Low Energy DRAM Controller for Computer Systems

Alberto González Trejo

Adrián Cristal Kestelman
Marco Antonio Ramírez Salinas

This dissertation is submitted for the degree of Master in Innovation and Research in Informatics
at the Universitat Politècnica de Catalunya

July 4th, 2019

Acknowledgements

First of all, as always, I want to say thanks to my family, who have always been my source of inspiration, motivation and I will always take care of them.

I am very grateful with the Universtitat Politècnica de Catalunya and the Instituto Politécnico Nacional, for provide me education and the opportunity to meet and collaborate with many incredible people, I am really thankful for this opportunity.

Last but not least, I really appreciate the support that CONACYT, a mexican government institution, provided me by means of a scholarship, allowing me to study abroad.

”Try not to become a man of success. Rather become a man of value.”
Albert Einstein

Abstract

”The Memory System: You can’t avoid it, you can’t ignore it, you can’t fake it.”
Bruce Jacob

The performance characteristics of modern DRAM memory systems are primarily impacted for two attributes: the device data rate and the row cycle time. Modern commodity DRAM devices data rates and row cycle times are scaling at different rates with each succesive generation (SDRAM, Open page DRAM, DDR, DDR2, DDR4). As a result, the performance characteristics of modern DRAM memory systems are becoming more difficult to evaluate, and at the same time the called *memory wall*, which points out the gap between the increment on CPU clock frequency and the lower increment in memory speed, is limiting the performance of modern computer systems, which are mainly memory intensive.

The memory controller is the part of the main memory system that is in charge of assuring the proper operation of the DRAM devices and at the same time, it manages the flow of data into and out the DRAM devices. However, due to the complexity of DRAM memory access protocols, the large number of timing parameters, the large number of combinations of memory system organizations, different workload characteristics and different design goals, the design of this specific part of the main memory system has as much freedom as the design space of a processor that implements a specific Instruction Set Architecture.

This specific part of the main memory system is specially treated as a *black box* since its implementation is mainly provided by the same companies that manufacture the computer systems chips (ASICs or FPGAs). This situation leads to a lack of knowledge of how this specific device controller really works.

In this work, we leverage an open source simulation framework to evaluate different memory scheduling algorithms, which allows us to choose the one that performs better in terms of power consumption.

As a result, we provide an architectural design of a memory controller, which is implemented in Verilog and tested on a FPGA platform.

Last but not least, this work provides a thoroughly journey of how a DRAM memory system works and how feasible it is to jump from a simulation framework to the world of actual implementation of a DDR3 DRAM memory controller.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

MICROSE Laboratory at *Centro de Investigación en Computación del Instituto Politécnico Nacional*, in Mexico city, has been working along the last 6 years developing innovative high performance computer architectures targetting FPGA-SoC. *Lagarto SoC* is an academic project which is aimed to create an educational environment on computer architecture and operating systems, ranging from high school to graduate school. This thesis is part of this effort.

The memory system behavior has become a focal point of computer architecture research. This is due this part of a computer system is a performance and energy bottleneck in almost all applications. Nowadays, system designers, application developers and many technological trends are pointing to the same direction: more capacity, bandwidth, efficiency and reliability out of the memory system is a must [1].

Recent research ranges from studies of smarter memory controllers and other recent work looks at the performance of different commercial DRAMs at the architectural level, and a wide range of DRAM bus and memory controller organizations at the system level. DRAM performance has relied on technology improvements and only bandwidth-related improvements on architecture.

The continuing increase of data intensive applications are today one of the main concerns regarding main memory systems. This is due mainly two reasons, one is the key design between the power consumed by the main memory system by itself and the other is the QoS *(Quality of Service)* that has to be delivered in order to be able to leverage the peak features that current modern DRAM devices like DDR3 provide to system designers.

DRAM provides parallelism by means of certain number of banks which operate concurrently. According to [2] all banks in a DRAM chip share the same data, address and command buses. In addition, a number of DRAM devices *(8 in DDR3)* can be connected to the same bus, this organization is called a *rank*, in order to improve parallelism. The complex internal organization of DRAMs leads to a cumbersome list of timing constraints that govern DRAM behavior.

A typical DDR memory is arranged in banks having multiple rows and columns along with prefetch buffers. For any data transaction, the memory address is split into bank address, row address and column address. The performance advantages of the DDR memory are mainly due to its prefetch architecture with burst oriented operation, which allows to exploit the Dual Rate data transfer intrinsic on these devices, where a memory access to a particular row of a bank causes the prefetch buffer to grab a set of adjacent data words and subsequently burst them on I/O pins on each edge of the data transfer memory clock, without requiring individual column addresses. Thus the higher the size of prefetch buffers, the higher is the bandwidth. Higher bandwidth is also achieved by creating modules with multiple DDR memory chips.

A modern DRAM memory controller employs diverse mechanisms to alleviate the complexity of DRAM devices, those sophisticated mechanisms are address mapping, memory scheduling algorithms and power management optimizations. All these mechanisms work together in order to improve the efficiency of a DRAM memory system.

1.1 Motivation

A state of the art main memory system is composed by a technology device like DDR3 DRAM, and also by a sophisticated memory controller that aims to deliver, between other things, maximum bandwidth. The aim of a DRAM memory controller is to ensure the correct operation of the DRAM module regarding refresh and timing requirements, by issuing commands to the DRAM chip to initiate the active, precharging and I/O phases. A small, seemingly inconsequential change in the sequence or timing commands to the DRAM can result in a huge change in the bandwidth we can get out of the DRAM memory system. Since memory bandwidth is often the bottleneck for critical memory intensive tasks, the change in bandwidth quickly becomes a change in system performance. Unfortunately, the command sequence and timing come from a complex interaction of the application and system software, the various elements of the system hardware, including cache controllers, memory managers, direct memory access (DMA) controllers, accelerators and the DRAM controller. This situation gets more complicated as *Systems on Chip* grow more powerful and complex.

In this work, a low power DRAM memory controller analysis, design and implementation is presented. The aim of this work is to provide a thoroughly journey through how a DRAM memory system works, the analysis of the features of a memory controller which we can take hand to lower power consumption, and also the development of a functional IP is presented.

This dissertation also utilizes a simulation framework based on gem5, which allows us to perform trace based simulation and full system simulation with an event-based memory controller model.

The aim of this work is to provide a complete framework of how a DDR3 DRAM memory controller is designed, ranging from simulation to implementation perspective. We make the following contributions on this thesis:

- We analyze the main memory system in detail.
- We propose a memory scheduling algorithm that optimizes power consumption.

- We contribute to the open source community and provide a basic open source DDR3 Memory Controller targeting FPGA.
- Design and implementation of an interface with the SoC Lagarto.

1.2 Organization of this dissertation

In this work a methodical analysis of a DRAM memory system is presented, this is done with the aim of providing enough background information of the tasks a DRAM memory controller is in charge of. In the first chapter, a brief introduction of this work is presented.

In the second chapter, the theoretical background of this work is presented, starting with the main memory subsystem description and the underlying technology of DRAM devices, afterwards the DRAM memory access protocol is described in detail. Also in this chapter the low power modes present on DRAM technology like DDR3 are described, and a generic DRAM memory controller is depicted, giving special attention to the row buffer management and address mapping policies, DRAM metrics and the memory scheduling algorithms; the related work is also presented in this chapter.

The methodology followed in this work is presented in chapter 3. In here the simulation framework, the traces used to evaluate the memory controller proposal and the description of the workloads are described, the memory controller proposal is described in detail in this chapter. The evaluation of the proposal and the results obtained out of the execution of the workloads and trace simulations are presented in chapter 4.

In chapter 5 the actual implementation of the memory controller targeting FPGA is presented. The architectural design and microarchitecure is described. Also, the interface with *SoC Lagarto* is presented. In chapter 6, the conclusions and future work of this thesis are presented.

Chapter 2

Theoretical Background

In order to ease the understanding of this work, it is provided the needed theoretical background in this section. The contents of this chapter is heavily based on the book ”Memory Systems: Cache, DRAM, Disk” by Jacob et al. [3]. This chapter describes the organization of the main memory subsystem and the DRAM technology in specific, giving as example a generic Fast Page Mode DRAM, since current DRAM devices are mainly based on this DRAM architecture. The DRAM memory system organization is described, taking into account different organizations and remarking the DRAM access protocol, the specific DDR3 device operation is presented, and at the end, the description of the memory controller. Also, this chapter describes specifically the architecture of DDR3 devices, since this is the target DRAM device of the final implementation.

The limited goal of this chapter is to provide a broad overview of the functionality of DRAM devices and memory controllers. With the understanding of these fundamentals we can advance in deeper discussions about architectural trade-offs of a memory controller, and memory scheduling algorithms.

2.1 Main memory subsystem

The memory subsystem is an important component in all computer systems, this is due this part of computer systems accounts for an important fraction of the computing time and energy consumption. When a processor fails to fetch data from caches, the LLC sends a request to the memory controller. This memory controller is connected to DRAM devices, for example to DIMMs (Dual In-line Memory Module) via memory channels. Then, the memory controller manages a queue of pending memory requests and it’s in charge of the scheduling of this requests, always obeying the timing constraints imposed by the DRAM technology.

In a conventional memory system, a memory controller on a processor (same chip) is connected to Dual In-line Modules (DIMMs) via an off-chip electrical memory channel. A modern DDR3 memory channel typically has a 64-bit data bus and a 23-bit address/command bus that can support 1-2 DIMMs. Each DIMM is typically organized in 1-4 ranks. When the memory controller issues a request for a cache line, all DRAM chips in a rank work together to service the request, i.e., a cache line is striped across all chips in a rank. A rank and its constituent DRAM chips are also partitioned into multiple (4-16) banks. Each bank can process a different cache line request in parallel, but all banks in the active rank must sequentially share the data and command wires of the memory channel. JEDEC, the leading developer of standards for the solid-state industry, is in charge of regulating the emerging memory technologies. [2]

2.2 DRAM Devices

Figure 1. illustrates the organization and structure of an FPM DRAM device. Internally, the DRAM storage cells in the FPM DRAM device shown in Figure 1 are organized as follows: 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; RAS signal is known as row-address strobe. Inside the DRAM device, the address on the address bus is buffered by the row address buffer and then sent to the row decoder. The row decoder then accepts the 12-bit address and selects 1 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 means, a 16 bit wide column is the basic addressable unit of memory in this device, and each column access that follows the row access would ordinary access (read or write) 16 bits of data from the same row of DRAM. The way that a column access is engaged is similar to the row access in that the memory controller would be place a 10 bit address on the address bus, but this time the *column access strobe* (CAS) siganl would be asserted. Internally, the DRAM chip then takes the 10 bit column address, decodes it and uses it to select one column out of 1024 available. 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, have similar basic organization. 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. All DRAM devices also have some logic circuit that control the timing and sequence how the device operates. In the case of the the FPM DRAM device shown in Figure 1, the chip has internal clock generators as well as 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 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, modern DRAM devices move data onto and off the device through a set of bi-directional input-output pins connected to the system. [3]

All DRAM devices contain some basic logic control circuitry to direct the movement of data onto, within, and off DRAM devices. This essential control logic accepts externally asserted signal and control, and then orchestrates a timed sequences of internal control signals to di-

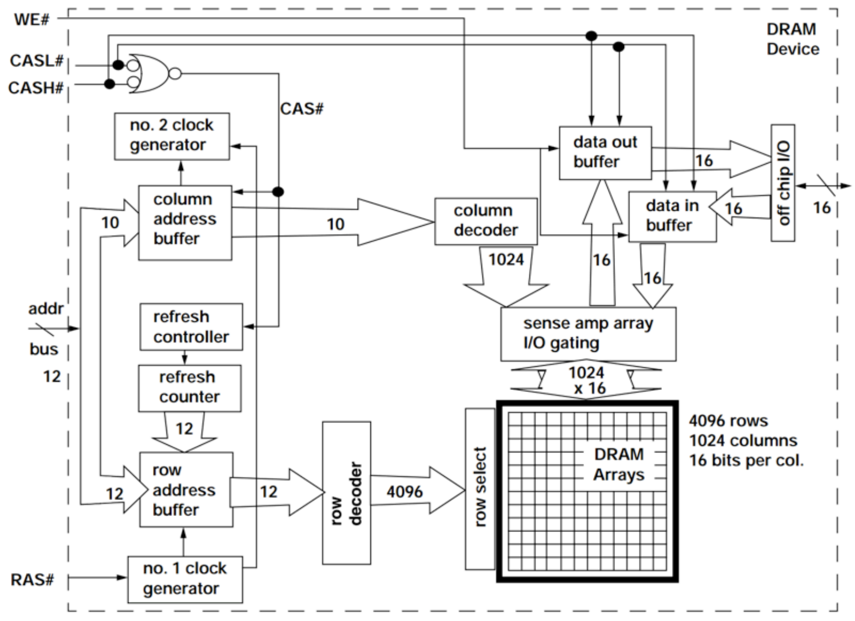


Figure 2.1: 64 Mbit Fast Page Mode DRAM Device (4096 x 1024 x 16).

rect movement of data on the FPM DRAM device illustrated in Figure 1.

Modern DRAM devices are controlled by synchronous state machines 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. Some parameters as CAS latency, burst type, burst length and low power modes are some examples of this configurable features. The value of the burst type determines the ordering of the data returned by the SDRAM device. The burst length determines the number of columns that a SDRAM device will return to the memory controller within a single read command.

DRAM devices are classified by the number of data bits in each device and typically that number quadruples from generation to generation. Also, in a given generation, a DRAM device may be configured with a different data bus width. For example, we can have a specific configuration of a 1 Gbit DDR2 SDRAM device with 16,384 rows per bank and each row consists of 8,192 bits. In the x8 configuration, we have 16,384 rows and each row consists of 1024 columns. This configuration is denoted as **256 Meg x 8**. The different configurations lead to a different number of bits per bitline, different numbers of bit per row activation, and different number of bits per column access. Also, the differences in current consumption characteristics in turn leads to difference in timing parameters designed to limit peak power consumption characteristics of DRAM devices.

The well known N-bit prefetch in SDRAM devices, implies that each time a column read command is issued by the memory controller, the control logic determines the duration and ordering of the data bursts, 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. This leads the following result, in successive generations of SDRAM, specially in DDRx SDRAM devices, successively large 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. illustrates the data I/O structure of a DDR SDRAM device.

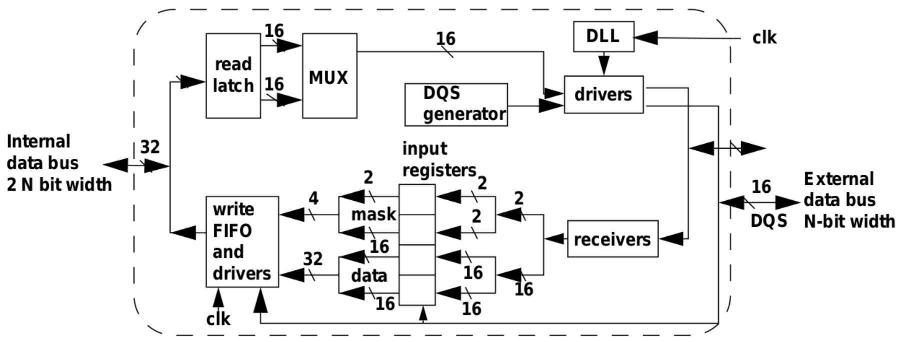


Figure 2.2: Data I/O in a DDR SDRAM devices illustrating 2-bit prefetch

Figure 2 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 bus. In DDR3 SDRAM devices, the number of bits prefetched by the internal data bus is 4N. The downside of the N bit prefetch architecture is that it is not possible to support short bursts, meaning that, in DDR3 devices, a minimum burst length of 8 columns of data are accessed per column read command.

DDR stands for Double Data Rate, this means data is transferred on both the rising and falling edges of the clock signal. This means that the transfer rate is roughly twice the speed of the I/O bus clock. For example, if the I/O bus clock runs at 800 MHz per second, then the effective rate is 1600 Megatransfers per second (MT/s), this is due there are 800 million rising edges per second and 800 million falling edges per second of a clock signal running at 800 MHz. The transfer rate refers to the number of operations transferring data that occur in each second in the data-transfer channel.

DDR architecture is aimed to achieve high-speed operation. The memory operates using a differential clock provided by the controller. Commands are registered at every positive edge of the clock. A bidirectional data strobe (DQS) is transmitted along with the data during reads and by the controller during writes. DQS is edge aligned with data for reads and center aligned with data for writes.

Read and write accesses to the DDR3 SDRAM devices are burst oriented. Accesses begin with the registration of an active command, which is then followed by a read or write command. The address bits registered with the active command are used to select the bank and row to be accessed. The address bits registered with the read or write command are used to select the bank and the starting column location for the burst accesses.

As we can see in the following table, memory vendors use different terminology to advertise their chips, sometimes we can find that bandwidth is indicated as MHz, other times is is expressed as time transfer per second in megabytes.

To calculate the data transmission rate, the transfer rate has to be multiplied by the information channel width, meaning

$$\textit{Channel width} * \textit{transfer rate} = \textbf{bits}\ \textit{transferred/second} \tag{2.1}$$

2.3 DRAM Memory System Organization

In the context of a complete memory system, the organization and operation of multiple devices is important to examine since this is how current DRAM memory subsystems actually work. 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 computer systems.

The organization of multiple DRAM devices into a memory system can impact the performance of the memory systems in such ways as latency, operating datarates, and bandwidth. This is why it is important to present the organization of such systems in detail. 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 a typical personal computer system controller.

Figure 2.3 shows 3 different single memory controller configurations. The one called ”Typical system controller” consists of a single physical memory channel, which is controlled by a single DRAM memory controller.

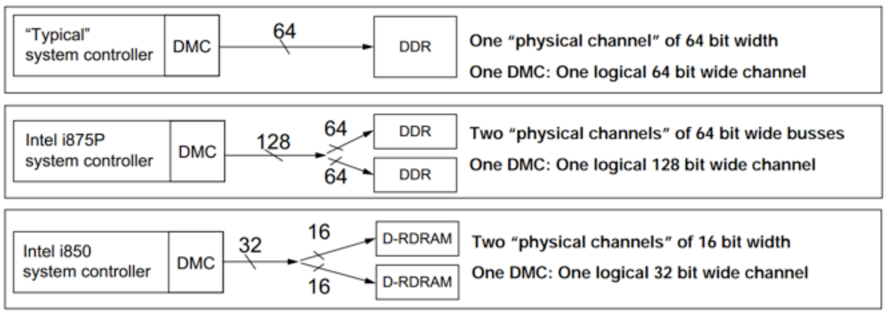

"Typical" system controller
DMC
64
DDR
One "physical channel" of 64 bit width
One DMC: One logical 64 bit wide channel
Intel i875P system controller
DMC
128
64
64
DDR
DDR
Two "physical channels" of 64 bit wide busses
One DMC: One logical 128 bit wide channel
Intel i850 system controller
DMC
32
16
16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

Figure 2.3: Systems with a single memory controller and different data bus widths.

In the example labelled as *Intel i875P*, the memory controller connects to a single memory channel with a 128 bit wide data bus, but since commodity DRAM module have a 64 bit wide data bus, it is necessary to use a configuration called paired-memory-module which is also referred as dual-channel configuration. It is important to note that since there is only one memory controller, both DRAM modules operate in parallel to store and retrieve data through the 128 bit wide data bus, so in fact, the paired-memory module configuration is in fact a 128 bit wide single channel memory system.

Modern DRAM memory systems with one DRAM memory controller and multiple physical channels of DRAM devices such as those illustrated in Figure 2.3 are typically designed with the physical channels operating in lockstep with respect to each other.

We define a **rank** essentially as a set of one or more DRAM devices that operate in lockstep in response to a given command. Those DRAM devices are controlled with the same command and address buses. Figure 2.4 illustrates a configuration of 2 ranks of DRAM devices, note that address and command buses are connected to every DRAM device in the memory system, but the wide data bus is partitioned and connected to different DRAM devices.

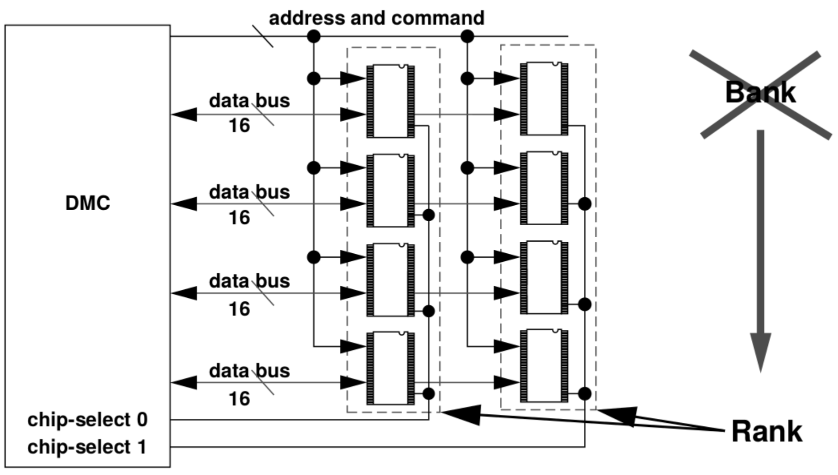


Figure 2.4: Memory system with 2 ranks of DRAM devices.

We use the word **bank** to denote an independent memory array inside a DRAM device. Figure 2.4 shows an SDRAM device with 4 banks of DRAM arrays. 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 a row access cycle, as we will show in the following sections. Some common resources, such as 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. Also, multiple banks in a given DRAM device can also be precharged or refreshed in parallel, depending on the design of the DRAM device.

We can think of a **row** in DRAM devices as the group of storage cells that are activated in parallel in response to a row activation command. As we already mentioned, multiple DRAM devices are typically connected in parallel as ranks of memory. The effect of having such configuration (rank of DRAM devices operating 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 called 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 by the DRAM memory controller.

In DRAM memory systems, a **column** of data is the smallest independently addressable unit of memory. In memory systems such as DDRx, the size of a column of data is the same as the width of the data bus. In DDRx SDRAM memory systems, each column access command loads and stores multiple columns of data depending on the programmed burst length. For example, in a DDR3 DRAM device, each memory read command returns a minimum of 8 columns of data.

2.4 DRAM Memory Access Protocol

In this section, the DRAM memory access protocol is examined thoroughly. 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. We examine a generic DRAM access protocol by focusing on basic DRAM commands common to all commodity DRAM devices.

2.4.1 Basic DRAM Commands

In this section, five basic commands are described. The descriptions if the basic commands form the foundation of the DRAM memory access protocol, also the interaction of the basic DRAM commands are then used to determine latency response and sustainable bandwidth characteristics of DRAM memory systems.

Figure 2.5 illustrates a generic SDRAM device which is used to define the basic memory access protocol. Different phases of operation occurs on the DRAM devices to facilite the movement of data for each command. The generic DRAM access protocol described in this section 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, like t_{RRD} and t_{FAW}.
Figure 2.5 illustrates four overlapped phases of operation for an abstract DRAM command. In phase one, the command is transported through the address and command buses 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, 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 time.

Figure 2.6 illustrates an abstract progression of a generic DRAM command. 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 2.6 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 [1]. In cases where the duration of an operation limits the timing of command issuance,

[1]CAS command excepted. t_{CAS} denotes the beginning of the CAS command to the beginning of the data transport phase.

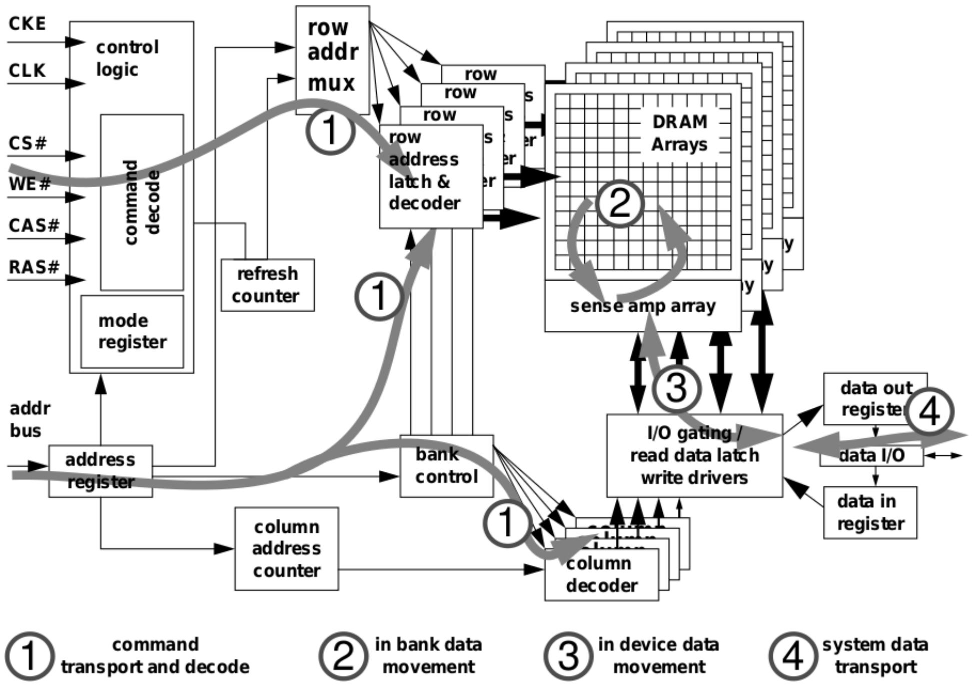


Figure 2.5: Command and data movement on a generic SDRAM device.

$t_{parameter}$ then defined the minimum time that commands may be placed onto the 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 $t_{parameter}$.

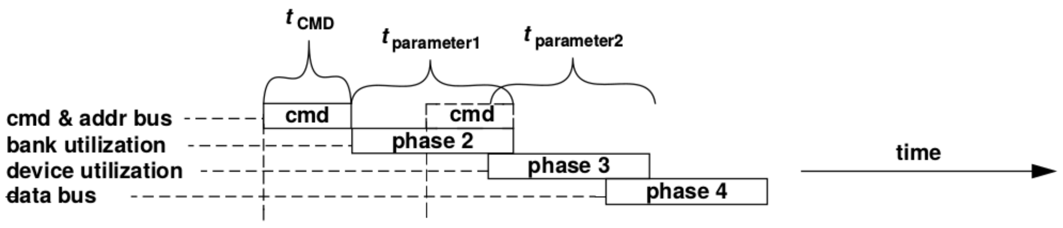


Figure 2.6: Different phase of an abstract DRAM command in a generic DRAM device.

The examination of the DRAM access protocol begins by a definition of the timing parameters. Table 2.1 summarizes the timing parameters used in the description of the DRAM memory access protocol. The timing parameters summarized in table 2.1 is far from a complete set of timing parameters used in the description of a modern memory access protocol. Nevertheless, the timing parameters described here is a minimum set of timing parameters whose use is sufficient to characterize and illustrate important interactions in modern DRAM memory systems.

Row Access Command

Figure 2.7 abstractly illustrates the progression of a generic row access command, also know as row activation command. The purpose of a row access command is to move data from the

Parameter	Description
t_{BURST}	Data **Burst** duration. The time period that data burst occupues on the data bus. Typically 4 or 8 beats of data. In DDR SDRAM, 4 beats of data occupies 2 cycles.
t_{CAS}	**C**olumn **A**ccess **S**trobe. Time interval between columnd access command and data returned by DRAM device(s). Also known as t_{CL}.
t_{CMD}	Command transport duration. Time period that a command occupies on the command bus as it is transported from the DRAM controller to the DRAM devices.
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.
t_{DQS}	**D**ata **S**trobe turnaround. Used in DDRx SDRAM memory systems. 1 full cycle in DDRx SDRAM.
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.
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.
t_{RC}	**R**ow **C**ycle. time interval between accesses to different rows in a given bank. $t_{RC} = t_{RAS} + t_{RP}$.
t_{RCD}	**R**ow to **C**olumn command **D**elay. Time interval between row access command and data ready at sense amplifiers.
t_{RFC}	**R**efresh **C**ycle **T**ime. Time interval between Refresh and Activation command.
t_{RRD}	**R**ow activation to **R**ow activation **D**elay. Minimum time interval between two row activation commands to the same DRAM device. Limits peak current profile.
t_{RP}	**R**ow **P**recharge. Time interval that it takes for a DRAM array to be precharged and be ready for another row access.
t_{WR}	**W**rite **R**ecovery time. Minimum time interval between end of a write data burst and the start of a precharge command. Allows sense amplifiers to restore data to cells.

Table 2.1: Summary of timing parameters used in generic DRAM access protocol.

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 know as the Row Column (Command) Delay, 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.
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 to 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

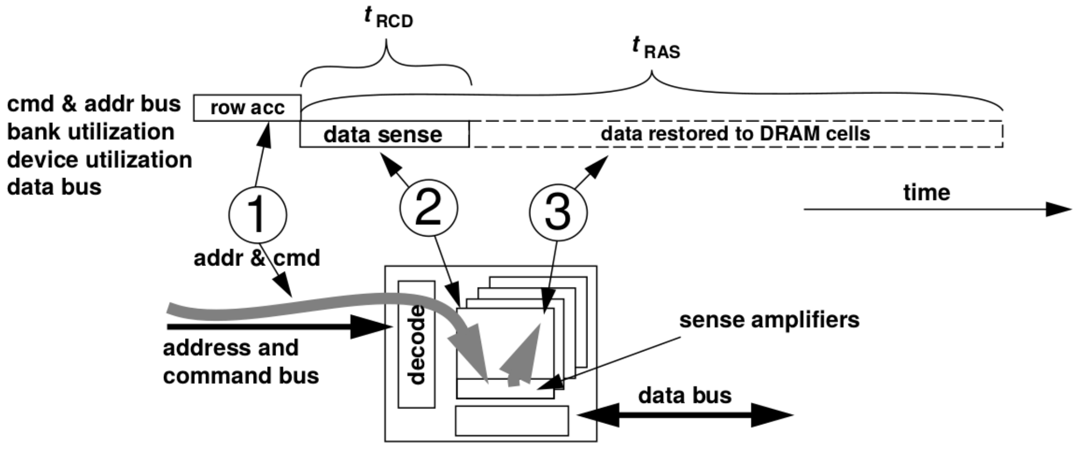


Figure 2.7: Row access command and timing.

row access to the same bank.

Column Read Command

Figure 2.8 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, the 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 rahter than the number of clock cycles. The data burst duration is labelled in Figure 2.8 as t$_{Burst}$.

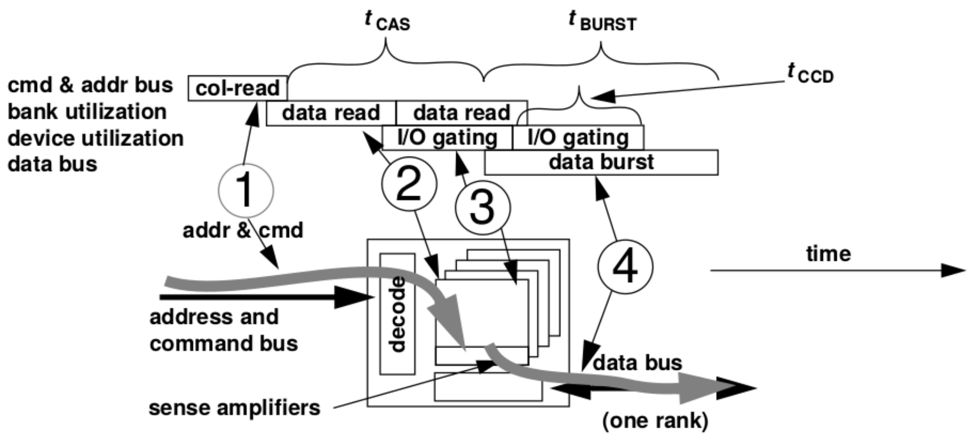


Figure 2.8: Column read command and timing.

Figure 2.8 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$_{Burst}$.

Column Write Command

Figure 2.9 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 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.

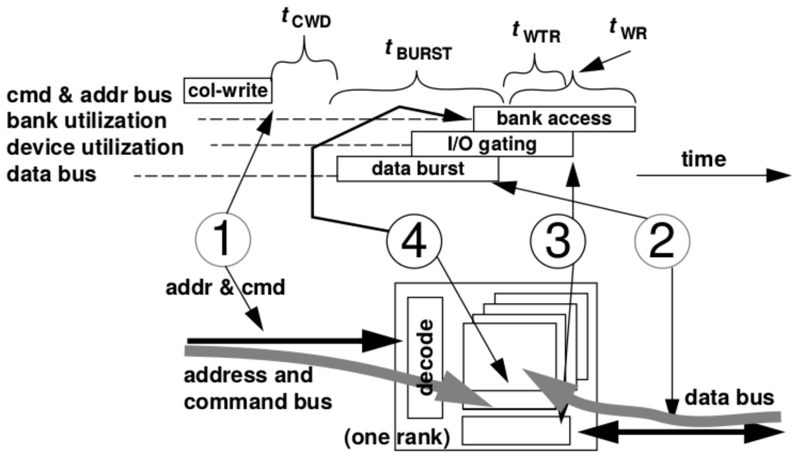


Figure 2.9: Column Write command and timing.

In Figure 2.9, 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. The 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 2.9 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.

Precharge Command

Accessing data on a DRAM device 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 2.10 illustrates the progression of a precharge command.

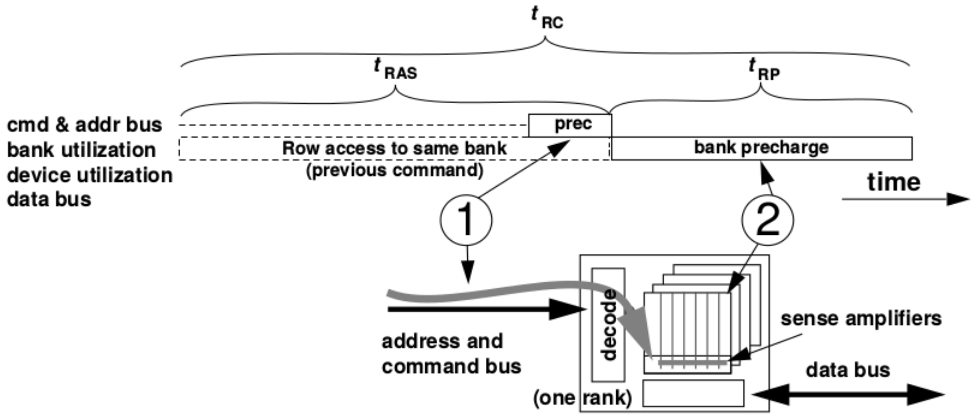


Figure 2.10: Row precharge command and timing.

Figure 2.10 shows that in the first phase, the precharge command is sent to the DRAM device, and in phase two, the selected bank is precharged. 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, the 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.

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 as the time passes, 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 mechanism used by different systems, some are designed to minimize controller complexity while others are designed to minimize bandwidth impact.

DDR3 SDRAM devices require Refresh cycles at an average periodic interval of t$_{REFI}$. All banks of the SDRAM must be precharged and idle for a minimum of the precharge time t$_{RP_{min}}$ before the Refresh Command can be applied. To simplify refresh management, each DRAM device has an internal refresh counter that tracks the rows to be refreshed during the next refresh operation. Normal memory operations resume only after the completion of an

Auto-Refresh command, when the refresh cycle has completed, all banks of the SDRAM will be in the precharged (idle) state. A delay between the Refresh Command and the next valid command must be greater than or equal to the minimum Refresh cycle time $t_{RFC_{min}}$, and this parameter depends on memory density.

Figure 2.11 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. The single refresh command to all banks take one refresh cycle time to complete. Figure 2.11 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}.

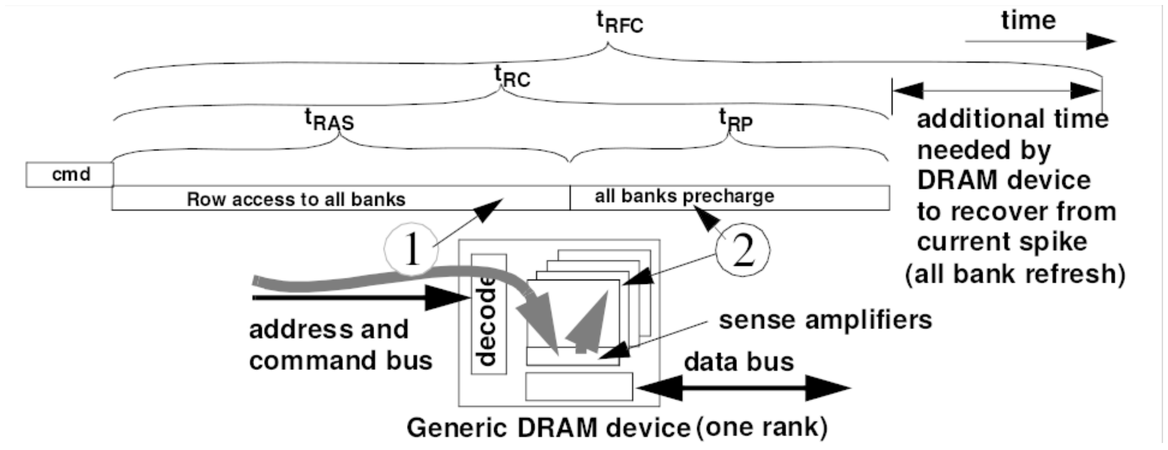


Figure 2.11: Row refresh timing.

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 2.11 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.

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 64 milliseconds for each row in a bank. That is, in a DRAM device with 8192 rows per bank and 64ms 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.

A Read Cycle

Figure 2.12 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 2.12, a sequence of commands in an abstract memory system designed for applications that do not to benefit from keeping rows of data in the sense amplifiers for subsequent accesses is illustrated. As show in Figure 2.12, 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 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.

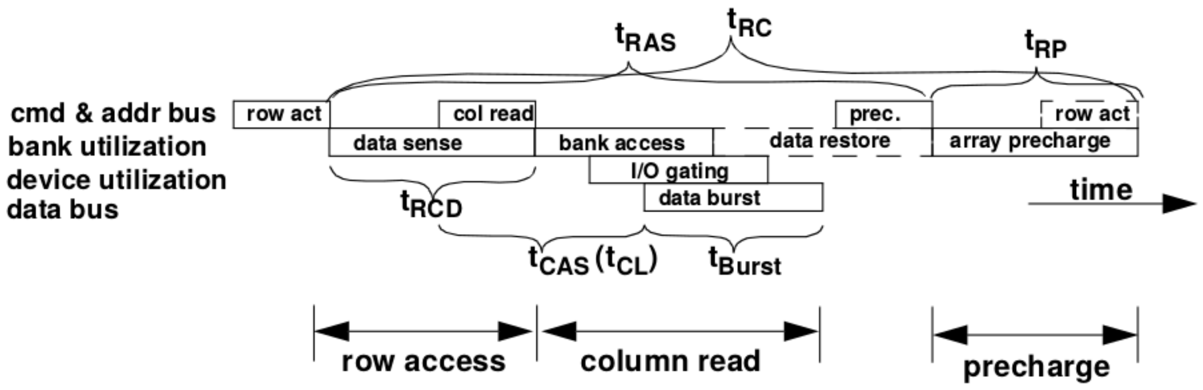


Figure 2.12: One read cycle in a ”close-page” memory system.

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.

Complex Commands

Some DRAM devices support commands that perform more complex series of actions. Figure 2.13 shows the same sequence of DRAM commands as presented in Figure 2.12, however, the simple column-read command is replaced with a compound *column-read-and-precharge* command. As the name implies, the latter command combines a column-read command and a precharge command into a single command. The advantages of a column-read-and-precharge command is that for close-page memory systems that precharge the DRAM bank immeadiately after a read command, the column-read-and-precharge command reduces the bandwidth requirement on the command and address bus. The implicit advantage of a combined command comes from the memory controller is now able to place a different command on the address and command bus that a separate precharge command would have otherwise occupied.

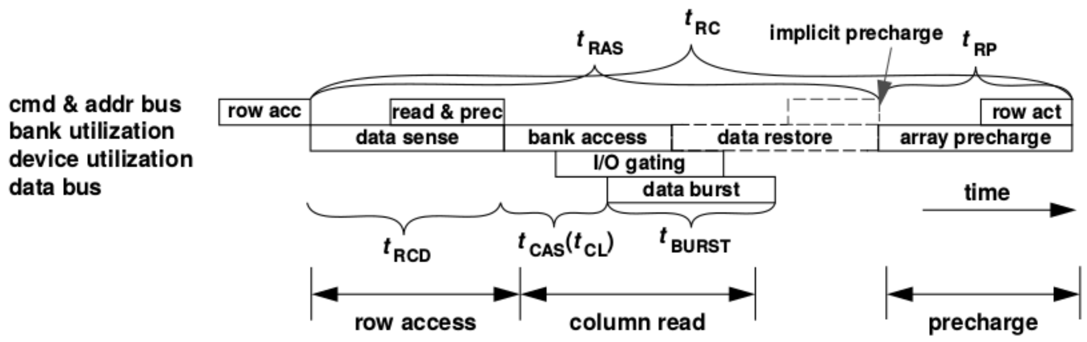


Figure 2.13: A read cycle with a row access command and a column-read-and-precharge command.

Figure 2.13 shows a colum-read-and-precharge command as issued by the memory controller to the DRAM device in the earliest time slot possible after the row access command while still respecting the t_{RCD} timing requirement, but the implicit precharge command is delayed so that it does not violate the t_{RAS} timing requirement.

Another type of complex command supported by DDR3 SDRAM memory system is the posted-CAS command. The posted-CAS command is simply a delayed column access command. Figure 2.14 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.

2.4.2 DRAM Command Interactions

By using 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 multiplexors, and the availability of off-chip resources such as the command, address and data buses. However, even with the availability of some shared resources, some other considerations have to be taken into account when scheduling DRAM commands.

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

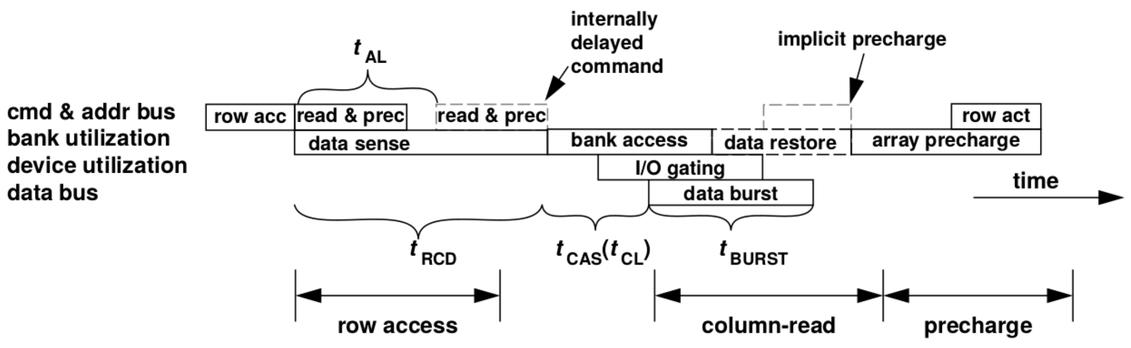


Figure 2.14: Posted CAS defers CAS commands in DRAM devices by a preset delay value, t_{AL}.

the 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. This leads to a more complex command interactions and higher degree of difficulty in scheduling command sequences in open-page memory systems. 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 later on.

Consecutive Reads to Same Rank

In modern DDR DRAM 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 availabiliry of the data bus. Figure 2.15 shows two read commands, labelled as read 0 and read 1, pipelined consecutively. As illustrated in Figure 2.15, t$_{CAS}$ after a read command is placed onto the command and address bus, the DRAM devices 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.

In DDR3 DRAM memory systems, 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

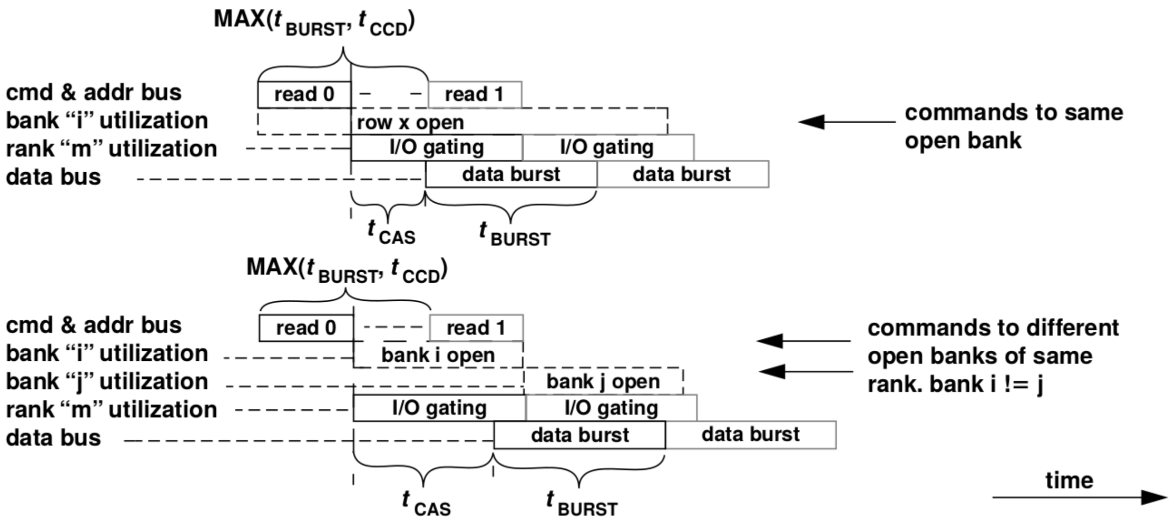


Figure 2.15: Consecutive column-read commands to the same bank, rank, and channel.

every t$_{Burst}$ time period, as shown in Figure 2.15.

Read to Precharge Timing

Figure 2.16 illustrates the minimum command timing for a precharge command that immediately follows a column-read command. The formula for minimum command timing is defined as $t_{Burst} + t_{RTP} - t_{CCD}$. We can notice that essentially, the timing parameter t$_{RTP}$ itself specifies the minimum amount of time that is needed between a column-read command and a precharge command.

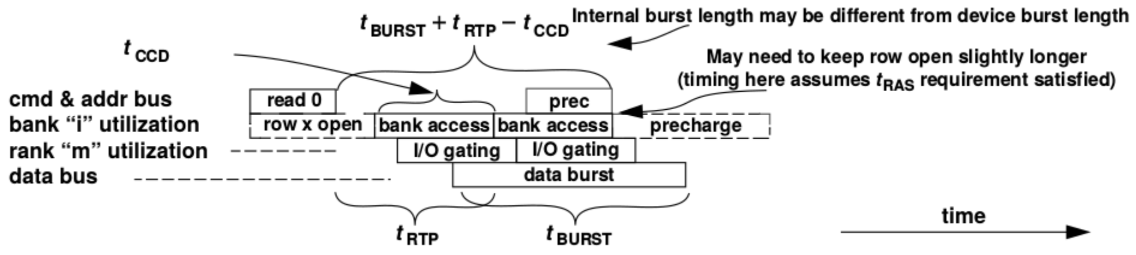


Figure 2.16: Read to precharge command timing.

Consecutive Reads to Different Rows of Same Bank

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.

Best Case Scenario: Figure 2.17 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 2.17 illustrates that consecutive column read accesses to different rows within the same bank could at best be scheduled with minimum timing of $t_{Burst} + t_{RP} + t_{RCD}$.

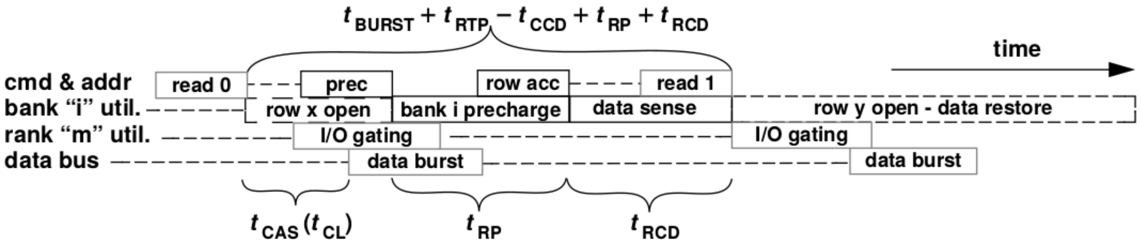


Figure 2.17: Consecutive column-read commands to different rows of the same bank: best-case scenario.

Worst Case Scenario: Figure 2.18 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 scenario shown in Figure 2.17, Figure 2.18 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.

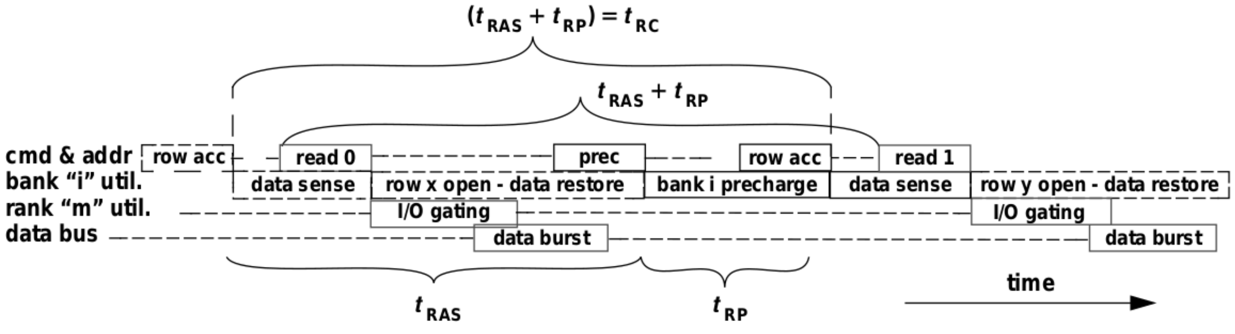


Figure 2.18: Consecutive column-read commands to different rows of the same bank: worst-case scenario.

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.

Consecutive Reads to Different Banks: Bank Conflict

In this section we examine the case of consecutive read requests to different banks with the second request hitting a bank conflict against an active row in that bank. This scenario 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 2.19 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. There are three implicit assumptions in Figure 2.19. The first assumption made 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 2.19 is that the t$_{RAS}$ requirement had been satisfied in bank j, and bank j can me immediately precharged. The third and final assumption 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.

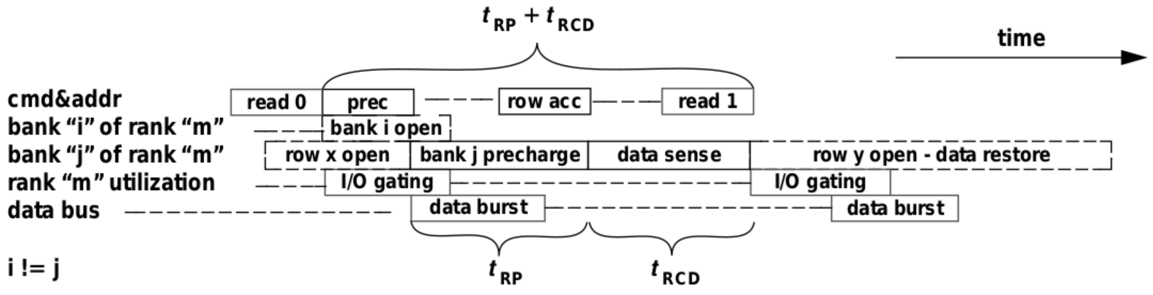


Figure 2.19: Consecutive DRAM read commands to different banks, bank conflict, no command reordering.

Figure 2.19 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 2.19 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 be at best be scheduled with the minimum timing distance of $t_{CMD} + t_{RP} + t_{RCD}$.

With Command Re-Ordering

Figure 2.20 shows that the DRAM memory system can obtain bandwidth utilization if the DRAM controller can interleave or re-order DRAM commands from different transaction requests. Figure 2.20 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 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 utlization 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 2.20 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 2.20 thus illustrates one way that a DRAM memory system can obtain better bandwidth utilization with advanced DRAM controller designs.

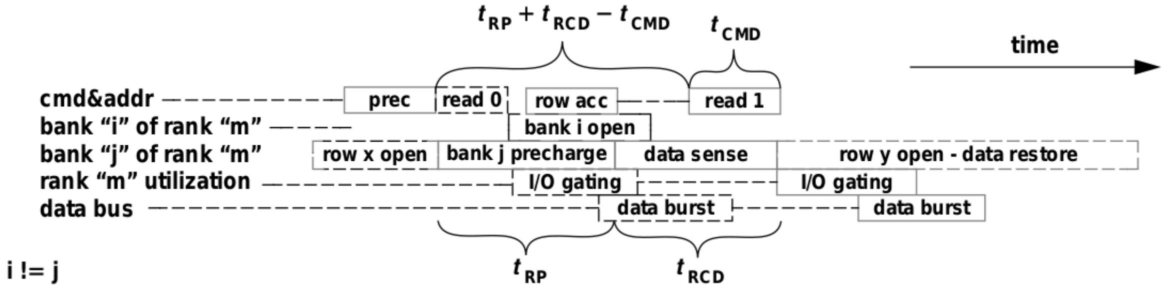


Figure 2.20: Consecutive DRAM read commands to different banks, bank conflict, with command reordering.

Consecutive Read Requests to Different Ranks

Consecutive read commands to the open banks of the same rank of a 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. Figure 2.21 illustrates the timing and command sequence of two consecutive read commands to different ranks. In Figure 2.21, 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. However, for 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.

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

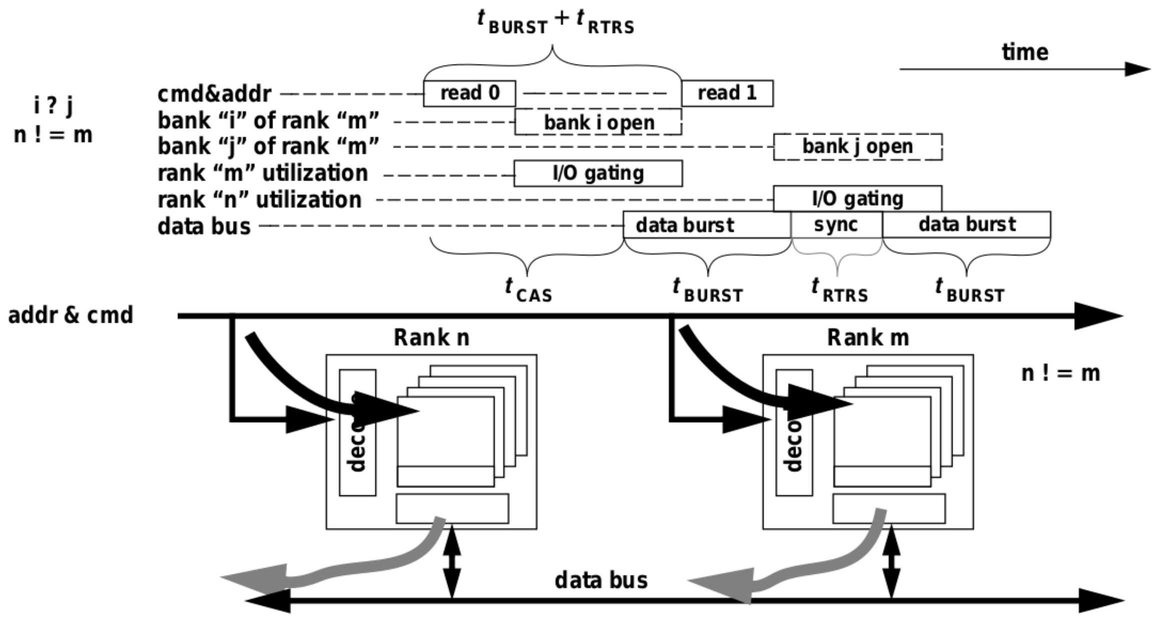


Figure 2.21: Consecutive column-read commands to different ranks.

read commands to different ranks of DRAM devices is that in case of consecutive 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 take control of the shared data bus and send its data to the DRAM memory 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 2.22 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.

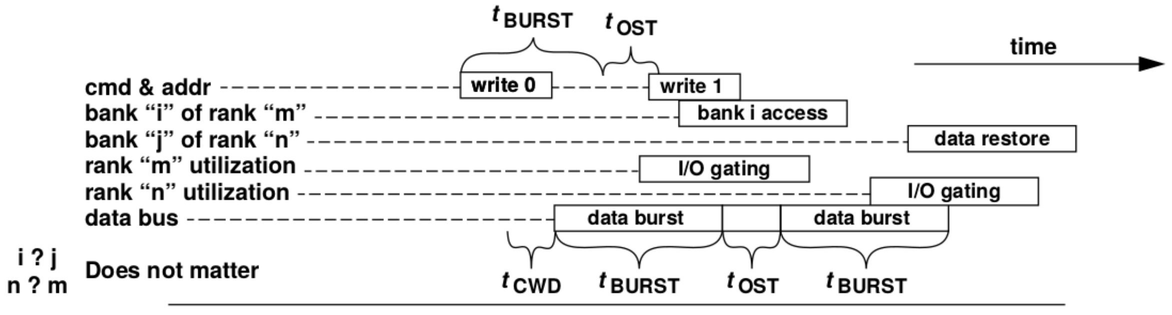


Figure 2.22: Consecutive column-write commands to different ranks.

Consecutive Write Requests: Bank Conflicts

Similar to the case of 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 respect the timing requirements of the write recovery time t$_{WR}$. In case of write commands to 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 2.23 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 $t_{CWD} + t_{Burst} + t_{WR} + t_{RP} + t_{RCD} - t_{CMD}$.

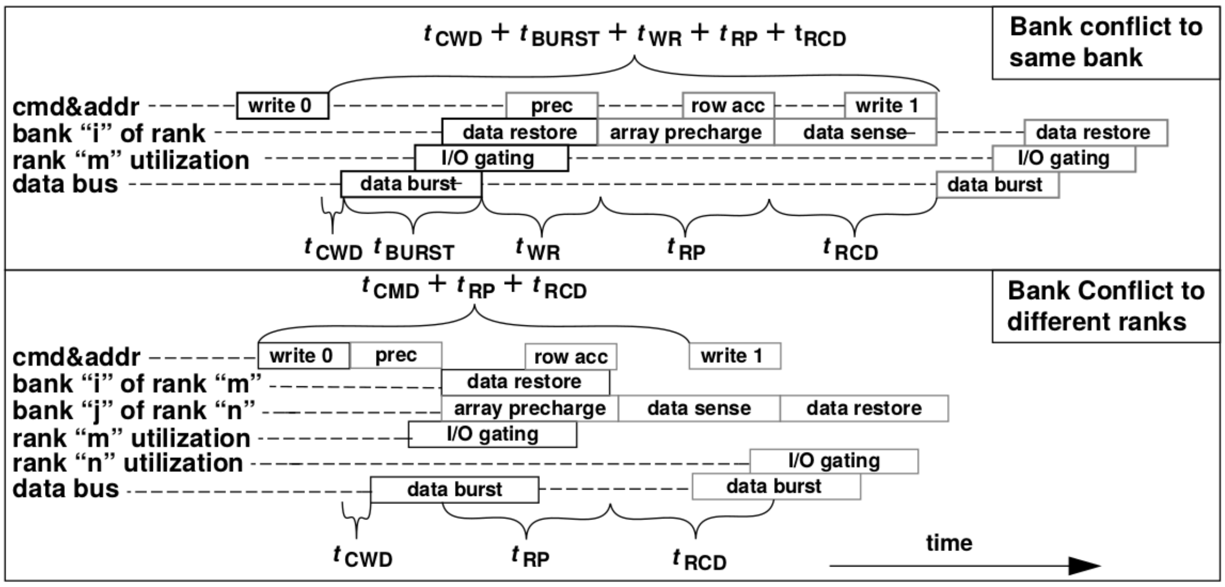


Figure 2.23: Consecutive write commands, bank conflict, best cases.

Figure 2.23 also shows the case where consecutive write requests are issued to different ranks of DRAM devices with the second write request resulting 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 immeadiately. 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.

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 2.24 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 memory controller of the DRAM device. Figure 2.24 shows that the minimum scheduling distance between a read command that follows a read command is $t_{CAS} + t_{Burst} + t_{DQS} - t_{CWD}$.

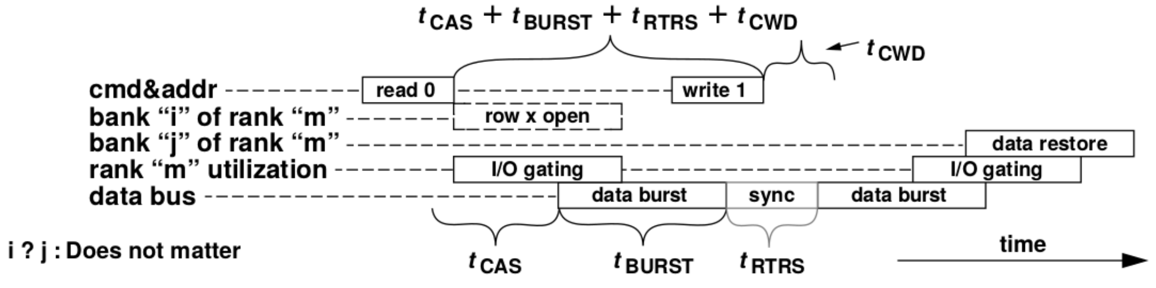


Figure 2.24: Write command following read command to open banks.

Write Following Read: Same Bank, Conflict, Best Case

Figure 2.25 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 2.25, 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 sastisfied for the row held by bank "i" before the read command illustrated in Figure 2.25 was issued into the DRAM memory system. Figure 2.25 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 abnk i is precharged. The column write command can then proceed after time t$_{RCD}$ following the row access command. Figure 2.25 thus shows that a read request that follows a read request to different rows of the same bank can at best occur wih the minimum scheduling distance of $t_{Burst} + t_{RP} + t_{RCD} - t_{CMD}$.

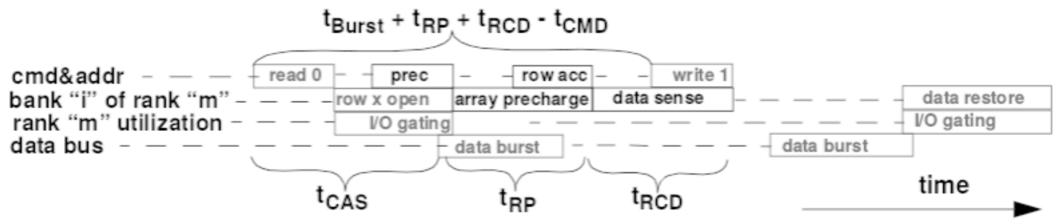


Figure 2.25: Write command following read command to same bank: bank conflict, best case

Figure 2.25 shows 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 case scenario, the minimum scheduling distance between the column read command and the column write command that follows it increases to an entire row cycle t$_{RC}$.

Write Following Read: Different Banks, Conflict, Best Case

Figure 2.26 illustrates the case where a write request follows a read request to different banks. Figure 2.26 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 2.24. However, the assumption given in Figure 2.26 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 memory controller must first precharge bank j and issue a new row access command to bank j before the column 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 2.26 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 $t_{CMD} + t_{RP} + t_{RCD}$.

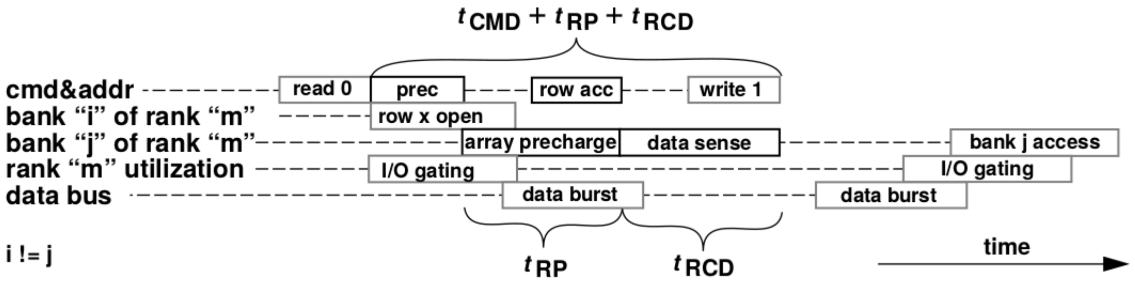


Figure 2.26: Write command following read command to different banks: bank conflict, best case

Figure 2.26 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 not constrained by the column read command to bank i. In a memory system with DRAM memory 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.

Read Following Write to Same Rank, Open Banks

Figure 2.27 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 2.27 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 the same rank of devices. Figure 2.27 also 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 gating resource is shared 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

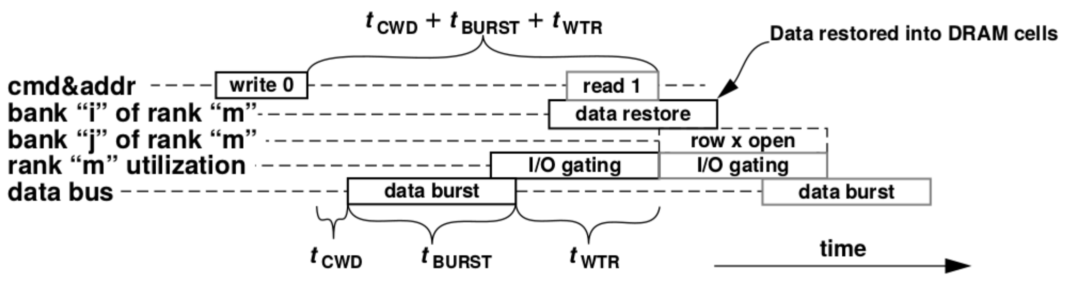


Figure 2.27: Read following write to the same rank of DRAM devices.

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 2.27 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 2.27, 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.

Read Following Write to Different Ranks, Open Banks

Figure 2.28 shows a slightly different case for a column read command that follows a column write command than the case illustrated in Figure 2.27. The combination of column read command issued after a column write command illustrated in Figure 2.28 differs from the combination of column read command issued after a column write command illustrated in Figure 2.27 in that the column write command and the column read command are issued 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.

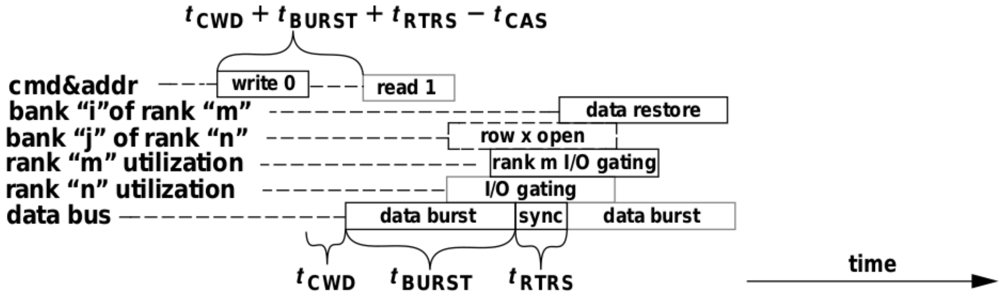


Figure 2.28: Read following write to different ranks of DRAM devices.

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 aread command to different ranks of memory is thus $t_{CWD} + t_{Burst} + t_{DQS} - t_{CAS}$.

Read Following Write to Same Bank, Bank Conflict

Figure 2.29 illustrates the case where a read request follows a write request to different rows of the same rank. In the best case scenario presented, the t$_{RAS}$ row restoration time requirement for the previous row has already been satisfied. Figure 2.29 shows that under this condition, the precharge command can be issued as soon as the data from the column write commands 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 2.29 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 $t_{CWD} + t_{Burst} + t_{WR} + t_{RP} + t_{RCD} - t_{CMD}$.

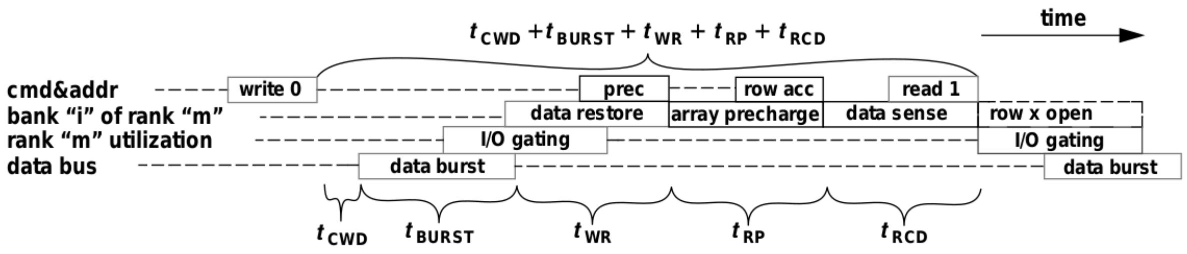


Figure 2.29: Read following write to different rows of the same bank: best case.

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

Figure 2.30 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 the bank j, and a different row is presently active in bank j than the row needed by the read request. Figure 2.30 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 2.30 shows that in this case, the precharge comamnd for the read request command can be issued as soon as the write command is issued. Thus, Figure 2.30 shows that the minimum scheduling distance in this case is $t_{CMD} + t_{RP} + t_{RCD}$.

Figure 2.30 also reveals several important points of note. One obvious point is that the DRAM command sequence illustrated in Figure 2.30 likely benefits from command re-ordering between different memory transactions. A second, less obvious point is that the computed minimum scheduling distance depends on the relative duration of the various timing parameters. That is, Figure 2.30 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

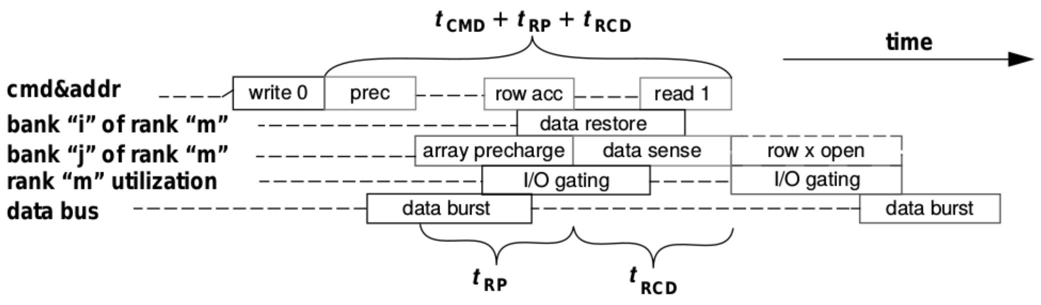


Figure 2.30: Read following write to different banks, bank conflict, best case.

can proceed. That is, the minimum scheduling 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 $t_{CWD} + t_{Burst} + t_{WR}$.

2.4.3 Minimum Scheduling Distances

In previous sections, the resource usage model for DRAM devices was applied to a basic DRAM commands and minimum scheduling distances between different combinations of DRAM commands were examined in detail. Table 2.2 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 2.2 summarizes the minimum scheduling distances between read and write requests rather than between row access, column read, column write and precharge commands. On this table, letter "R" represents a read request, letter "W" represents a write request, letter "s" means that the consecutive requests are made to same channel, rank, bank or row, and "d" means that the requests are made to different channel, rank, bank or row, "o" means open row and "c" means closed 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 the 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 2.2 shows both the best case and worst case minimum scheduling distances for consecutive request 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 [2].

[2]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 2.2 is that a row access is only used in combination with a column access command.

Prev	Next	Rank	Bank	Row	Minimum scheduling distance between DRAM commands open-page. No command reordering 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	$t_{CWD} + t_{RP} + t_{RCD}$	$t_{RC} - t_{Burst}$
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 2.2: Minimum timing for consecutive read and write transactions: open-page.

2.4.4 Power Constraints

Apart of the minimum scheduling distances between DRAM commands, there exist some additional constraints which limit the bandwidth utilization of modern DRAM based memory systems. One important constraint is related with the power consumption of DRAM devices. Due to the increasing demand of bandwidth, DRAM manufactures put high data transfer rates in each successive generation of DRAM devices. However, this increase in operating data frequencies leads to higher activity rates and all in all, higher power consumption.

One of the solutions that have been proposed over the years to lower the power consumption of DRAM devices is to constrain the activity rate of DRAM devices. By doing this, at the same time, we are limiting the capability to move data to/from DRAM devices, which leads to a poorer performance of the DRAM memory system.

In modern DRAM devices like DDR3, when a row is activated, the whole DRAM page consisting of thousands of cells are discharged, sensed, and then restored to their respective cell, all of this in parallel. As the outcome of this action, the row activation command is an energy intensive operation. Figure 2.31 shows the abstract current profile of a DRAM read cycle. An active DRAM device draws a relatively low and constant quiescent current level and for each different activity, the DRAM device draws some additional current, meaning that 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.

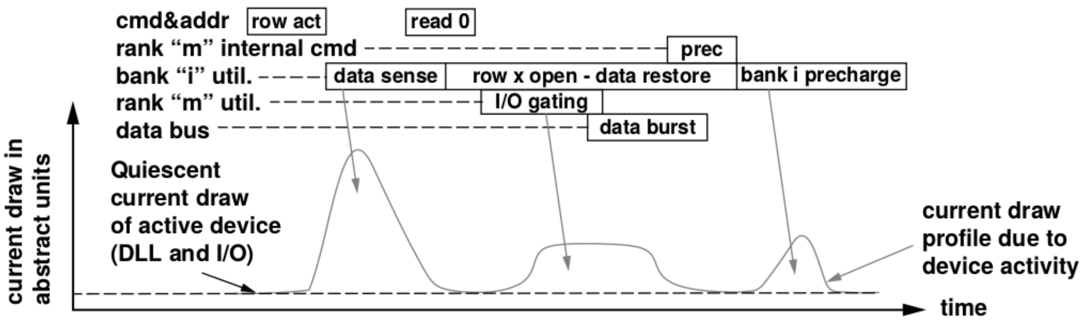

cmd&addr
row act
read 0
rank “m” internal cmd
prec
bank “i” util.
data sense
row x open - data restore
bank i precharge
rank “m” util.
I/O gating
data bus
data burst
current draw in abstract units
Quiescent current draw of active device (DLL and I/O)
current draw profile due to device activity
time

Figure 2.31: Current profile of a DRAM read cycle.

Since the magnitude of the current draw for a row activation command depends on the number of column bits per row and the magnitude of the current draw for data burst and data bus (which depends on the data bus width of the DRAM device), the current profile is shown in an abstract manner. As a result, the current profile of each command on each respective device depends not only on the specific command, but also on the internal organization and external configurations of the DRAM devices.

All modern DRAM devices contain multiple banks of DRAM arrays, which allows to pipeline DRAM commands in order to achieve higher performance. Unfortunately, since the current profile of a DRAM device is proportional to the activity rate, a high performance, highly pipelined DRAM device can also draw a large amount of current. Figure 2.32 shows the individual contributions to the current profile of two pipelined DRAM read cycles on the same device.

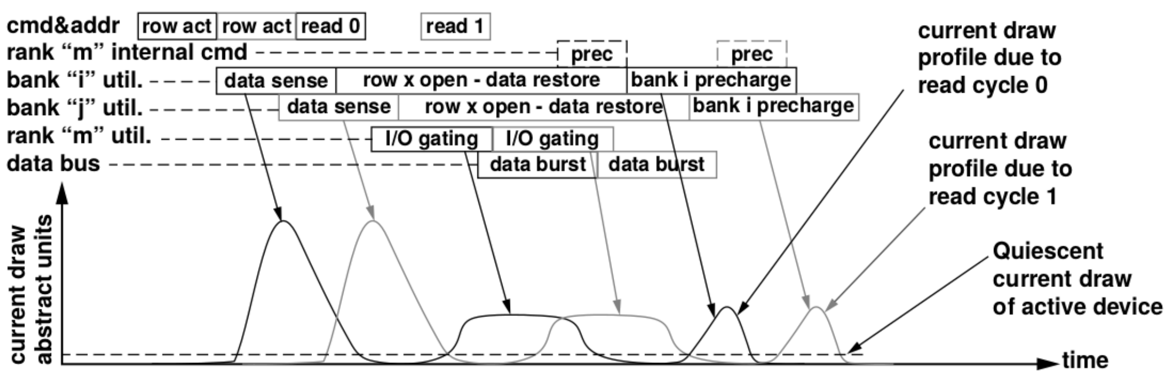


Figure 2.32: Current profile of two pipelined DRAM read cycles.

The total current profile of the pipelined DRAM device is not shown in Figure 2.32, but can be computed by the summation of the quiescent profile and the current profiles of 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 $\frac{t_{RC}}{t_{Burst}}$ 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, new timing parameters have been defined in DRAM devices like DDR3, which limit the activity rate and power consumption.

tFAW: Four Bank Activation Window

In DDR3 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 2.33 shows a sequence of row activation requests to different banks on the same DDR3 SDRAM device that respects both timing constraints t_{RRD} and t_{FAW}. figure 2.33 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.

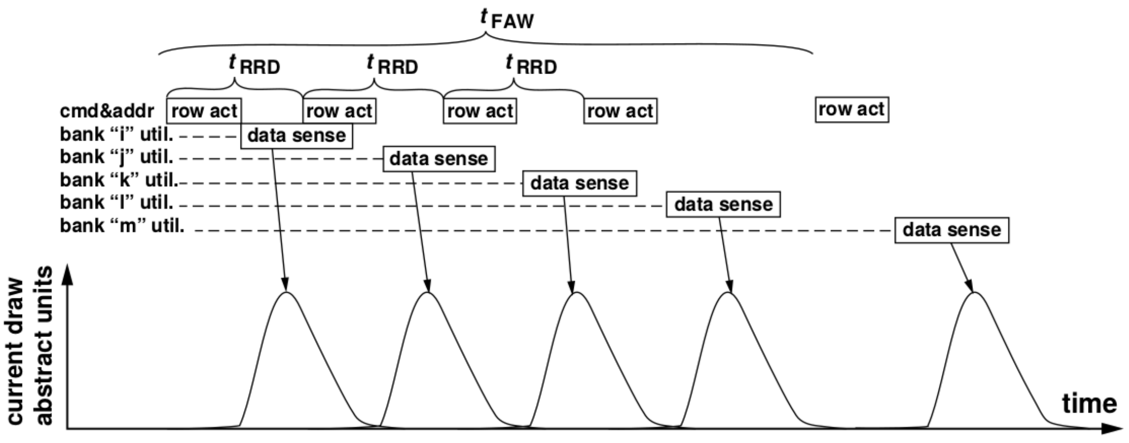


Figure 2.33: Maximum of four row activations in any t_{FAW} time frame.

For memory systems that use close-page policy (row buffer management), t_{FAW} places additional constraint on the maximum sustainable bandwidth of a memory system with a single rank of memory regardless of operating data rates.

2.5 DDR3 SDRAM Protocol

In previous sections, a generic DRAM access protocol was examined in a reasonable detail. In this section, we describe the DDR3 SDRAM memory access protocol in detail, since this is the DRAM device that we are targeting on this work, both in simulation and implementation. The goal of this section is to illustrate by example how the generic DRAM access protocol can be applied to a specific DRAM memory system.

Before any READ or WRITE commands can be issued to a bank within the DRAM, a row in that bank must be activated. This is accomplished via the ACTIVATE command, which selects both the bank and the row to be activated.

After a row is opened with an ACTIVE command, a READ or WRITE command may be issued to that row, subject to the t_{RCD} constraint. When at least one bank is open, any READ-to-READ command delay or WRITE-to-WRITE command delay is restricted to t_{CCD}, as shown in Figure 2.34.

A subsequent ACTIVATE command to a different row in the same bank (bank conflict) can only be issued after the previous active row has been closed (precharged). The minimum time interval between succesive ACTIVE commands to the same bank is defined by t_{RC}.

A subsequent ACTIVE command to another bank can be issued while the first bank is being accessed, which results in a reduction of total row-access overhead. The minimum time interval between successive ACTIVATE commands to different banks is defined by t_{RRD}. No more than four bank ACTIVATE commands may be issued in a given t_{FAW} period, and the t_{RRD} constraint sill applies. It is important to point out that the t_{FAW} parameter applies, regardless of the number of banks already opened or closed, as shown in Figure 2.35.

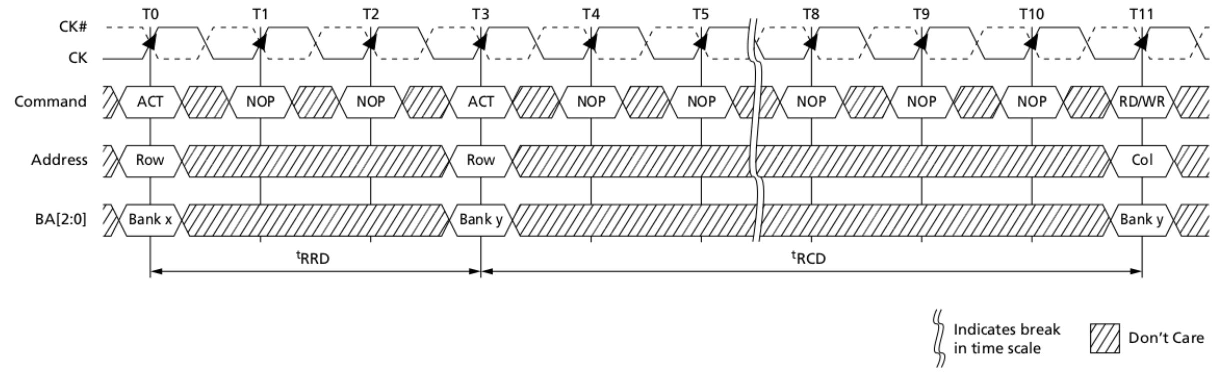


Figure 2.34: ACTIVATE command meeting t_{RRD} and t_{RCD}

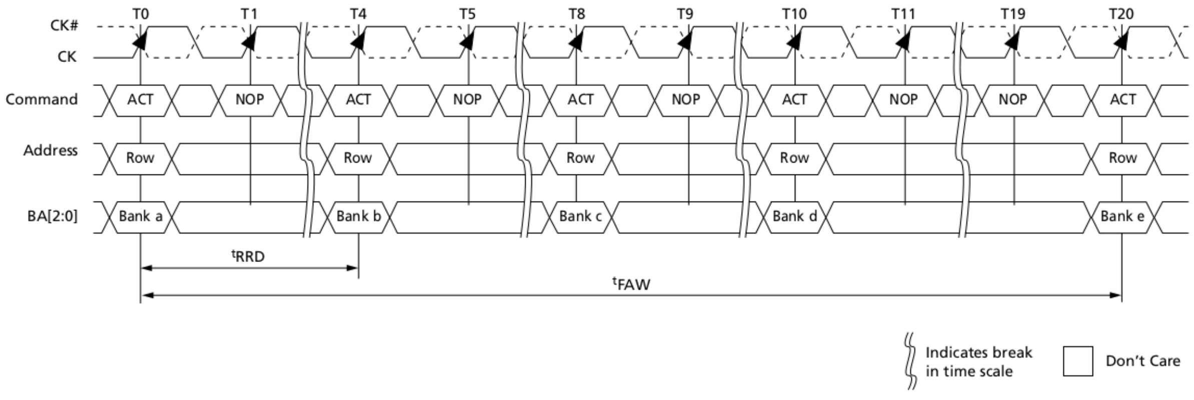


Figure 2.35: t_{FAW} example.

Figure 2.36 shows a read operation, where the read latency is depicted. READ bursts are initiated with a READ command. The starting column and bank addresses are provided at the same time as the READ command, also auto-precharge is either enabled or disabled for that burst access.

During READ bursts, the valid data-out element from the starting column address is available READ latency (RL) clocks later. RL is defined as the sum of posted CAS additive latency

(AL) and CAS latency (CL) ($RL = AL + CL$). The value of AL and CL is programmable in the mode register via the MRS command. Each subsequent data-out element is valid nominally at the next positive or negative clock edge (that is, at the next crossing of CK and CK#). Figure 3.36 shows an example of RL based on a CL setting of 8 and an AL setting of 0.

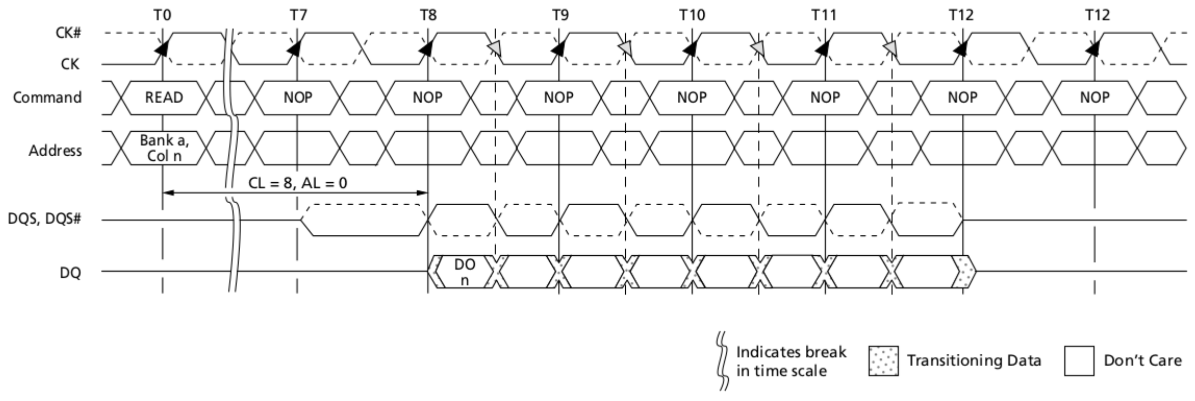


Figure 2.36: Read latency.

DQS, DQS# is driven by the DRAM along with the output data. Upon completition of a burst, assuming no other commands have been initiated, the DQ goes High-Z. Data for column red commands are sent by the DRAM devices and edge aligned to the data strobe signal, on the other hand, data for column write commands are sent by the DRAM controller and center aligned to the data strobe signal.

Data from any READ burst may be concatenated with data from a subsequent READ command to provide a continuous flow of data. The first data element from the new burst follows the last element of a completed burst. The new READ command should be issued t_{CCD} cycles after the first READ command. Figure 2.37 illustrates this for BL8, RL=5 ($CL = 5, AL = 0$).

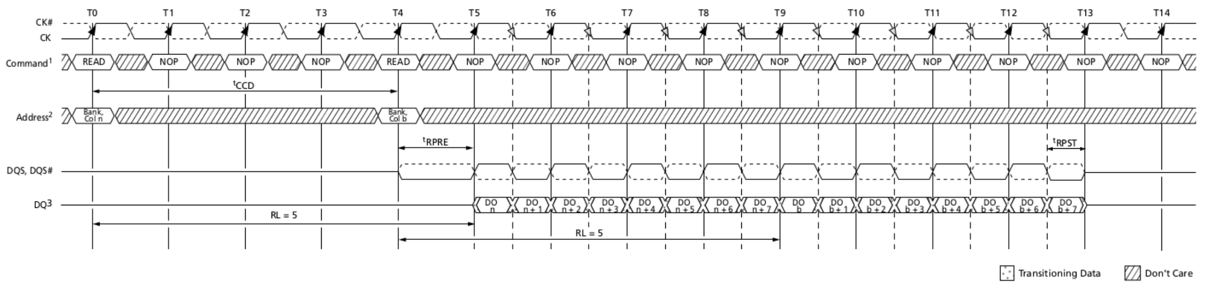


Figure 2.37: Consecutive READ bursts (BL8).

WRITE bursts are initiated with a WRITE command. The starting column and bank addresses are provided with the WRITE command, and auto precharge is either enabled or disabled for that access. After a WRITE command has been issued, the WRITE burst may not be interrupted. Figure 2.38 shows a generic WRITE command.

During WRITE bursts, the first valid data-in element is registered on a rising edge of DQS following the WRITE latency (WL) clocks later and subsequent data elements will be registered on successive edges of DQS. WRITE latency (WL) is defined as the sum of posted CAS additive latency (AL) and CAS WRITE latency (CWL): $WL = AL+CWL$. The values of AL and CWL are programmed in the MR0 and MR2 registers, respectively.

Data may be masked from completing a WRITE using data mask. The data mask occurs on the DM ball aligned to the WRITE data. If DM is LOW, the WRITE completes normally. If DM is HIGH, that bit of data is masked. Upon completion of a burst, assuming no other commands have been initiated, the DQ will remain High-Z, and any additional input data will be ignored.

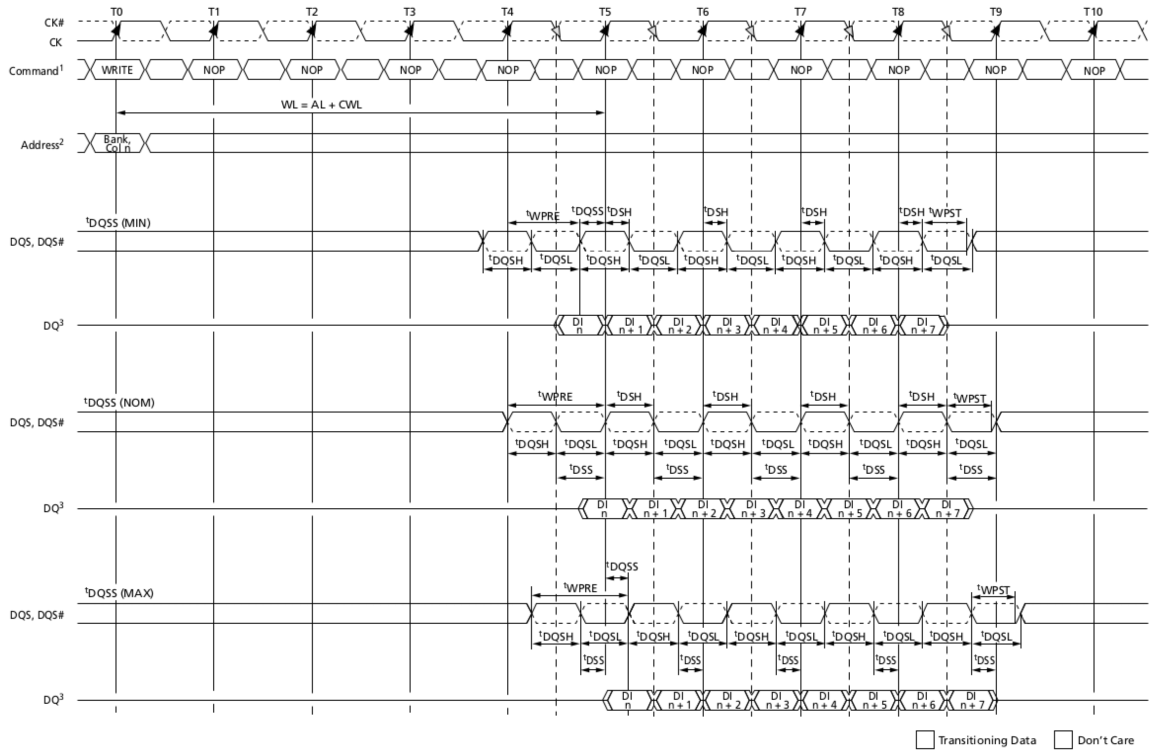


Figure 2.38: WRITE Burst.

Address input-bit 10 determines whether one bank or all banks are to be precharged and, in the case where only one bank is to be precharged, the input bank selects the bank. When all banks are to be precharged, the input bank is ignored. After the bank is precharged, it is in the idle state and must be activated prior to any READ or WRITE commands being used.

2.6 DDR3 memory low power modes

The memory community has been pursuing methods to improve the efficiency of the memory system. A key component of the DDR3 standard is the reduction of power. In DDR3 standard, there exit power modes which can be synchronously entered when CKE is registered low (along with a NOP command). CKE is along to go low while any other operation such as row activation, precharge or auto-precharge and refresh is in progress, but the IDD current effect will be applied until all the operations are finished.

DDL should be in a locked state when power-down is entered for fastest power-down exit timing. During Power-Down, if all banks are closed after any in-progress commands are completed, the device will be in *Precharge Power-Down* mode, otherwise, if any bank is open after in-progress commands are completed, the device will be in *Active Power-Down* mode [2].

Auto-refresh disipates substantial power since all the clocked circuitry in an SDRAM remains active dugin the entire refresh period. As a result, in addition to the power required for refresh, background power is consumed due to the delat locked loop (DLL) and peripheral logic. To save background power, a DRAM device has an option to enter Self-Refresh mode, where all external I/O pins and clocks are disabled, the DLL is truned off, and the devices preserves data without any intervention from the memory controller. Once the self-refresh mode is entered, all banks are precharged and closed, and the internal self-timed refresh is triggered [2]. Self-refresh is the lowest power mode for a DRAM device without losing data.

It is important to mention that the amount if power consumption lowered thanks to the power-down state modes depends on the duration of each power-down mode, and overall, power-down modes could reduce performance because of their *non-zero exit times*. Therefore, the power-down functionality included in a DRAM memory controller must identify the optimal point to enter a power-down state.

2.7 DRAM Memory Controller

In modern computer systems, processors and I/O devices access data in the memory system through the use of one or more memory controllers. Memory controllers manage the movement of data into and out of DRAM devices while ensuring protocol compliance, accounting for DRAM-device-specific electrical characteristics, timing characteristics, and, depending on the specific system, even error detection and correction. [3]

The design and implementation of the DRAM memory controllers determine the access latency and bandwidth efficiency characteristics of the DRAM memory system. The behavior of a contemporary DRAM memory controller is heavily dependent on the interaction of many aspects of the memory system as the memory access pattern and the various DRAM timing constraints. The memory controller employs sophisticated address mapping, command scheduling, and power management optimizations in order to perform the best possible.

The DRAM memory controller functions are to ensure the correct operation of DRAM, which involves to manage the refresh operation every certain period of time (64ms) and obeying the timing constraints imposed by the DRAM device. The memory controller also has to service the DRAM requests while obeying the timing constraints of DRAM chips, some of this constraints are resource contentions or conflicts (bank, bus and channel) and the minimum write-to-read delays. And also the memory controller is in charge of translating the requests to DRAM command sequences. The memory controller can provide better performance by buffering and scheduling the requests, this can be done by reordering the requests, use a different row-buffer policy and to change the way we manage the banks, ranks, and bus. The memory controller is also in charge to manage power consumption and thermals in DRAM devices by the means of managing power modes.

DRAM controllers can be designed to minimize die size, minimize power consumption, maximize system performance, or simply a reasonably optimum combination of various conflict design goals. The goal of this section is to examine, specifically, three of the most important actions that a DRAM memory controller has to accomplish with

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

Due to reasons presented on previous sections, many research has been devoted to the optimization of DRAM memory controllers. Specifically, *Address Mapping Scheme* designed to minimize bank address conflicts have been studied by Lin et. al. and Zhang et. al [4] [5] [6]. *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 [7] [8] [9] [10] [11].

Figure 2.39 illustrates an abstract DRAM memory controller. This memory controller accepts requests from one or more microprocessors and I/O devices. Provides an arbitration interface to determine which of the agents that requested the access to the memory will be able to place its request into the memory controller. This request arbitration is part of the memory controller user interface, even though sometimes it is implemented as part of the bus interface. By including this arbitration as part of the memory controller, it is possible to arbitrate both transaction and command queues, and schedule them in order to achieve better performance.

In Figure 2.39 we can see that once a transaction wins arbitration and enters into the memory controller, it is mapped to a memory address location and converted to a sequence of DRAM commands. In the case of a write operation, the corresponding DRAM commands could be an ACTIVATE followed by a WRITE command. All these commands are put into queues, which can correspond to a specific bank or rank. This queues may be arranged as a generic queue pool, as proposed in [8], where the controller will select one from pending commands to execute in an specific queue. After that, depending on the DRAM command scheduling policy, commands are scheduled to the DRAM devices through the electrical signaling interface, also called physical interface.

We can think of a modern DRAM memory controller as having three main blocks, the physical interface, the command processor and the transaction processor. The physical interface connects to the DRAM chips or memory modules. It takes a single stream of commands from the command processor, sends the commands with proper timing to the DRAM chip, and manages the associated flows of data bytes for read/write operations. The interface

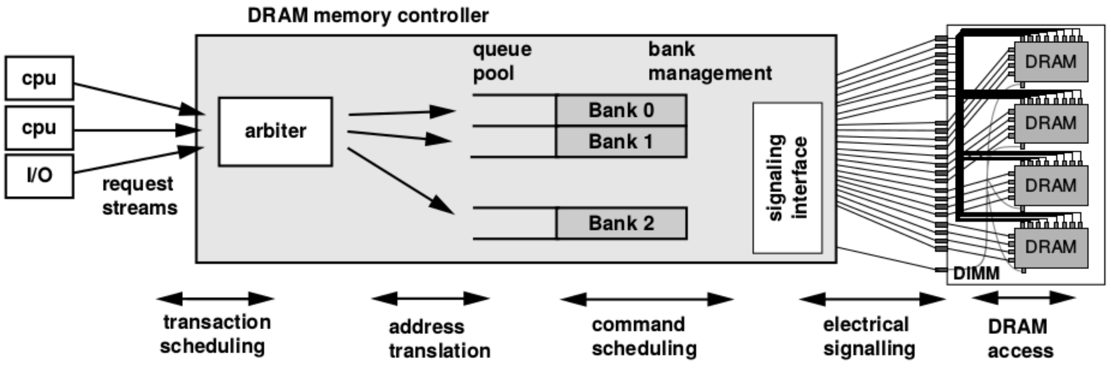


Figure 2.39: Abstract DRAM memory controller.

transceivers, synchronous buffers for commands and data, and a state machine to generate proper command and data timing are all included in this block.

Also, there is a state machine for the complex initialization and calibration sequences specified in the DDR3 DRAM standard [2]. In addition, the physical interface in some applications include self-test, diagnostic, and error-detection and correction hardware. The physical interface may be adjusted when you change to a different size or speed-grade of DRAM device.

The command processor keeps track of the state of the DRAM (ranks and banks), and translates incoming bus read and write cycles into the proper sequence of DRAM commands. For example, the command processor might find a series of bus reads of consecutive words scattered through its input queue, and in response, issue a precharge, an activate and a block-read command to its output queue. In order to do this, the command must know which rows in which banks of which ranks will be open when the new commands get issued.

As the need for bandwidth increases, the complexity of the command processor must increase, too. For example, the processor might look ahead through its input queue and attempt to reorder operations to stay on an active row as long as possible, to overlap reads with precharges, or to interleave banks. Above all, the processor will seek to avoid ping-ponging between rows within one bank (bank conflicts). All these adjustments must be identified and made on the fly.

Finally, the transaction processor sits between the command processor and the rest of the SoC. It typically has a number of channels connecting it to the SoC's high-speed central switch. The main job of the transaction processor is to blend together the streams of reads and writes coming in on the various channels, imposing a priority scheme so that each channel, i.e., each cache controller, DMA engine, accelerator, etc., on the other ends of those channels, get the latency and bandwidth they need.

Selecting such a priority scheme in a dynamic environment is not easy. It is more difficult if you can't accurately predict the characteristics of the traffic on each channel. Ideally, the workload would be fixed, so you could optimize a priority scheme for it or there would be several clearly identified modes of access, and provision for dynamically adjusting priorities as the traffic shifted.

2.7.1 Row Buffer Management Policy

Two of the most typically used row-buffer policies to manage the operation of the sense amplifiers are *open-page* and *closed-page* policies. There exist dynamic row-buffer management polices, like the ones mentioned below.

The *closed-page* policy, opens the row for every column access and then closes it by means of an auto-precharge command. The improve *adaptive closed page* keeps the row open if there are any pending accesses to that row in the queue, as suggested in [12].

On the other hand, the *open-page* policy leaves a row open until a bank conflict occurs, in which case the the open row is closed and the new row opened. As before, the *open adaptive page* does not wait until the conflict occurs, but instead closes the page in advance if there are accesses to a different row in the same bank (bank conflict), and also there are no queued accesses to the open row [12].

In more detail, the *open-page* row-buffer management policy is designed to improve the row hits. This policy favors the memory accesses to the same row by keeping the sense amplifiers open and holding an entire DRAM page (row of data) in order to be ready for access it. The primary assumption in this policy is that once a row of data is brought to the array of sense amplifiers, different columns of the same row can be accessed again in the future, meaning the some *spatial locality* intrinsic in the data access is expected.

So, in the case that another memory read access is made to the same row, that memory access could occur with the expense of the minimal latency which is t_{CAS}, this is due since the row is already active, a column access command timing is needed to move the data from the sense amplifiers to the memory controller. However, in the case of a bank conflict (an access to a different row of the same bank), the memory controller has to first precharge the DRAM array, perform another row access, and then perform the column access. The minimum access latency to access data in the case of a bank conflict is $t_{RP} + t_{RCD} + t_{CAS}$.

Contrary to the *open-page* policy, *closed-page* row buffer management policy does not assume any locality in the data access since it is designed to favor random accesses to different rows of memory. This means that after activating and accessing a row, it is precharged automatically my means of a READ-PRECHARGE command, this allows the bank to be ready to be access later on.

The bottom line is that the definition of the row-buffer access policy forms the foundation for the design of a memory controller. This specific design choice directly or indirectly impact the selection of the address mapping scheme, the memory command-reordering mechanism, and the transaction re-ordering mechanism.

2.7.2 Address Mapping Policies

The address mapping process does the following: take an address from system's memory address space and perform the mapping to the organization of the DRAM memory system. This means that a request address is translated to a *channel {L}*, *rank {R}*, *bank {B}*, *row {R}* and *column {C}* values, specifying the location of the data. The address mapping scheme determines which bits of the address are used for each specific "dimension" or "logical" representation of the DRAM memory system.

Elaborating the address mapping process, by requesting two different rows of the same bank, we would have to close the first row and open the second, which leads to a significant increment in latency. If the request instead were to access the same row but different columns, they could be executed one after the other after the initial access. This would take only t_{CAS}, in addition of using *less energy* by avoiding the extra ACTIVATE and PRECHARGE commands.

Now, another scenario is the following: we have two request to different banks, in which case we would have to pay the latency of an ACTIVATE command per each bank, but at the same time, the second ACTIVATE command would need to be delayed some cycles in order to satisfy t_{RRD}. In spite of being able to hide this latency because of the delay necessary in the bus, the energy consumed is higher, since two rows would be activated, instead of one. Accessing two different channels would be even more advantageous in terms of speed, since they are fully independent, but the energy problem would remain.

The data mapping policy determines the extent of parallelism that can be leveraged within the memory system. A cache line is placed entirely in one bank, the next cache line could be placed in the same row, or in the next row of the same bank, or in the next bank in the same rank, or in the next rank in the same channel, or in the next channel.

2.7.3 DRAM metrics

Apart from the already presented metrics used to assess performance in a DRAM memory system (bandwidth and latency), a set of parameters can be used to describe the DRAM usage pattern and performance in addition to achieve throughput, or bandwidth. Both *request pattern* and *memory controller* affect these parameters.

- DRAM utilization
- DRAM efficiency
- Row buffer locality

DRAM *utilization* is the percentage of cycles within a time period the data bus was employed to transfer data [13]. While at first sight we would like to have this value to be high, it can be misleading as a memory scheduler measure if there were significant intervals with no requests at the controller.

DRAM *efficiency* was defined to improve upon DRAM utilization [13] and is defined as the percentage of the non-idle DRAM cycles the data bus was utilized. This removes the time which during the memory controller and DRAM have no work to perform, and so, it presents a more indicative performance measure in the case when there are significant idle times.

Row Buffer Locality is defined as the percentage of requests which hit the already opened row. High locality means the cost of opening a new row is amortized over a large number of serviced requests. However, high locality can have a negative side-effect of delaying requests that access different rows of the same bank, overall in multithreaded systems that have a high chance of interthread interference by keeping a bank accupied through repeated access to a single row, as pointed by Kaseridis et al. [14].

2.7.4 Memory Scheduling

The term *memory scheduler* is often used in a mislead way that overlaps with the memory controller. In this work, the *memory scheduler* is the part of the memory controller that is in charge of the selection of specific DRAM commands to be issued. DRAM scheduling is a complex problem, requiring a delicate balance between circumventing access scheduling constraints, prioritizing requests properly, and adapting to a dynamically changing memory reference stream.

As mentioned in the latter paragraph, memory access scheduling is the process of ordering the DRAM operations (bank precharge, row activation, column access, between others). All these operations have to honor several timing constraints for each rank and bank. In memory scheduling has to be taken into account the data bus utilization for read and write operations on the same DDR bus. There is also the refresh time that is inherently tied to the DRAM technology. The aim of the algorithm used in memory scheduling is to maximize the row buffer hit rates and parallelism, leading to a maximum utilization of the bandwidth available in the DRAM memory system. Also, this algorithm can leverage the low power modes available in modern DRAM devices like DDR3 in order to lower power consumption.

The memory controller receives requests and place them into a single *Transaction Queue*. The write requests and their corresponding data is usually placed on a separate *Write Queue*. Then, any incoming incoming read request need to be checked against the write requests waiting in the write queue in order to maintain **consistency**. This action also provides the opportunity to immediately service a read request without the need of accessing the DRAM devices.

Then, requests from the queues aforementioned are translated into a series of DRAM commands. The simplest translation is to wrap every incoming request into a PRECHARGE, ACTIVATE and READ/WRITE command. After this translation, these commands are placed into queues, called *Command Queues*. As we already mentioned before, there can be a single queue per memory controller, one command queue per rank or even one command queue per each bank. Then, the memory scheduler issues the commands to the DRAM devices. It keeps track of the timing constraints, issuing commands only when the appropriate timing constraints are met and at the same time, it makes sure that no data is lost in the DRAM devices by issuing REFRESH commands at the appropriate time.

Some state-of-the-art memory scheduler are presented in the following subsections. First, the in-order scheuler is presented as a baseline, then the mostly used scheduler is the *First Ready, First Come, First Served*. Also, the Thread-Fair memory scheduler [15] is presented. More advanced schedulers have been studied by Nazm et. al [16], O. Multu et. al [17] and Y. Kim et. al [18].

In-order

Basically, the in-order scheduler issues commands in the strict order of arrival. A very good example of how inefficient this scheduling can be is illustrated by Rixner et. al at [8].

First Ready - First Come - First Served

This widely used scheduler is simple, a greedy request scheduling policy proposed by Rixner et. al at [8]. Due its popularity, it is often used as a baseline memory scheduling policy since its low complexity is coupled with its performance. There exist variations of this scheduler, but the base version is the following. The *FR-FCFS* scheduler chooses one command at very cycle from all the ACTIVE, READ and WRITE that satisfy the timing constraints, hence the *First Ready* part of the name. Out of that set of issuable commands, the oldest command will be picked, hence the *First Come, First Served*. This means that a newer command will be issued over an older one if the timing constraints prevent the former from issuing. In the case of the PRECHARGE commands, they are only considered if no other commands can be issued.

The specific timing of the PRECHARGE commands depends on something called the *Row-status policy*, which determined the rule of closing the row, therefore it is tied with the choice of the maximum number of row accesses. There are mainly three options: *single access*, *open row* and *closed-row*. As we can imagine, the single access only allows an access to a row before it is closed by a PRECHARGE command, leading to a very simple and poor scheduling. Then, if we want to allow multiple row accesses the question of when to close the row inevitable arises. If the row is closed once there are no more accesses to it in the command queueu, this could slow down future accesses to that row. Then, the *closed-row* policy sends a PRECHARGE command as soon as the command queue does not have accesses to it. And the *open-row* policy sends a PRECHARGE command only once the command queue contains no accesses to the open row and at least one access to a different row of the same bank.

Thread-Fair Memory Scheduler

The Thread-Fair scheduler presented by Fang et. al [15] establishes some simple rules, and leverages additional information to prioritize the oldest request from each thread. This is possible due the oldest request is expected to have the highest chance of blocking the *processor's* program. Figure 2.40 sumarizes the scheduling rules.

Under normal operation, read requests are prioritized over write requests. If the write queue is filled above the *high watermark*, or there are no read requests, enter the *write drain* mode in which writes have higher priority. While read requests are being serviced, read row hits have

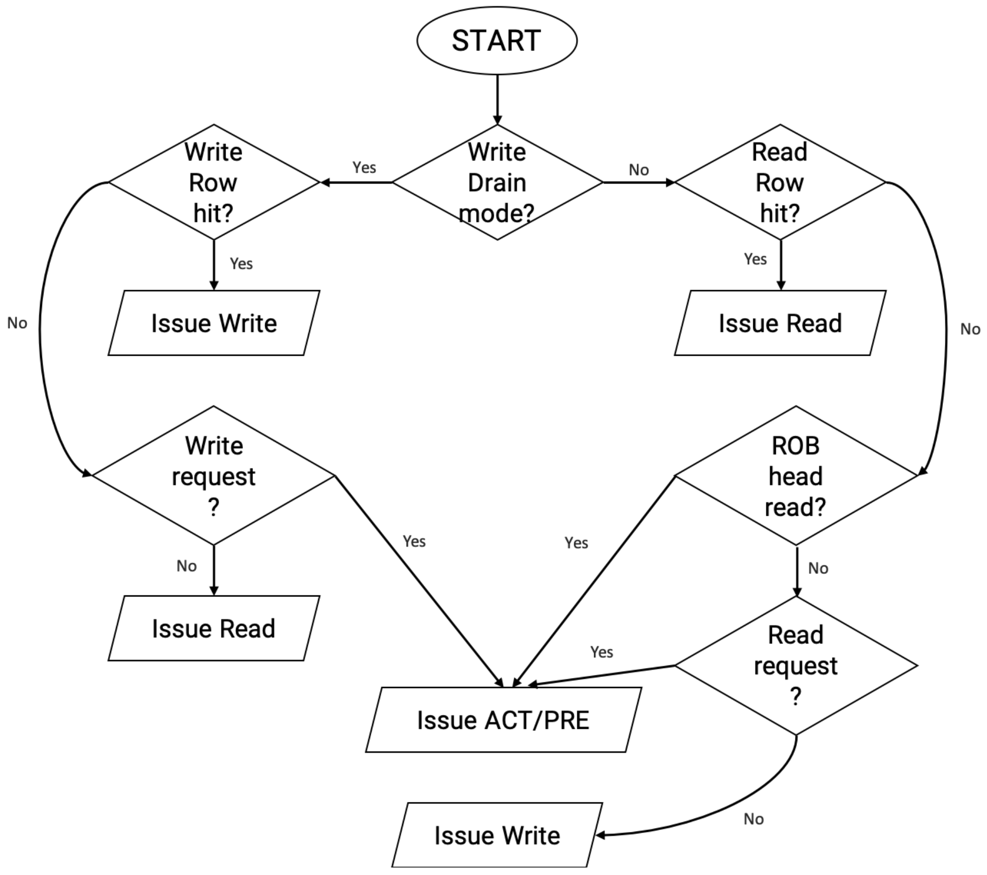


Figure 2.40: Thread Fair scheduler decision flowchart.

the highest priority. If no read row hits can be issued, the request generated by the reorder buffer (ROB) head is given the highest priority and appropriate row is opened. If multiple such request exits, they are considered in a round-robin fashion. If all requests generated by the ROB heads have the corresponding rows open, or the required banks are busy, rows are opened for other read requests by the oldests-first rule. Finally, if no row opening commands can be issued, write row hits are considered issuing.

Correspondingly, when in the write drain mode, write row hits have the highest priority. If the queue holds no row hits, remaining writes are serviced according to age. Finally, if no write requests can be serviced, read row hits will be issued. The Thread-Fair scheduler takes advantage of the auto-precharge commands. When issuing column access commands, if there are no more pending accesses that would hit in the open row, last column access is issued as the *auto-precharge* command. This way does have the potential of closing the row too soon.

2.8 Related work

In the previous section, a couple of schedulers were presented. In this section, we will describe another common design target which is power consumption.

2.8.1 Power and energy oriented schedulers

Since the energy usage is often tightly coupled with improving performance, whereas power considerations are often harmful to application performance. So, the peak power usage is already limited by the FAW (Four-Row Activation Window) in order to ensure the proper operation of the hardware.

One technique for controlling the power usage of the DRAM system is called throttling, and is examined by Hanson and Rajamani [19]. The basic idea behind memory throttling is to restrict read and write traffic to main memory as a means of controlling power consumption. This can be done by restricting the number of accesses within a fixed time period to some level below the peak (or normal) hardware capability. The impact of this restriction can vary from minimal to significant. As we could imagine, the downside of this technique is the performance degradation.

Hur and Lin propose to use a timeout counter to generate power-down commands [20]. And they extend the *Adaptive History-Based* scheduler [11] with additional policy which prefers issuing commands only to one rank at a time, allowing longer power-down periods.

Another two energy-saving proposal are presented by Huang et al. [21]. One of the proposals attempts to improve the self-refresh mode of DRAM since this mode saves large amount of energy, but as we already stated, the exit time of this mode is long. The basic idea is to use the history of accesses within last refresh interval to modify the time threshold for entering self-refresh mode. The other proposal is to move the most frequently accessed pages into a specific subset of ranks, which would allow longer and more frequent power-saving modes for the other ranks. The downside of this proposal is the penalty in terms of performance that comes with the page migration.

The refresh operation is one of the main DRAM characteristics that can be leveraged to lower power consumption. Having in mind that the REFRESH operation consists of an ACTIVATE and PRECHARGE command per row, we could think about a kind of *Smart Refresh*, like the one proposed by Ghosh and Lee [22], on which a mechanism keeps track of the rows' last refresh through either a normal activation or a REFRESH operation. Although rows are only refreshed when necessary, the counters used to keep track of the rows require a large amount of storage in the memory controlller.

2.8.2 Related work summary

In this section we gave an overview of the state of the art. The memory schedulers are mainly divided in three groups (performance, fairness and energy), but since this work is focused on low power consumption, the one presented was energy oriented scheduling. The schedulers presented in the former subsection are not the ones implemented on this thesis, with the exception of *FR-FCFS* and *FCFS*, but they are part of the theoretical background provided to understand this work.

Chapter 3

Methodology

In this chapter, we describe the infrastructure employed for the evaluation presented in the following chapter. We follow two approaches to demonstrate the behavior of the memory controller model proposal, a memory traffic generator and the execution of some PARSEC [23] benchmarks in a full-system simulation running Linux. In this chapter the memory controller proposal is described in detail as well.

3.1 Simulation framework

gem5 is an open source **discrete event simulator** framework [24]. To be able to study the DRAM's impact on system power and performance, it is necessary to have a controller model that is representative with respect to the architecture of the memory controller, and how this controller manages all DRAM timing constraints and also, the optimization goals and constraints.

gem5's main memory model captures a generic modern DRAM controller architecture, with split read and write queues, as illustrated in Figure 3.1. Instead of buffering the DRAM state by bank or rank, gem5's controller model does it per memory controller, so they do not model the actual DRAM device.

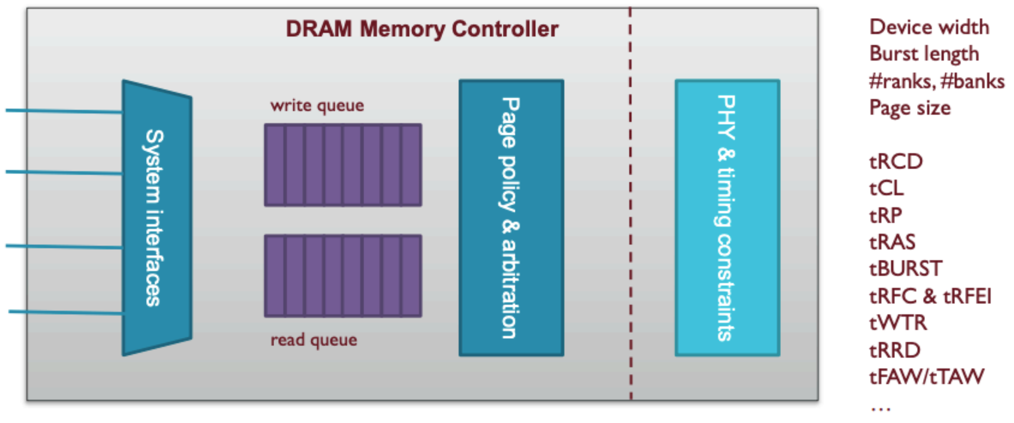


Figure 3.1: gem5 DRAM memory controller model.

The controller model has parameters that allow us to express the memory system organization. Some of these parameters are bus width, burst length, row buffer size, number of devices per rank, ranks and banks. Table 3.1 shows the DRAM controller parameters available in gem5's controller model.

Parameter	**Description (unit)**
Write buffer size	Number of write queue entries
Read buffer size	Number of read queue entries
Write high/low threshold	High/low watermark for write queue
Scheduling policy	FCFS or FR-FCFS
Address mapping	RoRaBaCoCh, RoRaBaChCo, RoCoRaBaCh
Page policy	Open or closed (adaptive or not)
Frontend latency	Static frontend latency (ns)
Backend latency	Static backend latency (ns)
Device bus width	Data bus with per DRAM device (bits)
Burst length	DRAM burst length (beats)
Row-buffer size	Device row buffer size (bytes)
Devices per rank	-
Ranks per channel	-
Banks per rank	-
Channels	Channel count for the address decoding
t_{RCD}	Row to column delay (ns)
t_{RAS}	Row access strobe (ns)
t_{RP}	Row precharge time (ns)
t_{CL}	Column access latency (ns)
t_{BURST}	Burst duration (ns)
t_{RFC}	Refresh cycle time (ns)
t_{REFI}	Refresh command interval (ns)
t_{WTR}	Write to read switching time (ns)
t_{RRD}	Row to row activation delay (ns)
t_{XAW}	Activation window (ns)
Activation limit	Number of activates in window

Table 3.1: gem5's DRAM model controller parameters

The model used on gem5 does not model any separate command queues, in contrast to other simulators like DRAMSim2 [25]. Therefore, the page policy and arbitration scheme operate directly on the read and write request queue. This model claims that capturing the split request and command queues adds unnecessary cost and detail to the model, and it is demonstrated at [26].

gem5's DRAM model controller models a simplified DRAM state machine, which is implicitly encoded in the controller code. The only timing constraints that are modelled are the ones that have significant contribution to system level performance, as show in Table 3.1. The model captures a subset of the DRAM bank state changes, along with the data bus occupancy, and return of DRAM read data.

The are two levels of scheduling, the first is the choice between reads and writes, on which a write drain mode is employed in order to minimize the cost of read/write switching (tWTR). The second level involves selecting a request either from the read or write queue, depending on the current direction and page policy.

3.1.1 Power Model

The simulated DDR3-1600 x64 consists of 8 chips each having a 8-bit interface (x8), with timing and power based on the Micron DDR3 SDRAM MT8JTF12864HZ-1GB [27]. gem5 employs a tool called DRAMPower tool [28] to calculate energy components based on the I$_{DD}$ current values of a specific device. This tool employs an improved Power Modeling presented on [29], which will be described briefly later on. This SDRAM power model estimates power consumption during the state transitions to power-saving states, employs an SDRAM command trace to get the actual timings between the commands issued and is generic and applicable to al DDRx SDRAMs and all memory controller policies and all degrees of bank interleaving.

This Power Model includes basic power components that add up and contribute to overall memory power consumption. These basic components include background power components (contributing mainly to static power consumption), such as Active Background (ACT$_{BG}$) and Precharged Background (PRE$_{BG}$) power, and active power components (contributing mainly to dynamic power consumption), such as Activate (ACT), Precharge (PRE), Read (RD), Write (WR) and Refresh (REF) power. Also, the transitions from stand-by modes to power-down modes power components contributions are taken into account on this model.

Background Power

If all memory banks are in the precharged stand-by state, the memory consumes a precharge background current (static power component) of I_{DD2N}. However, even if a single bank is in the active state, the memory consumes an active background current (also a static power component) of I_{DD3N}. Then, if a bank stays in the active state for a period of $tRAS_{new}$ cycles out of the total transaction length of tRC_{new} cycles, it consumes an average P(ACT$_{BG}$) static power per cycle for the entire transaction length, as shown in Equation (3.1). If on the other hand, all the banks remain in the precharged state for tRP_{new} cycles, the memory consumes an average P(PRE$_{BG}$) static power per cycle, given by Equation (3.2) for the entire transaction length.

$$P\left(ACT_{BG}\right) = \sum_{n=1}^{tRAS_{new}} I_{DD3N} \times \frac{V_{DD}}{t_{RC_{new}}} \tag{3.1}$$

$$P\left(PRE_{BG}\right) = \sum_{n=1}^{tRP_{new}} I_{DD2N} \times \frac{V_{DD}}{t_{RC_{new}}} \tag{3.2}$$

Activate and Precharge Command Power

I$_{DD0}$ is specified as the average current consumed by the memory when it executes an ACT command and a PRE command, within the minimum timing constraints. The I$_{DD0}$ current value also includes the active background current I$_{DD3N}$ for the minimum period for which the row is active (*tRAS*) and the precharge current I$_{DD2N}$ for the minimum period for which the row is precharged (*tTC* - *tRAS*). Hence, these should be substracted from I$_{DD0}$ for the appropiate durations and averaged over the transaction length t$_{RC_{new}}$ to identify the average power consumed only due to the ACT and PRE commands. Equations 3.3 and 3.4 show P(ACT) and P(PRE) providing estimates by using the same total average current of I$_{DD0}$ and apply it separately to the two components, based on the ratio of the number of active cycles to precharge cycles in the transaction.

$$P\left(ACT\right) = \sum_{n=1}^{tRAS} \left(I_{DD0} - I_{DD3N}\right) \times \frac{V_{DD}}{t_{RC_{new}}} \tag{3.3}$$

$$P\left(PRE\right) = \sum_{n=tRAS+1}^{tRC} \left(I_{DD0} - I_{DD2N}\right) \times \frac{V_{DD}}{t_{RC_{new}}} \tag{3.4}$$

Read and Write Command Power

A read command consumes I$_{DD4R}$ average current during the cycles of the data transfer, while a Write command consumes I$_{DD4W}$. Since these also include the active background current values consumed during the read or the write, I$_{DD3N}$ must be substracted from the I$_{DD4R}$ and I$_{DD4W}$ currents, to identify the power associated only with the Read and the Write commands, respectively. The power values are scaled over the transaction length t$_{RC_{new}}$ to get the average power consumed by a Read and a Write, given the Equations (3.5) and (3.6) respectively. tR and tW represent the number of cycles the data is on the data bus when reading and writing to the SDRAM, respectively.

$$P\left(RD\right) = \sum_{n=1}^{tR} \left(I_{DD4R} - I_{DD3N}\right) \times \frac{V_{DD}}{t_{RC_{new}}} \tag{3.5}$$

$$P\left(WR\right) = \sum_{n=1}^{tW} \left(I_{DD4W} - I_{DD3N}\right) \times \frac{V_{DD}}{t_{RC_{new}}} \tag{3.6}$$

Refresh Power

A refresh command consists of a single Refresh command along with a set of pre-refresh NOPs that give enough time (at least tRP cycles) to precharge all the banks each before executing the refresh. Accordingly, P(PRE) (Equation 3.4) is consumed (with a transaction length of tRP) for the number of precharges (N(PRE)) issued and I$_{DD2N}$ current is consumed for the tRP cycles associated with those Precharges. The refresh command by itself, consumes I$_{DD5}$ current over the refresh cycles (t$_{RFC}$). The refresh and pre-refresh power components add up over tREF ($= tRP + tRFC$) cycles to give the total refresh power, as shown in Equation (3.7).

$$P\left(WR\right) = \sum_{n=1}^{tRP} \frac{((I_{DD2N} \times V_{DD}) + (N(PRE) \times P(PRE)))}{tREF} + \sum_{n=1}^{tRFC} I_{DD5} \times \frac{V_{DD}}{tREF} \quad (3.7)$$

3.2 Traffic generators

gem5 provides a module called *TrafficGen* that allows synthetic traffic generation. The traffic generator has a single master port that is used to send requests, independent of the specific behavior. The behavior of the traffic generator is specified in a configuration file, and this file describes a state transition graph where each state is a specific generator behavior. Examples including idling, generating linear address sequences, random sequences and replay of captured traces. By describing these behaviors as states, it is straightforward to create very complex behaviors, simply by arranging them in graphs. The graph transitions can also be annotated with probabilities, effectively making it a Markov Chain.

3.3 PARSEC Benchmarks

The Princeton Application Repository for Shared-Memory Computers *PARSEC* Benchmark suite [23] was intended to fulfill the need of a benchmark suite that can be used to design the next generation of processors, *i.e.* Chip-Multiprocessors (CMPs). It consists of 13 workloads which were chosen from several representative application domains, ranging from scientific computing to engineering. PARSEC workloads were selected to include different combinations of parallel models, machone requirements and runtime behaviors. All benchmarks are written in C/C++ because of the continuing popularity of these languages in the near future.

We selected 4 out of all the workloads available to evaluate our memory controller proposal. These workloads are

- **blackscholes** This application is an Intel RMS benchmark. It calculates the prices of a portfolio of European options analytically with the Black-Scholes partial differential equation.
- **ferret** This application is based on the Ferret toolkit which is used for content-based similarity search.
- **fluidanimate** This Intel RMS application uses an extension of the Smoothed Particle Hydrodynamics (SPH) method to simulate an incompressible fluid for interactive animation purposes.
- **swaptions** The application is an Intel RMS workload which uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of swaptions. *Swaptions* employs Monte Carlo (MC) simulation to compute the prices.

3.4 Memory Controller proposal

The memory controller proposal is based on gem5's controller model. The low-power functionality implements a staggered powerdown similar to that described in [30]. This model captures a subset of the DRAM bank state changes, along with the data bus occupancy, and the return of DRAM read data. The events that are tracked are the following. First, the controller tracks when a bank is ready to execute a new command (possibly with auto precharge). Second, the controller records when a bank is able to precharge, i.e. when tRAS has been satisfied. Third, the controller tracks the earliest possible time a bank is allowed to be activated. With this, it is possible to approximate the DRAM state machine and respect the timings presented on Table 3.1.

In addition to the bank state, the model tracks the busy periods of the data bus, and also tracks when a refresh is due. Upon scheduling an access, the controller determines the time when response data is returned, and update the bank and data bus availability. [26]

We propose an **adaptive page policy** and a **write drain policy** which will be described in the following subsections.

3.4.1 Delayed Adaptive Closed Page Policy

We propose a *closed page adaptive policy*. When there is an access to a closed row, the controller opens the row for every column access, and **keeps the row open** if there is a certain number of queued accesses to the open row (threshold).

Per-bank delayed close scheduling algorithm references row hit rate periodically in order to determine whether postpone a precharge or not. The number of read commands and active commands per bank are observed, *increasing a read history counter* per read operation and *decreasing per active command*. For every 10k CPU cycles, if the value of the read history counter exceeds zero, this means that there exists some locality in the memory accesses and the closed page policy *change* has to be delayed, meaning that the row is kept open, according to the rules specified before. If on the other hand, the value of the counter is less than zero, this means that the number of active commands is higher than the number of read commands, changing to a closed page policy.

3.4.2 Write Drain Policy Exploiting Row Buffer Locality

Traditional Write Draining like the one proposed in [12] is to drain writes when there are no-pending read requests or when a number of write requests in the write queue exceeds a specific number. Then, traditional write draining room has more room to be improved by exploiting row buffer locality of both write and read requests.

The write drain algorithm is important for the memory controller performance. Write drain operation without considering row buffer hit status of pending requests in the read queue and write queue can increase the access or queuing delay by interrupting read requests.

A most efficient conventional write drain scheme is the delayed write drain algorithm [31]. Delayed write scheme assumes that read request will arrive soon when the read queue is empty, so that write draining is delayed for a short time to wait for potential read requests. Write drain is performed if no read request arrives. When the number of pending write requests in the write queue is compulsorily drained until the number of pending write requests in the queues reaches a *low watermark*. However, in the previous write drain methods, *row buffer locality* is not considered, so that the locality of issued request sequence can be broken by write drain operation.

Figure 3.2. illustrates that the conventional write drain mechanism has possibility to degrade performance. Write drain is started as the number of write requests in the write queue reaches high watermark, and it is continued to be drained until the number of write requests in write queue reaches low watermark.

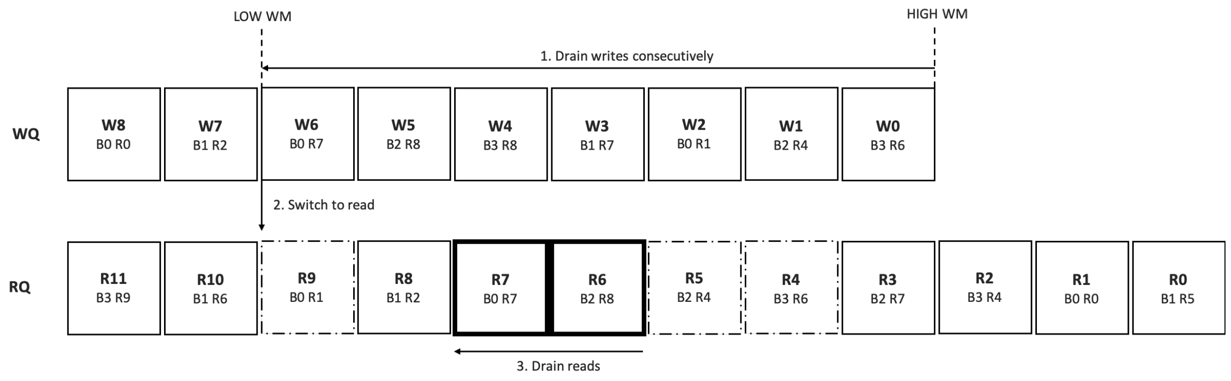


Figure 3.2: Conventional Write Drain Policy.

The Write requests are represented by Wx, and the Read requests are represented by Rx. While R4, R5 and R9 reference same pages with W0, W1 and W2, respectively, the pages are closed by W4, W5 and W6 with conventional write drain policy. This leads to unnecessary row activation when issuing R4, R5 and R9, thereby degrading performance significantly. In addition, row buffer locality is attacked when draining read requests.

In order to consider *row buffer locality* for write drain operation, we propose a *Write Drain Policy Exploiting Row Buffer Locality*. Figure 3.2 shows how the proposed algorithm utilizes the row buffer locality in write-to-read switching. While conventional write drain policy issues write requests until the number of the write requests in the write queue reaches low watermark, the proposed write drain policy issues write requests until the ”row hit” write requests in the write queue are completely consumed.

The flowchart of the algorithm is described on Figure 3.3. If the number of write requests in the write queue reaches high watermark, write drain is started. Once write drain is started, ”row hit” write requests are issued consecutively, even if there exists pending read requests in the read queue. Write to read switching is occured only when there is no ”row hit” write requests in the write queue with ”row hit” read request in the read queue. Same policy is applied for read to write switching so that row locality interference is to be reduced during request drain.

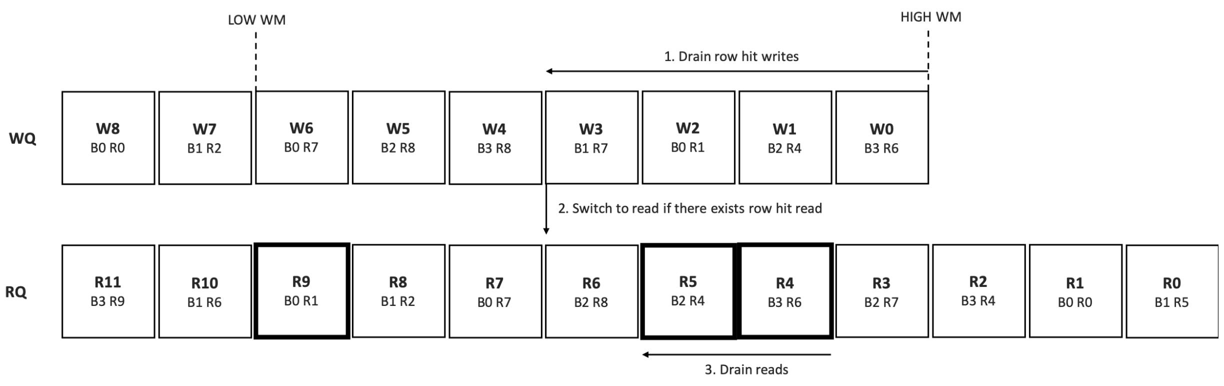


Figure 3.3: Proposed Write Drain Policy.

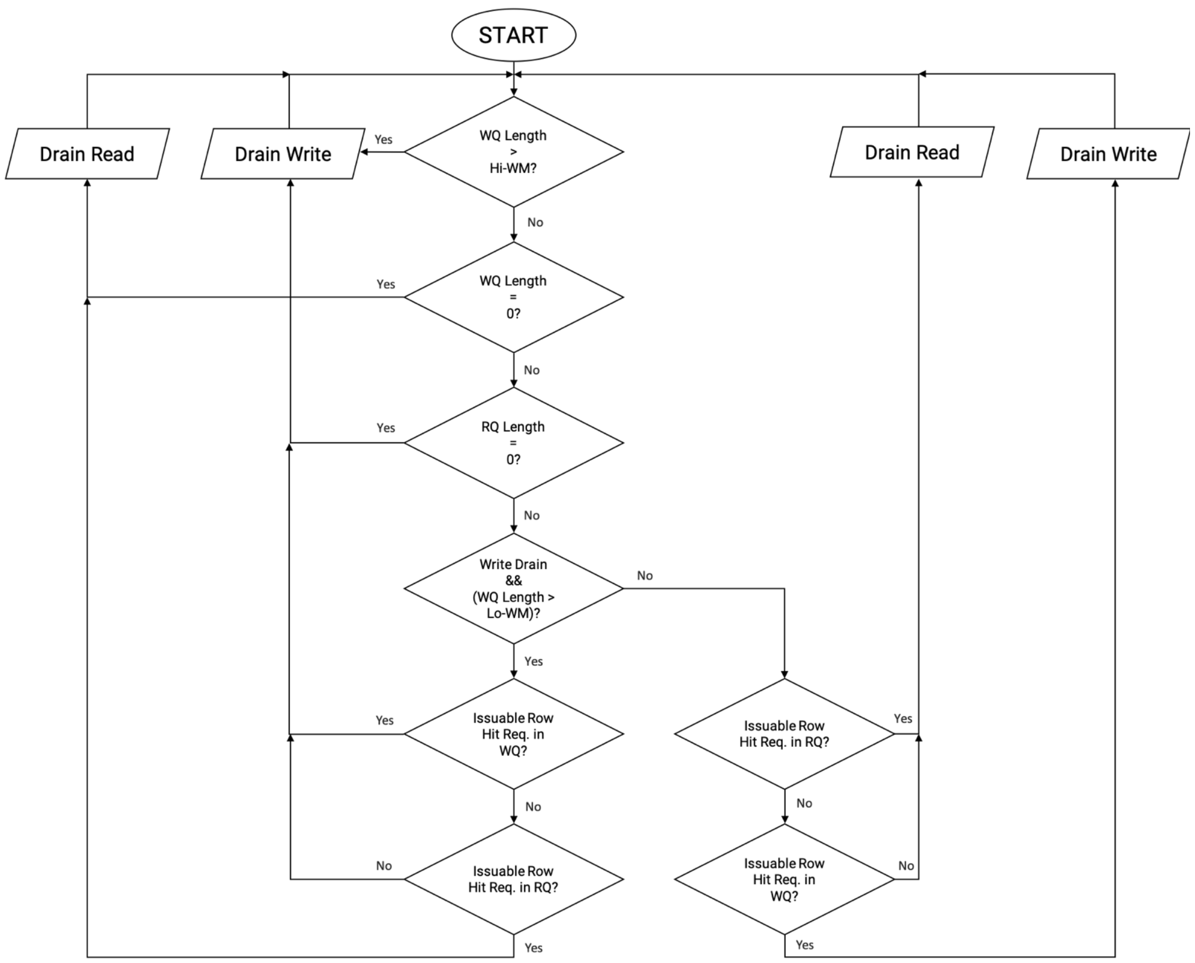

START
Drain Read
Drain Write
WQ Length > Hi-WM?
Yes
No
WQ Length = 0?
RQ Length = 0?
Write Drain && (WQ Length > Lo-WM)?
Issuable Row Hit Req. in WQ?
Issuable Row Hit Req. in RQ?
Drain Read
Drain Write
Issuable Row Hit Req. in RQ?
Issuable Row Hit Req. in WQ?

Figure 3.4: Flow Chart of the ”Write Drain Policy Exploiting Row Buffer Locality”

Chapter 4

Evaluation

In this chapter we analyze the memory controller proposal presented in the previous chapter. In order to evaluate the proposed DRAM controller, first, we show the behavior of the memory controller regarding the power-down modes employing a synthetic traffic generator. After that, we evaluate the behavior of the DRAM controller with some PARSEC benchmarks. All comparisons are made against two common memory scheduling algorithms, *FCFS* and *FR-FCFS*.

The power saving potential of the power-down modes depends directly on the times spent in these power-down modes. At the same time, power-down modes can reduce performance because of the latency of their exit-times. The experimental setup consists of a single traffic generator and a system bus that connects it to a DRAM controller, as show in Figure 4.1.

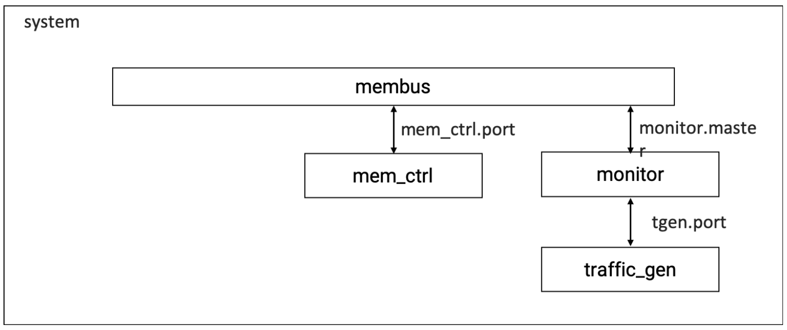


Figure 4.1: gem5 simple simulation setup.

In this work, we analyze a 1 GB DDR3 SDRAM memory system. This simulation allows us to analyze and demonstrate some concepts presented previously, and allows us to have a better idea of the actual implementation. The DRAM controller used is a single-ported DRAM controller model aiming to model the most important system-level performance effects of a DRAM without getting into much detail of the DRAM itself. The basic configuration of the controller corresponds to a burst for the specific DRAM configuration, e.g. x8 with burst length of 8 is 8 bytes. The configuration of the system is the following

- *Cache line size:* 64 bytes
- *Memory type:* DDR3-1600 8x8
- *Row Activation Windows:* 4

- *Banks:* 8
- *Ranks:* 1
- *Channels:* 1
- *Read percentage:* 80%
- *Burst length:* 8
- *Device bus width:* 8
- *Device row buffer size:* 1024
- *Device size:* 536870912 bytes
- *Devices per rank:* 8
- *Read buffer size:* 32
- *Write buffer size:* 64

The power and timing parameters based on the Micron DDR3-1600 1Gbit datasheet [27] are shown in Table 4.2.

Current	**Description**	**Value in mA**
I_{DD0}	Active precharge current	45
I_{DD2N}	Precharge standby current	23
I_{DD3N}	Active standby current	35
I_{DD4W}	WRITE current	103
I_{DD4R}	READ current	100
I_{DD5}	Refresh current	160
I_{DD3P1}	Active powerdown current	35
I_{DD2P1}	Precharge powerdown current	35
I_{DD6}	Self-refresh current	8

Table 4.1: gem5's DRAM model controller parameters

The *Inter-Transaction Time* or inter-request is proportional to the bus utilization, since as we increase the inter-transaction time, the bus utilization is reduced. One good opportunity to enter low power modes is after a refresh event, this is due there are no outstanding read or write requests in the controller's queue. To trigger power-down mode transitions, we have to manipulate the parameter *Inter-Transaction Time (ITT)*.

We can define ITT_{min} as the column to column delay t_{CCD} and the time to power-down-entry state as $t_{PDE} = t_{RAS} + t_{RP} + t_{CK}$. During a traffic generator phase, the generator selects a random value between ITT_{min} and a specified value of ITT_{max} which is calculated by multiplying t_{PDE} times a constant, these constants are 1 for very dense traffic and 50 for sparse traffic. For the target bank utilization, we use 1/8, 4/8 and 8/8, and finally, the number of bytes accessed sequentially ($N_{SeqBytes}$) by a traffic generator request are 64, 256 and 512 bytes.

Then, the traffic generation parameters are shown in table 4.2.

Description	**Value (unit)**
Request type	read (80 perc)
Request size	64B
Address range	0 to 1024 MB
ITT$_{min}$	t$_{CCD}$
ITT$_{max}$	t$_{PDE}$, 50xt$_{PDE}$, 100xt$_{PDE}$
N$_{SeqBytes}$	64, 256, 512
Bank utilization	1, 4, 8

Table 4.2: Traffic parameters in each memory configuration.

Table 4.3 shows the first memory controller configuration.

Memory Scheduler	FCFS
Page Policy	Close
Address Mapping	Ro:Co:Ra:Ba:Ch

Table 4.3: FCFS memory controller configuration.

Figure 4.2 shows the time spent in each power state for *ITT=1*, i.e. very dense traffic. Almost all the time is spent in the ACT state, this is due the close page policy, since for every access, we have to issue an ACT command. We have the same behavior in the different sequential accesses since we are not exploiting any row buffer locality. For bank utilization higher than one, in the case of 64 Bytes accesses, taking into account that the request size is 64B, every new request is to a random address, then it is possible to enter Precharge Power-Down (PRE$_P$DN) state since all banks are closed and precharged (because of the close page policy).

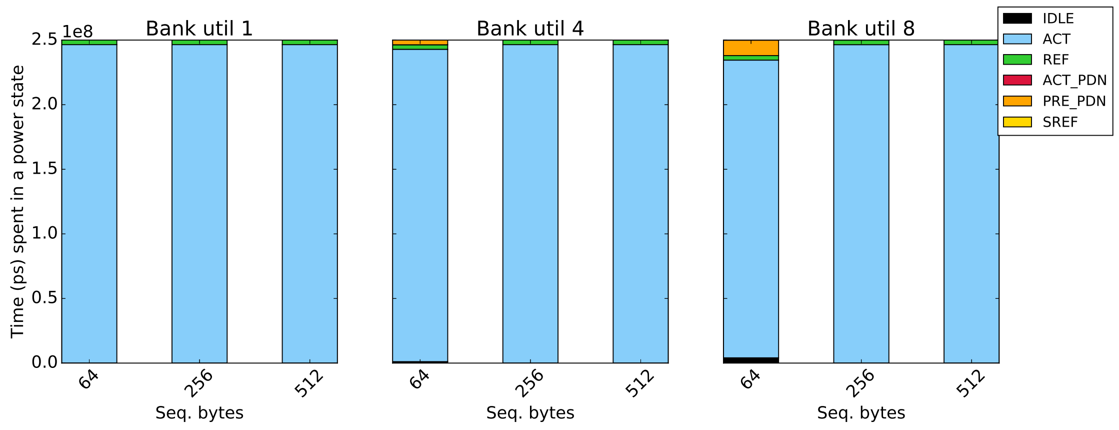


Figure 4.2: FCFS - Time spent in power states - very dense traffic.

Figure 4.3 shows the energy consumed by each power state. In bank utilization 1, the energy consumed is the same no matter the sequential accesses, as we could expect. On the other hand, when the bank utilization increases to 4 and 8, the Active Background (ACT_BACK) energy consumed is the same, but since we can activate more banks at the same time, the precharge and read energy consumed is more in the case of random accesses, it means 64 Bytes. This is an example of how in spite of entering a low power mode, the energy consumed is higher than the other cases when a low power mode is not entered.

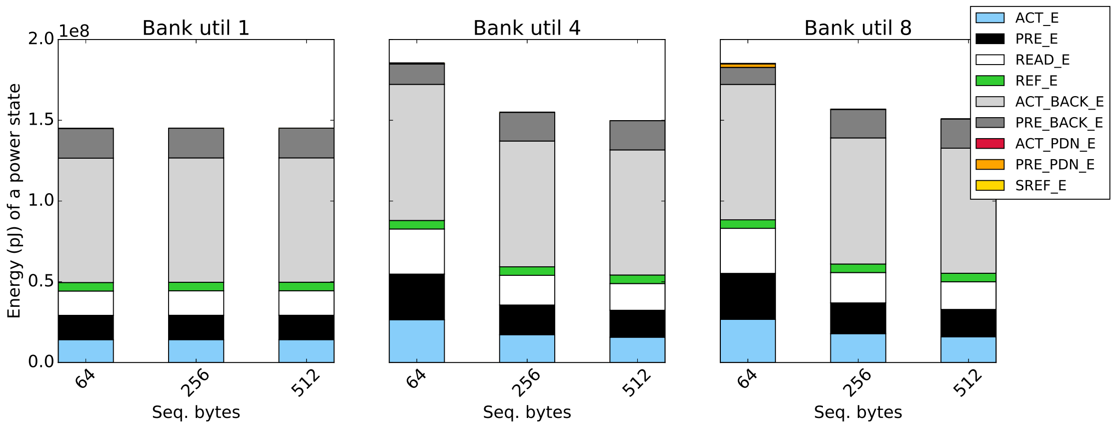


Figure 4.3: FCFS - Energy consumed by power states - very dense traffic.

Figure 4.4 shows the time spent for sparse traffic ($ITT = tCCD \times 50$). Since we met the conditions to enter two power modes, because the sparsity of the memory accesses, most of the time we are in state Precharge Power Down. As the sequential accesses increase, the time spent in Self Refresh mode increases since we can have all banks precharged and closed.

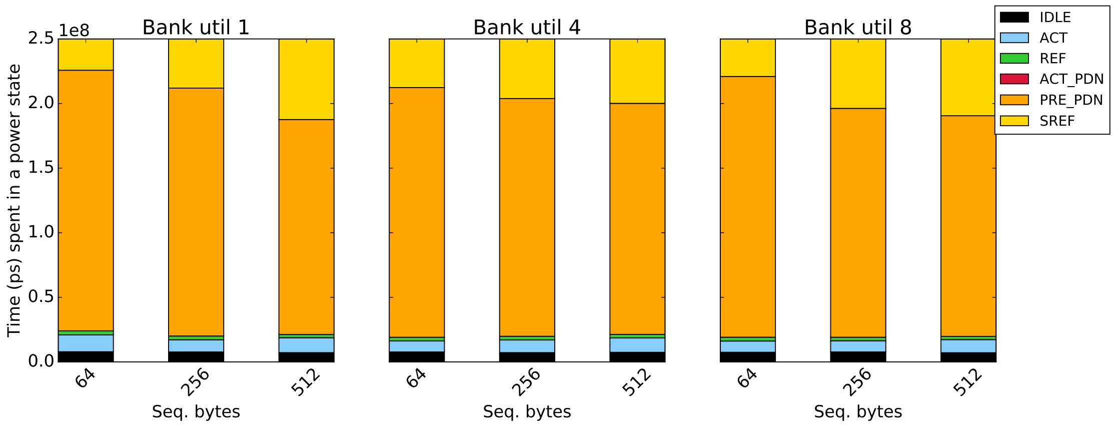


Figure 4.4: FCFS - Time spent in power states - sparse traffic.

As we could expect, the impact on the energy consumed is positive (see 1e7 mark). The energy un pJ consumed by each power state is proportional to the time spent in each state, pretty self-explanatory.

Table 4.4 shows the second memory controller configuration.

Figure 4.2 shows the time spent in power states for dense traffic. As the number of sequential bytes increases, the Active Power-Down mode can be accessed since row hits increase and other active banks (because of the open page policy) can enter to *PDNA* state.

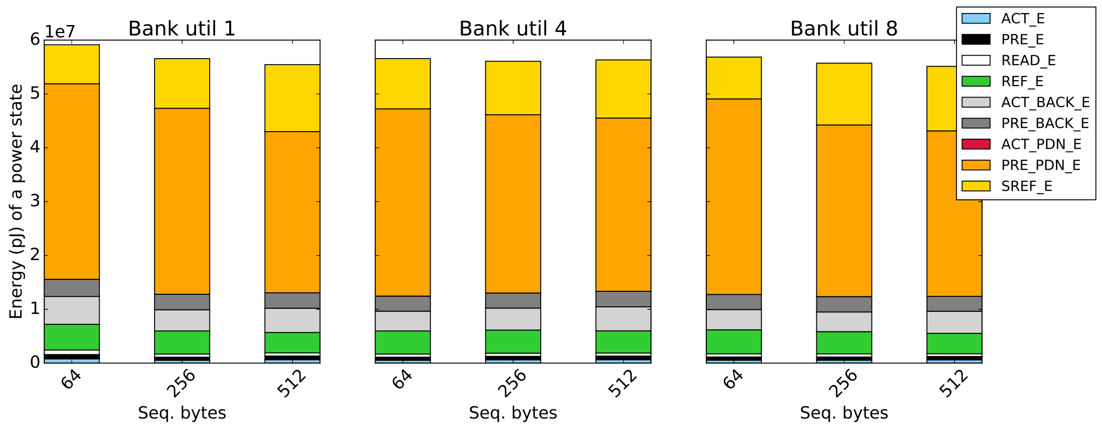


Figure 4.5: FCFS - Energy consumed by power states - sparse traffic.

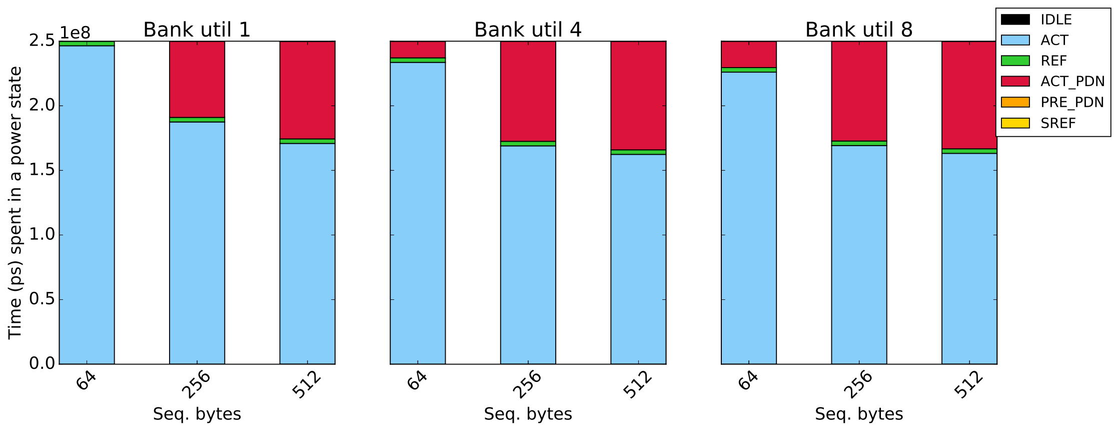


Figure 4.6: FR-FCFS - Time spent in power states - dense traffic.

Memory Scheduler	FR-FCFS
Page Policy	Open
Address Mapping	Ro:Ra:Ba:Co:Ch

Table 4.4: FR-FCFS memory controller configuration.

The energy consumed by power states for dense traffic is shown in Figure 4.7. One observations from this graphics is that the less row-hit, the more precharge energy consumed. This is reflected on every bank utilization configuration with 64 bytes accesses.

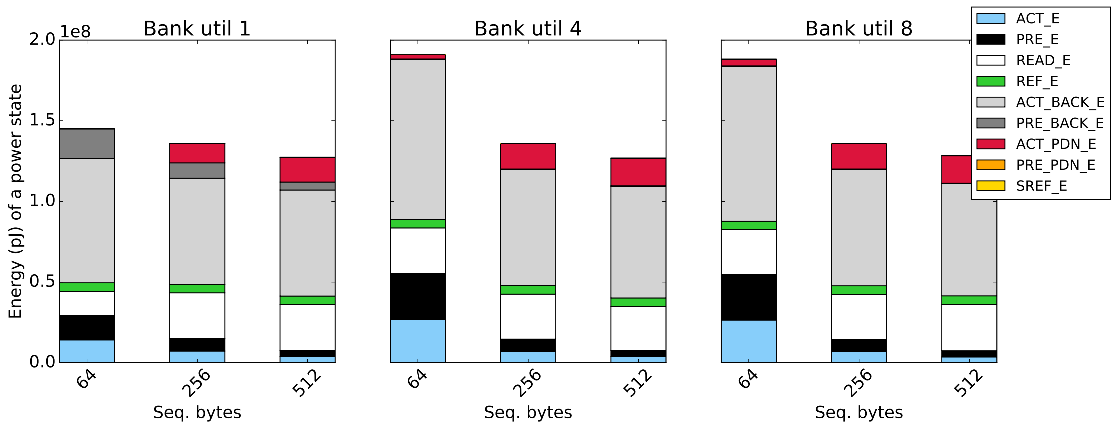


Figure 4.7: FR-FCFS - Energy consumed by power states - dense traffic.

For sparse traffic, Figure 4.8 show the time spent in power states. For 64 Bytes sequential accesses, the Active Power down state is higher because of the row misses and the open page policy of the controller.

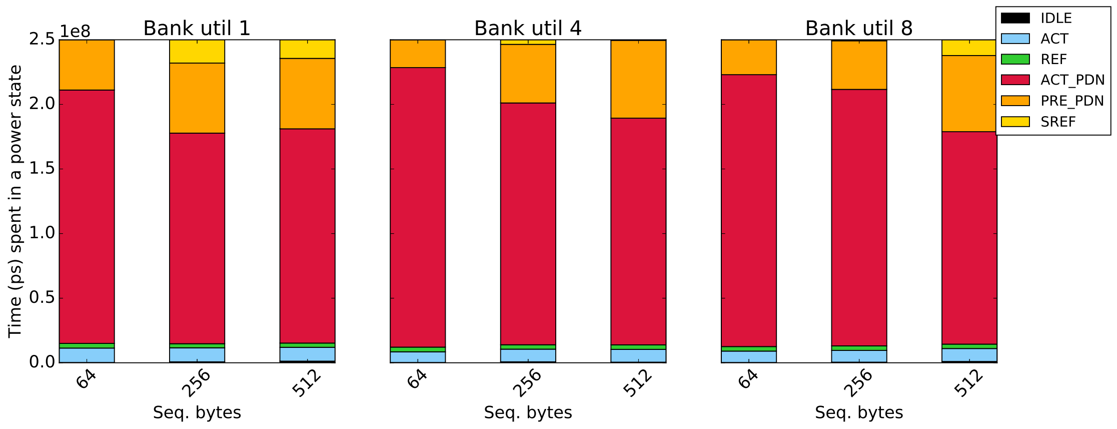


Figure 4.8: FR-FCFS - Time spent in power states - sparse traffic.

Figure 4.9 shows the energy consumed by power states for sparse traffic.

The Evaluation with PARSEC applications and kernels are shown in the following Figures. On the left Y-Axis, we have simulation time (execution time) in seconds and on the right Y-Axis we have the total energy consumed in pJ.

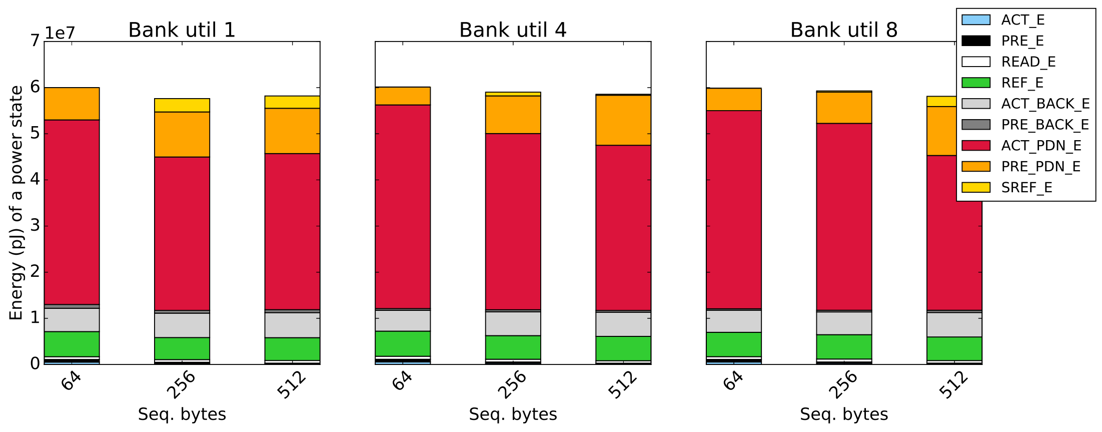


Figure 4.9: FR-FCFS - Energy consumed by power states - sparse traffic.

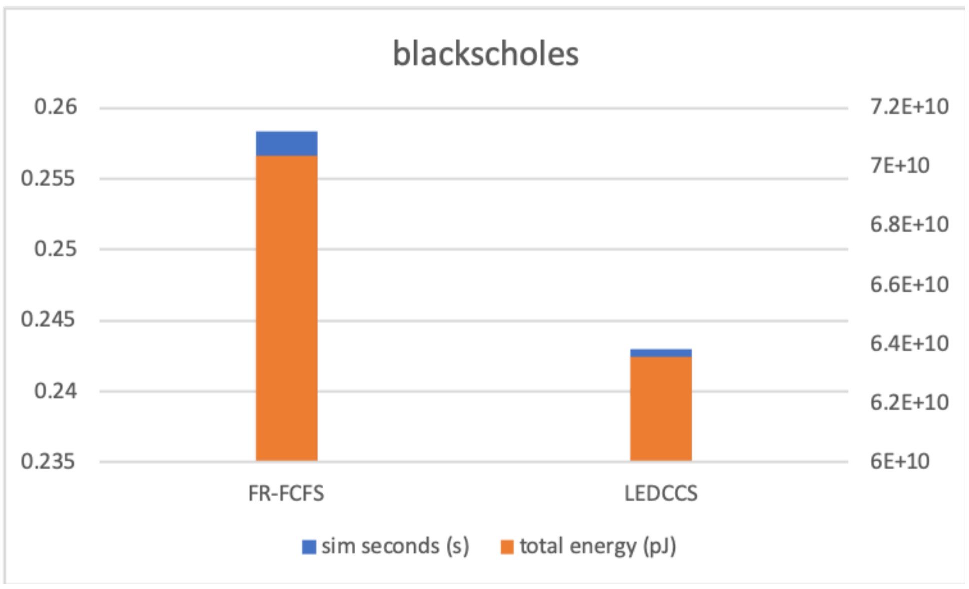


Figure 4.10: blackscholes

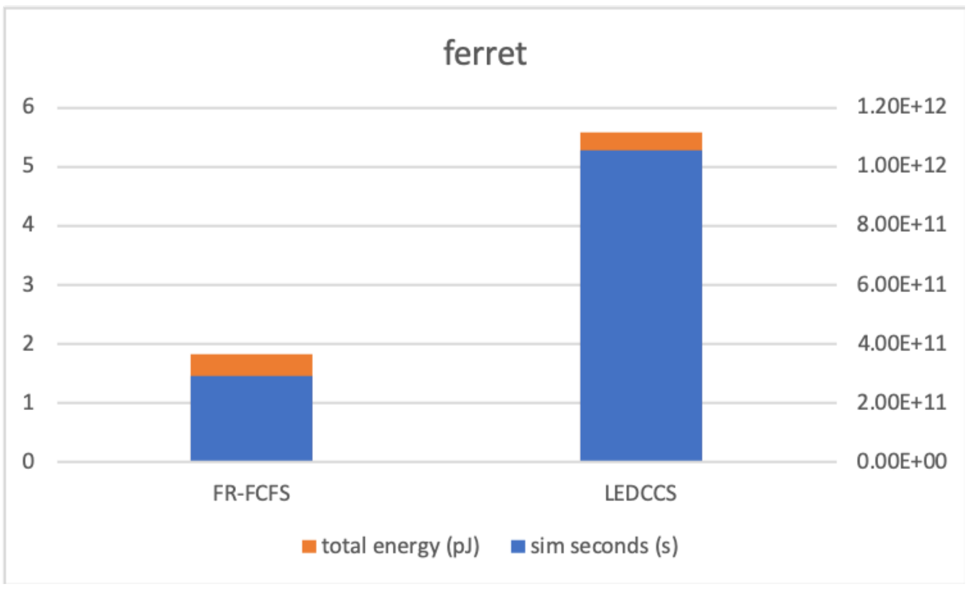


Figure 4.11: ferret

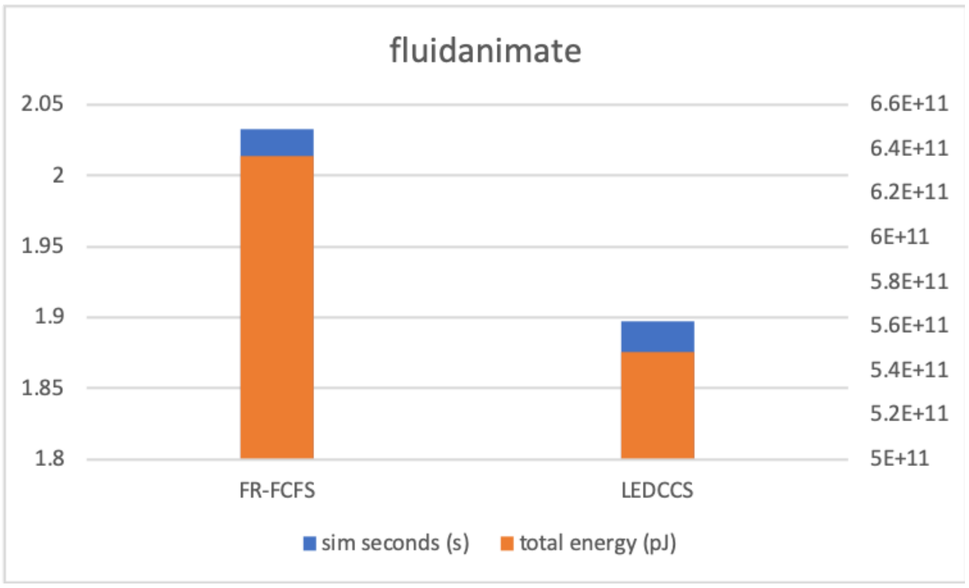


Figure 4.12: blackscholes

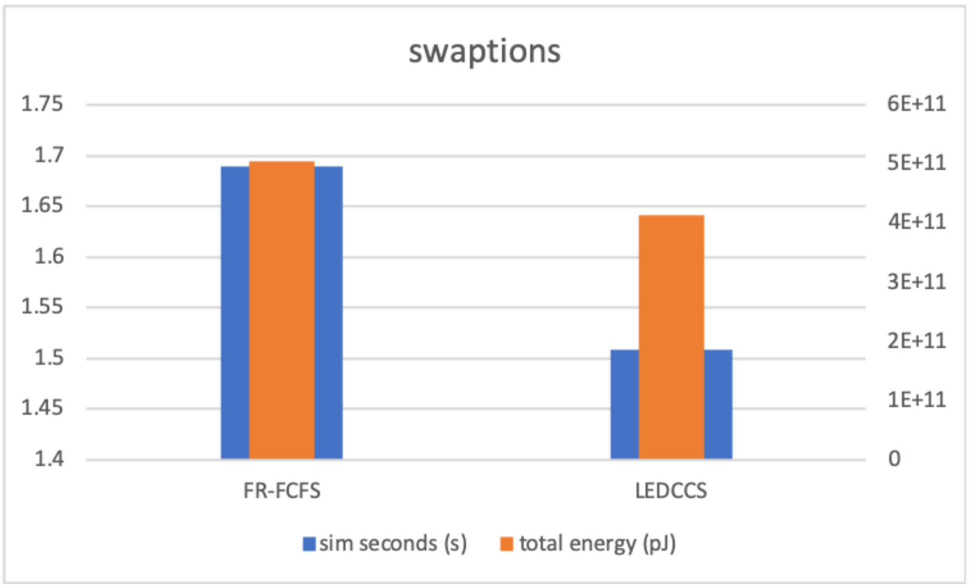


Figure 4.13: ferret

Chapter 5

Implementation

Figure 5.1. illustrates an abstract memory controller architecture. The *Interface Translation & Synchronization* represents the connection between the User Interface of the memory controller and the system bus. After that, the requests are put into a *Transaction Queue* which is basically a FIFO structure. After this, the process of *Address Mapping* is performed so the respective requests can be put into their respective *Command Queue*, which can execute command in order or out of order, depending on the memory scheduling algorithm. The block *Bank States* represents the part of the memory controller that keeps track of the states of the banks, this allows us to keep track of the rows that are active, so we can make scheduling decisions. The Block *Precharge/Refresh* is in charge of the Precharging/Refreshing process. After that, the commands are executed, this means that these commands are passed to the physical interface, which is in charge of performing all the timing signaling and the SSTL standard compliance.

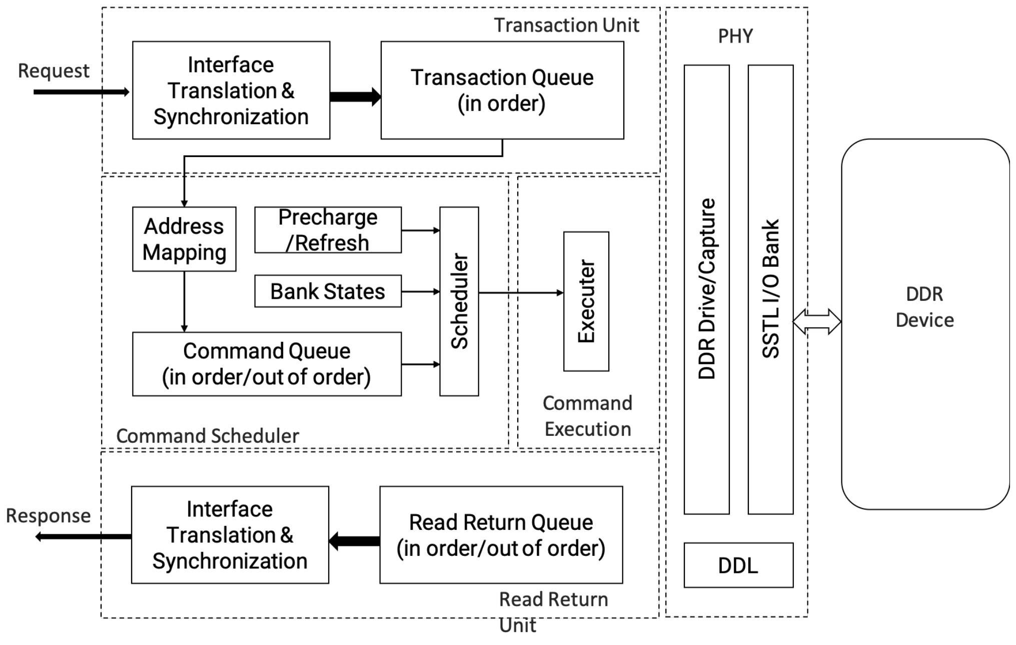


Figure 5.1: DRAM memory controller architecture.

5.1 Actual implementation

The actual implementation of the memory controller is showed in Figure 5.2. The following subsections will describe each of the blocks that comprise the memory controller implementation.

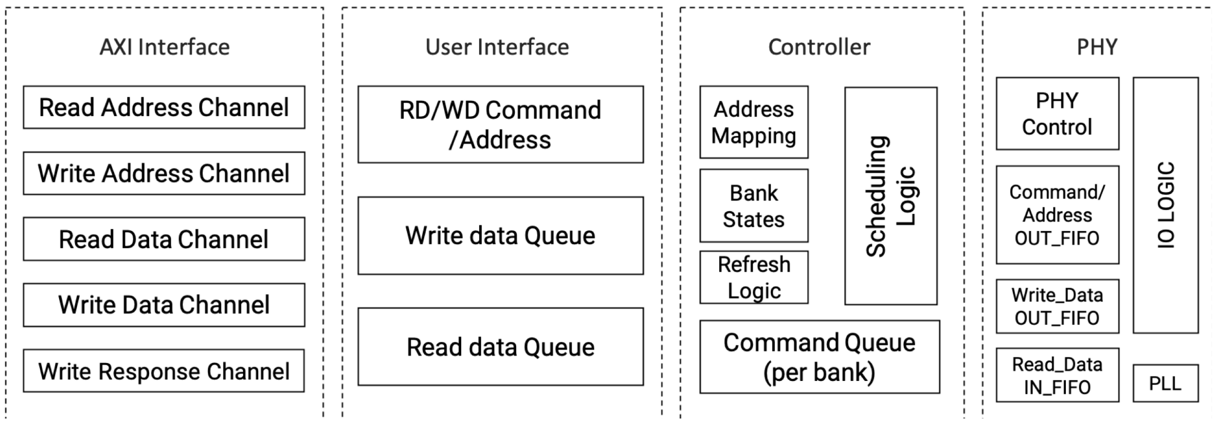


Figure 5.2: Actual implementation of DDR3 Memory Controller.

5.1.1 AXI Interface

AXI is part of the ARM AMBA, a family of micro-controller buses. There exist three types of AXI4 interfaces

- **AXI4**: it is intended for high-performance memory-mapped requirements.
- **AXI4-Lite**: it is intended for simple, low-throughput memory-mapped communication.
- **AXI4-Stream**: it is designed for high-speed streaming data.

The AXI specifications describe an interface between a single AXI master and AXI slave, representing hardware modules or IP cores that exchange information with each other. It is feasible to have multiple memory-mapped AXI masters and slaves connected together by using AXI infrastructure.

The AXI interconnect is architected using a traditional, monolithic crossbar approach. Both AXI4 and AXI4-Lite interfaces consist of five different channels: Read Address, Write Address, Read Data, Write Data and Write Response. Data can move in both directions between the master and slave silmutaneously, and data transfer sizes can vary.

The Interface implemented on this work is a Slave AXI4 interface. This choice was made based on the fact that the SoC Lagarto employs AXI as the Local Memory Bus, implementing AXI4 and AXI4-Lite. The AXI interface maps AXI4 transactions to the User Interface. The transaction-level arbitration provided by this interface is *Read Priority*, so Read and Write channels are served with equal priority, and the requests from the write address channel are processed when one of the following conditions is met

- There are no pending requests from the read address channel.
- Read wait limit is reached (16).

5.1.2 User Interface

The user interface block forms the bridge between the memory bus and the DDR3 memory controller and PHY layer logic. This UI comprises two FIFOs: the Address/Command FIFO and the Write Data FIFO. Commands (Read/Write) and target addresses to the memory controller are written to the Address/Command FIFO. If the issued command is a write, the corresponding output data must be written into the Write Data FIFO. The Address/Command FIFO is basically a synchronous FIFO, and the Write Data FIFO is configured as an asynchronous FIFO. During reads, the read data is presented at the output of the User Interface block on the read data bus and is qualified by a read valid signal.

5.1.3 Controller

This is the main module of the DD3 Memory Controller. This module processes the commands from the User Interface block and issues the required read, write, activate, precharge, and auto refresh commands to the DDR3 SDRAM. The controller machine is held in an idle state while the initialization state machine in the PHY layer initializes the DDR3 SDRAM device and issues read and writes for read data timing calibration. The initialization machine asserts the *phy_init_done* signal after completition of the initialization and calibration. The controller state machine remains in the IDLE state until assertion of *phy_init_done*.

The user of the memory controller issues read and write requests through the Use Interface Address/Command FIFO. This FIFO indicates with an *almost empty and empty* signals, the state of the pending requests. The controller state machine processes user command requests passed through this FIFO.

Memory Address Mapping

The user provides the target address to the user interface Address/Command FIFO, and this block is in charge of mapping the specific column, row and bank bits of the address provided by the user interface according to one the following memory address mapping schemes

- **Row/Bank/Column**: this scheme provides bank-level parallelism, by using the lower bits (the ones with more entropy) to map the banks. Then we can map a cache line in different banks and access them in a pipelined way, so increasing the throughput we get out of the DRAM memory system.
- **Bank/Row/Column**: this scheme provides us a way of row-level parallelism. By using this scheme, we are accessing the same bank, so if we employ a close page policy, we can leverage this scheme to map data to the same bank, while other banks might be in a low-power mode.

Bank Management

The controller is able to keep up to four bank/rows open at any given time. The banks are opened when commands are issued from the UI block. When a command is received, the controller makes a decision as to what it needs to do to support this access

- *Bank conflict*: the target bank address does not match any of the currenttly opened banks.
- *Bank hit*: the target bank address matches one of the four currently opened banks.
- *Row conflict*: the target bank address matches one of th currently opened banks, however, th target row address is different that the currently open row in that bank.

The controller indicates a conflict has occurred if there is a bank conflict or if there is a bank hit and a row conflict. For a bank conflict, if the controller has already opened four banks, it will close the least recently opened bank using the precharge command and open the new bank using an active command. If the controller has not already opened four banks, then the controller will issue and activate command to open the new bank. In the case of a bank hit and row conflict, the controller will close the bank that had the bank hit using a precharge command and will open the new row in the bank using the activate command. The controller closes all opened banks before issuing an auto refresh command. After the completion of the auto refresh command, accesses from the User Interface block again determine which banks are opened. Figure 5.3 shows the bank management logic.

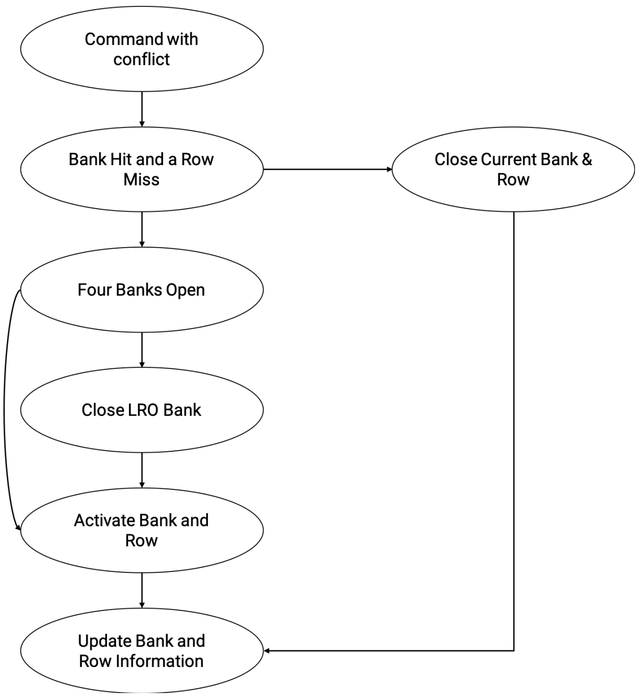


Figure 5.3: Bank Management Logic.

Commands

Table 5.1 shows the commands that the controller can issue to the DDR3 SDRMA. The user through the User Interface can issue only read and write commands. The controller issues the required SDRAM commands listed below to perform the read and write transaction.

Command	Description
Activate	An activate command to an unopened bank/row is issued for write and read accesses from the User Interface block. The controller also issues an activate command for the last accessed bank/row following an auto-refresh.
Read	A read command is issued to an open bank/row for read requests from the User Interface block.
Write	A write command is issued to an open bank/row for write requests from the User Interface block.
Precharge	A precharge command is issued under these conditions: A PRECHARGE_ALL command (to close all the banks) is issued before an auto refresh command; and a precharge of a particular bank is issued when a bank/row conflict is detected.
Auto Refresh	An auto refresh command is issued periodically based on the memory auto-refresh timing requirements.

Table 5.1: gem5's DRAM model controller parameters

The controller state machine logic is shown in Figure 5.4.

5.1.4 PHY

For the PHY, since we are targetting a Xilinx based FPGA, we used the PHY provided by Xilinx 7 Series, which allows us to leverage all the hard blocks of the FPGA. Xilinx supports using the PHY only portion of the MIG 7 Series IP [32] to interface to the custom controller.

The PHY provides a physical interface to an external DDR2 or DDR3 SDRAM. The PHY generates the signal timing and sequencing required to interface to the memory device. [3] It contains the clock, address, and control generation logic, write and read datapaths, and state logic for initializing the SDRAM after power-up. In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic analysis. Figure 5.5 shows a single bank DDR2/DDR3 PHY block diagram.

5.1.5 Simulation results

In this subsection we show the timing diagrams obtained from Vivado Simulator.

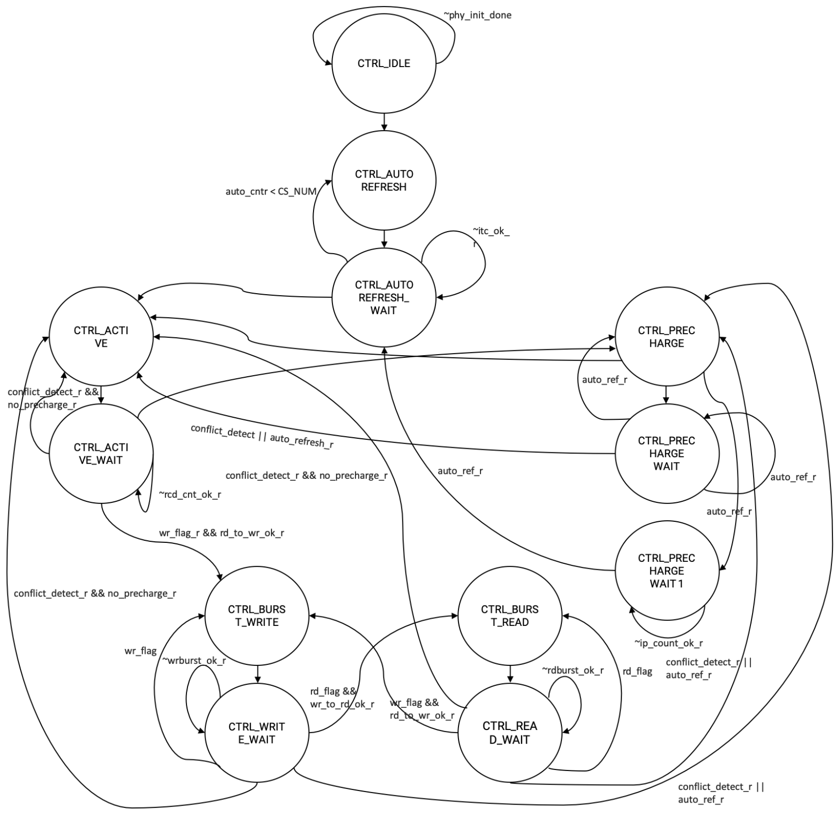


Figure 5.4: Controller State Machine Logic.

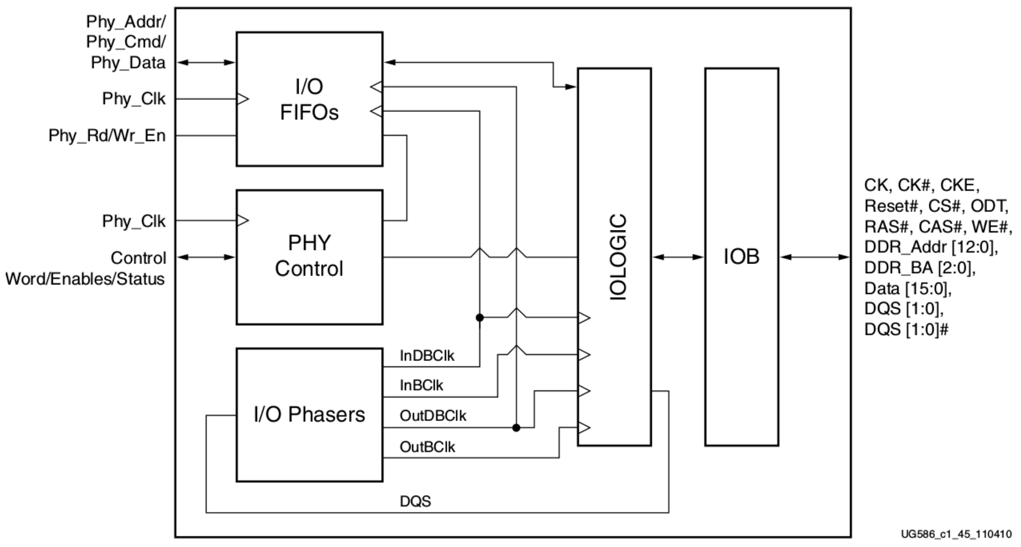
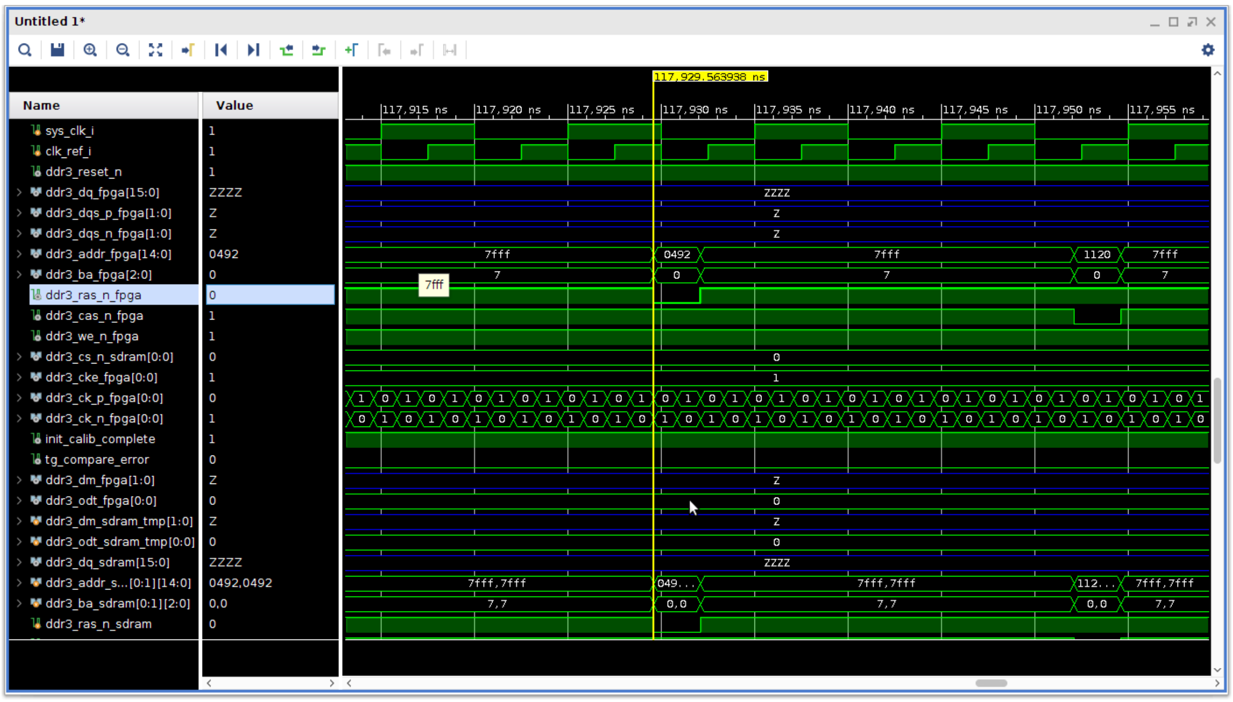
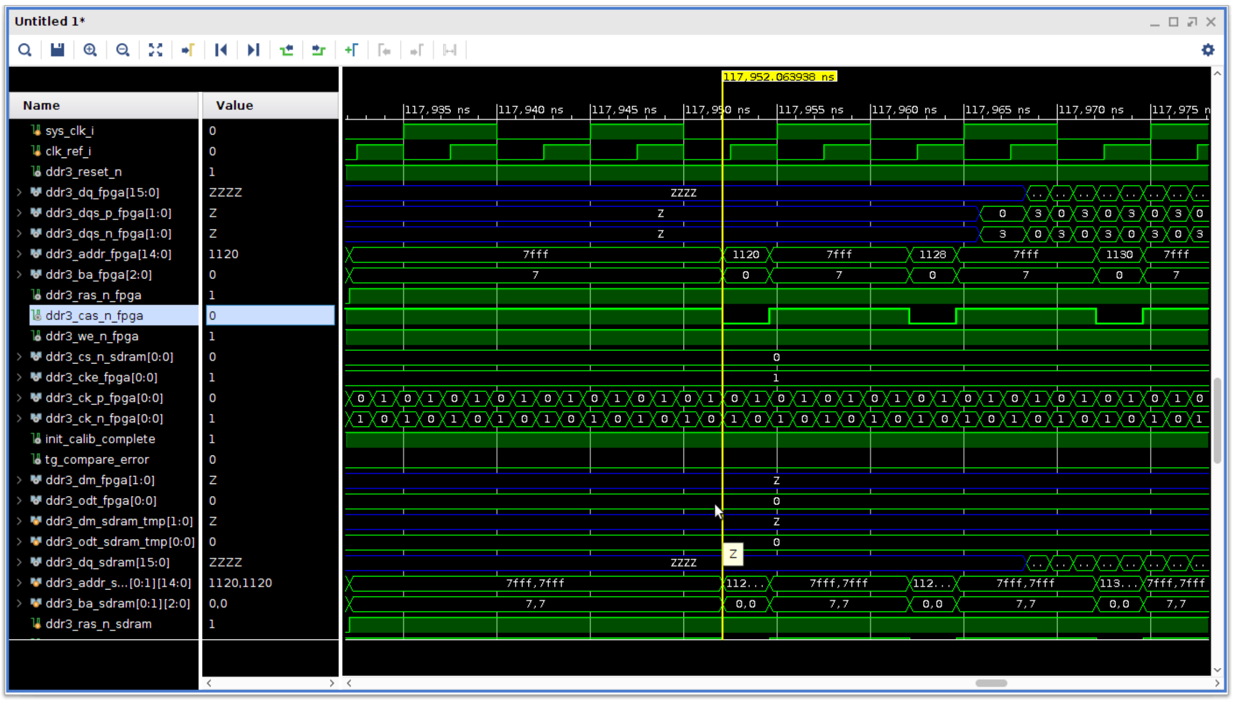

Phy_Addr/ Phy_Cmd/ Phy_Data
Phy_Clk
Phy_Rd/Wr_En
Phy_Clk
Control Word/Enables/Status
I/O FIFOs
PHY Control
I/O Phasers
InDBClk
InBClk
OutDBClk
OutBClk
DQS
IOLOGIC
IOB
CK, CK#, CKE, Reset#, CS#, ODT, RAS#, CAS#, WE#, DDR_Addr [12:0], DDR_BA [2:0], Data [15:0], DQS [1:0], DQS [1:0]#
UG586_c1_45_110410

Figure 5.5: Single Bank DDR2/DDR3 PHY Block Diagram.



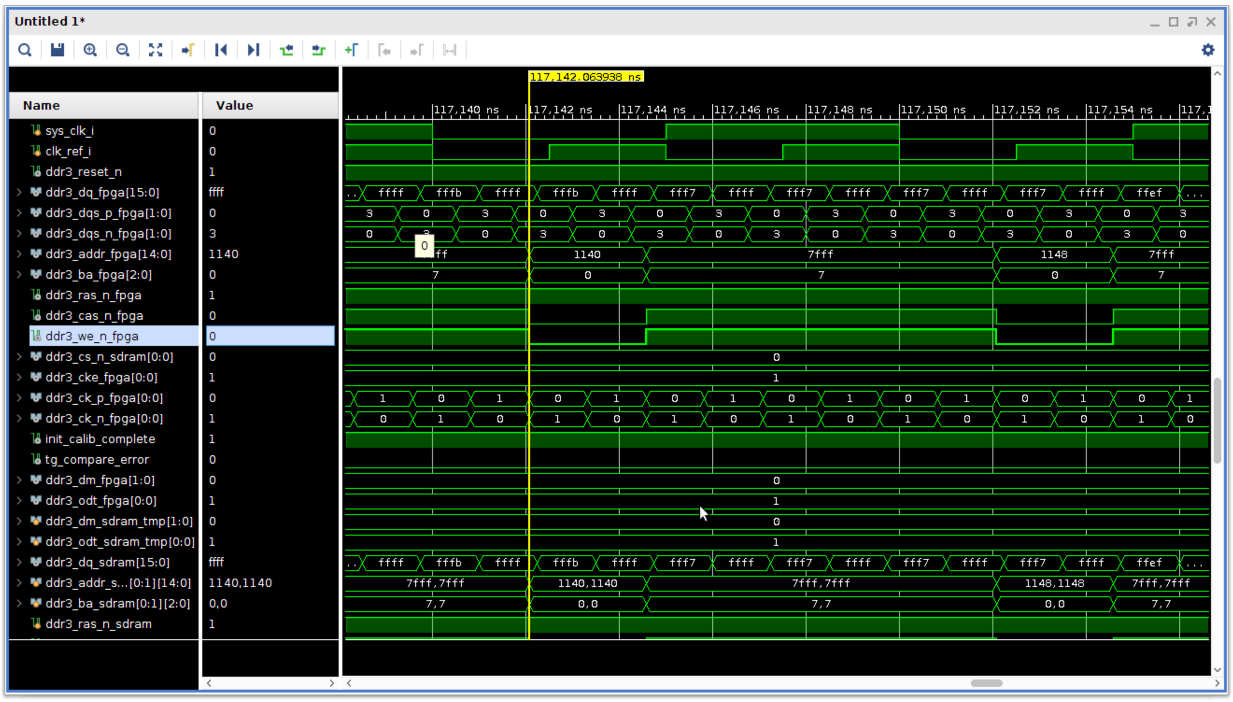
Name	Value
sys_clk_i	1
clk_ref_i	1
ddr3_reset_n	1
ddr3_dq_fpga[15:0]	ZZZZ
ddr3_dqs_p_fpga[1:0]	Z
ddr3_dqs_n_fpga[1:0]	Z
ddr3_addr_fpga[14:0]	0492
ddr3_ba_fpga[2:0]	0
ddr3_ras_n_fpga	0
ddr3_cas_n_fpga	1
ddr3_we_n_fpga	1
ddr3_cs_n_sdram[0:0]	0
ddr3_cke_fpga[0:0]	1
ddr3_ck_p_fpga[0:0]	0
ddr3_ck_n_fpga[0:0]	1
init_calib_complete	1
tg_compare_error	0
ddr3_dm_fpga[1:0]	Z
ddr3_odt_fpga[0:0]	0
ddr3_dm_sdram_tmp[1:0]	Z
ddr3_odt_sdram_tmp[0:0]	0
ddr3_dq_sdram[15:0]	ZZZZ
ddr3_addr_s...[0:1][14:0]	0492,0492
ddr3_ba_sdram[0:1][2:0]	0,0
ddr3_ras_n_sdram	0

Figure 5.6: Activate command.



Name	Value
sys_clk_i	0
clk_ref_i	0
ddr3_reset_n	1
ddr3_dq_fpga[15:0]	ZZZZ
ddr3_dqs_p_fpga[1:0]	Z
ddr3_dqs_n_fpga[1:0]	Z
ddr3_addr_fpga[14:0]	1120
ddr3_ba_fpga[2:0]	0
ddr3_ras_n_fpga	1
ddr3_cas_n_fpga	0
ddr3_we_n_fpga	1
ddr3_cs_n_sdram[0:0]	0
ddr3_cke_fpga[0:0]	1
ddr3_ck_p_fpga[0:0]	0
ddr3_ck_n_fpga[0:0]	1
init_calib_complete	1
tg_compare_error	0
ddr3_dm_fpga[1:0]	Z
ddr3_odt_fpga[0:0]	0
ddr3_dm_sdram_tmp[1:0]	Z
ddr3_odt_sdram_tmp[0:0]	0
ddr3_dq_sdram[15:0]	ZZZZ
ddr3_addr_s...[0:1][14:0]	1120,1120
ddr3_ba_sdram[0:1][2:0]	0,0
ddr3_ras_n_sdram	1

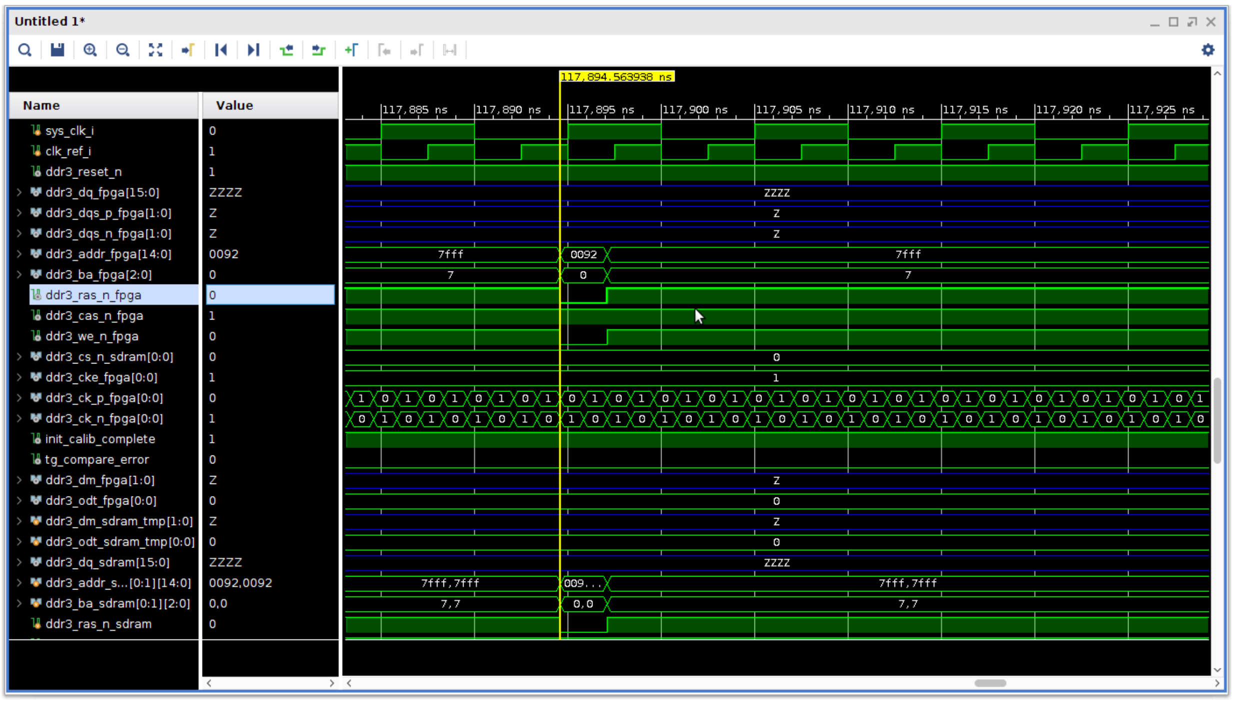
Figure 5.7: Read command.



Untitled 1*

Name	Value
sys_clk_i	0
clk_ref_i	0
ddr3_reset_n	1
ddr3_dq_fpga[15:0]	ffff
ddr3_dqs_p_fpga[1:0]	0
ddr3_dqs_n_fpga[1:0]	3
ddr3_addr_fpga[14:0]	1140
ddr3_ba_fpga[2:0]	0
ddr3_ras_n_fpga	1
ddr3_cas_n_fpga	0
ddr3_we_n_fpga	0
ddr3_cs_n_sdram[0:0]	0
ddr3_cke_fpga[0:0]	1
ddr3_ck_p_fpga[0:0]	0
ddr3_ck_n_fpga[0:0]	1
init_calib_complete	1
tg_compare_error	0
ddr3_dm_fpga[1:0]	0
ddr3_odt_fpga[0:0]	1
ddr3_dm_sdram_tmp[1:0]	0
ddr3_odt_sdram_tmp[0:0]	1
ddr3_dq_sdram[15:0]	ffff
ddr3_addr_s...[0:1][14:0]	1140,1140
ddr3_ba_sdram[0:1][2:0]	0,0
ddr3_ras_n_sdram	1

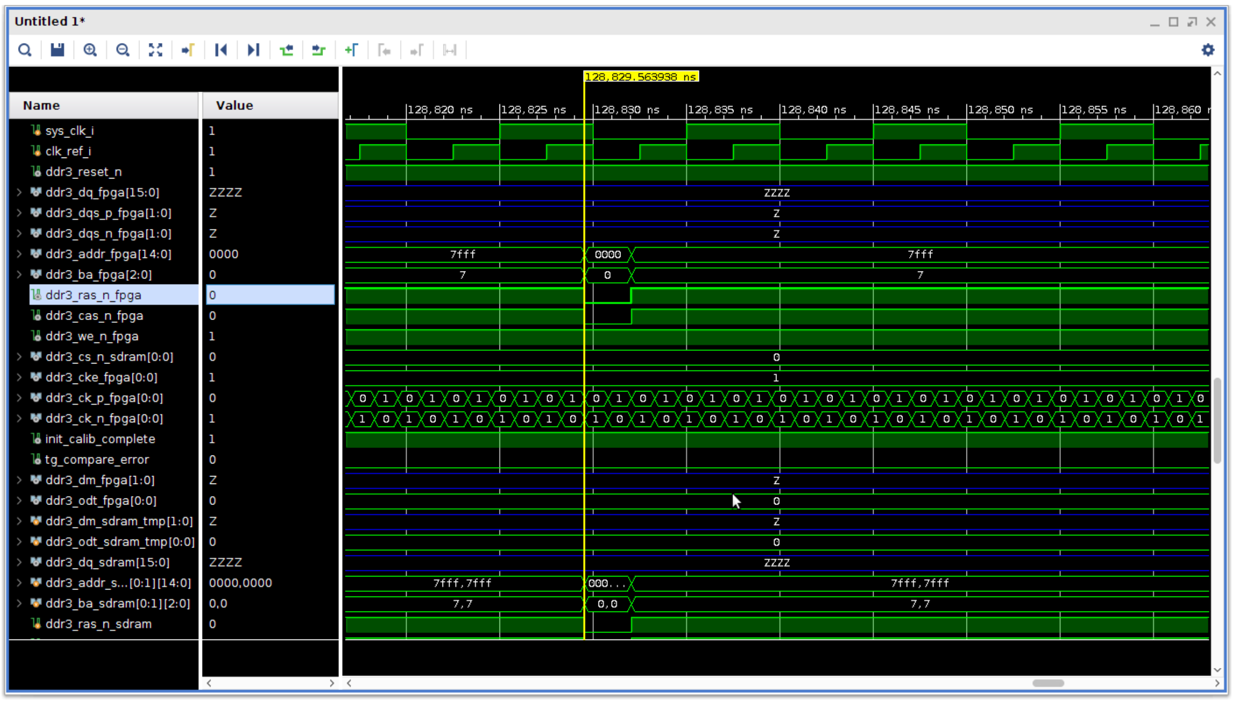
Figure 5.8: Write command.



Untitled 1*

Name	Value
sys_clk_i	0
clk_ref_i	1
ddr3_reset_n	1
ddr3_dq_fpga[15:0]	ZZZZ
ddr3_dqs_p_fpga[1:0]	Z
ddr3_dqs_n_fpga[1:0]	Z
ddr3_addr_fpga[14:0]	0092
ddr3_ba_fpga[2:0]	0
ddr3_ras_n_fpga	0
ddr3_cas_n_fpga	1
ddr3_we_n_fpga	0
ddr3_cs_n_sdram[0:0]	0
ddr3_cke_fpga[0:0]	1
ddr3_ck_p_fpga[0:0]	0
ddr3_ck_n_fpga[0:0]	1
init_calib_complete	1
tg_compare_error	0
ddr3_dm_fpga[1:0]	Z
ddr3_odt_fpga[0:0]	0
ddr3_dm_sdram_tmp[1:0]	Z
ddr3_odt_sdram_tmp[0:0]	0
ddr3_dq_sdram[15:0]	ZZZZ
ddr3_addr_s...[0:1][14:0]	0092,0092
ddr3_ba_sdram[0:1][2:0]	0,0
ddr3_ras_n_sdram	0

Figure 5.9: Precharge command.



Name	Value
sys_clk_i	1
clk_ref_i	1
ddr3_reset_n	1
ddr3_dq_fpga[15:0]	ZZZZ
ddr3_dqs_p_fpga[1:0]	Z
ddr3_dqs_n_fpga[1:0]	Z
ddr3_addr_fpga[14:0]	0000
ddr3_ba_fpga[2:0]	0
ddr3_ras_n_fpga	0
ddr3_cas_n_fpga	0
ddr3_we_n_fpga	1
ddr3_cs_n_sdram[0:0]	0
ddr3_cke_fpga[0:0]	1
ddr3_ck_p_fpga[0:0]	0
ddr3_ck_n_fpga[0:0]	1
init_calib_complete	1
tg_compare_error	0
ddr3_dm_fpga[1:0]	Z
ddr3_odt_fpga[0:0]	0
ddr3_dm_sdram_tmp[1:0]	Z
ddr3_odt_sdram_tmp[0:0]	0
ddr3_dq_sdram[15:0]	ZZZZ
ddr3_addr_s...[0:1][14:0]	0000,0000
ddr3_ba_sdram[0:1][2:0]	0,0
ddr3_ras_n_sdram	0

Figure 5.10: Refresh command.

Chapter 6

Conclusion

In this work it was intended to provide a thoroughly journey in the world of DRAM memory systems, specifically focused on DDR devices. One of the goals of this work is to bring up the difference between a simulation framework and actual implementation in hardware.

I strongly believe that both approaches are important as a computer architect and the following are some remarks about the work done through this thesis.

- The more sophisticated the memory controller is, the more hardware has to be employed when implementing, and this leads to more power consumption.
- Adaptive-policies and algorithms deliver small improvements over fixed-traditional policies.
- Sometimes, simple changes give us significant improvements, like in the case of the Write-Drain and adaptive page policy, part of this work.

In computer architecture there is no right answer for everything and most of the work in this area has to be with insight, that's one of the beauties of this job. Memory controllers form part of the options to improve memory systems performance, and as technology advances and the need of more powerful systems keeps growing, there will be always a chance to make some kind of improvements.

Bibliography

[1] S. A. McKee, “Reflections on the memory wall,” in *Proceedings of the 1st Conference on Computing Frontiers*, CF ’04, (New York, NY, USA), pp. 162–, ACM, 2004. 1

[2] J. S. S. T. Association, “Jedec standard: Ddr3 sdram specification,” 2008. Last accessed 16 September 2017. 1, 5, 38, 40

[3] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. 4, 5, 38, 67

[4] Z. Zhang, Z. Zhu, and X. Zhang, “Breaking address mapping symmetry at multi-levels of memory hierarchy to reduce dram row-buffer conflicts.” 39

[5] and and, “A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality,” in *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pp. 32–41, Dec 2000. 39

[6] and S. K. Reinhardt and D. Burger, “Reducing dram latencies with an integrated memory hierarchy design,” in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 301–312, Jan 2001. 39

[7] V. Cuppu and B. Jacob, “Organizational design trade-offs at the dram, memory bus, and memory controller level: Initial results,” November 1999. 39

[8] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” *SIGARCH Comput. Archit. News*, vol. 28, pp. 128–138, May 2000. 39, 44

[9] V. Cuppu and B. Jacob, “Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor dram-system performance?,” in *Proceedings 28th Annual International Symposium on Computer Architecture*, pp. 62–71, June 2001. 39

[10] S. A. McKee, W. A. Wulf, J. H. Aylor, R. H. Klenke, M. H. Salinas, S. I. Hong, and D. A. B. Weikle, “Dynamic access ordering for streamed computations,” *IEEE Transactions on Computers*, vol. 49, pp. 1255–1271, Nov 2000. 39

[11] and C. Lin, “Adaptive history-based memory schedulers,” in *37th International Symposium on Microarchitecture (MICRO-37’04)*, pp. 343–354, Dec 2004. 39, 46

[12] G. Narancic, M. Papadopoulou, and J. Zebchuk, “A preliminary exploration of memory controller policies on smartphone workloads,” 2012. 41, 52

[13] G. L Yuan and T. Aamodt, “A hybrid analytical dram performance model,” 01 2009. 42, 43

[14] D. Kaseridis, J. Stuecheli, and L. K. John, “Minimalist open-page: A dram page-mode scheduling policy for the many-core era,” in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, (New York, NY, USA), pp. 24–35, ACM, 2011. 43

[15] N. E. Z. S. Z. S. Fang K., Iliev N., “Thread-fair memory request ordering,” July 2012. 44

[16] M. N. Bojnordi and E. Ipek, “Pardis: A programmable memory controller for the ddrx interfacing standards,” in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 13–24, June 2012. 44

[17] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, “Mise: Providing performance predictability and improving fairness in shared main memory systems,” in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 639–650, Feb 2013. 44

[18] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, “Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers,” in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 1–12, Jan 2010. 44

[19] H. Hanson and K. Rajamani, “What computer architects need to know about memory throttling,” in *Proceedings of the 2010 International Conference on Computer Architecture*, ISCA’10, (Berlin, Heidelberg), pp. 233–242, Springer-Verlag, 2012. 46

[20] I. Hur and C. Lin, “A comprehensive approach to dram power management,” in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pp. 305–316, Feb 2008. 46

[21] and K. G. Shin, C. Lefurgy, and T. Keller, “Improving energy efficiency by making dram less randomly accessed,” in *ISLPED ’05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, pp. 393–398, Aug 2005. 46

[22] M. Ghosh and H. S. Lee, “Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams,” in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 134–145, Dec 2007. 46

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The parsec benchmark suite: Characterization and architectural implications,” in *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 72–81, Oct 2008. 47, 51

[24] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The gem5 simulator,” *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011. 47

[25] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “Dramsim2: A cycle accurate memory system simulator,” *IEEE Computer Architecture Letters*, vol. 10, pp. 16–19, Jan 2011. 48

[26] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. N. Udipi, “Simulating dram controllers for future system architecture exploration,” in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 201–210, March 2014. 48, 52

[27] Micron, “Micron ddr3 sdram part mt8jtf12864hz,” 2006. Last accessed January 18 2017. 49, 56

[28] e. K. Chandrasekar, C. Weis, “Drampower: Open-source dram power & energy estimation tool.” Last accessed June 24 2019. 49

[29] K. Chandrasekar, B. Akesson, and K. Goossens, “Improved power modeling of ddr sdrams,” in *2011 14th Euromicro Conference on Digital System Design*, pp. 99–108, Aug 2011. 49

[30] M. Jung, C. Weis, N. Wehn, M. Sadri, and L. Benini, “Optimized active and power-down mode refresh control in 3d-drams,” in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, Oct 2014. 52

[31] C. Natarajan, B. Christenson, and F. Briggs, “A study of performance impact of memory controller features in multi-processor server environment,” in *Proceedings of the 3rd Workshop on Memory Performance Issues: In Conjunction with the 31st International Symposium on Computer Architecture*, WMPI ’04, (New York, NY, USA), pp. 80–87, ACM, 2004. 53

[32] Xilinx, “Mig 7 series ddr3 - phy only design,” 2014. Last accessed December 10 2018. 67