP processors should be coupled with further enhancements, such as a pseudo-open-page optimized address mapping scheme and a distributed refresh mechanism to minimize the impact of refresh.

CHAPTER A

Appendix: Workload Descriptions

A.1 Trace Fundamentals

The performance analysis performed in this study relies on trace based workloads. This section summaries the traces used this study and provides a cursory examination into the characteristics of workloads used in this work. There are two types of traces used in this work. One type of traces consists of memory request sequences logged by using the **M**icro-**A**rchitectural **S**imulation **E**nvironment (MASE) from the University of Michigan [29]. The second type of traces used in this text consists of processor bus traces captured with a digital logic analyzer during the execution of workloads on a test computer system. Figure A.1 shows trace segments from both types of traces.


Address
Request Type
Time Stamp
Address
Request Type
Time Stamp
Mase simulator address trace
Logic analyzer: processor bus trace

```
0x404361C0 IFETCH  2007226645
0x200261C0 IFETCH  2007226645
0x404374C0 IFETCH  2007226720
0x200274C0 IFETCH  2007226720
0x404438C0 READ    2007226819
0x400038C0 READ    2007226819
0x40435700 IFETCH  2007228119
0x20025700 IFETCH  2007228119
0x40439FC0 READ    2007228279
0x40009FC0 READ    2007228279
0x200276C0 IFETCH  2007228521
0x404438D0 READ    2007228766
0x400038D0 READ    2007228911
0x4E435A70 WRITE 2007229102
0x4002E7E0 WRITE 2007229292
0x40439FE0 READ    2007229382
```

```
0x01495F78    P_FETCH       54.782,500 us
0x01495F70    P_FETCH       54.792,500 us
0x01495F68    P_FETCH       54.802,500 us
0x01495F60    P_FETCH       54.812,500 us
0x01FC84E0    P_MEM_RD      54.942,000 us
0x01FC84E8    P_MEM_RD      54.952,000 us
0x01FC84F0    P_MEM_RD      54.962,000 us
0x01FC84F8    P_MEM_RD      54.972,000 us
0x01C2B290    P_MEM_RD      55.241,500 us
0x01C2B298    P_MEM_RD      55.251,500 us
0x01C2B280    P_MEM_RD      55.261,500 us
0x01C2B288    P_MEM_RD      55.271,500 us
0x01F7B2A0    P_MEM_WR      55.311,500 us
0x01F7B2A8    P_MEM_WR      55.321,000 us
0x01F7B2B0    P_MEM_WR      55.331,000 us
0x01F7B2B8    P_MEM_WR      55.341,000 us
```

Figure A.1: Sample Memory Request Trace Segments.

Figure A.1 shows two trace segments from the two types of traces used in this study. Figure A.1 shows that each trace logged the address, type, and timestamp for each trace. In the address trace captured from MASE, each transaction request is recorded as a single event, and the time stamp for each request is recorded in the form of CPU cycle count. In the address trace captured by the digital logic analyzer on a processor bus, each entry of the trace segment represents an active cycle on the processor bus. In the test system, each cacheline is 32 bytes in length and each cacheline transaction occupies four bus clock cycles on the processor bus, and the cacheline transfer appears as four separate events in Figure A.1[1]. The timestamp logged by the digital logic analyzer also differs from the timestamp logged by the architectural simulator in that the digital logic analyzer records timestamp in terms of microseconds, and since the processor bus of the test system operates at 100 MHz, the timestamp increments with the minimum granularity of 10 nanoseconds for each event in the processor bus trace. Finally, one last difference between the two different types of traces is that traces created by the architectural simulator record virtual addresses as generated by the simulated workloads, whereas the processor bus traces record physical addresses as they appear on the processor bus. In the context of the performance analysis based on these traces, traces with virtual addresses may be subjected to additional virtual to physical address page mapping considerations whereas traces recorded with physical addresses need not deal with virtual to physical address mapping considerations.

[1] Some I/O transactions seen in the bus trace captured by the digital logic analyzer only occupies a single bus clock cycle, so not all transactions go through four bus clock cycles.

A.2 Description of Workloads

In this work, address traces from nine different SPEC benchmarks and four sets of processor bus traces were used. The address traces for the SPEC benchmarks were generated by MASE, and the processor bus traces were captured on a personal computer with an AMD K6-III processor while four different applications were running on the host system. Restricted by the relatively shallow memory buffer depth of the digital logic analyzer, traces captured through the use of the logic analyzer are relatively short trace segments. Each trace segment consists of approximately four million active processor bus cycles that represents approximately one million transaction requests. To ensure that the captured traces adequately represent the performance characteristics of the workload, multiple trace segments for each workload were captured for each workload. Rather than merging the independent trace segments into one large trace for a given workload, the results for each trace segment are presented individually to illustrate the degree of variance within each workload. The traces used in this study are as follows:

MASE Traces (SPEC CPU 2000 Benchmarks)

164.gzip, 176.gcc, 197.parser, 255.vortex.

MASE Traces (SPEC CPU 2000 FP Benchmarks)

172.mgrid, 178.galgel, 179.art, 183.equake, 188.ammp.

AMD K6 Processor Bus Traces

JMark 2.0 - CPU, AWT, Complex Mathematics: 3 Segments, 3DWinbench CPU: 1 Segment, SETI@Home: 3 Segments, Quake 3: 5 Segments.

A.2.1 164.gzip: C Compression

164.gzip is a benchmark in the SPEC CPU 2000 integer suite. 164.gzip is a popular data compression program written by Jean-Loup Gailly for the GNU project. 164.gzip uses Lempel-Ziv coding (LZ77) as its compression algorithm. In the captured trace for 164.gzip, four billion simulated instructions were executed by the simulator over two billion simulated processor cycles, and 2.87 million memory requests were captured in the trace. Figure A.2 shows that 164.gzip undergoes a short duration of program initialization, then quickly enters into a repetitive loop. Figure A.2 shows the memory system activity of 164.gzip through the first 1.8 billion processor cycles. Figure A.2 also shows that 164.gzip is typically not memory intensive, since it averages less than one memory reference per thousand instructions. Moreover, in the time periods when 164.gzip fills the memory system with transaction requests, the transaction requests appear to be bursts of memory write requests.

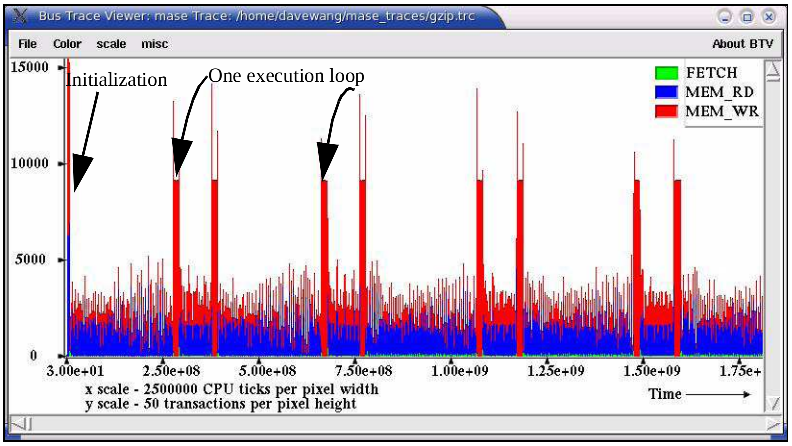


Figure A.2: : 164.Gzip trace overview.

A.2.2 176.gcc: C Programming Language Compiler

176.gcc is a benchmark in the SPEC CPU 2000 Integer suite that tests compiler performance. 176.gcc is based on gcc version 2.7.2.2 and generates code for a Motorola 88100 processor. In the captured trace for 176.gcc, 1.5 billion simulated instructions were executed by the simulator over 1.63 billion simulated processor cycles, and 4.62 million memory requests were captured in the trace. Unlike 164.gzip, 176.gcc does not enters into a discernible and repetitive loop behavior within the first 1.5 billion instructions. Figure A.3 shows the memory system activity of 176.gcc through the first 1.4 billion processor cycles. Figure A.3 shows that 176.gcc, like 164.gzip, is typically not memory intensive, although it does averages more than three memory references per thousand instructions. Moreover, in the time frame illustrated in Figure A.3, 176.gcc shows a heavy component of memory access due to instruction fetch requests, and relatively fewer memory write requests. The reason that the trace in Figure A.3 shows a high percentage of instruction fetch requests is that the trace was captured with the L2 cache of the simulated processor set to 256 KB.

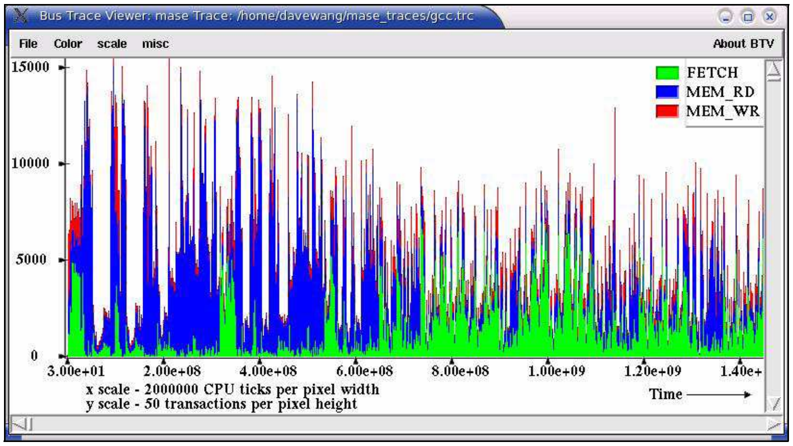

Bus Trace Viewer: mase Trace: /home/davewang/mase_traces/gcc.trc
File
Color
scale
misc
About BTV
15000
10000
5000
0
FETCH
MEM_RD
MEM_WR
3.00e+01
2.00e+08
4.00e+08
6.00e+08
8.00e+08
1.00e+09
1.20e+09
1.40e+
x scale - 2000000 CPU ticks per pixel width
y scale - 50 transactions per pixel height
Time

Figure A.3: : 176.gcc trace overview.

A.2.3 197.parser: C Word Processing

197.parser is a benchmark in the SPEC CPU 2000 integer suite that performs syntactic parsing of English, based on link grammar. In the captured trace for 197.parser, 4 billion simulated instructions were executed by the simulator over 6.7 billion simulated processor cycles, and 31.2 million requests were captured in the trace. Similar to 164.gzip, 197.gzip undergoes a short duration of program initialization, then quickly enters into a repetitive loop. However, 197.gzip enters into loops that are relatively short in duration and is difficult to observe in an overview. Figure A.4 shows the memory system activity of 197.parser through the first 6.7 billion processor cycles. Figure A.4 shows that 197.parser is moderately memory intensive, since it averages approximately eight memory references per thousand instructions. Figure A.5 shows that each loop lasts for approximately 6 million CPU cycles and the number of reads and write request are roughly equal.

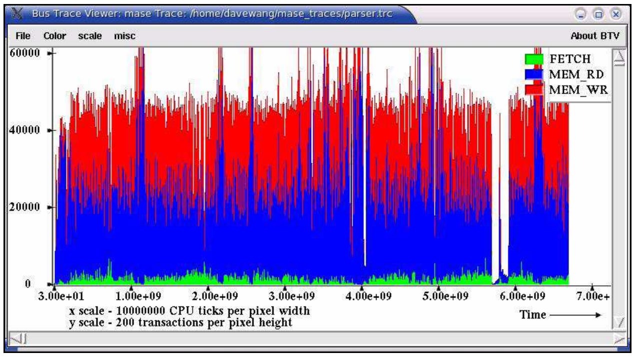


Figure A.4: : 197.parser trace overview.

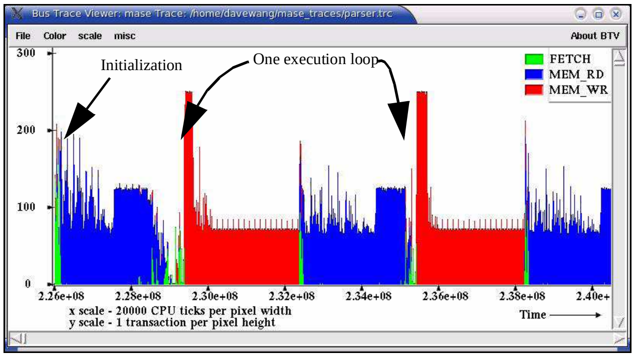


Figure A.5: : 197.parser trace view closeup.

A.2.4 255.vortex: C Object-oriented Database

255. vortex is a benchmark in the SPEC CPU 2000 Integer suite. In the captured trace for 255.vortex, 4 billion simulated instructions were executed by the simulator over 3.3 billion simulated processor cycles, and 7.2 million requests were captured in the trace. In the first 3.3 billion processor cycles, 255.vortex goes through several distinct patterns of behavior. However, after a 1.5 billion cycle initialization phase, 255.vortext appears to settle into execution loops that lasts for 700 million processor cycles each, and each loop appears to be dominated by instruction fetch and memory read requests with relatively fewer memory write requests. Figure A.6 shows the memory system activity of 255.vortex through the first 3.3 billion processor cycles. Figure A.6 also shows that 255.vortex is typically not memory intensive, since it averages less than two memory reference per thousand instructions.

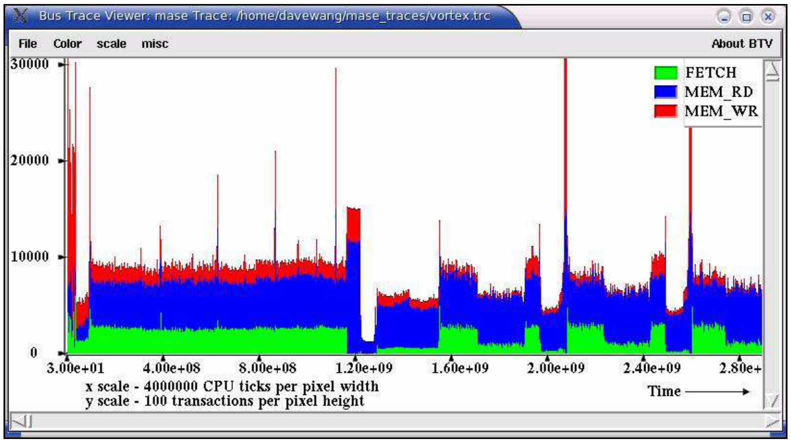

Bus Trace Viewer: mase Trace: /home/davewang/mase_traces/vortex.trc
File
Color
scale
misc
About BTV
30000
20000
10000
0
🟩 FETCH
🟦 MEM_RD
🟥 MEM_WR
3.00e+01
4.00e+08
8.00e+08
1.20e+09
1.60e+09
2.00e+09
2.40e+09
2.80e+
x scale - 4000000 CPU ticks per pixel width
y scale - 100 transactions per pixel height
Time

Figure A.6: : 255.vortex Overview.

A.2.5 172.mgrid: Fortran 77 Multi-grid Solver: 3D Potential Field

172.mgrid is a benchmark that demonstrates the capabilities of a very simple multigrid solver in computing a three dimensional potential field. It was adapted by SPEC from the NAS Parallel Benchmarks with modifications for portability and a different workload. In the captured trace for 172.mgrid, 4 billion simulated instructions were executed by the simulator over 9 billion simulated processor cycles, and 47.5 million requests were captured in the trace. 172.mgrid is moderately memory intensive, as it generates nearly twelve memory requests per thousand instructions. Figure A.7 shows that after a short initialization period, 172.mgrid settles into a repetitive and predictable loop behavior. The loops are dominated by memory read requests, and memory write requests are relatively fewer.

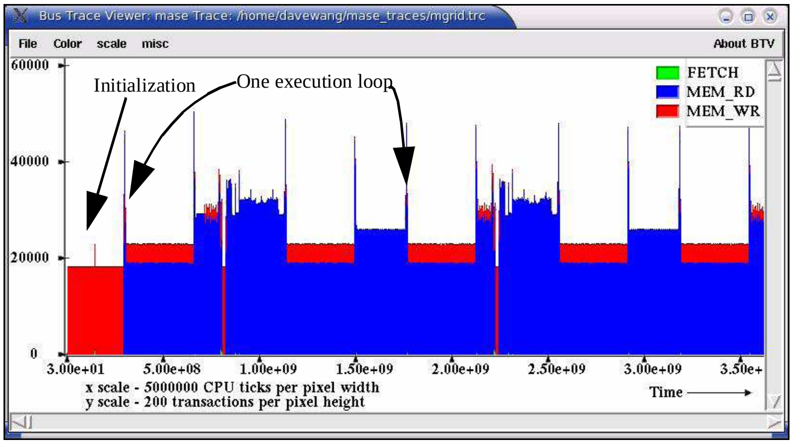


Figure A.7: : 172.mgrid trace overview.

A.2.6 178.galgel: Fortran 90 Computational Fluid Dynamics

In the captured trace for 178.galgel, 4 billion simulated instructions were executed by the simulator over 2.2 billion simulated processor cycles, and 3.1 million requests were captured in the trace. Relatively, 178.galgel is not memory intensive, as it generates less than one memory requests per thousand instructions. Figure A.8 shows that after a short initialization period, 178.galgel settles into a repetitive and predictable loop behavior. The loops iterations are bursty and goes through several different phases.

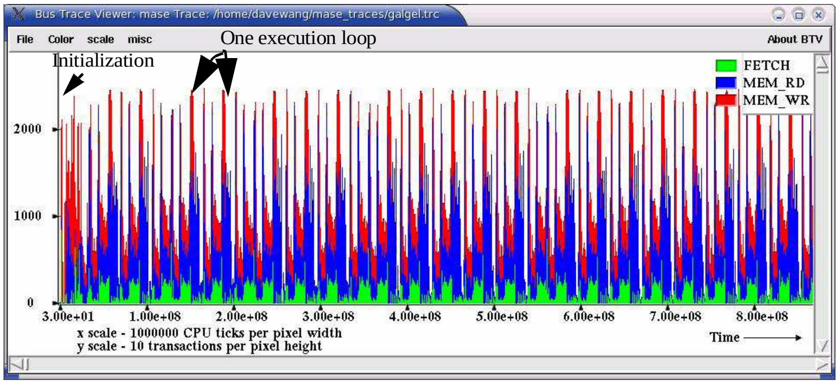
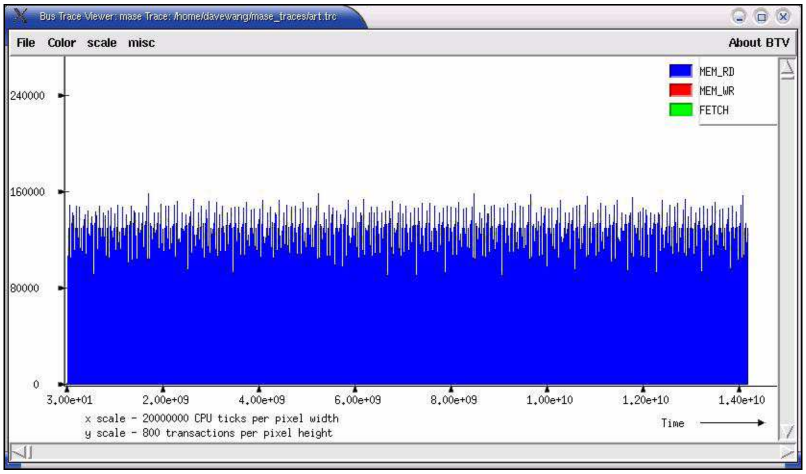


Figure A.8: : 178.galgel trace overview.

A.2.7 179.art (SPEC CPU 2000 FP Suite)

179.art is a benchmark derived from an application that emulates a neural network and attempts to recognize objects in a thermal image. In the captured trace for 179.art, 450 million simulated instructions were executed by the simulator over 14.2 billion simulated processor cycles, and 90 million requests were captured in the trace. 179.art is extremely memory intensive, and it generates almost two hundred memory requests per thousand instructions. Figure A.9 shows that 179.art is dominated entirely by memory read transactions.



x scale - 20000000 CPU ticks per pixel width
y scale - 800 transactions per pixel height

Figure A.9: : 179.art trace overview.

A.2.8 183.equake: C Seismic Wave Propagation Simulation

183.equake simulates the propagation of elastic waves in large, highly heterogeneous valleys. Computations are performed on an unstructured mesh that locally resolves wavelengths, using a finite element method. In the captured trace for 183.quake, 1.4 billion simulated instructions were executed by the simulator over 1.8 billion simulated processor cycles, and 7.9 million requests were captured in the trace. Figure A.7 shows that after a long initialization period, 183.equake settles into a repetitive and predictable loop behavior. The loops are dominated by memory read requests, and memory write requests are relatively fewer outside of the initialization phase. 183.equake is moderately memory intensive, as it generates almost six memory references per thousand instructions.

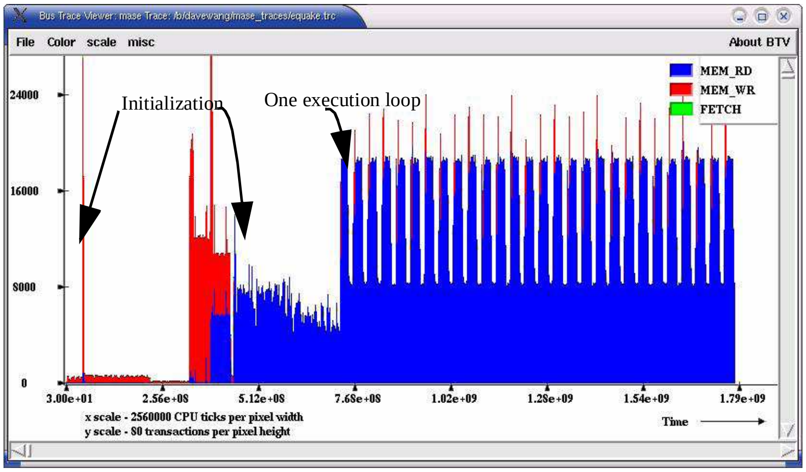


Figure A.10: : 172.mgrid trace overview.

A.2.9 188.ammp: C Computational Chemistry

188.ammp is benchmark in the SPEC CPU 2000 FP suite that runs molecular dynamics on a protein-inhibitor complex embedded in water. unstructured mesh that locally resolves wavelengths, using a finite element method. In the captured trace for 188.ammp, 4 billion simulated instructions were executed by the simulator over 10.5 billion simulated processor cycles, and 60 million requests were captured in the trace. Figure A.11 shows that 188.ammp is moderately memory intensive. It generates approximately 15 memory references per thousand instructions. 188.ammp is somewhat unique in that the rate of memory requests appears to follow a pattern, yet that pattern is not readily discernible in the trace that captures 10.5 billion processor cycles of execution time.

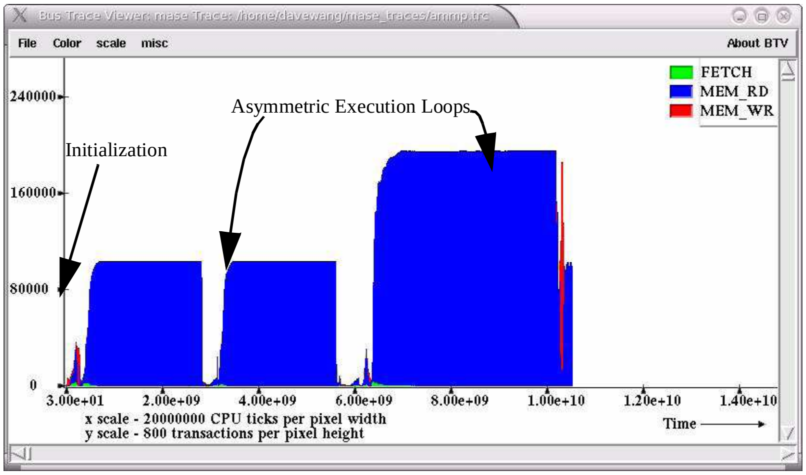

Bus Trace Viewer: mase Trace: /home/davewang/mase_traces/ammp.trc
File
Color
scale
misc
About BTV
FETCH
MEM_RD
MEM_WR
240000
160000
80000
0
Asymmetric Execution Loops
Initialization
3.00e+01
2.00e+09
4.00e+09
6.00e+09
8.00e+09
1.00e+10
1.20e+10
1.40e+10
x scale - 20000000 CPU ticks per pixel width
y scale - 800 transactions per pixel height
Time

Figure A.11: : 188.ammp trace overview.

A.2.10 JMark 2.0 - AWT, CPU and Complex Arithmetic

JMark 2.0 is a suite of benchmarks designed to test the performance of Java virtual machine implementations. The CPU, AWT and Complex Mathematics benchmarks are independent benchmarks in this suite of benchmarks. Compared to other workloads examined in this work, the benchmarks in JMark 2.0 accesses memory only very infrequently. Ordinarily, the relatively low access rate of these benchmarks would exclude them as workloads of importance in a study of memory system performance characteristics. However, the benchmarks in JMark 2.0 exhibit an interesting behavior in access memory in that they repeatedly access memory with locked reads and locked write requests at the exact same location. As a result, they are included for completeness to illustrate a type of workload that performs poorly in DRAM memory systems regardless of system configuration. Figure A.12 shows an overview of a trace segment from the Abstract Windowing Toolkit (AWT) benchmark trace.

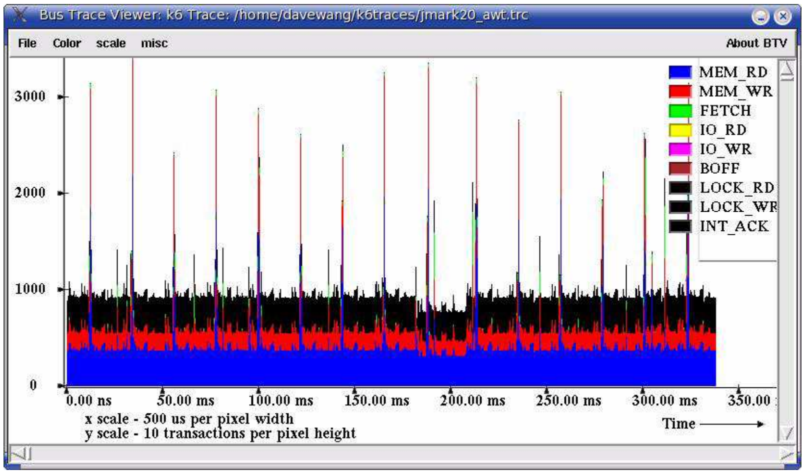
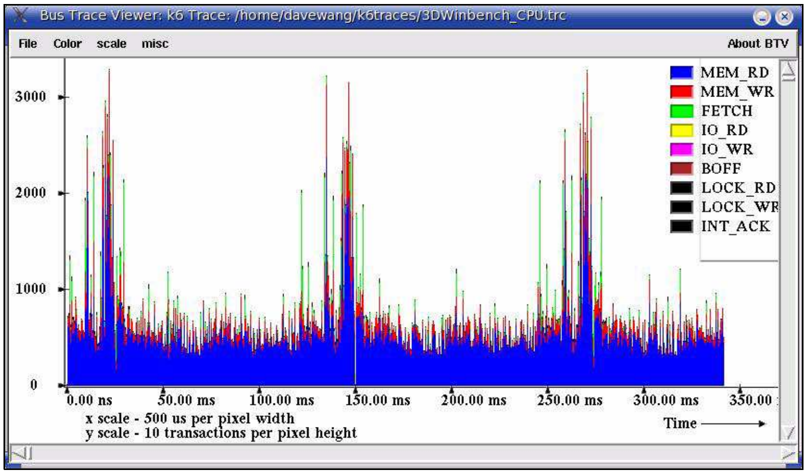


Figure A.12: : JMark Abstract Windowing Toolkit Benchmark trace overview.

A.2.11 3DWinbench - CPU

3D Winbench is another suite of benchmarks that is designed to test 3D graphics capability of a system. The CPU component tests the processor capability and it is moderately memory intensive. Figure A.13 shows an overview of the 3D Winbench trace.



Bus Trace Viewer: k6 Trace: /home/davewang/k6traces/3DWinbench_CPU.trc

x scale - 500 us per pixel width
y scale - 10 transactions per pixel height

Figure A.13: : 3D Winbench trace overview.

Figure A.13 shows that 3D Winbench achieves sustained peak rate of approximately 5 transactions per microsecond during short bursts, and it sustains at least 1 transactions per microsecond throughout the trace.

A.2.12 SETI@Home - 3 Segments

SETI@Home is a popular program that allows the SETI institute to make use of spare processing power on idle personal computers to search for signs of extraterrestrial intelligence. The SETI@HOME application performs a series of fast fourier transforms on captured electronic signals to look for existence of extraterrestrial intelligence. The series of FFT’s are performed on successively larger portions of the signal file. As a result, the size of the working set for the program changes as it proceeds through execution. Figure A.14 shows a portion of the SETI@HOME trace, and in this segment, the read and write transactions have an approximate 1:1 ratio, and the memory reference rate is approximately 0.2 requests per microsecond. Figure A.15 shows a different portions of the SETI@HOME

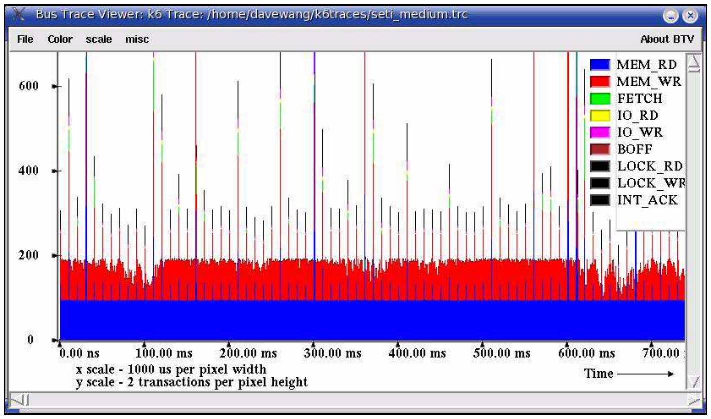


Figure A.14: : Portions of SETI@HOME Workload. Medium Memory Access Rate.

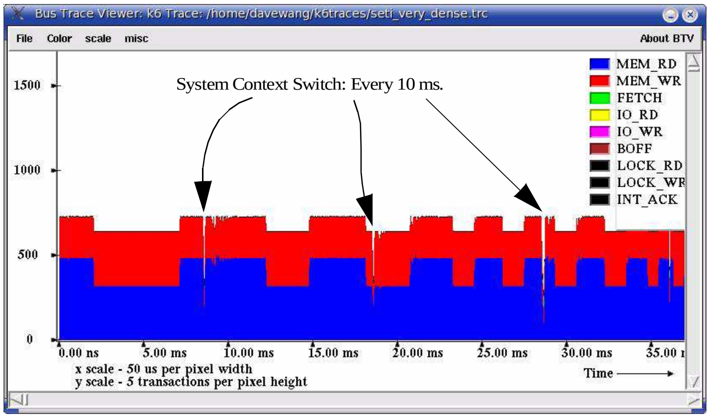


Figure A.15: : Portions of SETI@HOME Workload. Very High Memory Access Rate.

workload. In this segment, the memory reference rate increases to approximately 12~14 transactions per microsecond. The workload also alternates between read to write transaction ratios of 1:1 and 2:1. Finally, the effects of the disruption caused by the system context switch can be seen in this trace segment.

A.2.13 Quake 3 - 5 Segments

Quake 3 is a popular game for the personal computer. It is also relatively memory intensive. Figure A.16 shows a short segment of the Quake 3 processor bus trace, randomly captured as the quake 3 game progressed on a personal computer system. Figure A.16 shows that the processor bus activity of the game is very bursty. However, a cyclic behavior appears in the trace with a frequency of approximately once every 70 milliseconds. Interestingly, the frequency of the cyclic behavior coincides with the frame rate of the Quake3 game on the host system.

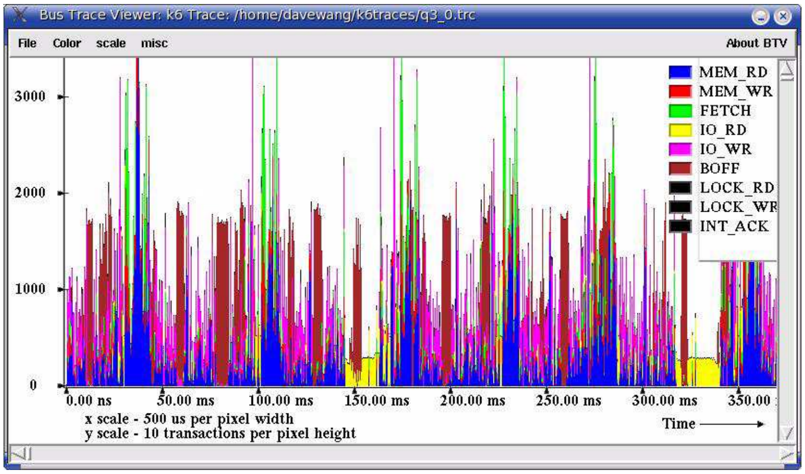


Figure A.16: Quake 3: Random trace segment.

Figure A.17 shows another short segment of the Quake 3 game labelled as segment 4. In Figure A.17 the cyclic pattern of bursts of instruction fetches remain, but the level of activity on the processor bus is significantly higher than shown in segment 0. In Figure A.17, the level of I/O transactions are also significantly higher. In this work, a total of 5 segments of Quake 3 traces are used to examine the overall behavior of the workload.

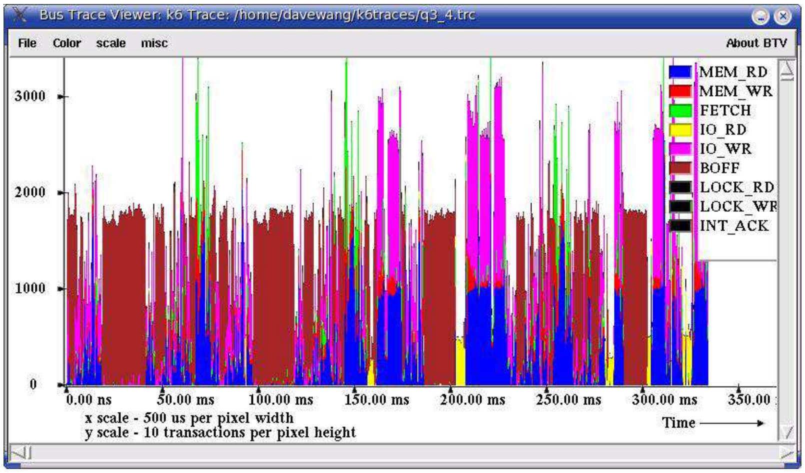


Figure A.17: : Quake 3: Random Segment 4.

CHAPTER B *Glossary of Terminology*

BRR: Acronym Bank Round Robin; A scheduling algorithm that rotates scheduling priority around different banks in a close-page memory system to optimize for maximum temporal distance between accesses to the same bank. See page 188.

CAS: Column Access Strobe; Command to move a column of data in a DRAM memory system. See page 71.

Close-Page policy: A row buffer management policy designed to optimize DRAM memory bandwidth in a memory system, and typically supports applications that access data in DRAM memory systems with very little spatial and temporal locality. The sense amplifier is immediately precharged after the column access command is processed. See page 115.

CPRH: Command Pair Rank Hopping; a scheduling algorithm that alternately schedules row activates to different ranks of DRAM devices and group schedules column accesses to the same rank of memory to maximize DRAM bandwidth. See page 191.

DDR: Dual Data Rate SDRAM device. See page 107.

DDR2: Second Generation Dual Data Rate SDRAM device. See page 107.

DDR3: Third Generation Dual Data Rate SDRAM device. See page 107.

DDRx: Denotes DDR SDRAM, DDR2 SDRAM, DDR3 DRAM and future DDR SDRAM variant devices.

Differential Sense Amplifiers: Circuitry in DRAM devices that senses, amplifies a row of data. Also referred to as a row buffer since it can hold the resolved row of data for a lengthy period of time, until the next refresh command is issued. See page 15.

DIMM: Dual Inline Memory Module. See page 56.

Precharge: Command to prepare a given DRAM array for a row access. See page 19.

Open Page Policy: A row buffer management policy designed to optimize DRAM memory bandwidth in a memory system, and typically supports applications that access data in DRAM memory systems with high degrees of spatial and temporal locality. The sense amplifier is kept active for as long as possible. See page 114.

Rank: One or more DRAM devices that act as a single entity in reponse to a given DRAM command. In essence, it is a “bank” of DRAM devices. However, since the terminology of “bank” denotes separate DRAM arrays inside of a given DRAM devices, a “bank of DRAM devices” is now referred to as a rank of memory. See page 48.

RAS: Row Access Strobe; Command to activate a row of DRAM cells in a memory system. See page 70.

Row Cycle Time: The amount of time it takes to access a row in a given bank of DRAM array and restore the data to the DRAM array so that another row in the same bank can be accessed. See table 4.1 on page 69.

Row Buffer: See Differential Sense Amplifiers, page 15.

SDRAM: Synchronous DRAM device. See page 34.

Sense Amplifiers: See Differential Sense Amplifiers, page 15.

SIMM: Single Inline Memory Module. See page 55.

[1] W. Wakamiya, T. Timori, H. Ozaki, H. Itoh, K. Fujiwara, T. Shibano, H. Miyatake, A. Fujii, T. Tsutsumi, S. Satoh, T. Katoh, "Fully planarized 0.5 µm technologies for 16M DRAM", International Electron Devices Meeting (IEDM) Technical Digest, 1988.

[2] H. Yamauchi, T. Yabu, T. Yamada, "A Circuit Design to Suppress Asymmetrical Characteristics in High-Density DRAM Sense Amplifiers", IEEE Journal of Solid State Circuits, Vol. 25, No. 1, Feb. 1990.

[3] G. Bronner, H. Aochi, M. Gall, J. Gambino, S. Gernhardt, E. Hammerl, H. Ho, J. Iba, H. Ishiuchi, M. Jaso, R. Kleinhenz, T. Mii, M. Narita, L. Nesbit, W. Neumueller, A. Nitayama, T. Ohiwa, S. Parke, J. Ryan, T. Sato, H. Takato, S. Yoshikawa, "A fully planarized 0.25 µm CMOS technology for 256 Mbit DRAM and beyond", 1995 Symposium on VLSI Technology.

[4] H. Geib, W. Raab, D. Schmitt-Landsiedel, "Block-Decoded Sense-Amplifier Driver for High-Speed Sensing in DRAM's", IEEE Journal of Solid State Circuits, Vol. 27, No. 9, Sept. 1992.

[5] J. Ziegler, M. Nelson, J. Shell, R. Peterson, C. Gelderloos H. Muhlfeld, C. Montrose, "Cosmic Ray Soft Error Rates of 16-Mb DRAM Memory Chips", IEEE Journal of Solid State Circuits, Vol. 33, No. 2, Feb. 1998.

[6] J. Amon, A. Kieslich, L. Heineck, T. Schuster, J. Faul, J. Luetzen, C. Fan, C. Huang, B. Fischer, G. Enders, S. Kudelka, U. Schroeder, K. Kuesters, G. Lange, J. Alsmeier, "A Highly Manufacturable Deep Trench based DRAM Cell Layout with a Planar Array Device in a 70nm Technology", International Electron Devices Meeting Technical Digest, 2004.

[7] S. Kuge, T. Kato, K. Furutani, S. Kikuda, K. Mitsui, T. Hamamoto, J. Setogawa, K. Hamade, Y. Komiya, S. Kawasaki, T. Kono, T. Amano, T. Kubo, M. Haraguchi, Z. Kawaguchi, Y. Nakaoka, M. Akiyama, Y. Konishi, H. Ozaki, "A 0.18 µm 256 Mb DDR-SDRAM with low-cost post-mold-tuning method for DLL replica", International Solid State Circuits Conference (ISSCC) Technical Digest, Feb. 2000.

[8] C. Radens, S. Kudelka, L. Nesbit, R. Malik, T. Dyer, C. Dubuc, T. Joseph, M. Seitz, L. Clevenger, N. Arnold, J. Mandelman, R. Divakaruni, D. Casarotto, D. Lea, V. Jaiprakash, J. Sim, J. Faltermeier, K. Low, J. Strane, S. Halle, Q. Ye, S. Bukofsky, U. Gruening, T. Schloesser, G. Bronner, "An orthogonal 6F2 trench-sidewall vertical device cell for 4 Gb/16 Gb DRAM", International Electron Devices Meeting Technical Digest, 2000.

[9] H. Jeong, W. Yang, Y. Hwang, C. Cho, S. Park, S. Ahn, Y. Chun, S. Shin, S. Song, J. Lee, S. Jang, C. Lee, J. Jeong, M. Cho, J. Lee, K. Kinam, "Highly manufacturable 4 Gb DRAM using using 0.11 µm DRAM technology", International Electron Devices Meeting Technical Digest, 2000.

[10] Y. Kim, S. Lee, S. Choi, H. Park, Y. Seo, K. Chin, D. Kim, J. Lim, W. Kim, K. Nam, M. Cho, K. Hwang Y. Kim, S. Kim, Y. Park, J. Moon, S. Lee, M. Lee, "Novel capacitor technology for high density stand-alone and embedded DRAMs", International Electron Devices Meeting Technical Digest, 2000.

[11] J. Sim, H. Lee, K. Lim, J. Lee, N. Kim, K. Kim, S. Byun, W. Yang, C. Choi, H. Jeong, J. Yoo, D. Seo, K. Kim, B. Ryu, H. Yoon, C. Hwang, K. Kim, "A 4Gb DDR SDRAM with Gain-Controlled Pre-Sensing and Reference Bitline Calibration Schemes in the Twisted Open Bitline Architecture", International Solid State Circuits Conference (ISSCC) Technical Digest, Feb. 2001.

[12] T. Kirihata, G. Mueller, M. Clinton, S. Loeffler, B. Ji, H. Terletzki, D. Hanson, C. Hwang, G. Lehmann, D. Storaska, G. Daniel, L. Hsu, O. Weinfurtner, T. Boehler, J. Schnell, G. Frankowsky, D. Netis, J. Ross, A. Reith, O. Kiehl, and M. Wordeman, "A 113µm2 600Mb/s/pin 512Mb DDR2 SDRAM with Vertically-Folded Bitline Architecture," International Solid State Circuits Conference, Digest of Technical Papers, 2001, pp. 382–383.

[13] T. Takahashi, T. Sekiguchi, R. Takemura, S. Narui, H. Fujisawa, S. Miyatake, M. Morino, K. Arai, S. Yamada, S. Shukuri, M. Nakamura, Y. Tadaki, K. Kajigaya, K. Kimura, K. Itoh, "A Multigigabit DRAM Technology with 6F2 Open-Bitline Cell, Distributed Overdriven Sensing and Stacked-Flash Fuse", IEEE Journal of Solid State Circuits, Vol. 36, No. 11, Nov. 2001.

[14] T. Kirihata, G. Mueller, B. Ji, G. Frankowsky, J. Ross, H. Terletzki, D. Netis, O. Weinfurtner, D. Hanson, G. Daniel, L. Hsu, D. Storaska, A. Reith, M. Hug, K. Guay, M. Selz, P. Poechmueller, H. Hoenigschmid, M. Wordeman, "A 390 mm2 16-bank 1 Gb DDR SDRAM with Hybrid Bitline Architecture", IEEE Journal of Solid State Circuits, Vol. 34, No. 11, Nov. 1999.

[15] R. Smith, J. Chlipala, J. Bindels R. Nelson, F. Fischer, T. Mantz, "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM", IEEE Journal of Solid-State Circuits, Vol. SC-16, No. 5, Oct 1981.

[16] "Construction Analysis: Samsung KM44C4000J-7 16 Megabit DRAM", Integrated Circuit Engineering Technical Report, No. SCA 9311-3001.

[17] "Construction Analysis: Hitachi 5165805A 64 Mbit (8 Mb x 8) Dynamic RAM", Integrated Circuit Engineering Technical Report, No. SCA 9712-565.

[18] "Construction Analysis: Mitsubishi M5M465405AJ 64Mbit DRAM (16M x 4 bit)", Integrated Circuit Engineering Technical Report, No. SCA 9712-566.

[19] J. Mandelman, R. Dennard, G. Bronner, J. DeBrosse, R. Divakaruni, Y. Li, and C. Radens, "Challenges and future directions for the scaling of dynamic random-access memory", IBM Journal of Research and Development, vol. 46, No. 2, Mar. 2002.

[20] J. Barth, J. Dreibelbis, E. Nelson, D. Anand, G. Pomichter, P. Jakobsen, M. Nelms, J. Leach, and G. Belansek, "Embedded DRAM design and architecture for the IBM 0.11-µm ASIC offering", IBM Journal of Research and Development, vol. 46, No. 6, 2002.

[21] E. Adler, J. DeBrosse, S. Geissler, S. Holmes, M. Jaffe, J. Johnson, C. Koburger, J. Lasky, B. Lloyd, G. Miles, J. Nakos, W. Noble, S. Voldman, M. Armacost, R. Ferguson, "The Evolution of IBM CMOS DRAM Technology," IBM Journal of Research and Development, vol. 39, p167–188, 1995.

[22] C. Kim, "Memory World in the Next Decade", Memory Devision, Device Solution Network Business, Samsung. Seminar presentation to Po-hang University of Science and Technology (POSTECH), Kyungpook, Korea. May 2003.

[23] J. Lee, Y. Ahn, Y. Park, M. Kim, D. Lee, K. Lee, C. Cho, T. Chung, K. Kim, “Robust Memory Cell Capacitor using Multi-Stack Sotarge Node for High performance in 90nm Technology and Beyond”, Proceedings of the 2003 Symposium on VLSI Technology.

[24] Z. Zhang, Z. Zhu, X. Zhang, “Breaking Address Mapping Symmetry at Multi-levels of Memory Hierarchy to Reduce DRAM Row-buffer Conflicts”, The Journal of Instruction-Level Parallelism, Vol. 3, 2002.

[25] Z. Zhang, Z. Zhu, X. Zhang, “A Permutation-based Page Interleaving Scheme to Reduce Row-buffer Conflicts and Exploit Data Locality”, Proceedings of the 33rd IEEE/ACM International Symposium on Microarchitecture, Dec. 2000. pp32-41.

[26] Z. Zhu, Z. Zhang, X. Zhang, "Fine-grain Priority Scheduling on Multi-channel Memory Systems". Proceedings of the 8th International Symposium on High Performance Computer Architecture, Feb. 2002.

[27] W. Lin, S. Reinhardt, D. Burger, “Reducing DRAM Latencies with an Integrated Memory Hierarchy Design”. Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Jan. 2001.

[28] V. Cuppu, B. Jacob, “Organization Design Trade-offs at the DRAM, Memory Bus and Memory Controller Level: Initial Results”, University of Maryland Systems and Computer Architecture Group Technical Report UMD-SCA-TR-1999-2, Nov. 1999.

[29] E. Larson, S Chatterjee, T. Austin “MASE: A Novel Infrastructure for Detailed Microarchitectural Modeling”, International Symposium on Performance Analysis of Systems and Software, 2001

[30] J. Janzen, “Calculating Memory System Power for DDR SDRAM”, Micron Designline Vol. 10, issue 2, 2Q01.

[31] J. Janzen, “Calculating Memory System Power for DDR2”, Micron Designline Vol. 13, issue 1, 1Q04.

[32] S. Rixner, W Dally, U. Kapasi, P. Mattson, J. Owens, “Memory Access Scheduling”, Proceedings of the 27th International Symposium on Computer Architecture, June 2000.

[33] V. Cuppu, B. Jacob, "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?", Proceedings of 28th International Symposium on Computer Architecture, June 2001

[34] S. McKee, "Dynamic Access Ordering: Bounds on Memory Bandwidth, " Univ. of Virginia, Technical Report CS-94-38, Oct. 1994.

[35] M. Franklin, G. Sohi, “ARB: A Hardware Mechanism for Dynamic Reordering of Memory References”. IEEE Transactions on Computers. Vol. 45, No. 5, 1996.

[36] S. Sair, M. Charney, “Memory Behavior of the SPEC 2000 Benchmark Suite”, IBM Research Report. October, 2000.

[37] S. Hong, S. McKee, M. Salinas, R Klenke, J Aylor, W. Wulf, “Access Order and Effective Bandwidth for Stream on a Direct Rambus Memory”, The 5th International Symposium on High-Performance Computer Architecture. Jan. 1999.

[38] 1 Gbit DDR2 SDRAM device datasheet, Micron Inc.

[39] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, M. Irwin, “Hardware and Software Techniques for Controlling DRAM Power Modes”, Seventh International Symposium on High-Performance Computer Architecture (HPCA'01), Jan. 2001.

[40] J. McCalpin, “STREAM: Sustainable Bandwidth in High Performance Computers”, http://www.cs.virginia.edu/stream/ .

[41] “Intel 875P Chipset: Intel 82875P Memory Controller Hub (MCH) Datasheet”, http://www.intel.com .

[42] C. Zhang, S McKee, “Hardware-Only Stream Prefetching and Dynamic Access Ordering”, Proceedings of the 14th international conference on Supercomputing, 2000.

[43] J. Alakarhu, “A Comparison of Precharge Policies with Modern DRAM Architectures”, Proceedings of the 9th International Conference on Eletronics, Circuits and Systems, Vol. 2, pp. 823-826, Sept 2002.

[44] K. Hiraki, T. Shimada, S. Sekiguchi, “Empiracle study of latency hiding on fine-grain parallel processor”, Proceedings of the 7th International Conference on Supercomputing, pp. 220-229, 1993.

[45] D. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", Proceedings of the 22nd International Symposium on Computer Architecture, June 1995.

[46] S. Rixner, “Memory Controller Optimizations for Web Servers”, Proceedings of the 37th International Symposium on Microarchitecture. Dec. 2004.

[47] L. Zhang, Z. Fang, M. Parker, B. Mathew, L. Schaelicke, J. Carter, W. Hsieh, S. McKee, “The Impulse Memory Controller”, IEEE Transactions on Computers Vol. 50 , Issue 11, Nov. 2001.

[48] C. Natarajan, B. Christenson, F. Briggs, “A Study of Performance Impact of Memory Controller Features in Multi-Processor Server Environment”, Proceedings of the 3rd Workshop on Memory Performance Issues (WMPI-2004)

[49] F. Briggs, M. Cekleov, K. Creta, M. Khare, S. Kulick, A. Kumar, L. Looi, C. Natarajan, S. Radhakrishnan, L. Rankin, “Intel 870: A Building Block for Cost-Effective, Scalable Servers”, IEEE Micro, ,Vol. 22, No. 2, Mar. 2002.

[50] Z. Zhu, Z. Zhang, “A Performance Comparison of DRAM Memory System Optimizations for SMT Processors”. Proceedings of the 11th International Symposium on High-Performance Computer Architecture, Feb. 2005.

[51] I. Hur, C. Lin, “Adaptive History-Based Memory Schedulers”, Proceedings of the 37th International Symposium on Microarchitecture. Dec. 2004.

[52] S. Leibson, “Jesse Lipcon: Interview with a pioneer”, Microprocessor Reports, Volume 14, Archive 12, December, 2000.

[53] K Krewell, “Alpha EV7 Processor: A High Performance Tradition Continues”, Compaq EV7 Whitepaper, Reed Electronics Group. In-stat MDR, April 2002.

[54] J. Pawlowski, “Memory Performance Tutorial”, Tutorial given at the 11th International Symposium on High-Performance Computer Architecture, Feb. 2005.