ABSTRACT

Title of Thesis: FBSIM AND THE FULLY BUFFERED DIMM MEMORY SYSTEM ARCHITECTURE

Rami Marwan Nasr, Master of Science, 2005

Thesis directed by: Professor Bruce Jacob
Department of Electrical and Computer Engineering

As DRAM device data rates increase in chase of ever increasing memory request rates, parallel bus limitations and cost constraints require a sharp decrease in load on the multi-drop buses between the devices and the memory controller, thus limiting the memory system's scalability and failing to meet the capacity requirements of modern server and workstation applications.

A new technology, the Fully Buffered DIMM architecture is currently being introduced to address these challenges. FB-DIMM uses narrower, faster, buffered point to point channels to meet memory capacity and throughput requirements at the price of latency.

This study provides a detailed look at the proposed architecture and its adoption, introduces an FB-DIMM simulation model - the FBSim simulator - and uses it to explore the design space of this new technology - identifying and experimentally proving some of its strengths, weaknesses and limitations, and uncovering future paths of academic research into the field.

FBSIM AND THE FULLY BUFFERED DIMM MEMORY SYSTEM ARCHITECTURE

by

Rami Marwan Nasr

Thesis submitted to the Faculty of the Graduate School of the
Univeristy of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2005

Advisory Committee:

Associate Professor Bruce Jacob, Chair / advisor
Associate Professor Manoj Franklin
Associate Professor Shuvra Bhattacharyya

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr Bruce Jacob for introducing me to the interesting and important field of DRAM research and the brand new Fully Buffered DIMM technology, for his shared knowledge, expert guidance, and for helping me see the big picture when all I saw were details.

I would like to thank David Wang of Dr Jacob’s research group for his lectures - without which all of this would have taken an order of magnitude longer for me to understand and put into perspective. David is one of the most knowledgeable people out there on anything with ‘RAM’ in it. Also thanks for the slides and class notes – a few pieces of which are used in this paper, and especially for the table on page 69 (Chapter 3 of course)!

I would like to thank all the members of my thesis committee for their time and help.

I would like to thank Dr Donald Yeung for encouraging me to get on the research bandwagon with a Master’s Thesis, and for helping me get started.

I would like to thank Dr Hassan Diab for first sparking my interest in the great field of computer architecture.

Huge thanks go out to my dear parents Amale and Marwan Nasr, wonderful sister Carine, my friends, and my fabulous girlfriend for their limitless support, encouragement, and patience.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION AND MOTIVATION 1

1.1 Memory Bottlenecks 1

1.2 Current State of FB-DIMM Implementation and Literature 5

1.3 Overview of Paper 6

CHAPTER 2: EVOLUTION OF HIGH-SPEED BUS TECHNOLOGIES 9

2.1 Futurebus+ and Bus Limitations 9

2.2 Scalable Coherent Interface (SCI) 11

2.3 Adoption of High Speed Serial Technologies 13

CHAPTER 3: MODERN MEMORY TECHNOLOGY 14

3.1 Introduction to DRAM Devices and Organizations 14

3.2 Ranks, Memory Modules and Bussing 18

3.3 Row Buffer Management and Address Mapping 20

3.4 Memory Access Protocol and Timing 21

3.5 Memory Architecture Limitations 28

3.6 Registered DIMMs 31

3.7 Rambus (DRD and XDR RAM) Solutions 32

CHAPTER 4: FULLY BUFFERED DIMMS 35

4.1 The Fully Buffered Architecture Introduced 36

4.1.1 Architecture Overview 37

4.1.2 FB-DIMM Overview 38

4.1.3 Southbound Channel Frames 40

4.1.4 Northbound Channel Frames 43

4.1.5 Pins and System Board Routing 46

4.1.6 Power and Heat 46

4.1.7 FB-DIMM Quick Facts 48

4.2 Serial Differential Signaling 48

4.3 The FB-DIMM Channel 50

4.3.1 Channel Initialization 50

4.3.2 Channel Timing 53

4.3.3 Data Rates and Peak Theoretical Throughput 54

4.4 The Advanced Memory Buffer 56

4.5 Reliability, Availability, Serviceability 59

4.6 Summary; Pros and Cons 61

4.7 Current Industry Adoption and Future Projections 63

CHAPTER 5: FBSIM SIMULATION MODEL AND IMPLEMENTATION 64

5.1 Simulator and Simulation Model Overview 64

5.1.1 Objective of Study 65

5.1.2 Simulation Model - Assumptions and Abstractions 66

5.1.3 Input Specifications 69

5.1.4 Output Specifications and Program Overview 72

5.2 Input Transaction Generation Model 76

5.3 Address Mapping Algorithm 82

5.4 The Scheduler 87

CHAPTER 6: FBSIM VERIFICATION AND RESULTS 95

6.1 Case Study I –Constant Capacity 97

6.1.1 DIMM and System Board Parameters 97

6.1.2 Ramp Input 98

6.1.3 The 1x8 Case 100

6.1.4 Comparison between the Cases 108

6.2 Debug Mode – A Microscopic Look at Channel Activity 112

6.2.1 Two Reads to Every Write 113

6.2.2 Read Only Traffic 116

6.2.3 Write Only Traffic 117

6.2.4 Four Reads to Every Write 118

6.3 Variable Latency Mode 119

CHAPTER 7: CONCLUSION AND FUTURE WORK 122

REFERENCES 126

LIST OF TABLES

2.1 High Speed Parallel Bus Limitations 10, 11

3.1 Important Synchronous DRAM Timing Parameters 22

3.2 Minimum Inter-Command Scheduling Distances in Open Page Row Buffer Management (Copyright by [18]) . . 25

4.1 Excess Power Consumption in FB-DIMMs 47

4.2 Pros and Cons of the FB-DIMM Architecture 61, 62

4.3 Some FB-DIMM Technology Manufacturers 63

5.1 .sim Input File Specifications 70, 71

5.2 .dis Input File Specifications (see Section 5.2) 72

5.3 Mapping Algorithm Output (map modulus is 9) 84

6.1 DIMM Parameters Specified for Case Study 97

LIST OF FIGURES

1.1 Market Projections for FBDIMM (From Infineon [26]) 5

3.1 A Memory Cell (from [18]) 14

3.2 A Section of a Memory Bank (from [18]) 15

3.3 Components of a DRAM Device (from [18]) 17

3.4 Memory System Topology (from [18]) 19

3.5 Consecutive Reads to Open Rows (from [18]) 23

3.6 Write After Read (top) and Read After Write (bottom) to Different Banks of the Same Rank (from [18]) . . 27

3.7 Server Capacity Demand vs. Supply (Intel [34]) 30

3.8 System Board Routing Patterns for Path Length Matching 31

3.9 DRDRAM (from [18]) 33

4.1 Elpida Test FB-DIMMs 35

4.2 Simple Routing Pattern from Memory Controller to FB-DIMMs 36

4.3 The FB-DIMM Architecture (from [34]) 37

4.4 FB-DIMM Heat Spreading [14] 47

4.5 Differential Pair Signaling (from [27]) 49

4.6 Channel Initialization FSM (from [15]) 51

4.7 The Advanced Memory Buffer (Intel [27]) 56

5.1 SETI@home in Bus Trace Viewer (from [18]) 80

5.2 SETI@home Simulated in FBsim 80

5.3 Vortex Benchmark in Bus Trace Viewer (from [18]) 81

5.4 Example Normal Distribution Transaction Rates in FBsim 81

5.5 Example Configuration 83
5.6 Memory Space Division amongst DIMMs 85
5.7 Closed Page Mapping 86
5.8 Open Page Mapping 87
6.1 Input Request Rate Specification; 1x8 Configuration 99
6.2 Data Throughput Response; 1x8 Configuration 100
6.3 Average Scheduling Window Size 101
6.4 FBsim Percentage Makeup of Scheduling Rejections 104
6.5 Average Read Latency Distribution 106
6.6 Average Read Latency vs. System Throughput 107
6.7 Throughput Distributions for the Four Configurations 108
6.8 Latency Throughput Curves for the Four Configurations 110
6.9 Causes for Scheduling Rejections in Each Configuration 111
6.10 Channel Utilization with Read : Write Ratio = 2:1 113
6.11 Channel Utilization with Reads Only 116
6.12 Channel Utilization with Writes Only 117
6.13 Channel Utilization with Read : Write Ratio = 4:1 118
6.14 Throughput Distribution for Variable Latency Mode 120
6.15 Latency/Throughput Curve for Variable Latency Mode 120

CHAPTER 1: INTRODUCTION AND MOTIVATION

1.1 Memory Bottlenecks

As Amdahl's law would have us remember, a computer system is only as good as its slowest part [11]. Even as tremendous attention and resources are funneled into the development and evolution of microprocessor technology, achieving remarkable and unrelenting increases in processing speed, and as computer systems are becoming more and more parallel in nature and the speed and independence of I/O devices increases, the memory subsystem is somehow expected to keep up in speed (response time) and with ever increasing request rates (throughput), and on top of that, to increase in capacity, and above all, to stay cheap. As the gap between memory and processor speed continues to grow, the architecture, organization and resulting performance of the memory subsystem becomes more and more critical.

The access time of the memory subsystem has traditionally been held back by three major factors: the innate switching limitations of memory process technology and its associated capacitance requirements for high density bit storage, the limitations of wide inter-silicon high speed communication and signaling physics, and of course, the all powerful dollar.

Due to the different process technologies involved than those of logic devices, high capacity memory needs to reside on its own chip, and to meet capacity requirements, many of these chips are needed. Of course all these chips

must be connected, and centrally controlled. Yet bus and interconnect technology has been struggling since the eighties to keep up with the staggering increases in silicon speed. Major issues are multi capacitive load and transmission line physics, multiple (reflective) physical ‘board’ discontinuities in the signal path between silicon and silicon, and inter-signal and inter-device drift, skew and synchronization. All these effects become more pronounced as communication speeds increase and signal wavelengths decrease below the physical distances involved in the interconnects. Memory interconnect technology has traditionally relied on wide, parallel multi-drop buses to meet command, addressing, and data transfer requirements. But due to the issues listed above, improvements in parallel bus technology have long been slowing and approaching a ceiling. Exacerbating the situation further is the fact that memory devices are grouped and mounted on memory modules (usually Dual Inline Memory Modules, or DIMMs) to conserve horizontal real estate on motherboards. This introduces more distance, asymmetry, and signal discontinuity into the data paths.

Due to the relative simplicity of memory devices, and the dominating requirements of high capacity and expandability, mainstream memory devices have traditionally overwhelmingly followed commodity market forces, and the logic of quantity, low profit and low cost over quality and speed. This has made the memory device market remarkably resilient to change. While DRAM has been evolving (FPM, EDO, SDRAM, DDR, DDR2), this evolution of memory devices has been forced to be slow and conservative. Radical shakeups of the memory

subsystem needed to meet modern processing requirements have met with heavy market resistance when necessitating drastic changes in memory device design (e.g. Rambus). Thus, a more ideal solution is to meet speed and capacity requirements by instead radically altering the organization of the memory subsystem, but while touching the memory devices themselves as lightly as possible, if at all.

In April of 2004, Intel proposed the Fully Buffered DIMM (FB-DIMM) architecture to do just that. Rather than communicating with memory devices over wide, parallel, multi-drop buses, in the Fully Buffered architecture an advanced buffer is added to each DIMM, and the memory controller communicates with the buffers through a daisy chained, narrow, point-to-point, more serial interface. Serialization (narrowing) and point-to-point buffering allows the channel to reliably operate at much higher data rates than a multi drop bus, to support many more connections (although more connections come at the cost of added latency), accommodates much lower pin counts, and dramatically simplifies motherboard routing and so decreases system board and memory controller packaging costs. Best of all, the old memory interface is maintained on the DIMM between the buffers and the DRAM devices, allowing the DRAMs to remain untouched!

This important memory technology shift did not come out of the blue. Across the spectrum of high speed digital interconnect interfaces, the trend over the last decade has been towards more narrow and serial communication, from USB, to Serial ATA, SAS, and most recently, PCI Express [30]. Other examples

involving radical changes in memory interconnect architecture of mainstream computing have been proposed and even implemented – Rambus DRDRAM and XDR being the most notable amongst these. These have also mostly been based on narrowing memory communication channels and serializing data transfer. For a myriad of reasons however, these technologies have never made big inroads into the mainstream memory market.

Many arguments however seem to suggest that FB-DIMM technology will be a different story. First is the inevitability and ultimate necessity of a change in memory inter-connect technology, which in this paper will be demonstrated to be failing to keep up with modern memory system requirements in the high end, and soon also in the mainstream of computing. Second is the above stated fact that the FB-DIMM architecture will not require any change in the memory devices, and so DRAM device design can continue at its (relatively) leisurely evolutionary pace. In fact, the FB-DIMM philosophy will be shown to be an important separator between the DRAM device design domain, and the domain of inter-chip signaling and system board interconnects. This fact should help guarantee financial viability. Finally and perhaps most importantly, is the fact that the FB-DIMM initiative from its inception has been a collaborative one. Intel started the engine, and all the mainstream memory device manufacturers have jumped on board. "There's incredible industry momentum around FB-DIMM and multi-core processor technologies that enable Intel architecture platforms to achieve even higher performance," says Jim Pappas of the Intel Digital Enterprise group [17].

JEDEC – the Joint Electron Device Engineering Council, an international body of semiconductor manufacturers that sets the official standards for memory systems – is currently in the process of standardizing the technology.

Fully Buffered DIMMs will enter the server market full force in the final quarter of 2005, and the workstation market soon thereafter. The graph in figure 1.1, given at an Infineon presentation [26], shows the projected takeover of the mainstream server/workstation memory market by FB-DIMMs.

In this paper, the evolutionary progressions that led us to the FB-DIMM architecture are described. Modern memory devices and trends are introduced, and with that background, the FB-DIMM architecture is explained and discussed in detail. The FBsim simulator which was written to explore this new architecture is also introduced, along with preliminary results generated and identified paths of future academic research into the FB-DIMM domain.

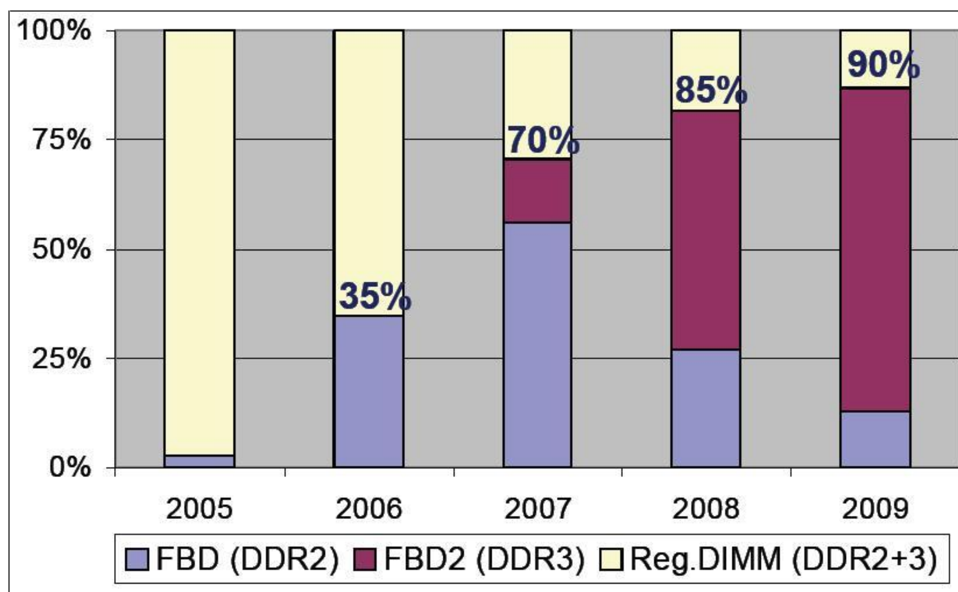


Figure 1.1 - Market Projections for FBDIMM (From Infineon [26])

1.2 Current State of FB-DIMM Implementation and Literature

The FB-DIMM concept was introduced to the world in February of 2004, at the Spring Intel Developers Forum (IDF). Since then, industry leaders have been collaborating behind closed doors to standardize the technology, and implement it. The final months of 2004 saw the first few FB-DIMMs demonstrated for test and marketing purposes. However, design decisions and implementation details have to date been cloaked in secrecy. As such, information on FB-DIMM technology has so far remained in the domain of high level industry slide shows and speculative magazine articles. It is to the best knowledge of this author at the date of this writing that no academic studies into the FB-DIMM architecture have yet been published. As such, this is believed to be amongst the first of academic studies exclusively in FB-DIMM, and the role of this research is more one of a pioneering synthesis of a variety of industry sources and identification of future research paths than a specific narrow study. However as stated above, FB-DIMM technology did not arrive from a vacuum, but rather a progression of evolutionary events in memory design, on which a rich and wide body of research exists, some of which is cited in this paper as it is encountered.

A few weeks before the writing of this paper, Infineon has released the first datasheet of an Advanced Memory Buffer (AMB) – the primary component of the FB-DIMM implementation [15]. The datasheet is preliminary and incomplete, but contains an interesting body of data revealing to those outside the circle of memory developers important implementation details of the FB-DIMM

standard-to-be. Another datasheet was released by Micron in April 2005, in the middle of this paper's writing. Combined with other industry literature, details from the datasheets have opened the gates for academic research into the FB-DIMM field. Although the first FB-DIMM compatible products will not hit the markets until the end of 2005, they will hit the market with force, and the sooner this new technology can be academically analyzed, the better the understanding will be, and the better the facts can be separated from the marketing hype.

1.3 Overview of Paper

In Chapter 2 of this paper, the evolution of high speed bus technology is discussed. The Scalable Coherent Interface (SCI) is reviewed. The SCI standard was developed in the late eighties and early nineties by a concerned group of high speed bus designers who saw parallel busses fast approaching a brick wall. This standard began the shift of silicon interconnect philosophy from wide parallel to high speed serial.

Chapter 3 introduces DRAM memory devices and identifies in more detail some of the architectural challenges that high speed memory system design is facing, and how the evolution of modern DRAM has so far met some of these challenges.

In Chapter 4, I introduce and describe in all available detail the new Fully Buffered memory architecture, while exploring design and performance issues, design trade-offs, and industry adoption.

Chapter 5 is a detailed overview of my Fully Buffered simulation model and FBsim simulator - how it works, what it does, and any simplifying assumptions made, as well as its application and scope of study. Chapter 6 performs three studies under FBsim, presenting interesting and important insights into the FB-DIMM design space and validating the correctness of FBsim along the way.

Chapter 7 concludes by summarizing the achievements of my study, and identifying future paths of research into the domain, whether using FBsim or not.

I hope you find the Fully Buffered memory design space as exciting and interesting to explore as I did.

CHAPTER 2: EVOLUTION OF HIGH-SPEED BUS TECHNOLOGIES

This chapter describes the beginning of the end for high speed wide parallel bus technology in the late nineteen eighties, and the gradual migration across the computer system to high speed serial (where 'serial' is taken to mean significantly narrower than the 32 or 64 bit wide data bus).

2.1 Futurebus+ and Bus Limitations

Sometime in 1987, Paul Sweazey, a leader of the IEEE 896 Futurebus project took time to think about where things were going. Futurebus was a high performance parallel bus technology supporting arbitration, locks, mutual exclusion, and even cache coherence. Futurebus was aiming to become a high-end multiprocessing backplane bus standard [10].

Sweazey, using his insights as an expert bus designer, as well as some foresight, calculated that due to signaling limitations, within a few short years even one microprocessor would be able to saturate any bus, even Futurebus. Parallel data transport technology simply scaled badly. Table 2.1 on the next page lists some of the major limitations to high speed parallel multi-drop signaling.

Although such parallel buses would remain sufficient in the near future to meet the needs of I/O subsystems, it became clear to him that for higher bandwidths, a fundamental change in direction was needed. And thus in 1987 the Scalable Coherent Interface (SCI) project was born.

Transmission Line Load and Reflective Characteristics	It is impossible to perfectly terminate a transmission line with a matched load, especially when that load varies over time, EM field, and temperature. Imperfect termination produces signal reflections - a ringing effect. This effect gets more pronounced as the number of loads on the line increases, and as a result, each bit sent down the line takes a certain time to 'settle'. As data-rates increase, this settling time becomes a non-negligible limiting factor in the signal's bit period.
Signal Discontinuities	The above effect is exacerbated further by the material discontinuities that inevitably exist between silicon and silicon. The signal travels through different materials as it travels from the sending chip to its pin, possibly to a chip mounted connector, through a soldered joint to a printed circuit board, through a stripline in or on the board, and again through soldered connectors and pins to the receiving silicon. Each material discontinuity represents an interface for signal reflection and degradation. The higher the number of connections to a transmission line, the worse this effect.
Current Loops and EM Field	Every signal must have a return path. This closed loop creates an EM field, and EM fields of adjacent loops interfere with one another. This effect, known as cross-talk, is a limitation to spatially adjacent data paths. Systems have to be engineered so that return paths are provided as close to the signal as possible to minimize the field. The wider the data path, the more interleaved return paths are needed to shield from cross-talk.
Inter-Signal Skew, Clock Skew and Path Matching	The wavelength of a signal decreases as its data-rate (frequency) increases. Assuming a transmission speed of two thirds the speed of light, a 1 GHz signal has a wavelength of just 20cm. Hence as transmission rates increase, system board distances become a non-negligible factor. Wide bus lines must be path-length matched, so that parallel signals arrive in a synchronized manner rather than arriving skewed in time. For synchronous communication, the clock path must also match the data path to achieve synchrony. Unfortunately, signal path length is not just a factor of distance, but also of temperature,

Table 2.1 – High Speed Parallel Bus Limitations

	material purity, load, random EM interference, etc. The wider the bus, the more chance that these factors will be varying across the bus. Advanced de-skewing techniques exist and continue to improve, but again the wider the data path, the costlier the de-skewing overhead.
Inter-device 'Skew' and Synchronization	As bit periods get smaller and distance becomes a bigger factor, multiple devices connected to a multi-drop bus inevitably end up operating in different (phase skewed) clock domains. This creates more problems for overall system synchronization.

Table 2.1 (Cont'd)

2.2 Scalable Coherent Interface (SCI)

The Scalable Coherent Interface [9] [10] was born as an alternative to wide parallel buses, one that could scale much better, and could transfer data much faster. A small team of ambitious and competent 'believers' led the development of SCI, and this optimal grouping combined with the low profile and low credibility of the project meant it operated very efficiently and under the radar of the usual bickering and industry politics. The standard was finalized in 1990, well before Futurebus – from which it was spawned – was complete!

SCI was a fully distributed solution to bussing. The 'bus' actually consisted of high speed point-to-point serial links between nodes, of which the standard supported up to 64K earning its 'Scalable' name (along with its support for LAN-like distances). A node could be anything, an SMP processing element, a processor with cache, a memory node, or an I/O node. Nodes automatically acted as repeaters or network hubs, repeating and re-synchronizing signals over

outgoing links. Each point-to-point link supported concurrent (bidirectional) transfers. The standard was defined to be topology independent. Highly advanced for its day, SCI defined a 64-bit addressing scheme, where the first 16 bits represent the node-ID, leaving the other 48-bits open to addressing within that node. SCI allowed imposing read or write locks on memory locations. An optional cache coherence layer was defined. The cache coherence was based on a distributed directory scheme with doubly linked lists. It supported multiple read but single write of shared locations, and was invalidation based. Data distribution and routing was hardware based, and transparent to the nodes. Other incorporated features were a transaction priority scheme, message broadcasting support, and a 16-bit CRC across the links for strong reliability. Thus SCI in actuality acted as a super high speed, low overhead local area network for device interconnect, with protocol layers defined (in hardware) over the network to make it look like a bus. As an interface, it meant that connected devices would only have to know how to talk to SCI, rather than know explicitly how to talk to each other as on a shared parallel bus. This 'separation of powers' allowed much of the signaling and compatibility headaches and overhead to be taken away from connected devices.

It was a very simple and elegant solution that was unfortunately well ahead of its time. While industry was accommodated to step solutions that increase performance and decrease price incrementally, here was a technology that required radical change in interconnect design. And this is why SCI never reached the mainstream – it was perhaps too much and too early. Politics caught

up with SCI as it was neglected and criticized by a large number of influential people with stakes in Futurebus, and other technologies such as FibreChannel. Although it was adopted to a small degree by some manufacturers and reached some end products, SCI was poorly marketed and overwhelmingly ignored. As an open standard, subsets of SCI along with its core design philosophy were more or less slowly and silently stripped away and copied as the limits of parallel communications were reached by different segments of the computer device spectrum. As a serial 'translation' of a previously parallel interconnect technology, the FB-DIMM architecture owes a good deal of its heritage to these pioneering ideas.

2.3 Adoption of High Speed Serial Technologies

As parallel signaling has reached limitations across the computer system interconnect spectrum, high speed serial technologies have become the norm. The old parallel port to external I/O devices has been long replaced by Universal Serial Bus (USB) standards. SAS (Serial Attached SCSI) has been overtaking parallel SCSI in the high end (enterprise) server hard-drive interconnect market just as Serial ATA is replacing parallel ATA for mainstream computers [30]. Graphic card and other I/O device makers are embracing PCI Express as the new peripheral interconnect technology. Only DRAM memory devices still rely on wide parallel interconnects, but with FB-DIMM, this is soon to change.

CHAPTER 3: MODERN MEMORY TECHNOLOGY

In this chapter, basic concepts of memory device design and memory system architecture are introduced for background. For details, see [18].

3.1 Introduction to DRAM Devices and Organizations

DRAM devices are the basic building blocks of a Dynamic Random Access Memory system. A DRAM device is a chip whose sole function is to store and retrieve as many bits as possible, and as quickly as possible. The basic element of a DRAM device is the memory cell. A memory cell is traditionally made of up of one transistor and one capacitor, as depicted in figure 3.1.

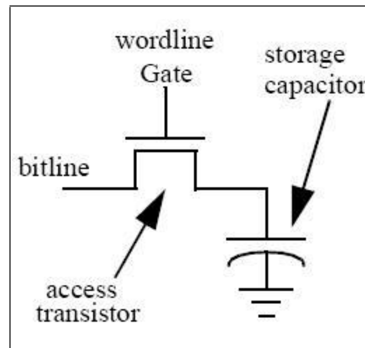


Figure 3.1 – A Memory Cell (from [18])

Memory cells are arranged into large two dimensional arrays named banks. The horizontal divisions of the bank are called rows. Each row features a wordline connected to the gates of the transistors of all memory cells in that row. Vertically arranged across the bank are the bitlines, connecting vertically adjacent cells across rows, as in Figure 3.2.

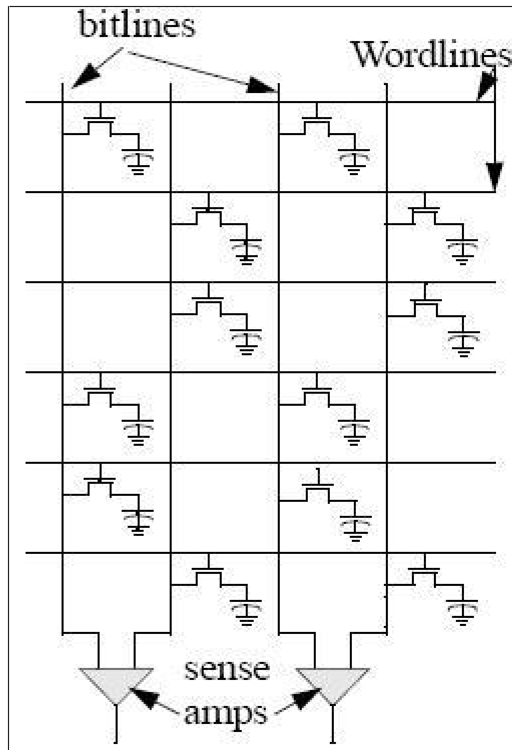


Figure 3.2 – A Section of a Memory Bank (from [18])

A memory operation to a bank begins when all bitlines in that bank are charged to a reference voltage V_{REF}. This is a bank precharge. Next, the voltage is raised on the required wordline, effectively 'opening the gates' of all access transistors on that wordline and connecting their capacitors to their bitlines. This is a row activation. Depending on the states of the connected capacitors, they will either discharge into their bitline raising its voltage slightly above V_{REF}, or charge from their bitline, lowering its voltage slightly below $V_{REF.}$ Sense amplifiers connected to the bitlines will sense the polarity of these voltage differentials, lock on to them, and amplify them to full voltage. Thus a bitline slightly below V_{REF} will drop down to LO voltage, drawing all charge out of the capacitor on the bitline, while a bitline slightly above V_{REF} will be driven up to HI voltage, topping

up the charge in the capacitor. Once this charging or discharging has stabilized, the wordline voltage can be removed, thus ‘closing the gates’ on the capacitors, and sealing their charge until the next access of that row (although leakage does occur, necessitating memory refresh cycles).

In a memory read operation, a precharge followed by a row activation reads all the bits in the activated row into the sense amps. In a write operation, the row activation is followed by driving the bit pattern representing the data being written onto the sense amplifiers, effectively overwriting the existing charges on the targeted capacitors of that row.

While there are usually thousands of rows in a memory bank, each row is also usually a few thousand bits wide. We know however that each memory address is logically 8 bits wide. This necessitates the division of a row into columns, where column ID specifies which bits of the row are to be read, or where in the row given bits are to be written. Columns are not necessarily 8 bits wide. Depending on the device, they can be more or less than 8 bits in width. The width of every column in a DRAM device is equal to the number of data in/out pins on the device.

The limitation of this grid architecture is that only one row of a memory bank can be accessed at a time. The precharging takes some time, the row activation and data sensing/overwriting takes some time, and the recovery of the capacitors on that row takes some time. During this cycle time, other rows in the bank cannot be touched. DRAM devices circumvent this restriction by dividing

their area into multiple banks, each of which can be accessed independently. However, the data paths between these banks and the outside world are traditionally shared, thus somewhat limiting parallel operation.

Figure 3.3 illustrates the main components of a synchronous DRAM device.

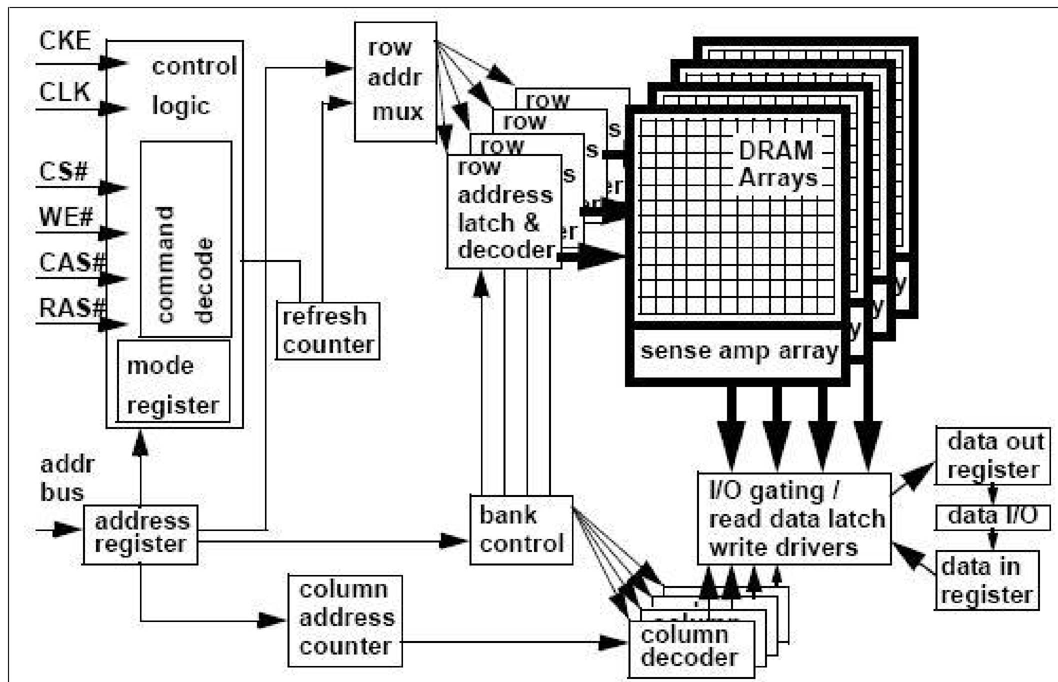


Figure 3.3 – Components of a DRAM Device (from [18])

The CS# signal is a chip select/inhibit. WE# is the write enable signal, which is activated when the device should latch in data and write it into the addressed column in the DRAM array, and deactivated when the device should read from the addressed column and latch out the read response data. RAS# is the Row Address Strobe signal. It is activated after a row address is published on the address bus, signaling the device to activate the addressed row (bank ID

information is contained in the row address). CAS# is the Column Address Strobe signal. It is activated when a column address is published on the address bus, telling the DRAM device to either read from or write to that column. Unused combinations of these signals and/or additional control signals are used to initiate bank precharge, refresh, and other operations.

3.2 Ranks, Memory Modules and Bussing

DRAM devices usually have 1, 2, 4, 8 or 16 data in/out pins. Hence to accommodate the 64-bit width of the data bus, multiple DRAM devices are connected to the data bus side by side as a rank. All devices in the rank share the address and command bus, so if column x in row y of bank z is accessed in one device in the rank, the same bank, row and column is simultaneously accessed on all the other devices in the rank.

Thus the memory controller is connected identically to all the devices in a rank through a single address and command bus, with a segment of the 64-bit data bus going to each. This combination is called a channel. Multiple ranks of devices can sit on a channel, with each rank enabled or disabled with a strobe signal. Thus all DRAM devices in all ranks of a channel share the address and command bus, while segments of the data bus are shared across ranks. This topology is summarized in figure 3.4.

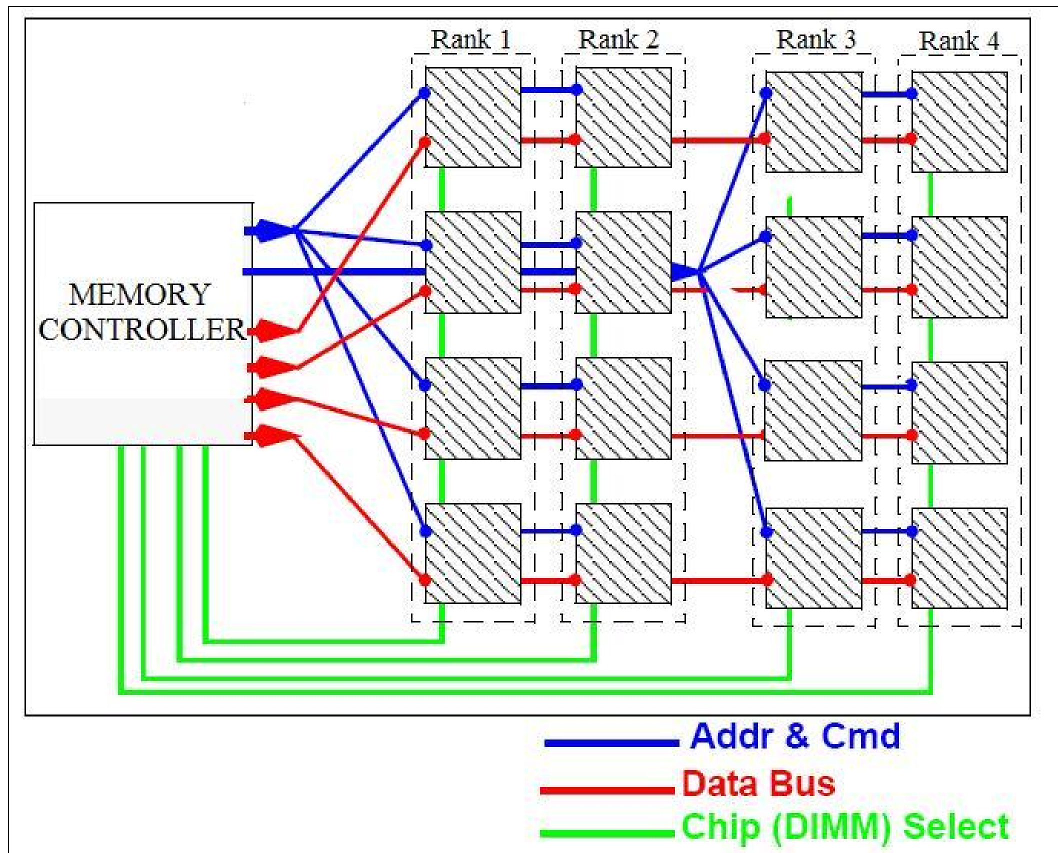


Figure 3.4 – Memory System Topology (from [18])

To save on system board real estate, and to simplify memory expandability and interchangeability, ranks are mounted together on memory modules, mostly Dual Inline Memory Modules (DIMMs). DIMMs stand perpendicular to the plane of the system board. A DIMM contains one or more whole ranks. Multiple DIMMs however may still share the same channel, effectively increasing the number of ranks on that channel. As different DIMMs have different capacities, and can be organized and controlled in different ways, DIMMs contain small flash memory SPD (Serial Presence Detect) devices which

store information relevant to the DIMM, such as capacity, number of ranks, banks, rows, columns, etc., and timing parameters (see section 3.4).

3.3 Row Buffer Management and Address Mapping

As discussed in section 3.1, a row activation of a DRAM bank leads to the sensing of the thousands of bits on a row, while a read or write command only accesses a single column of that row. To decrease command overhead, modern DRAM devices employ burst mode, which means that a fixed number of columns (for example 4 or 8) are burst in to or out of the device in a pipelined manner at each read or write command. But still, a large number of bits on the sense amps remain untouched. The array of sense amps is also called the row buffer. There are two extreme philosophies of row buffer management.

The **open page** philosophy dictates that once a row is activated, it should be left open until another row in that bank needs to be activated. This means that future read or write requests to that open row will not incur the precharging and row activation delay while it remains open. This assumption puts faith into having good locality in the memory request stream. A side effect however is that request times become variable, depending on whether or not the requested row is open. This complicates timing and memory command pipelining. It also means that the memory controller has to keep track of which rows are open in which banks of what ranks. And finally, as the sense amps remain open and the wordlines remain charged, power is being consumed.

The **closed page** philosophy dictates that once a row has been activated and read from or written to, it is immediately closed, and the bank precharged. This implies a regular pattern of 'Activate, Operate (read or write), Precharge' per operation, which is easier to control and pipeline. The trade-offs of open and closed page buffer management are non-trivial, and their performances vary over different request streams.

Mapping schemes for translating physical memory addresses into rank, bank, row and column Ids can be optimized according to the row buffer management policy used. For example, in open page mode, you would want the row ID bits to be as far to the left (most significant) as possible, so that as large as possible a chunk of contiguous memory is mapped to the same row.

3.4 Memory Access Protocol and Timing

A myriad of timing parameters are involved in control of and communications with DRAM devices. These form the basis of what is called the memory access protocol. The main parameters related to the studies in this paper are described in table 3.1. Timing parameters are usually specified as (rounded up) integer multiples of t_{cycle}, the memory clock cycle time. Table 3.1 provides typical values of these parameters for DDR 2, normalized to cycle time. Other important timing parameters exist, such as parameters that limit current draw and power consumption over time, but these are beyond the scope of this paper.

t_{cycle}	Synchronous DRAM clock cycle time. In Double Data Rate (DDR) RAM, data is transmitted to and from the devices at both rising and falling edges. The period of data transmission here is called a beat, where a beat is half of t_{cycle}.	3ns
t_{Burst}	Burst time. The number of consecutive cycles a data burst in or out of the DRAM device occupies.	2 or 4
t_{CMD}	The amount of time a command must remain on the command bus before it is latched into the DRAM device.	1
t_{RP}	Row Precharge. The time a DRAM bank needs to precharge its bitlines to V_{REF} (in preparation for row activation) after a precharge command has been received.	5
t_{RCD}	Row to Column command Delay. The time needed after a row activation to sense row data into the sense amps and become ready for a column read or write operation.	5
t_{RAS}	Row Access Strobe. The time needed from the end of a row activate command until the capacitor charges in the row are recovered and the bank is ready for another precharge.	14
t_{CAS}	Column Access Strobe. The time needed from the beginning of a column READ command receipt until the DRAM device begins bursting data out of its data pins.	5
t_{CWD}	Column Write Delay. The time expected between the memory controller's signaling of a write command, and the beginning of the write burst into the device data pins.	4
t_{WR}	Write Recovery. The time needed from the beginning of receipt of a write burst until when the capacitors in the row being written have been charged and the bank is ready to be precharged.	5
t_{RC}	Row Cycle. $t_{RC} = t_{RP} + t_{RAS}$. The time needed to activate a row, read from or write to the row once, and then precharge the bank.	19
t_{DQS}	Data strobe turnaround time. When a wide parallel bus is operated at high speeds such as in DDR and DDR2, the device sending data takes control of a 'source synchronous' strobe signal that acts as a synchronizing clock for the data transmitted. When control of the data bus is switching from one device to another, a turnaround time must be respected for the data strobe to prevent mis-synchronization and data collisions.	2

Table 3.1 – Important Synchronous DRAM Timing Parameters

Each of these time parameters must be known and respected by the memory controller. The DRAM device is a 'dumb' device, responding identically

to a given command no matter when the command is given, thus it is up to the memory controller to avoid unpredictable results by not violating these times between commands.

However, a lot of pipelining potential can be exploited by the controller. The top half of figure 3.5 illustrates how two consecutive read commands to the same open row in a bank can be pipelined so that they are only t_{BURST} apart, while the lower half shows the same is true for consecutive reads to different open rows in different banks in the same rank. The same holds for consecutive writes.

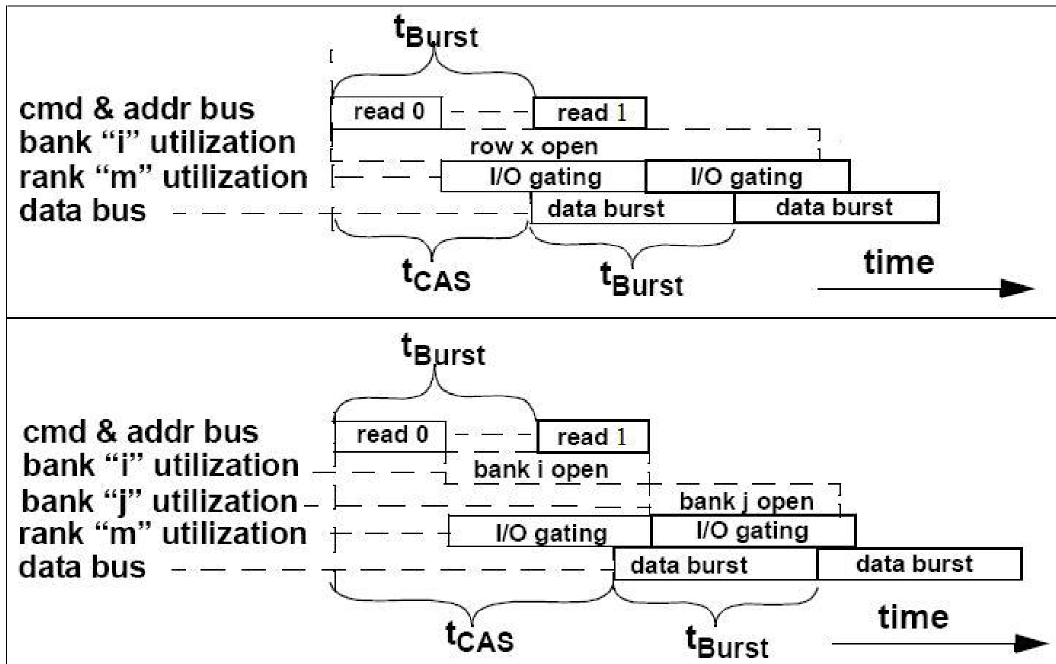


Figure 3.5 – Consecutive Reads to Open Rows (from [18])

In the case of consecutive reads or writes to different rows in the same bank however, the bank must be precharged and the new row activated before the new read or write can take place. And the bank cannot be precharged of course until the old row has recovered from its activation. So depending on the case, t_{RAS},

t_{RP}, and t_{RCD} come into play, inevitably leaving the data bus idle for that period of time. A comprehensive and in-depth study of minimum scheduling distances between DRAM commands in all cases can be found in [18] (Jacob, Wang). Their work is summarized in a table, included here as table 3.2.

Table 3.2 assumes open page row buffer management. It gives us the minimum scheduling distance between consecutive DRAM reads and writes. The 'previous' column tells whether the first instruction is a Read or a Write, and the 'next' column tells the same for the next one. The 'rank' column tells if the instructions are accessing the same rank (s), or different ranks (d). The 'bank' column tells if the same (s) or different (d) bank is being accessed. The 'row' column tells us whether the second instruction's row is already open (o), or if it encounters a row conflict (c). In the case of a row conflict, either the row to be closed is already recovered since its last row activation and can be closed immediately ('best case' column), or it has not yet had time to recover to recover and must first completely recover before a new precharge command can be sent ('worst case' column).

In closed page row buffer management, every DRAM instruction begins with a row activation command, which is followed by a read or a write command, and the instruction ends with a precharge command leaving the row closed and the bank precharged. So in closed page mode, an instruction is always made of three identically spaced. I found that inter-instruction scheduling times in closed page mode can be mapped out of table 3.2, as is described by the colored boxes I

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

Table 3.2 – Minimum Inter-Command Scheduling Distances in Open Page Row Buffer Management (Copyright by [18])

added. The ‘row’ column is ignored as a row in this mode will always be closed, but already precharged. So a case falling within a red box will require the minimum scheduling time between instructions denoted by the corresponding red box, blue case with the corresponding blue box, etc. The rationale for this mapping is that consecutive instructions to the same rank and bank in closed page mode must be a full row cycle apart, corresponding to the worst case in open page mode. Consecutive instructions to the same rank but different bank in closed page mode can use the same scheduling distance as the best case in open page mode because there is no such thing as a row conflict in closed page mode, and the three commands within the two consecutive instructions can be pipelined. The same argument applies for consecutive instructions to different ranks.

It is important to note that minimum scheduling distances as given in the table apply not only to consecutive commands (or instructions in CPM), but rather between all commands. Thus before a new command is issued, it must be checked off against all previous commands that might potentially be a hazard, not just the one before it.

You will notice from table 3.2 that consecutive reads and consecutive writes to DRAM can be pipelined quite nicely, allowing good utilization of the data bus. However, the cost of a write after read is higher, and the cost of a read after a write is greater yet. Figure 3.6 illustrates these cases for write after read (top half) and read after write (bottom half) to different banks in the same rank.

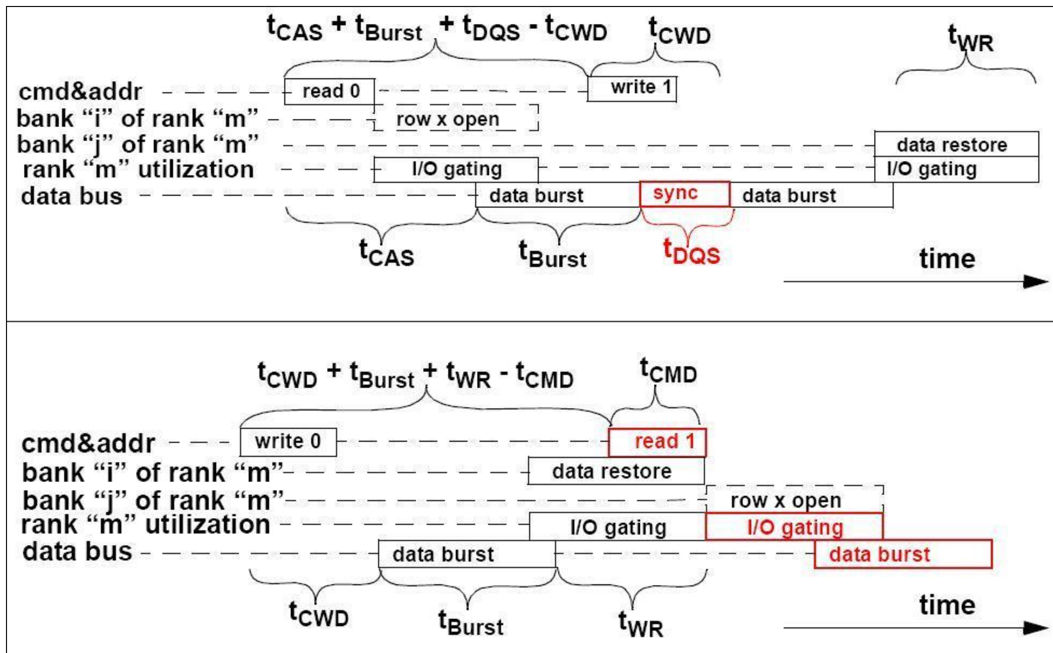


Figure 3.6 – Write After Read (top) and Read After Write (bottom) to Different Banks of the Same Rank (from [18])

In a write after read, the pipelining diagram follows a 'V' shape which isn't so bad. The data bus - the most important (shared) resource - is the bottom of the 'V' and is utilized quite efficiently, the only restriction being a t_{DQS} period as the memory controller, preparing to burst write data, takes control of the data strobe from the DRAM device which has just finished bursting its read data. In a read after write however, the 'V' is inverted leaving a wide opening on the data bus between the write data send burst, and the read data receive burst. The main cause of this gap is the fact that the write and the read must share the same I/O gating circuitry that routes data around the device. The overheads just described are known as read/write turnaround time, and write/read turnaround time.

The concepts simplistically introduced in this section are very important to understand how the Fully Buffered architecture will manage to hide some of these

debilitating turnaround times to achieve better channel utilization under large memory request rates.

3.5 Memory Architecture Limitations

As was described in section 3.2, interconnects between the memory controller and the DRAM devices have been implemented on wide, parallel multi-drop buses both to relay commands and addresses, and to send data back and forth. The following reasoning explains this reliance: A single memory devices cannot contain enough capacity for a modern memory system. Thus multiple devices must exist. To optimize performance, these devices are then made to work in parallel synchrony as ranks, necessitating multiple connections of the same address bus, and synchronization across a wide data bus. Capacity can only be (simply) added to a memory channel in rank granularity (i.e. on DIMMs), necessitating a multi-drop databus between the memory controller and DIMMs, and additional rank enable signals.

This complex topology (visualized above in figure 3.4) becomes very difficult to operate at high bit-rates due to all the limiting factors described in section 2.1. As this difficulty or inertia increases, implementation complexity to overcome it increases too, as do overheads. To give a small example, t_{DQS}, the data strobe turnaround time (which only became necessary when data transfers on the buses started operating close to the hundreds of megahertz), is more or less a fixed time related to the channel length and the I/O circuitry response time. As the

buses are driven to higher bit-rates, this time remains fixed, making its overhead proportionally increase, and so utilization efficiency of the parallel bus decreases.

Another Achilles heel of the multi-drop bus architecture is the large numbers of signal discontinuities in the data path: memory controller to pin, pin to connector, connector to system board (solder), system board to DIMM connector (solder), DIMM connector to DIMM printed circuit board, DIMM board to DRAM device pin (solder), and DRAM device pin to silicon.

These combining factors have made high speed memory buses a non-scalable proposition. As the data rate goes up, the number of devices sitting on a memory channel is thus forced to go down. With SDRAM 100, up to 288 devices were supported on a channel (in 8 DIMMs each of 2 ranks). With DDR-200, this dropped to 144 devices, and again to 72 devices with DDR2-400 (2 DIMMs each of 2 ranks, or 4 DIMMs each of 1 rank) [26]. Today, at the threshold of DDR2-667 and 800, the number of supported devices threatens to halve again, and yet again for the soon to arrive DDR3. In servers - which depend strongly on high memory capacities for performance, the memory capacity given by 36 or less devices per channel is not near enough. Figure 3.7 illustrates Intel's prediction of the growing gap between server capacity demand, and the supply that the multi-drop (stub) bus architecture is able to provide as data-rates increase.

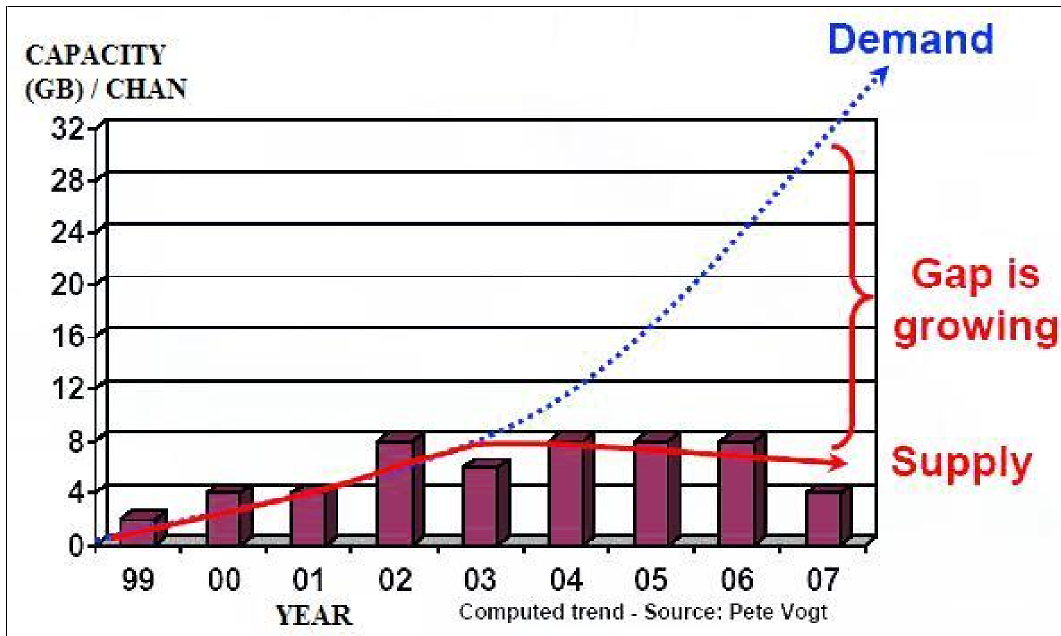


Figure 3.7 – Server Capacity Demand vs. Supply (Intel [34])

One possible solution is to dramatically increase the number of memory channels. Increasing the number of memory channels however means increasing the pin count on the memory controller as well as increasing system board real estate, both expensive propositions.

Another factor helping to make high speed parallel buses a non scalable proposition is system board real estate. At high data-rates, parallel bus signals need to be path length matched, resulting in wasteful, serpentine routing patterns taking up multiple layers on the system board, as in figure 3.8.

Continuing to meet the growing demands on modern memory systems will therefore require a drastic change of direction in overall memory system architecture/organization.

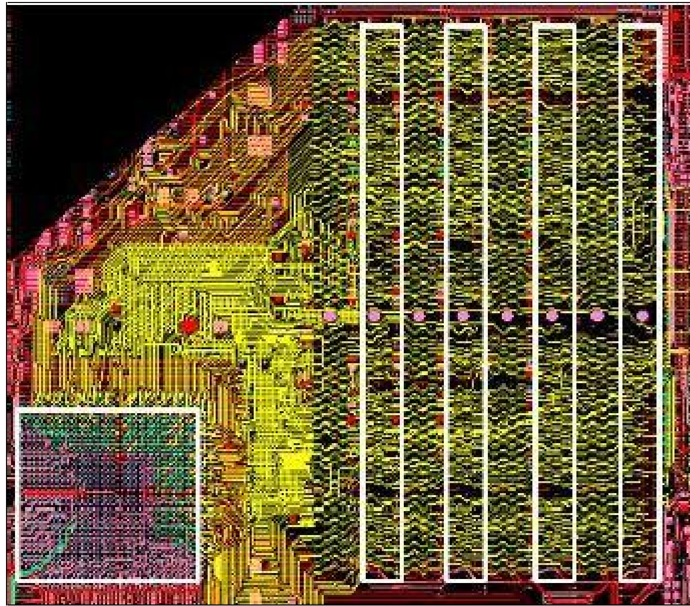


Figure 3.8 – System Board Routing Patterns for Path Length Matching

3.6 Registered DIMMs

A temporary quick fix to the multi-drop bus load problem was introduced with Registered DIMMs. A registered DIMM contains a repeater for the address, command, and data signals to the DRAM devices on that DIMM [18]. Thus the memory controller will only see one load per DIMM on the buses, and more DIMMs can be supported. This repeater buffer will of course introduce a small added latency to the signal paths. Furthermore, this solution only partially redresses the load issue of multi-drop bussing (as each registered DIMM is still a drop on a multi-drop bus), and does not comprehensively redress the other issues listed in Table 2.1. As such, it is only a partial fix, or patch. In fact, the statistics given in section 3.5 for DDR-200 and on apply to registered DIMMs!

3.7 Rambus (DRD and XDR RAM) Solutions

The Rambus company has proposed, implemented, and marketed the radically different Direct Rambus DRAM solution, and more recently, XDR. In its first attempt (DRDRAM), Rambus encountered only fleeting success and small inroads into the main memory market. Some lessons were learned for XDR, and its market performance remains to be seen. Although Rambus were the first to propose such radical change, most of their innovations and ideas weren't necessarily new or original; many had been seen before in different technologies or standards. However Rambus managed to patent these technologies, and collect royalties from anyone who wanted to implement them.

The DRDRAM solution was to change the memory system architecture and signaling organization in a big way [18]. The 64-bit wide data bus is replaced with a multiplexed 16-bit bus that operates at least four times faster. Rather than acting like a multi-drop bus where data has to settle across all the loads, the 16-bit wide bus supports multiple bits simultaneously in flight. Figure 3.8 illustrates the DRDRAM organization.

Note that there are two clock lines, one headed to the memory controller (clock to master), and one heading out of the memory controller (clock from master). Data sent to the memory controller is synchronized with the clock to master, and data sent out of the memory controller is synchronized with the clock from master. Thus channels are narrower and the signaling protocol is spatially aware, allowing for much higher data-rates to be achieved.

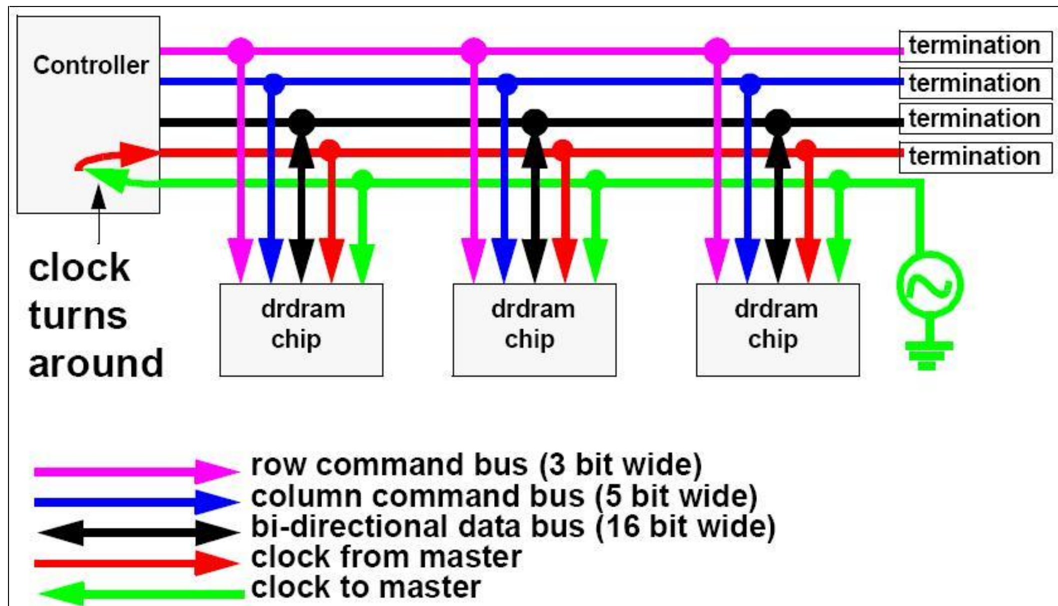


Figure 3.9 – DRDRAM (from [18])

The main downfall of DRDRAM was that it required DRAM devices to significantly change. As described above, the DRAM device market is a commodity market operating at very low profit margins, and as such is highly resistant to change. This fact, coupled with the ill will that Rambus engendered with its licensing fees, inhibited the widespread adoption of Rambus technology in the DRAM market.

XDR, the next generation Rambus technology, essentially maintains the ideas of DRDRAM, while correcting some of the (smaller) mistakes Rambus made the first time around and taking evolutionary steps forward. The main change from DRDRAM is the adoption of point to point serial links for the data 'bus' between the DRAM devices and memory controller. This essentially means

1 rank per channel. However, the address and command buses remain parallel multi-drop. Low voltage high speed differential pair signaling is used for the point-to-point data connections allowing for high data-rates, while the address and command buses operate at a much lower data-rate. Complex de-skewing circuitry in the memory controller ensures incoming and outgoing data is compensated for the differences in signal path length.

As you will see in the next chapter, some of the ideas adopted in XDR are quite similar to those of the FB-DIMM architecture.

CHAPTER 4: FULLY BUFFERED DIMMS

The Fully Buffered DIMM (FB-DIMM) architecture was introduced in February of 2004 at the Intel Developers Forum [34]. As of the writing of this paper, test FB-DIMMs have been produced, and FB-DIMMs are projected to enter the server memory market in full force at the end of this year (2005), and the workstation market soon thereafter. Figure 4.1 is a photograph of two types of Elpida test model FB-DIMMs.

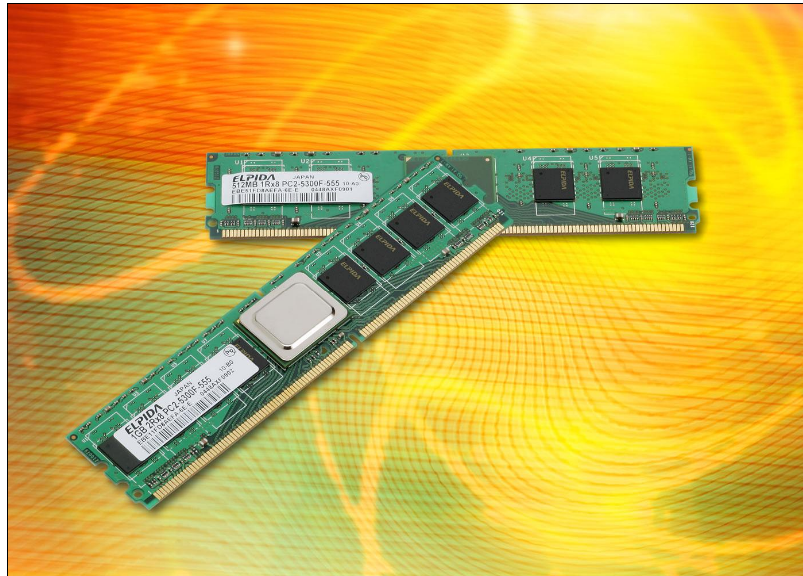


Figure 4.1 – Elpida Test FB-DIMMs

As well as being highly scalable and reliable, the FB-DIMM architecture is a very cost effective memory solution, and one that is viable in the long term. Industry is already aligned around this emerging technology with the immediate goal of smoothing the transition from DDR2 to DDR3, but experts expect it to be around for multiple memory generations.

4.1 The Fully Buffered Architecture Introduced

Rather than communicating with memory devices over wide, parallel, multi-drop buses, in the Fully Buffered architecture an Advanced Memory Buffer (AMB) is added to each DIMM, and the memory controller communicates with the AMBs through a daisy chained, narrow, point-to-point serial interface. This narrowing of the memory channel along with point-to-point buffering allows the channel to reliably operate at much higher data rates than a multi drop bus, and to support many more connections (at the added price of serial latency). It also accommodates much lower pin counts per channel, and dramatically simplifies motherboard routing. Figure 4.2 shows the routing traces on a fully buffered memory channel. Contrast this with the complex parallel bus routing of figure 3.8!

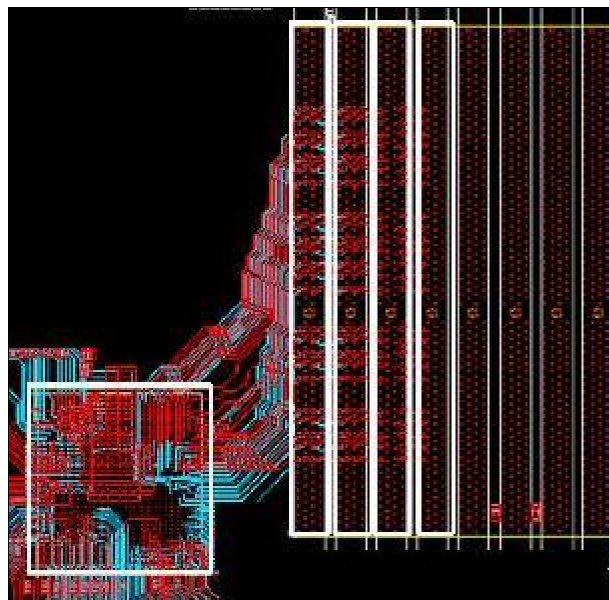


Figure 4.2 – Simple Routing Pattern from Memory Controller to FB-DIMMs

But perhaps the best thing about FB-DIMM is that the old parallel interface is maintained on the DIMM between the buffers and the DRAM devices, allowing the DRAMs to remain untouched! The short distances and limited number of loads on a DIMM allow the parallel interface to be much faster.

4.1.1 Architecture Overview

The FB-DIMM channel architecture is summarized in figure 4.3.

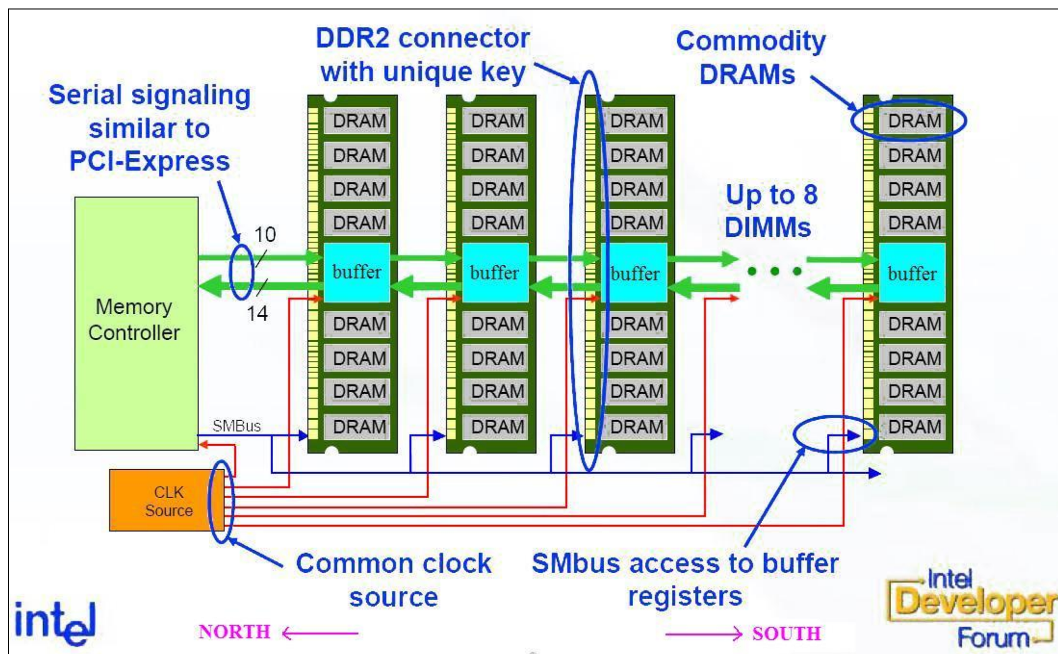


Figure 4.3 – The FB-DIMM Architecture (from [34])

In the FB-DIMM architecture, the classic parallel data, command, and address buses to the DIMMs are exchanged for two narrow, unidirectional, point-to-point high speed channels. The channel that caries command, address and write

data from the memory controllers to the DIMMs is called the SouthBound (SB) channel, and the channel that caries read responses back from the DIMMs is called the NorthBound (NB) channel. Together the NB and SB components form the overall memory channel.

The SB channel is composed of 10 serial differential pair links, and the NB channel of 14. The signal pairs aren't clocked together, thus obviating the need for path length matching on the system board. Instead, de-skewing circuitry on either side of the link can be used to synchronize the incoming bit-streams. The Low Voltage Differential Signaling (LVDS) mechanism – which is also used of PCI Express and other high speed serial protocols - is explored in more detail in section 4.2.

A common base clock is driven to the memory controller and all the DIMMs. This is only a base clock used for frequency multiplication, and does not directly relate to signaling on the NB/SB channels. A synchronous packetized frame protocol defines communication on these links.

On each DIMM sits an Advanced Memory Buffer that understands this frame protocol, and if it is being addressed, it will translate the protocol into the old DDR interface to the DRAM devices on its DIMM, and vice versa for read responses. This preserved interface is how the FB-DIMM architecture is able to support commodity DRAM devices.

The memory controller only drives its outgoing (SB) signal to the first AMB on the channel. Each AMB is responsible for immediately replicating and

forwarding the signal to the next AMB on the channel, until the last DIMM on the channel is reached. The same applies for Northbound communication. This is what is meant by a daisy chained, point-to-point link.

Also present in the architecture is a slow, multidrop serial connection that operates using the SMBus protocol. The SMBus (System Management Bus) is a two wire 100 KHz interface that gives the memory controller access to the control registers on the AMBs (all the AMBs are slaves on the SMBus), and data on the SPD chips of the DIMMs (see section 3.2). One wire is the data line, and the other is the data clock. Three address lines are also included in the SMBus to address the up to 8 DIMMs on the channel. The SMBus is used at system startup to configure AMB registers with data on NB/SB link speed, strength, and other link parameters. It is essentially to the FB-DIMM system as a car ignition circuit is to the car's motor.

4.1.2 FB-DIMM Overview

Other than the added AMB chip and heat shield, little difference can be immediately observed between a fully buffered DIMM and a normal DIMM in terms of appearance. Like normal DIMMs, FB-DIMMs contain either one or two ranks of DRAM devices, with each rank containing 8 or 9 x 8-bit wide DRAMs, or 16 or 18 x 4-bit wide DRAMs. The 8x8 and 16x4 configurations constitute the 64-bit wide data bus, while the 9x8 and 18x4 configurations add 8 ECC error correction code bits (the extra density needed to store ECC bits is not counted

towards the DIMM's capacity. Instead, the DIMM is called an ECC-DIMM). Also like normal DIMMs, an FB-DIMM contains an SPD device that stores configuration bits as well as descriptions and parameters of the DIMM relevant to the memory controller.

The FB-DIMM's inner topology however is of course very different than a normal DIMM. All data paths in and out of the DIMM pass first through the AMB. However, data transfers, commands, and addressing between the AMB and the DRAMs utilize the old parallel bus configuration and protocol described in Chapter 3 over the small distances on the DIMM. A further difference is that every FB-DIMM is notched on the bottom to prevent insertion into a non buffered channel, and to prevent non buffered DIMMs from fitting into FB-DIMM connectors. FB-DIMMs can stand alone on the system board, or in the case of limited space, FB-DIMMs support mounting on riser boards.

Featured in the FB-DIMM specification is the ability to add and remove FB-DIMMs while the system is powered up without halting the system. This is known as hot-add and hot-remove.

4.1.3 Southbound Channel Frames

The SouthBound (SB) channel is composed of 10 signal pairs, each operating at twelve times the frequency of the DRAM clock. A SB frame cycle is defined to encompass 12 bit cycles, so that the frame rate is equal to the DRAM clock frequency. This makes synchronization easier between the high speed

channels and the DRAM buses (prevents crossing of clock boundaries). Multiplying 10 by 12 gives us 120 bits in a SB frame.

There are two modes of operation on the southbound channel. In normal channel mode - when all 10 lines are up, 22 bits of the 120 are used as a powerful CRC (Cyclic Redundancy Check - error detection code). Note that this CRC is implemented over the entire channel frames, and is independent of any ECC data that may exist in the frame if the DIMMs are ECC DIMMs. These are multiple layers of protection. In fail-over channel mode – when 1 of the 10 signal pairs is down and 9 x 12 = 108 bits are left per frame, the 12 bit difference is taken out of the CRC.

In any case, 98 bits remain in a frame once the CRC is removed. How these bits are allocated depends on the type of frame. 2 bits of the 98 specify the frame type, leaving 96. Differing in their usage of these 96 bits are two types of SB frames, DRAM frames, and channel frames. The following are the two different types of DRAM frame:

- A command frame contains room for three commands. Examples of commands are precharge, activate, read, write, refresh, and NOPs, although more complex DRAM commands such as precharge all, enter/exit self refresh, and enter/exit power down mode are supported. Each command segment is called a ‘frame third’ in this paper. Of course 3 bits out of each command address the up to 8 DIMMs on the channel. Further bits identify the command type, rank ID, and bank ID of DRAM commands. If the command is a ‘row

activation', it must include the row address, and if it is a read or a write command, it must include the column address. Note that a maximum of one command can be sent to each DIMM in each southbound frame, as the frame period is equal to the entire DRAM clock period. DRAM devices on FB-DIMMs must obey '1n command timing', which means that commands reside on the DIMM's command bus for one DRAM clock. This issue of frame third vs. DRAM period means that a synchronization convention must be kept between the frames and the DRAM clocks. According to [15] (datasheet), the DRAM command period begins at the end of the first third of SB frames, meaning the two are phase shifted by 120º. This is done for optimal pipelining, as in most cases one command or less will arrive on each frame.

- A write data (Wdata) frame allocates 72 bits for write data (whether ECC is used or not in the DIMM), and the remaining 24 bits house a command (it is as yet unclear at the time of this writing why if commands are 24 bits, a command frame cannot fit four commands in its 96 bit payload rather than just three. Perhaps extra space is reserved for future use). Recall here that DDR devices receive write data in bursts. In these bursts, data is clocked on both the rising and falling edges of the DDR clock, hence the name Double Data Rate. Thus two 64 (or 72) bit parallel words of data must be sent to the DRAM devices in a DDR clock period. But the channel frame period is equal to the DDR clock period, and only one parallel word arrives per SB frame, meaning that in terms of write data, the SB channel is operating at half the

DDR data-rate. This necessitates the usage of a FIFO write buffer in each AMB. The memory controller sends Wdata frames to an AMB's write buffer whenever it can (i.e. in a not necessarily contiguous manner), and once enough data has been buffered to allow a non-interrupted DRAM burst, the memory controller can any time subsequently send a write command to initiate the transfer.

Different SB channel frame types are needed for channel initialization, differential signal clock synchronization frames (see section 4.1.4), AMB configuration register read commands or write commands + data, channel debug or soft reset frames, or plain old IDLE frames. The memory controller also directly controls the CKEs (clock enables) of all the DRAM devices through channel frames to the DIMM's AMB. Options are enable/disable over DIMM, and enable/disable over rank. This is used for channel power control and standby mode.

As we will see, DRAM read commands as well as some channel commands like 'Read Config Reg' and 'Sync', generate traffic on the northbound channel. These commands must be timed by the memory controller to avoid collisions on the NB channel. These issues are discussed further in section 4.3.2.

4.1.4 Northbound Channel Frames

The NorthBound (NB) channel is composed of 14 signal pairs, each operating at twelve times the frequency of the DRAM clock. A NB frame cycle is also defined to encompass 12 bit cycles, so that the frame rate here is also equal to the DRAM clock frequency. Multiplying 14 by 12 gives us 168 bits in a NB frame. Three overall frame modes are supported for the NB channel.

- In the normal mode of operation, two lines are dedicated to CRC, meaning 24-bit CRC over a frame. The remaining 144 bits comprise two 72-bit (64+ECC) data payloads. This mode fails over to the second mode.
- In the second mode of operation, one of the NB lines is down. 1 line of the remaining 13 is dedicated to CRC, meaning a 12-bit CRC over a frame. The remaining bits are allocated as above. This mode fails over to no CRC protection if another line goes down.
- In the third mode of operation, two NB lines are down. In this mode, the ECC bits in the frame are removed, and replaced with 12 CRC bits instead (which differentiates this from the failover of the second mode). The third mode of operation has no failover.

In addition to DRAM read data frames, NB frames can also be Idle, Alert, Register Data or Status frames. These are not protected by CRCs.

- Idle frames contain a permuting data pattern. They are intentionally designed to generate CRC errors at the memory controller so as not to confuse them

with Read Data frames, but the controller recognizes the permuting pattern and so ignores the Idle frame. In [15] a 12 bit Linear Feedback Shift Register (LFSR) produces the permuting data pattern through $2^{12}-1 = 4095$ unique Idle frames before the pattern is repeated (the all zero pattern is not generated). Each bit of the LFSR maps to one of 12 bit lanes (the two others are not used).

- An Alert frame contains the Boolean inverse of the LFSR data pattern. An AMB detecting a CRC error on its SB input will start and continue generating Alerts until it receives a Soft Channel Reset or a Fast Reset. More on resets and channel initialization follows in section 4.3.3.
- A Register Data Frame contains 32 bits of usable data. These 32 bits contain returned data from AMB registers.
- A Status frame is returned by an AMB in response to a Sync command from the host (Section 4.1.3). It refers to the status of the AMB regarding commands it received before the Sync. AMB errors occurring subsequent to a Sync command are not reported until the next Sync frame. Errors here are operational errors, not data transmission errors (which are dealt with by Alert frames as described above). All AMB's on the channel respond to a Sync by merging their Status bits onto the NB channel, each onto a 'dedicated' bit lane. Thus the Status frame consists of a group of status bits from each AMB. Each AMB protects its status bits with a single parity bit. Thus 12 bits are delivered into the Status frame by each AMB, 11 status and 1 parity. The most Southbound of AMBs on the channel fills its bit lane, but also fills the other

bit lanes with all zeros and an invalid parity bit for each, so that the host can detect if an AMB on the channel does not respond by filling in status bits on its designated bit lane.

4.1.5 Pins and System Board Routing

Due to the more serialized topology, fully buffered channels require a much lower pinout on the memory controller side, and on the DIMM side. In addition to the 24 differential signal pairs (48 pins), a fully buffered channel from the memory controller side will need about 6 pins for power [x], 12 for ground, 5 for the SMBus, and about 3 for clocking, making a total of around 75 pins (as contrasted with non buffered DDR2 channels which each require ~240 memory controller pins). This lowered pin count translates to large cost savings, as packaging contributes a large factor to memory controller cost due to the large pinout needed. It also allows support for multiple fully buffered channels to be implemented in the memory controller at the same cost of one non buffered channel. Finally, system board costs are also reduced as the number of connections needed is lowered and the need for path length matching is obviated. Consequently, the number of system board layers needed is also reduced.

4.1.6 Power and Heat

An issue that has to be addressed in fully buffered DIMMs is power dissipation and heat. Higher data-rates generally mean higher power consumption for signaling, although this effect can be dampened by using lower voltages. AMB's represent an added logic device to the DIMM. As all the data paths in and

out of the DIMM pass through the AMB, it is a central component, and as such will heat up the quickest. A typical AMB will burn up around 4 Watts when operating at the front or middle of a channel, and closer to 3 Watts when operating at the end of a channel [14] [27]. In contrast, a typical DDR2 DRAM device consumes just 0.3 W. It is important not to heat up DRAM devices close to the AMB, as this can affect their performance or corrupt their data. For this purpose, FB-DIMMs will feature heat spreaders clipped on to their AMBs. Figure 4.4 illustrates heat spreading on FB-DIMMs. Heat spreaders can also be seen in the middle of the FB-DIMMs pictured in figure 4.1. Table 4.1 describes excess power usage of FB-DIMMs. AMB's also contain on-chip thermal sensors, which report to the memory controller through Status frames. In case of overheating, the memory controller will throttle back the request rate to that DIMM.

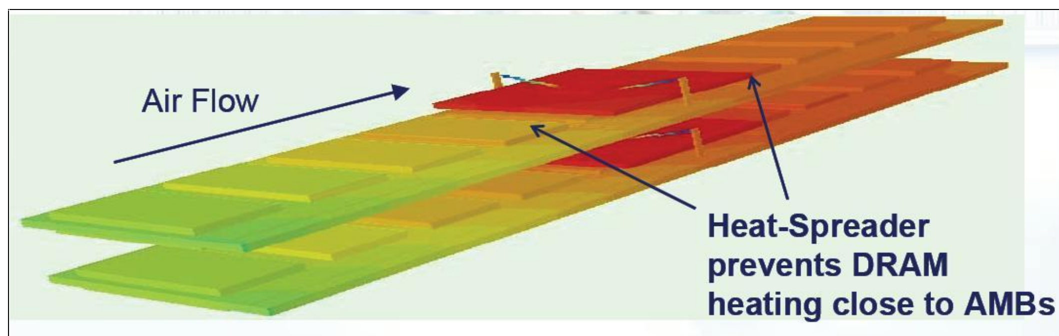


Figure 4.4 – FB-DIMM Heat Spreading [14]

No. DRAM devices	9	18	36
DIMM Org.	1rank x 8	2ranks x 8 or 1rank x 4	2 ranks x 4
DRAM power	2.7W	5.4W	10.8W
FB-DIMM power	6.7W	9.4W	14.8W
=> FB-DIMM Excess Power	148%	74%	37%

Table 4.1 – Excess Power Consumption in FB-DIMMs

4.1.7 FB-DIMM Quick Facts

- FB-DIMM in its current state is expected to support practical motherboard memory capacities of up to 192GB; that's eight 4GB FB-DIMMs in each of six channels.
- For first generation FB-DIMM implementations, sustained data throughput is expected to be around 6.7 GBps per channel.
- Data-rates on first generation FB-DIMM channels will be driven to 3.2, 4.0 or 4.8 Gbps for each differential signal pair (depending on the speed of the DDR technology used).

4.2 Serial Differential Signaling

An important component of the FB-DIMM interconnect architecture is the differential signaling used on the NB and SB channels. In differential signaling, signals are transmitted over pairs of wires. On one wire the transmitter (Tx) sends the original signal, and on the other, its inverse is simultaneously transmitted. Thus the receiving (Rx) end derives the logic HI or LO information by comparing the voltages of the two lines. This obviates the need for a shared V_{ref} signal or clean ground, along with all the engineering challenges that come along with these. Lower transmission voltages are thus supported – capital that can be either used for lower power consumption, or to support higher datarates.

Differential signaling features good common noise rejection properties; differential pairs are sent along adjacent lines so that any jitter or skew effects will

most likely hit them both, and the difference between their voltages will be conserved.

The differential signaling technology used in fully buffered channels is Low Voltage Differential Signaling (LVDS). Figure 4.5 shows the transmitter and receiver circuits.

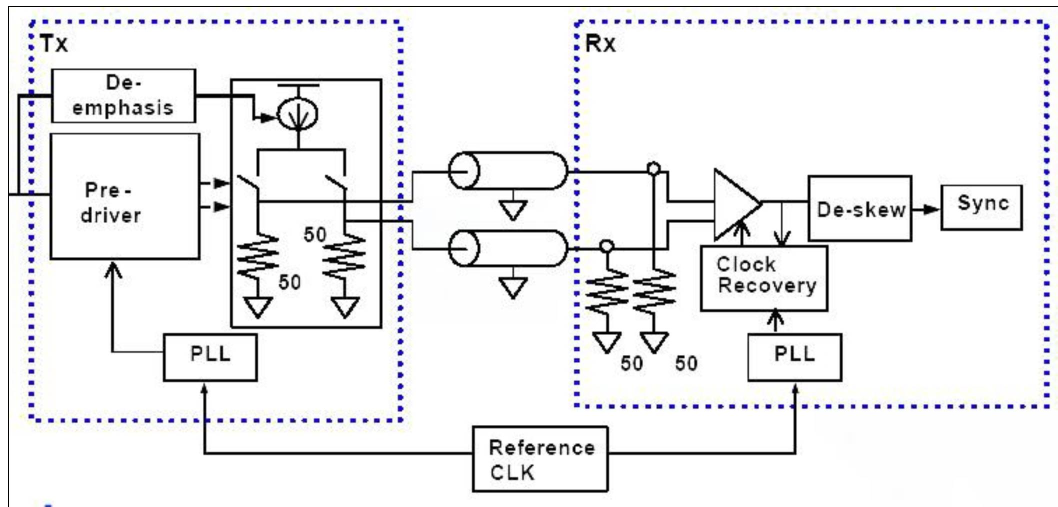


Figure 4.5 – Differential Pair Signaling (from [27])

As can be observed from figure 4.5, clocking for incoming bits at the receiver is derived from the differential signal itself, with the aid of a phase lock loop (PLL) and a slow, common reference clock running at 1/24 of the data frequency. This type of data clocking is known as mesochronous clocking, and eliminates the clock skew factor. A price to pay for this however is that the link bit stream needs to maintain a minimum transition density of 3 transitions per 256 transmitted bits to maintain phase tracking in the PLLs.

You might also notice de-emphasis circuitry on the TX side. With de-emphasis, when a voltage transition is made it is made to the maximum magnitude, but the voltage is then quickly lowered – or de-emphasized - for subsequent non-transitioning bits . This saves transmission power, and more importantly, lessens the effect of Inter Symbol Interference (ISI), or the electromagnetic interference between physically adjacent signal lines (cross-talk).

4.3 The FB-DIMM Channel

The frame protocol on the SB and NB channels has already been discussed in depth in sections 4.1.3 and 4.1.4. In this section, fully buffered channel initialization, timing, and theoretical and practical bounds are explored.

4.3.1 Channel Initialization

Figure 4.6 shows the channel initialization FSM (Finite State Machine) implemented for FB-DIMM channels. When the channel is powered up - or any time it does a soft reset – it is immediately put in the Disable state. As there are no dedicated control signals in the channel, the memory controller 'indicates' Disable mode both by its forcing both signals in each differential pair to ground (called 'electrical idle'), and by the lack of the usual periodic Sync voltage transitions (called a 'training violation') in the lines that are used to maintain phase lock. The memory controller then drives Training state (TS0) patterns onto

the SB channel. This pattern is mainly a '101010' type pattern to quickly lock on the phase trackers in the channel. Every TS0 frame however begins and ends with

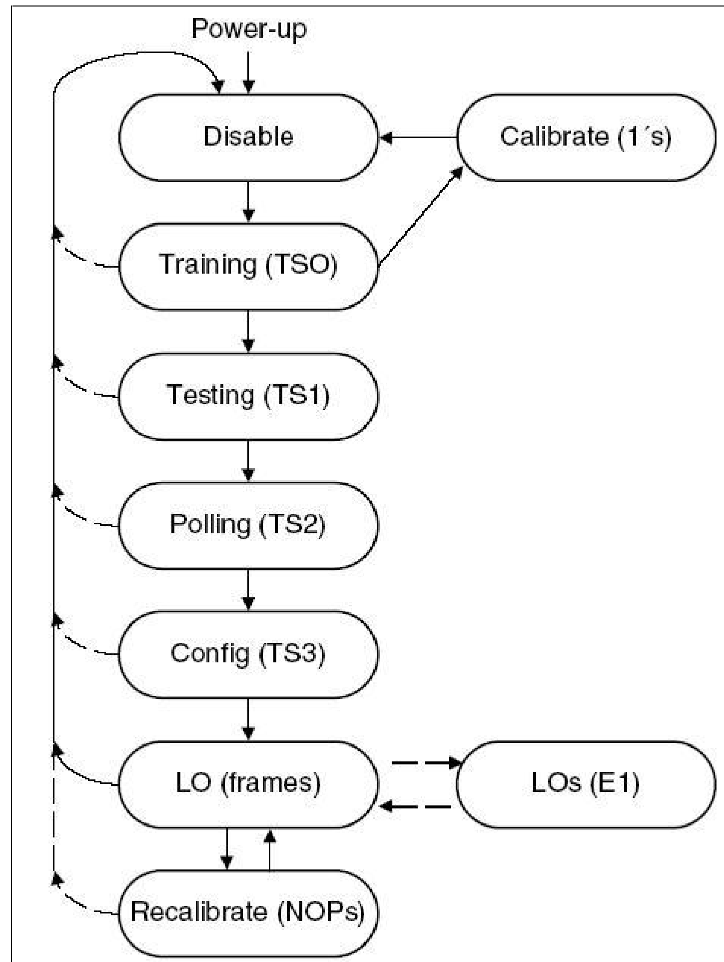


Figure 4.6 – Channel Initialization FSM (from [15])

a different pattern, and the AMBs use these to synchronize the frame boundaries, and to in turn synchronize the DRAM clocks with the frame boundaries (or some known phase shift from them). Once the memory controller receives the training pattern back on the required number of its inputs, it knows that all the AMBs in the channel are in 'bitlock' and 'framelock'. It then moves on to the Testing state.

In the Testing (TS1) state, the most electrically stressful bit patterns are sent down the channel and back along with CRCs, and the memory controller must ensure that these come back unaltered before moving on.

In the Polling (TS2) state, the memory controller sends patterns addressed to different DIMMs on the channel, and calculates the round trip latencies to these. In the Configuration (TS3) state, the memory controller is sending collected latency data and other collected channel data back to the AMBs for proper configuration.

Once configuration is complete, the memory controller sends the first Sync frame and the channel enters the normal state of operation, denoted LO. As the channel requires a minimum periodic transition density (as described in section 4.2), and as datastreams are otherwise unpredictable, the memory controller must send a Sync frame down the channel at least once every 42 frames. All AMBs on the channel subsequently reply with a joint Status frame (as described in section 4.1.4). The AMBs can reduce power and heat by switching off internal circuits during Sync frames. The Sync requirement is represented in figure 4.6 as a Recalibrate state. The LOs state in figure 4.6 is the channel low power standby mode.

To better deal with unpredicted channel issues and failure problems, FB-DIMM channel initialization and configuration is implemented with firmware.

4.3.2 Channel Timing

Utilization of the channel and the associated timing issues are solely the responsibility of the memory controller in the FB-DIMM architecture. AMB's are 'dumb' forwarding devices. If AMBs are accidentally mistimed by the controller to output data to the same NB frame, the AMB closer to the memory controller (more northerly) will simply overwrite the existing frame with its own. Thus the memory controller must still understand all the DRAM timing parameters and scheduling distances discussed in chapter 3 to avoid sending commands to DIMMs whose responses will overwrite each other, and to avoid sending resource conflicting commands to the DRAM devices. It must also factor into the equation the latency delay of the DIMMs in question.

Fully buffered channel latency is defined as the time elapsed from when a read command is driven onto the SB channel by the memory controller, and when the first frame of the read response arrives at the memory controller on the NB channel. The FB-DIMM channel can be operated in two different latency modes.

In Fixed Latency Mode, the memory controller measures the worst case DIMM latency L_w – namely, the latency of the southernmost DIMM, and for every DIMM i north of the southernmost DIMM, it measures the latency L_i of DIMM i, and sends it the delay value $L_w – L_i$. The AMB of DIMM i subsequently delays the response to any and every command by this delay. The same is done for all the DIMMs north of the southernmost DIMM. The end effect of this

technique is that to the memory controller, all DIMMs on the channel will appear to have a fixed latency, thus simplifying timing for the memory controller.

In Variable Latency Mode, the DIMM AMBs don’t implement any delays, so that to the memory controller, each has a variable latency, and the controller factors these variable latency values into its calculations to ensure no frame conflicts on the NB channel.

4.3.3 Data Rates and Peak Theoretical Throughput

As described earlier, the NB and SB high speed serial channels in the FB-DIMM architecture are clocked so that the frame rate equals the DRAM clock rate. In DDR, the data rate is double the clock frequency. Thus the frame rate of the fully buffered channel is half the DDR data rate, and as each channel frame consists of 12 bit periods, each line in the channel operates at 6 times the DDR data rate. For example, for DDR2-800, the channel operates at 800 Mbps x 6 = 4.8 Gbps.

By virtue of the fact that a NB channel frame contains two (64 or 72 bit) data payloads, and a SB channel frame contains one data payload, the peak theoretical throughput of the NB channel is equal to the peak theoretical throughput of DDR, and the peak theoretical throughput of the SB channel is half that. Thus the overall peak theoretical throughput of the fully buffered channel is 1.5 times that of a classical DDR channel.

With respect to DDR datarate, the peak theoretical throughput of DDR is the datarate multiplied by the 64 bits of the databus. So for DDR2-800, the peak theoretical throughput is 64 x 800 Mbps = 51.2 Gbps = 6.4 GBps. Thus when utilizing DDR2-800, the peak theoretical throughput of the fully buffered channel is 1.5 x 6.4 GBps = 9.6 GBps per channel.

Of course due to a myriad of factors, this throughput or anything close can never be reached. Such factors include command overhead (commands share the SB bus with Write data), minimum scheduling distances due to technology constraints and resource conflicts (see chapter 3), DRAM refresh overhead, DRAM power constraints, and fully buffered channel overhead such as channel synchronization, re-initialization and error recovery overhead, AMB power constraints, and timing issues that avoid frame conflict. The address stream of the incoming requests to the memory controller also plays a large role in determining scheduling conflicts and pipelining potential. Part of this latter factor can be amortized by the design more intelligent and reorder capable memory controllers, but of course at the cost of design complexity and price. The research in chapter 6 of this paper experimentally studies some of these limitations, and how much of the peak theoretical throughput can be achieved.

4.4 The Advanced Memory Buffer

Figure 4.7 provides a more in-depth look at the internal components of an Advanced Memory Buffer.

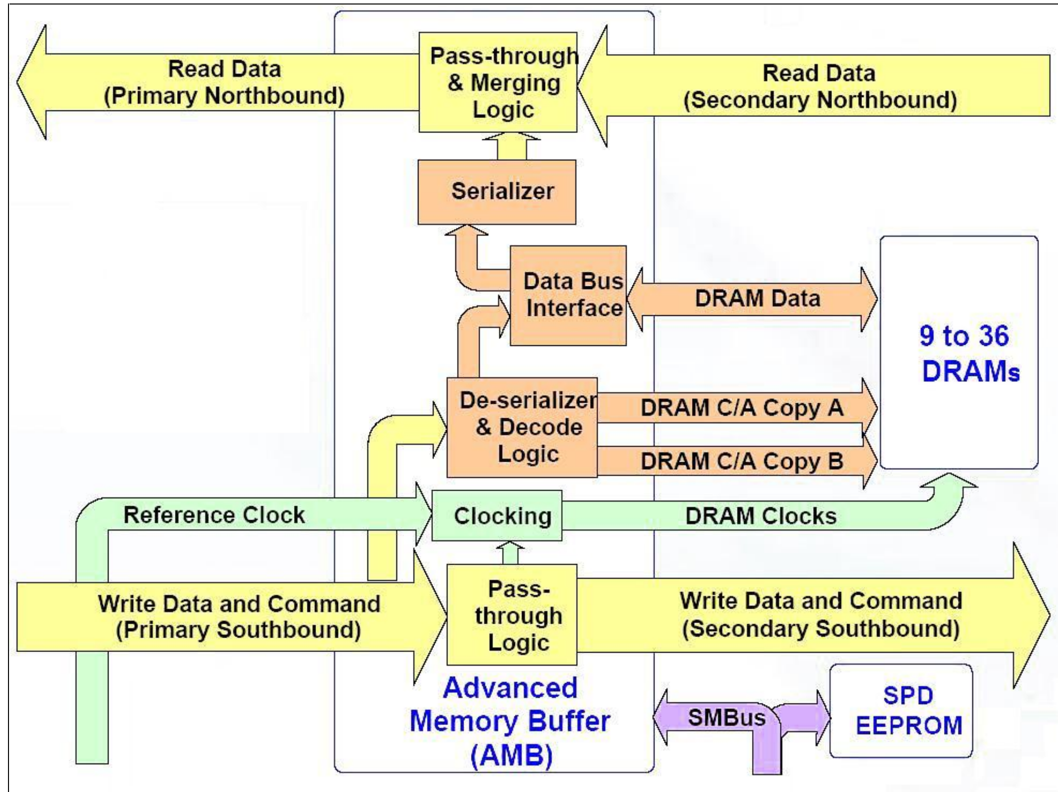


Figure 4.7 – The Advanced Memory Buffer (Intel [27])

Every AMB contains a SB input (Primary Southbound), a NB output (Primary Northbound), a SB output (Secondary Southbound), and a NB input (Secondary Northbound). The primary side connectors are on one side of the DIMM (the north side), and the secondary connectors on the other. If the DIMM is the last in its channels, its secondary side is switched off.

A critical aspect of the AMB is the super fast pass through logic between the primary and secondary sides of each channel. As described above, there are 12 bit periods in a channel frame, and it is the job of the pass through logic to pass these bits through as soon as they arrive, no matter what they contain. So even if a SB frame is addressed to DIMM i, it will still reach every DIMM j downstream (south) of i. This fast pass through logic is critical to minimize the detrimental latency effects of the serialized FB-DIMM architecture. Current generation AMB's can pass through signals with a delay of just 2 ns. The link on the other side of the pass through is totally independent physically, so that any signal degradation that is encountered on the incoming side is removed by the fresh differential transmitter on the outgoing side. This is what is meant by point-to-point connections. The total worst case round trip latency of a channel command frame (i.e. to the last DIMM on the channel through the SB lane, and back on the NB lane) is then:

$$\sum_{i=0}^{N-1} \left(2t_i + 2t_{Li}\right) + t_{DN} + t_{SN} + t_{RN}$$

N is index of the last DIMM on the channel (for example if there are 8 DIMMs, N = 7), i is the DIMM index, t_i is the pass through time on DIMM i, t_{Li} is the time needed for the signal to travel across the transmission line from DIMM (i-1) to DIMM i (if i=0, then 'DIMM (i-1)' is the memory controller), t_{DN} and t_{SN} are de-serialization (packet decode) and serialization (packet encode) AMB latencies (typically ~5ns and ~8ns), t_{RN} is the response time to the command of DIMM N.

On the Northbound channel of the AMB, in addition to passing through frames, the AMB has the ability to merge bits into passing frames, or to overwrite the passing frame with its own frame. This merge logic has no time to look at what it is overwriting, which is why the prevention of frame collisions is the timing responsibility of the memory controller.

Very simply put, the AMB's job is to decode every incoming frame by translating the incoming serialized word into a parallel one, and if that frame (or part of it) is addressed to that AMB, its command is immediately acted upon. DRAM commands are translated into the relevant DDR protocol, and the DRAM devices are interfaced through the classical parallel multi-drop bus interface. To limit loads on the internal command and address buses, there are two electrical copies of each. Clocking the DRAM devices on the DIMM is also the responsibility of the AMB, and it derives the DRAM clock from the memory system input reference clock, as described previously. Note that whether or not a particular frame is addressed to a particular AMB, that AMB will check the frame CRC and respond to any detected channel error with an Alert frame as described previously. On receiving Alert frames, the host controller retries a fixed number of times, and if Alerts continue to be received, a soft channel reset is performed and the controller attempts to reinitialize the channel (see section 4.3.1).

If a response is later generated to an incoming channel frame or DRAM command frame, the response is collected, serialized into a frame along with a computed CRC, and merged onto the NB channel towards the memory controller.

In the FB-DIMM architecture, the AMB not only acts as a serial-parallel-serial converter, but it can also be seen as a separating buffer between the channel interface domain, and the DRAM interface domain, so that changes made to one domain will not effect the design of the other. This is a very important 'separation of powers'.

4.5 Reliability, Availability, Serviceability

One of the important engineering philosophies of the FB-DIMM architecture is to provide reliability, availability, and serviceability (RAS – not to be confused with the Row Access Strobe of chapter 3!).

You will have noted from previous sections the strong CRC on every transmitted frame. This CRC is only for channel transmission, and is supplemental to the ECC code in the data payloads (assuming the DIMMs are ECC DIMMs). While the ECC protects only the data payload, the CRC protects the whole frame including the command and address information.

When errors are detected in CRCs, the FB-DIMM standard defines that the memory controller will retry multiple times, and if this is fruitless, it will attempt to reset and reinitialize the channel, all while the system is still online. A Bit Error Rate (BER) of 10^{-16} is specified for the differential pair links during normal operation. This corresponds to about one error per differential pair per link per 24 days of operation. However, due to the strong CRC protection and other

measures described in this section, the Fully Buffered system as a whole proscribes just 1 silent (undetected/corrected) error every 100 years!

Also, as described in sections 4.1.3 and 4.1.4, there is a fail over mode for the SB channel and there are two failover modes for the NB channel in case of line failure, whether transient or long term. The memory controller decides which mode to operate in on channel initialization.

If a DIMM fails, the pass through logic on its AMB will most probably still function and the channel will remain online. The memory controller will quickly notice the lack of responses from that DIMM and can, for example, trigger a system alarm. FB-DIMM supports hot adding and removal of DIMMs, so that DIMMs can be replaced while the system remains online. Of course fresh data on the removed DIMM will unavoidably be lost, unless it is first copied out. DIMMs downstream (south) of a removed DIMM will be deactivated until the removed DIMM is replaced. During a hot add or remove, the system board is responsible for removing power from the targeted DIMM socket, and in the case of hot add, the system board must notify the memory controller that a new DIMM has been added; the memory controller won't auto detect it. One means of doing this is a hard channel reset signal to the memory controller.

Every AMB has an on-chip thermal sensor, which will alert the memory controller if the AMB overheats. The memory controller will subsequently throttle back command rates to that AMB. AMBs also contain error registers

(accessible by the memory controller) that store data in case of errors in DRAM or other operations. This helps speed up fault isolation.

To support testing and debugging, AMB's feature a Logic Analyzer Interface (LAI) that allows the high speed channel contents to be monitored by a logic analyzer without disrupting its contents. AMBs also support 'transparent mode', which allows external testers to directly communicate with the DRAM devices. Information in this mode is directly passed back and forth through the AMB (between the DRAMs and the outside world) without the usual serial-parallel-serial conversions. The differential signal inputs and outputs in this mode are reconfigured into single ended low speed receivers and transmitters.

4.6 Summary; Pros and Cons

Table 4.2 summarizes the major pros and cons of the new first generation FB-DIMM architecture when compared to older memory system organizations.

PROS
Capacity. Supports up to 8 DIMMs (or 32GB) per channel. Up to 6 channels are supported.
Throughput. Separate independent read and write channels yield a peak theoretical throughput 1.5 times that of the old DRAM interface.
Turnaround Hiding. Consecutive Read/Write, Write/Read, and any other turnaround times related to data strobing are removed when the consecutive operations are addressed to different DIMMs on a channel. In fact, other than preventing frame conflicts on the NB bus, there is no minimum scheduling distance between commands addressed to different DIMMS. This is due to the 'buffer' effect between DIMMs.

Table 4.2 – Pros and Cons of the FB-DIMM Architecture

Scalability. The old DRAM interface can scale much better with the small signal paths on a DIMM, and without all the signal discontinuities of the system board. The memory controller to DIMM interface can also easily scale well due to the adoption of point-to-point buffering and high speed serial differential signaling. The DIMM interface is also much narrower now, meaning fewer memory controller pins per channel and lower system board costs per channel, which also helps scalability.
Narrow Channels. Seen from a high level, a narrow channel means longer data burst lengths to achieve the same data transfer, which in turn means less command overhead with respect to the overall channel usage. Put simply: better per line efficiency.
Cost and compatibility. FB-DIMM technology is compatible with current DRAM devices. It does not necessitate drastic change in future DRAM device implementations beyond their current evolutionary path.
Support. Due in part to the above fact, almost all major manufacturers in the memory industry have rallied around this technology. Unlike Rambus technologies, the FB-DIMM standard is an open, collaborative endeavor.
CONS
Latency. Buffering, along with support for deeper channels, adds a lot of latency to the memory channel. However, some or all of this added latency is hidden due to the removal of some turnaround times as well as the 1.5x throughput of the channel and independent reading and writing across DIMMs. Thus this latency difference will only hit you at lighter loads, where latency is generally not so important anyway. In fact, at heavier loads the latency on an FB-DIMM channel can be much less than that of an old DRAM channel under the same load. Chapter 6 of this paper explores experimentally load/latency relationships in FB-DIMM.
Price. FB-DIMMs will of course be more expensive than normal DIMMs due to the added AMB and heat spreader. New memory controllers will also have to be developed. However, some or possibly all of these price increases will be offset by the decreasing cost of the memory controller packaging due to less pins, and the decreased per channel cost of system board design and fabrication. Any step towards a new technology will be expensive however, and FB-DIMM - for the reasons described in this paper – has shown itself to be a very cost effective solution.

Table 4.2 – Cont'd

4.7 Current Industry Adoption and Future Projections

FB-DIMMs are already under production and are slated to enter the server memory market in the final quarter of 2005, and the workstation memory market soon thereafter. Test FB-DIMMs released at the end of 2004 have proven that the technology is implementable, and that the standard works. Figure 1.1 in chapter 1 shows industry projections for the FB-DIMM share in the memory module market. Industry is touting FB-DIMM technology as a long term strategic direction, and almost all the arguments I've encountered and created in this paper show this to be possible and likely.

Like any new technology transition, FB-DIMM is a trade-off with pros and cons, and will start out with raised prices, but in this case it seems the benefits strongly outweigh the costs, and the transition is long overdue. In terms of price, manufacturers are suggesting that FB-DIMMs will only have a small premium over non buffered modules. The industry politics have fallen into place, and it seems that FB-DIMM is very likely to become a success in the long term.

Table 4.3 lists some of the confirmed industry names behind the different components of the FB-DIMM system at the time of the writing of this paper.

Technology Component	Manufacturers
FB-DIMMs	Elpida, Hynix, Infineon, Kingston, Micron, Nanya, Samsung, Texas Instr.
AMBs	ICS, IDT, Infineon, NEC
Compatible Memory Controllers	Intel
Clock and Timing Devices	ICS

Table 4.3 – Some FB-DIMM Technology Manufacturers

CHAPTER 5: FBSIM SIMULATION MODEL AND IMPLEMENTATION

To study the inherent strengths and limitations of the Fully Buffered architecture, I have programmed a simulator called FBsim. This chapter gives an overview of the simulation model adopted – its objective and any assumptions made, the resulting program and its input and output specifications, and some of the more interesting implementation details such as the input transaction generation model, the address mapping algorithm, and the all important channel scheduler.

5.1 Simulator and Simulation Model Overview

FBsim is a standalone product, and does not interface with CPU or other simulators at the time of this writing. It supports four main simulation modes; fixed channel latency closed page mode, fixed channel latency open page mode, variable channel latency closed page mode and fixed channel latency open page mode. Open and closed page modes (described in section 3.3) each have their own address mapping mode (section 5.3). Fixed and variable channel latency modes are described in section 4.3.2.

This section will give you a better idea of what the simulator is studying, and what it does.

5.1.1 Objective of Study

The main objective of this simulator is to study read latency vs. throughput channel characteristics in the Fully Buffered architecture, in the four main modes described above. In my model, read latency is defined as the time (in nanoseconds) from when a read transaction reaches the memory controller, to the time the first frame of the read response is received by the memory controller on the NB channel (further frames will follow consecutively).

Due to the command overhead, inter-transaction scheduling distances due to resource conflicts in the DRAM devices, pipelining pattern 'bubbles', and error/synchronization channel overhead (all described earlier in this paper), a fully buffered channel will never be able to reach its peak theoretical throughput of 1.5 x the peak theoretical throughput of a DRAM device, just as the old parallel bus interconnect architecture was never able to meet (1 x) this throughput. As input transaction frequency to the memory controller is ramped up, at some point the fully buffered channel will saturate, and transactions will beginning queuing inside the memory controller scheduler. As transactions queue, the latency of read transactions starts growing rapidly. If the input request rate is maintained or even increased, this queue is likely to keep growing, and read latency will shoot up asymptotically.

Amongst other things, FBsim can help study why this saturation happens in the Fully Buffered architecture, and compare the saturation points of the four modes described above for different DIMM/channel configurations.

5.1.2 Simulation Model - Assumptions and Abstractions

Perhaps the most consequential or controversial abstraction made by my simulator is that it does not get memory requests from a CPU or other simulator, it generates its own 'imaginary' input (see section 5.2 for details). This approach has many strengths and weaknesses, and perhaps a whole thesis could be written on memory transaction simulation models. But to summarize, the main advantage of this approach is speed and simplicity. CPU simulators, although excellent in their own rites, are slow and very complex to interface with, and slapping a memory simulator on top will make things even slower.

Another approach would be to use memory traces as input to a memory simulator. These traces can be taken from a CPU simulator, or a program can be written to generate a trace straight out of a running application. Information to be saved for every memory transaction is the transaction's type (read or write), its physical address, and its time of issue. However, in the fully buffered architecture (which is primarily being designed for servers), multiple gigabytes/second of throughput is possible, and such high throughputs are required to saturation the channels. Thus one would have to find benchmarks able to generate such a massive transaction rate, and to simulate one second in the memory system, trace files would have to be in the gigabyte range. This huge file access would also drastically slow the simulator.

The fact is things would be orders of magnitude faster if the memory simulator can dynamically generate its own input transaction on the fly. This

would also allow the simulator to easily vary its input generation rate to study saturation effects on the channels. Future high-memory-bandwidth applications can be modeled in this way. Different channel configurations can be studied, and the conclusions reached will in all probability apply to most real applications too. It is true that request stream locality (which is application dependent) is a primary factor in memory latency, but it can be countered that a lot of locality relationships are lost and randomized when requests coming from multiple processing cores are being simultaneously received and interleaved. Furthermore, in the Fully Buffered architecture, each DIMM operates independently so that there is no minimum scheduling distance to be considered between requests addressed to different DIMMs on a channel other than that of sharing the SB and NB channels. This means that if the DIMMs can be load balanced well by the address mapping function (as I attempt in section 5.3), locality in the request stream becomes less of a factor, and inter-DIMM channel contention becomes a much larger factor.

As a final point on this matter, an excellent memory system simulator that interfaces with CPU simulators to study real applications already exists. It is called Sim-DRAM, and was developed at the University of Maryland [32]. Sim-DRAM can be used to study the response of the Fully Buffered architecture to real applications at lower request rates, and to compare these with the older parallel architecture, while FBsim can be used more to study future applications

and saturation points of the channels, and to compare different channel configurations. The two simulators can also be used to validate one another.

Since there is no real application running in FBsim waiting for a memory response, the data or bits stored in the memory are meaningless to the model, and absolutely no benefit is gained out of keeping track of them. Thus FBsim does not actually store and retrieve memory data, but rather simulates transactions to different memory locations and how they interact. This also makes the simulator much faster, and stops it from constantly paging to disk.

As you will see in section 5.4, FBsim uses quite a complex scheduling algorithm to attempt to get maximum utilization of the channels and minimize average latency. Hardware implementations of scheduling algorithms in an actual memory controller are of course severely limited by timing, area, and power constraints. The same argument applies for the address mapping algorithm described in section 5.3, albeit to a lesser extent. Again, a whole thesis could be written on hardware implementations of memory scheduling; it is beyond the scope of this paper. FBsim is a 'good' scheduler that tries to get the best possible channel utilization to analyze saturation limitations inherent to the Fully Buffered interconnect architecture, as opposed to the memory controller itself.

FBsim at this point does not simulate: channel initialization, channel frames (sync, status, AMB configuration register writes and reads), channel CRC errors and retries, DIMM power dissipation and throttling down of command rates due to AMB heating, DRAM power draw limitations (eg. t_{FAW}), and DRAM

refresh. At the time of this writing, insufficient information on most of these effects is available.

Despite the limitations of FBsim described above, it is a complete enough model to generate many interesting studies and results, as the rest of this paper will hopefully prove. There are already plenty of parameters and variables to study as you will see in sections 5.1.3 and 5.1.4. Furthermore, FBsim is quite simple and modular in construction, and future additions can be made to simulate further parameters.

5.1.3 Input Specifications

The inputs to the FBsim model and what they mean are very important to understanding the simulator and what it does. Inputs to FBsim are provided through two text files, a .sim file for simulation parameters, and a .dis file for input transaction generation parameters.

An important definition for his simulator is that of a **tick**. A tick in my simulator is defined as the frame period in the fully buffered channel, which is equal to the period of the DRAM command clock on the DIMMs, which is equal to one half of the DRAM device datarates.

Table 5.1 describes the simulation parameters in the .sim file at the date of this writing, and table 5.2 does likewise for the .dis file.

pageMode (int)	0 for Closed Page Mode, 1 for Open page mode.
DDRdataRate(int)	The DDR data rate in Mbps. Ex. for DDR2-800, it would be 800. The channel frame rate and the DRAM command cycle rate in the simulator will be set to one half of this frequency.
simtime (float)	Desired simulation time (in seconds, but can be a fraction)
theSchedulerWindow (int)	Maximum number of input transactions that map to the same channel that can be simultaneously considered for scheduling at each tick.
queueSize (int)	Maximum queue size for each channel for channel mapped transactions. If a transaction mapped to a certain channel cannot fit in that channels scheduler window (i.e. its full), the transaction enters the channel queue. A queue overflow halts the simulation. To best study the area of channel saturation, this queue size should be specified very high (ex. 500000). However, if the program runs out of virtual memory due to all the queued transactions, it will also halt.
howPatient (int)	Transactions entering a channel scheduler are given this much patience. Patience decrements by one at every tick. When a transaction runs out of patience, any transactions that came after it cannot be considered by the scheduler until it is scheduled.
debugMode (int)	0 for no debug, 1 for debug mode. Debug mode is discussed in section 5.1.4, and in chapter 6.
commenceDebug (float)	If the debug mode is 1, this is the time (in seconds) during the simulation at which the simulator will start outputting debug data.
numberDebugTicks (int)	If the debug mode is 1, this is the number of ticks for subsequent to the commenceDebug time that the simulator will output debug data. Sane values for this are 1,000 to 10,000.
randomSeed (int)	Input transaction generation uses a random number generator to simulate input probabilities. This number seeds that generator, so that keeping it fixed will generate the exact same input transactions over different simulator runs. Setting this to 0 means that the time( ) function is used to ‘randomly’ seed the random generator.
latencyMode (int)	0 for Fixed Latency Mode, 1 for Variable Latency Mode.

Table 5.1 - .sim Input File Specifications

Per DIMM Specifications (encapsulated in [] brackets)	
PTL (float)	Pass through latency (in nanoseconds) of the AMB.
channel (int)	Channel to insert DIMM on. The first DIMM spec given for a given channel will become 1st in its channel (DIMM 0), the second will become DIMM 1, etc.
nRanks (int)	Number of DRAM ranks on DIMM.
nBanks (int)	Number of banks per DRAM.
nRows (int)	Number of rows per bank.
nColumns (int)	Number of columns per bank.
colWidth (int)	Column width (data width of the DRAM).
deSerialLat (float)	Latency (in nanoseconds) of the AMB deserialization circuitry that decodes SB frames.
serialLat (float)	Latency (in nanoseconds) of the AMB serialization circuitry that packs NB frames.
tCAS (int)	**Timing parameters for the DRAM devices.** Note that in FBsim, t_{CMD} is always equal to 1 (see 1n command timing, section 4.1.3), and that t_{BURST} is always equal to 4, which yields a burst length of 8. Eight 64-bit data bursts equals 64 bytes, or the width of an average cacheline. *All transactions in Fbsim are 64 byte transactions.*
tCWD (int)	
tDQS (int)	
tRAS (int)	
tRC (int)	
tRCD (int)	
tRP (int)	
tWR (int)	
System Board Specifications	
0DIMMLatency (float)	Signal transmission latency from mem controller to 1st AMB on channel (AMB 0). The next variables only matter if those DIMMs actually exist on the channel. Board latency specs are replicated for all channels.
1DIMMLatency (float)	Signal transmission latency (ns) from DIMM 0 to 1
2DIMMLatency (float)	Signal transmission latency from DIMM 1 to 2
3DIMMLatency (float)	Signal transmission latency from DIMM 2 to 3
4DIMMLatency (float)	Signal transmission latency from DIMM 3 to 4
5DIMMLatency (float)	Signal transmission latency from DIMM 4 to 5
6DIMMLatency (float)	Signal transmission latency from DIMM 5 to 6
0DIMMLatency (float)	Signal transmission latency from DIMM 6 to 7

Table 5.1 (cont'd)

n (int)	Number of transaction generating distributions
Per Distribution Specifications (encapsulated in [] brackets)	
Type (int)	Distribution type. 0 is a step distribution, 1 is a normal (Gaussian) distribution. See section 5.2.
left (float)	The time the distribution begins (in seconds), which is the left cutoff
right (float)	The time the distribution ends (in seconds), which is the right cutoff
alpha (float)	The probability amplitude of the distribution if step, and the peak probability amplitude of the distribution if normal.
readFraction (float)	The fraction of transactions generated by this distribution that will be reads. Between 0 and 1.
mean (float)	If step distribution, this is left blank. If normal, this is the time mean of the distribution (in seconds), namely, the time that will create the peak probability amplitude alpha. Must be between left and right cutoff times.
sigma (float)	If step distribution, this is left blank. If normal, the standard deviation of the probability amplitude distribution about the mean (in seconds)
locMean (int)	If step distribution, this is left blank (step distributions are defined to yield perfect sequential locality). If normal, this is the mean of the locality probability distribution. See section 5.2.
locRange (int)	If step distribution, this is left blank. If normal, this is the cutoff range about the mean (symmetric) of the locality probability distribution.
locSigma (int)	If step distribution, this is left blank. If normal, this is the standard deviation about the mean of the locality probability distribution.

Table 5.2 - .dis Input File Specifications (see Section 5.2)

5.1.4 Output Specifications and Program Overview

FBsim is written in ANSI C, input parameters are just text in text files, and all outputs appear in text files too. As such, it should be able to compile and run on any platform with minimal or no difficulty. The command line to run FBsim is

simply: FBsim <simulationFile>.sim <distributionFile>.dis . FBsim runs quite fast, with at most 1 second needed (per channel) to simulate 1 ms of time under saturation conditions on a 2.4 GHz Pentium 4 running Win XP. Pre channel saturation, the simulator runs ten to fifty times faster. FBsim occupies about 4500 lines of code, over one .C file, and 4 .H header files. The simulator is basically divided as follows:

- FBsim.c : main() function. Calls functions from the headers to parse in the inputs and initialize the simulation. Contains the main simulator loop, where each iteration is done over one channel frame (tick). Calls the transaction generator at the beginning of a tick to probabilistically generate transactions, and then calls the scheduler three times for each channel (once for each SB frame third). If it is dumptime, the proper functions will be called to dump generated data to the output files and reset the corresponding variables.
- FBDIMM.h : Contains structs pertaining to an FBDIMM. Parses the .sim file to generate, intialize and organize the specified DIMMs.
- Trans.h : Contains structs pertaining to distributions and generated transactions. Parses the specified distributions from the .dis file, and organizes them into data structures. Contains function to generate transactions based on the distribution data. See section 5.2. Saves generated transaction data to arrays, and contains functions to dump these to the trans.txt output file.

- Mapping.h : Contains the mapping function that takes in a physical address, and derives the transaction's channel, DIMM, rank, bank, row and column Ids from it. There are separate mapping functions for CPM and OPM.
- MemC.h : The memory controller. This is the biggest code file. Contains functions and data structures that generate minimum scheduling distance arrays from Table 3.2 for each DIMM (timing parameters can vary by DIMM). Contains two scheduling functions, one for CPM and one for OPM. See section 5.4 for more on the schedulers. Scheduling functions generate and save throughput, latency, queue-size and scheduling rejection data, and tick-by-tick state data for channel 0 when in debug mode. Contains dump functions to dump this generated data to the output.txt, qsize.txt, rejects.txt and channels.txt output files.

The FBsim output files are designed to be copied and pasted into Microsoft Excel spreadsheets, or any spreadsheet software that you can paste text into, and that automatically separates the pasted text into columns based only on tab ('\t') characters, and rows based only on new-line ('\n') characters. Separating data manually into rows and columns would be an extremely arduous task due to the large quantities of data output. A template Excel spreadsheet (template.xls) has been created into which you simply paste the contents of the trans.txt, output.txt rejects.txt and qsize.txt files generated by FBsim, and the following charts automatically appear:

- Overall input transaction rate vs. time (separated into reads and writes).
- SB channel write data throughput distribution over time per channel and for all channels
- NB channel read data throughput distribution over time per channel and for all channels
- Total (SB plus NB) system throughput distribution over time per channel and for all channels
- Av. read latency time distribution per channel and for all channels
- Av. read latency vs. total system throughput datapoints plotted against each other (forming a latency/throughput curve)
- Av. queue size distribution over time for the scheduler queues of all channels
- Av. schedule window size over time for the scheduler windows of all channels
- Time distribution of scheduling rejection reasons (as percentages of the overall scheduling rejections)
- Average number of impatience stalls distribution over time

The charts are each made up of 200 data points. FBsim divides the total simulation time specified into 200 equal ‘dump segments’. At the end of each dump segment, accumulated simulation data is output to the output text files. This limits file access by the program, as well as the output file sizes.

The only exception to the ‘dump segments’ is debug mode output, which saves and outputs to channels.txt the SB and NB channel states as well as the

scheduler state (only for channel 0) for *every tick* in the debug interval you specified. A template Excel spreadsheet (debug.xls) has been created into which you paste the contents of channels.txt. Debug.xls will visualize the activity on the NB and SB channels.

5.2 Input Transaction Generation Model

In FBsim, input transactions are probabilistically generated according to user defined transaction-rate/time distributions. As of this writing, two types of distributions are supported, step distributions, and normal (Gaussian) distributions.

Step distributions are intended to simulate writing or reading large consecutive chunks of data to or from memory. This would happen for example when transferring virtual memory pages back and forth between memory and hard drive(s), in LAN activity, in any other DMA style I/O such as between graphics card and memory, and perhaps during context switching if the CPU wishes to push or pop L2 cache to/from a stack. For a step distribution, only the first five distribution parameters of table 5.2 are supplied, namely, the distribution type (1), starting time (left cutoff), ending time (right cutoff), alpha, and readFraction. readFraction can only be 0 (all writes) or 1 (all reads) for a step distribution. As for alpha, it is the amplitude, or ‘height’ of the step. Alpha is given as a fraction (between 0 and 1) of the peak theoretical throughput of the system.

The peak theoretical throughput of a channel was given in section 4.3.3 to be 1.5 times that of DDR, which means 1.5 x DDR datarate x 64 bits per second. To convert to transactions per second, you divide this by 64 x 8, as every transaction in FBsim is 64 bytes. Thus it becomes (3 x DDRdatarate) / 16 transactions per second per channel. Knowing that a tick (frame) period is half of the DDR datarate period, this becomes 3/8 transactions per tick per channel, or 3C/8 transactions per tick overall, where C is the number of channels with DIMMs on them.

Thus, the probability to generate a transaction from a step distribution in a given tick is $\frac{3C\alpha}{8}$. The first transaction generated from a step distribution will be given a random address, and all consecutive transactions from that distribution will be sequential in address (going up in increments of 64, as a transaction writes or reads 64 bytes of memory). The transactions will either all be reads, or all writes according to the readFraction given.

A long step distribution represents a device with a constant memory request rate, whereas very narrow step distribution can be used to simulate 'spikes' in the request rate.

Normal distributions (Gaussian) are a way to simulate more random memory behavior - such as that of programs – whose overall request rates rise and then ebb. Normal distributions are specified by all ten distribution parameters listed in table 5.2. A 'type' of 1 says that the distribution is normal. The 'mean' is

the time at which the distribution's request rate is at a peak, where the peak, alpha, is defined as a fraction of the system's maximum theoretical request rate. Sigma, the standard deviation, is specified to define how narrow (spiky and steep) or wide (blunt) the distribution will be. The time specified by the 'left' cutoff parameter is the time at which that distribution begins to be considered, and the 'right' cutoff parameter is the time at which it is considered no more. This can be looked at as clipping the distribution on each side (not necessarily symmetrically).

Thus the chance of generating a transaction from a normal distribution at each tick t is:

$$chance\ per\ tick = \begin{cases} 0 & ;\ t < left \\ \left(\frac{3C\alpha}{8}\right)\left(\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(t-\mu)^2}{2\sigma^2}}\right) & ;\ left < t < right \\ 0 & ;\ t > right \end{cases}$$

$$(\ \mu \text{ is mean, } \sigma \text{ is standard deviation}\ )$$

Next is the issue of locality. Locality in FBsim is defined as the 'number of consecutive transactions generated whose addresses are incrementing 64 bytes at a time', i.e. adjacent in memory. When a transaction is first generated from a normal distribution, it is randomly assigned a 'locality count' which determines how many consecutive transactions generated by that distribution will by local to each other. This is called a 'locality burst'. After the locality count is exhausted, a new locality count is randomly generated for the next transaction. The locality count is drawn from a normal distribution of its own, that is defined (in positive

integers) by locMean and locStdDev. locRange is a range centered at the mean, and provides the left and right cutoffs. Thus locality count is given by a generated random variable of normal probability distribution. Programs with good locality are simulated by a high locality mean, and programs with bad locality are simulated b a low locality mean. Programs with regular patterns of locality are simulated with a low standard deviation and range, while programs with irregular patterns of locality are simulated with high standard deviation and range. Finally, at the generation of each locality burst, the chance that the locality burst will be all reads or all writes is determined by the readFraction variable of the normal distribution.

FBsim allows distributions to intersect (as many as the user wishes, and the probability that each provides stacks) provided that the sum of all their alphas never exceeds 1. When FBsim is under multiple distributions, it interleaves generated transaction data (i.e. address, read/write, etc.) between those distributions according to the contribution percentage of each to the total current probability

Using these two basic tools, step distributions and normal distributions, the user can shape the input transaction frequency over time of their simulated application. The generated transaction rates are output by FBsim into trans.txt, so that the user can validate this shape. Real program transaction rates can be obtained from bus trace graphing programs, such as the Bus Trace Viewer by David Wang [18]. Figure 5.1 shows the memory trace of the SETI@home

application, and Figure 5.2 shows the input transaction rate graph generated by FBsim after I copied the first 30ms of SETI@home with step distributions.

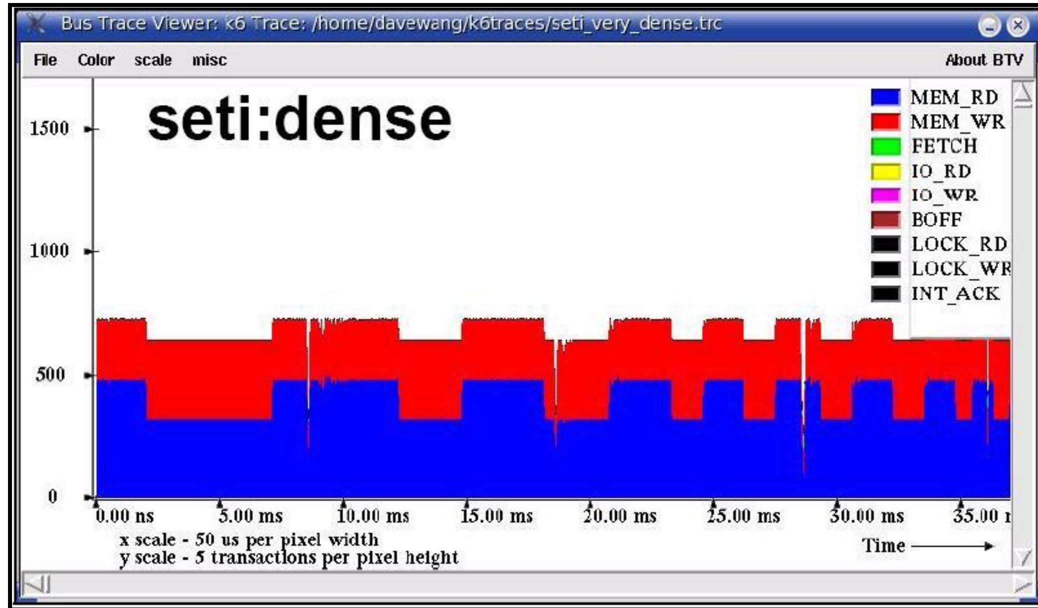


Figure 5.1 – SETI@home in Bus Trace Viewer (from [18])

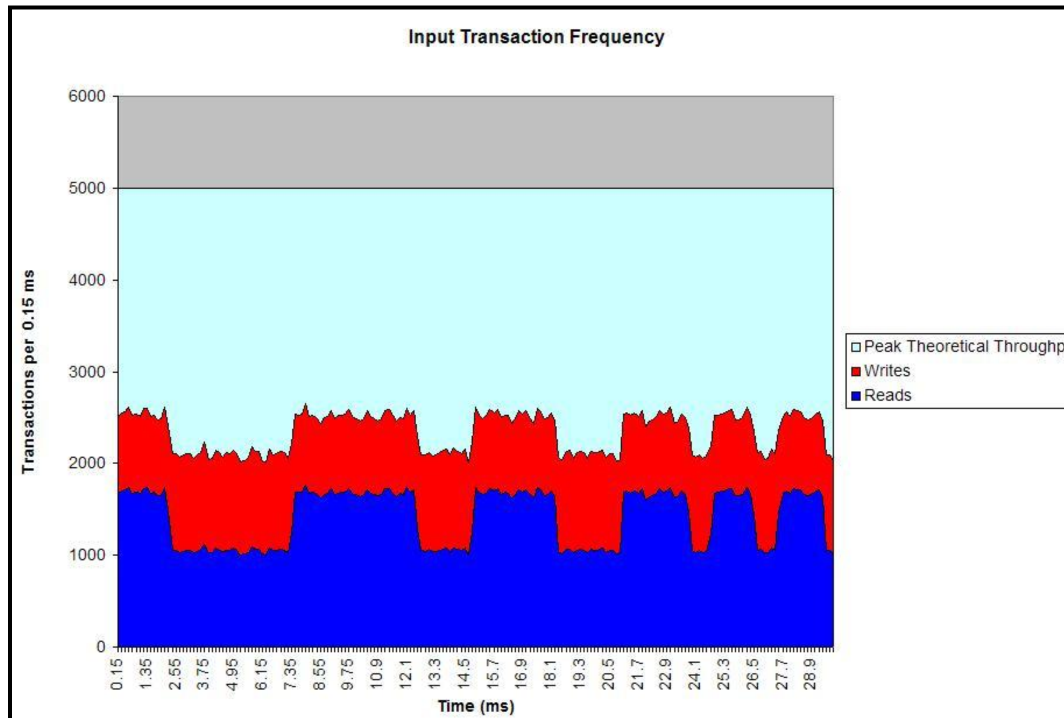


Figure 5.2 – SETI@home Simulated in FBsim

More difficult to copy would be the spikier vortex benchmark of figure 5.3, although this would be doable. And finally, in figure 5.4 is a normal distribution generated by FBsim, with a readFraction of 0.66, and an alpha of 0.5.

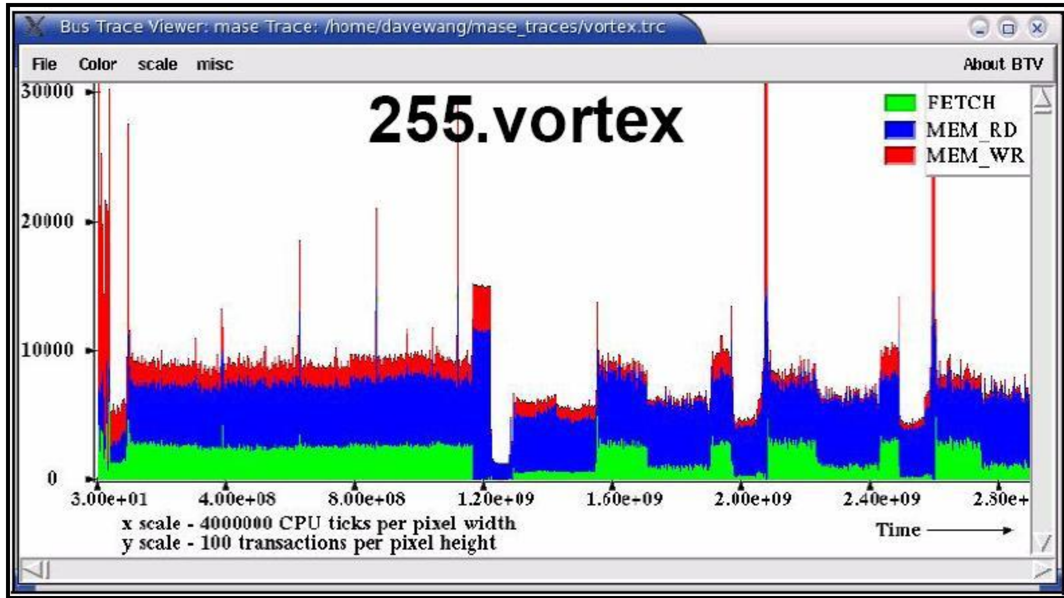


Figure 5.3 – Vortex Benchmark in Bus Trace Viewer (from [18])

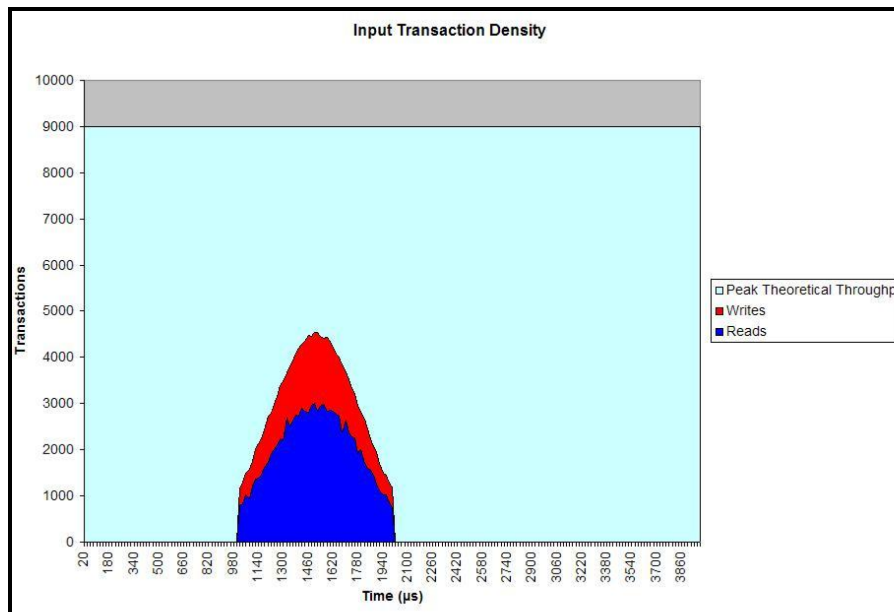


Figure 5.4 – Example Normal Distribution Transaction Rates in FBsim

5.3 Address Mapping Algorithm

A critical function of any memory controller is its memory mapping algorithm. With multiple channels, you want to balance incoming requests across the channel. With open page mode, you want to map adjacent addresses to the same row address to minimize row conflicts. In the FB-DIMM architecture, since DIMMs are buffered from each other, you also want to load balance the DIMMs on a channel, because although they share the same channel, transactions to different DIMMs do not conflict with each other, whereas transactions to the same DIMM must obey minimum scheduling distances due to resource sharing.

The first issue to consider in FB-DIMM is that DIMM capacities in terms of memory size are not necessarily homogenous, and so the same follows for channels. You can arrange DIMMs of the same or different memory capacities in any way you like. For example, in Fig 5.5, DIMM A on channel 0 holds 4 GB, DIMM B on channel 1 holds 1 GB, DIMM C on channel 1 holds 2 GB, and DIMM D on channel 2 holds 2 GB. The total memory available is 9 GB. One idea is to simply partition the 9 giga address space into contiguous blocks, with each block mapping to a DIMM, but this scheme would of course yield very poor channel load balancing as most or all of a program or DMA's memory requests would map to the same block. A much better idea is to map out the channel IDs to the finest granularity (which in FBsim is 64 memory locations for each 64 byte transaction), meaning that address x will map to one channel, address x+64 will map to another, etc. This would yield ideal channel balancing for request streams

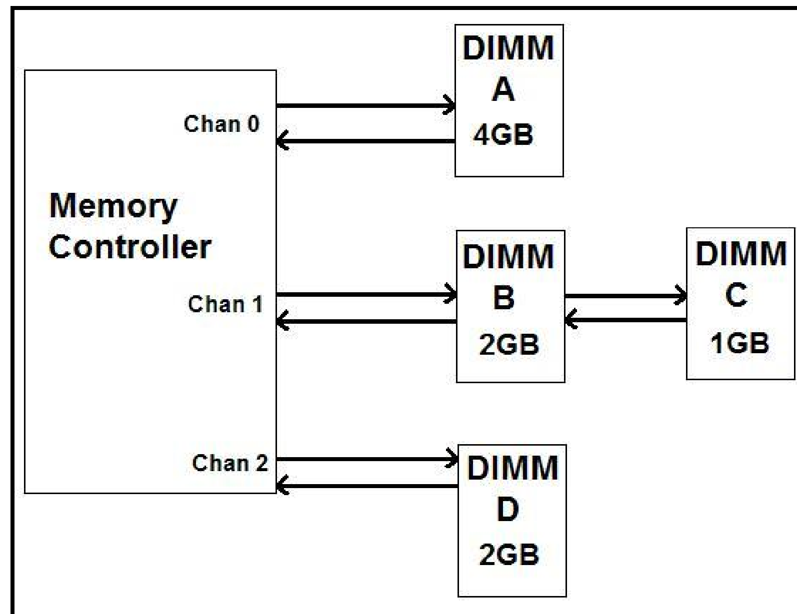


Figure 5.5 – Example Configuration

with perfect locality and, on the average, also for request streams with no locality (random). But it must be taken into consideration that the channels may not be balanced in terms of memory capacity, as in figure 5.5. The following is my algorithm for mapping physical address to channel ID and DIMM ID:

- Find the DIMM of least capacity. This is the lowest common denominator. In our case, it is the 1 GB DIMM.
- Sum the capacities of all the DIMMs, and divide by the least common denominator. Let the value yielded be called the **map modulus**. In our example, map modulus = 9.
- Translate each DIMM capacity into a factor of the smallest DIMM's capacity (i.e. divide by the smallest DIMM's capacity). Form these into a

matrix, and add a column to the right of the matrix that keeps the row sum.

For our example, we get:

$$\begin{array}{cc:c} & & \Sigma \\ \end{array}\left[\begin{array}{cc:c} 4(A) & & 4 \\ 1(B) & 2(C) & 3 \\ 2(D) & & 2 \end{array}\right]$$

- WHILE (a non zero row sum exists)

  {

  WHILE (visit each channel with a non zero row sum exactly once)

  {

  The next 'result' is channel DIMM with the highest number.

  Decrement that DIMM's number by 1.

  Decrement the row sum by 1.

  }

  }

- Table 5.3 shows the result of running the algorithm on our example.

Result 0 is	DIMM A (chan 0)
Result 1 is	DIMM C (chan 1)
Result 2 is	DIMM D (chan 2)
Result 3 is	DIMM A (chan 0)
Result 4 is	DIMM B (chan 1)
Result 5 is	DIMM D (chan 2)
Result 6 is	DIMM A (chan 0)
Result 7 is	DIMM C (chan 1)
Result 8 is	DIMM A (chan 0)

Table 5.3 – Mapping Algorithm Output (map modulus is 9)

Consider any memory address. The first step is to shift the address 3 bits to the right because the 3 least significant bits are unusable (due to the data bus width of 8 bytes…; 2^3=8). Shift again 3 bits to the right, but these bits are saved as the low order column ID. These 6 bits together account for the 64 memory address transaction granularity (2^6 = 64) in FBsim. Let 'address' = the resulting address after the six right shifts. To find out which DIMM it maps to, let x = 'address' **modulo** 'map modulus'. Find the table entry for 'result x' and it will tell you what DIMM and channel to map that transaction to. The result of using this method is that your memory space will be divided as in figure 5.6.

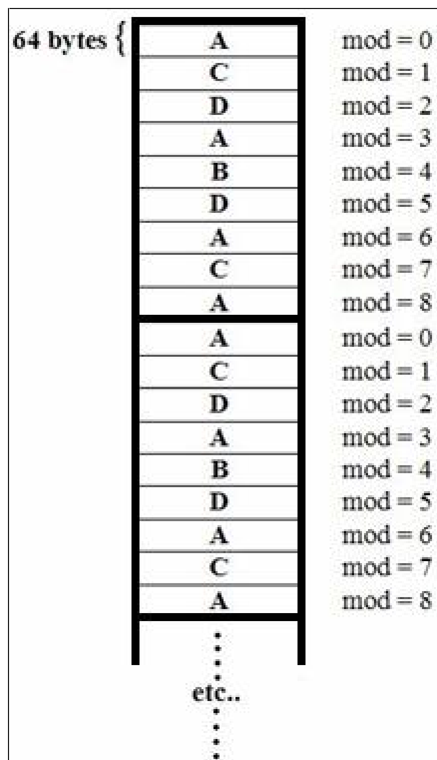


Figure 5.6 – Memory Space Division amongst DIMMs

FBsim contains some small improvements in this algorithm to ensure best case mapping for matched channels, and to prevent adjacent copies of a DIMM on the extrema if possible. For example, in our case DIMM A appears at result 0 and result 8 – these are adjacent. This could be prevented by switching result 0 (A) with result 1 (C).

Now that channel and DIMM ID have been obtained from the address, the modulus result is subtracted from the address, and it is divided by 'map modulus' effectively shifting the bits right (because this information has been used). The modulus result is now added back on. The result up till now specifies the memory address within that DIMM. Through alternating sequences of integer modulus and division operations (where the remaining address is the left operand and the right operand is based on the DIMM data, i.e. total number of ranks, banks, etc.), the remaining rank, bank, row and column data is extracted (note that a division by n is just a right shift by lg(n) bits, and all the DIMM parameters just listed are powers of 2). For closed page mapping, the data is extracted as in figure 5.7 (from right to left), and for open page mapping, as in figure 5.8.

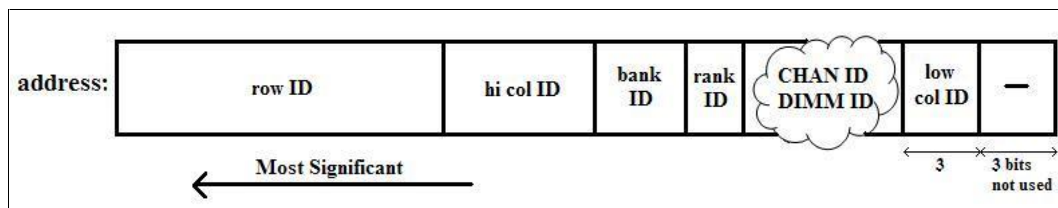


Figure 5.7 – Closed Page Mapping

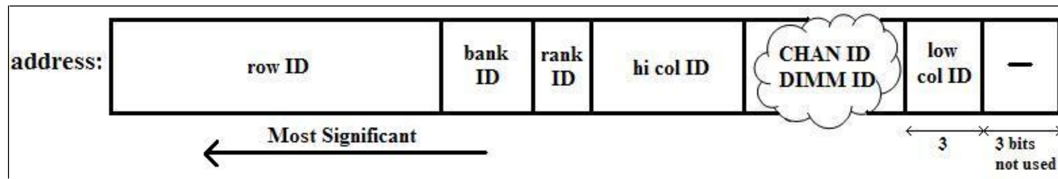


Figure 5.8 – Open Page Mapping

5.4 The Scheduler

The transaction scheduling functions in FBsim (there are two, one for Closed Page Mode, and one for Open Page Mode) operate on one channel. In each simulator tick (frame iteration), the appropriate scheduling function is called once on each active (non-empty) channel.

The basic organization is as follows: The scheduler consists of a scheduling window with a logical size of ‘x’ transactions (‘x’ is specified in the .sim input file – table 5.1), and a FIFO queue for when the scheduling window is full. When the transaction generator generates transactions, it attempts to enter them into the scheduling window, but if it is full, it puts them in the queue.

The scheduling window represents transactions that can be considered for scheduling in this tick. Transactions in the queue cannot be scheduled until they have entered the window. Thus the size of the window can be seen as the maximum amount of reordering that can conceivably take place amongst arriving transactions. The three possible reasons for limiting the size of the scheduling window are time (the scheduling decision must be made and the frame packed within a frame period), hardware complexity (considering more transactions at a

time for scheduling quickly increases hardware complexity and resource usage), and coherence (we don't want a read coming after a write to the same location being put ahead and reading the stale value). The coherence reason can be dealt with by checking read addresses off against write addresses in the window when scheduling reads, and vice versa for writes, but this would cost more complexity and time.

The scheduler window is made up of two linked lists, one for read transactions, and one for writes. When summed together, the size of these lists cannot exceed the specified schedule window size. When a transaction enters the scheduling window, it is given an integer 'patience' value equal to that specified in the .sim input file (table 5.1). At the end of every tick, the scheduler decrements all the patience values in the read and write lists. How this plays into the scheduling algorithm will be explained soon, but suffice to say for now that new transactions are entered into the ends of their linked lists, so that the linked lists remain 'sorted' in order of patience.

Recall that on the SB channel, up to 3 commands can fit into a frame. Recall also that each frame arriving at a DIMM maps directly into one DRAM clock cycle (which necessarily equals 1 DRAM command cycle), so only one command per DIMM is allowed onto a SB frame.

In closed page mode, a read transaction translates to a row acativation command, followed by t_{RCD} frame times, followed by a read command, followed by $t_{RAS} – t_{RCD}$ frame times, followed by a precharge command (for a review of

these time parameters, see table 3.1. Note that in FBsim, all these parameters can vary even across different DIMMs sharing the same channel). The addressed DRAMs in the targeted DIMM respond with data t_{CAS} frames after they receive the read command from their AMB. Thus from the frame of reference of the memory controller, 4 read response frames begin arriving on its NB channel input exactly t_{CAS} + latency frames after it sent out the read command, where 'latency' is either the round trip latency of that DIMM (variable latency mode), or the round trip latency of the last DIMM on the channel (fixed latency mode). Note that in either case, latency is a DIMM parameter that is calculated for all DIMMs at the beginning of the simulation.

For write transactions in closed page mode, first the memory controller must send the 8 frames of write data to the targeted DIMM's AMB write buffer. Once this has been done, the controller then sends a row activation command, followed $t_{RAS} - t_{WR} - t_{BURST} - t_{CWD} + t_{CMD}$ frames later by a write command (in FB-DIMM again t_{CMD} is always 1, and in FBsim, t_{BURST} is always 4). And then $t_{CWD} + t_{BURST} + t_{WR} - t_{CMD}$ frames after the write command, a precharge command is sent.

In open page mode, the reasoning is similar. The difference is that if the row is open, only one command (read or write) needs to be sent, whereas if another row is open in the addressed bank, the bank must be precharged, activated, and only then is the read or write command sent. In open page mode, every DIMM data structure keeps a two dimensional integer array for open row

tracking, where the first dimension is the rank index, and the second is the bank index.

The above arguments demonstrate that even in closed page mode, there is no good regularity to determine pipelining on the SB and NB channels; there are just too many possibilities - even more so in variable latency mode. This in my opinion necessitated a brute force ‘pattern matching’ approach to scheduling.

In FBsim, the scheduler keeps track of the future utilization of the channel through a circular rolling array, rolling meaning that at the end of every simulated frame period, the ‘current time’ index (pointer) to the array is incremented, and circular meaning that the end logically loops back to the beginning. Before the current time index is incremented, the contents of the array it indexes are zeroed. After incrementing, those contents represent utilization in the ‘distant’ future. The NB and SB arrays share the ‘current time’ index. The NB array must be larger in size than t_{RCD} plus the maximum DIMM latency on the channel plus four, so that if you schedule a read to that DIMM, you can mark the NB channel busy that much in future. As for the SB arrays, there are three of them (each one represents a ‘command third’ of each frame), and their size must be roughly equal to the maximum t_{RC} (row cycle time), so that the up to three commands (precharge, activate, read/write) can mark the appropriate array as busy in the appropriate place in a read transaction, or a write transaction (after the write data has been sent). Thus to share the same time index for simplicity, the size of the above arrays must be larger than the maximum between t_{RCD} + latency + 4, and t_{RC}.

When a transaction is scheduled, it fills in the necessary channel array elements in the corresponding future times with ‘1’ for busy. When a transaction is considered for scheduling in a given frame third, the scheduler checks to make sure that the right spaces in the arrays are zero (i.e. channel free at those times). If they aren’t, the transaction can’t be scheduled yet. This is what is meant by ‘brute force pattern matching’.

Before scheduling a transaction, the scheduler must also consider DRAM minimum scheduling distances. Minimum scheduling distances are given for open page mode and closed page mode (colored boxes) in table 3.2. As described above, in FB-DIMM, DRAM minimum scheduling distances only apply between transactions being scheduled to the same DIMM. To accommodate this requirement in FBsim, when a transaction is scheduled, it is inserted into a ‘hazards’ list contained in the DIMM data structure. It only remains in that list for ‘maxHazard’ ticks, where maxHazard is the largest minimum scheduling time found in table 3.2. So when a transaction to a DIMM is to be scheduled, it is first checked off against all the transactions in the DIMM’s ‘hazards’ list to make sure no minimum scheduling distance is violated.

In FBsim, minimum scheduling distance for a transaction in consideration to be scheduled is checked out before the channel utilization possibility is checked. When a scheduling rejection occurs for either of the above reasons, it is saved, and when a dump time is reached, these different rejections are summed and each divided by the sum to give a percentage that indicates what percentage

of negative scheduling decisions were made negative due to that reason. These percentages are saved into 'rejects.txt'.

Now that the scheduling decision has been covered, one thing remains. Where do we look first, in the read list, or the write list?

The first thing to consider is that a write data frame can fit one command, whereas a command frame can fit three commands but no write data. Thus write data should be given first priority – if it exists in the scheduler, send it. A point to keep in mind however is that two reads are required to simultaneously fill the NB channel during the sending of one block (8 frames) of write data. In closed page mode, each read transaction needs three commands (activate, read, precharge) for a total of 6 commands. Meanwhile, once the write data is sent, the write transaction needs 3 commands of its own (activate, write, precharge). Hence if all frames are write data frames, 9 commands are needed while 8 slots are available. This means that every once in a while, an 'overflow' command frame is needed, or the scheduler will either fill up with all read commands or all write commands (as opposed to write data which is being immediately sent) depending on which of the two are given first priority. To counter this possibility, when it detects an imbalance between read and write commands in the scheduling window (60% to 40%), FBsim will reserve frames for command only to prevent write data from being sent on that frame unless more than 1 command cannot be scheduled for it. This will adaptively redress the balance between reads and writes in the scheduling window and keep it at close to 50-50. A problem is however that this

50-50 design is optimized for an input transaction ratio of two reads to each write. If the input ratio is different than this, keeping the scheduling window half full of reads and half full of writes is not the best solution. A more optimal solution would be to adapt the threshold where the command frame reservation kicks in to change according to the incoming transaction read/write ratio, which the scheduler would consequently have to measure. This is studied in section 6.2.4, and proposed as a future study.

Taking everything into consideration, the overall scheduling algorithm I adopted works as follows:

- If the current frame is not ‘command reserved’, and it contains no more than one command from a previously scheduled transaction, then attempt to load it with write data.
- If one or more command spaces remain empty, attempt to schedule read commands, and then write commands starting with this frame. With write commands, minimum scheduling distances in the addressed DIMM are tested, and if that test is passed, the SB command utilization pattern of the write is tested. With read commands, minimum scheduling distances are similarly testing, followed by testing the SB command utilization pattern, followed by testing the NB read response pattern.
- If the frame was command reserved and skipped the first step, yet only one or less commands have managed to find their way onto the frame, attempt to load it with write data.

In Chapter 6, the debug mode is discussed, and FBsim generated channel utilization diagrams for this scheduling scheme are shown for verification of this scheduling algorithm. As you've seen, a lot of design choices were made for this scheduler, and a study could be done on how making different choices would affect the scheduling efficiency, and the complexity/efficiency tradeoffs involved.

CHAPTER 6: FBSIM VERIFICATION AND EXPERIMENTS

The Fully Buffered DIMM architecture was proposed and initially defined by a small group of engineers from Intel. As such, many very important design decisions on a technology that is very likely to soon make a large impact on the memory market, were made privately by a small group of people. Based on their core propositions and assumptions, a closed group of joint experts from across the memory industry went to work fine tuning and standardizing the proposed technology, and even beginning production along the way, with the first FB-DIMMs to be sold at the end of this year. All the while, targeted consumers and interested academics have only been given teasing hints about the technology, and very little of substance with which to validate the few performance figures thrown at them by the FB-DIMM developers and with which to understand and study this new design space.

In March of 2005 that changed, when Infineon released the first FB-DIMM datasheet. In April, a second datasheet was released, this time from Micron. These datasheets are preliminary and still somewhat hazy in areas, whether purposefully or not. Furthermore, JEDEC have yet to publish the FB-DIMM standard. But finally, enough information about the technology is available to begin accurately studying this important design space. Much of the available information has been collected and assimilated into chapter 4 of this paper.

As FB-DIMM is a brand new and somewhat revolutionary technological step, the avenues available for study are numerous. FBsim aims to be one of the tools available to aid in studying this design space. Only a small subset of the FB-DIMM design space is studied in this paper, but it is my hope to give a clearer idea of how this technology performs, and to identify the interesting avenues of study. It is also my hope to validate the operation of FBsim and the choices I made in implementing it through some of these studies and their results.

Due to its specialized nature, very fast runtime, and easy control of memory request rates, FBsim is ideally suited to study the saturation point of the Fully Buffered channel, namely the 'breaking' point at which the channel speed and frame protocol becomes the limiting factor of the memory system, i.e., the peak sustainable bandwidth of a Fully Buffered channel. This saturation point shows the true colors of the FB-DIMM technology design – i.e. where and why different design decisions might have done better, or worse. It can also teach us how best to use this technology, since it has been presented to us as a fait accompli.

The preliminary studies performed in this chapter will show that the design decisions made for FB-DIMM have generally been very robust ones, and that the technology is indeed very much scalable. However, the studies also show that a slightly wider southbound channel would allow for even better sustained bandwidth and that at least two FB-DIMMs are needed on a channel to take advantage of the FB-DIMM channel architecture rather than be limited by it.

6.1 Case Study I –Constant Capacity

The Fully Buffered architecture allows us to go deeper with up to 8 DIMMs on a channel, but by making channels narrower, it also reduces the cost of expanding the number of channels in a system, both by reducing pin counts, and cutting down on system board real estate per channel. In this study, there are 8 hypothetical 512 MB DDR2-800 DIMMs for a total capacity of 4GB. The DIMMs can either be all placed on one channel, split over two channels, four channels, or eight (eight is academic as the Fully Buffered standard specifies up to six channels only). These configurations are respectively called 1x8, 2x4, 4x2 and 8x1. In this study, Closed Page Mode and Fixed Latency Mode are used.

6.1.1 DIMM and System Board Parameters

Table 6.1 specifies the parameters for the DIMMs used in this study.

DRAM Type	DDR2-800	**Column Width**	4 bits
DIMM Density	512 MB	**Number of Ranks**	1
Device Type	256Mb (64M x 4)	**Devices Per Rank**	16 (18 with ECC)
Banks Per Device	8	**AMB Pass Through Latency**	2.2 ns
Rows Per Bank	8192	**AMB Deserialization Latency**	8.1 ns
Columns Per Row	1024	**AMB Serialization Latency**	5.0 ns
t_{CAS}	5	t_{RC}	19
t_{CWD}	4	t_{RCD}	5
t_{DQS}	2	t_{RP}	5
t_{RAS}	14	t_{WR}	5

Table 6.1 – DIMM Parameters Specified for Case Study

The signal path from the memory controller to the first FB-DIMM is given to be 12 cm. At two thirds the speed of light, this implies a 0.6 ns delay. The signal path from one FB-DIMM to another is given to be 4cm, yielding a latency of 0.2 ns.

The following equation gives the round trip latency of a read command once it has been scheduled, taking into account that this is fixed latency mode, and closed page mode (so the read must be preceded by a row activation, adding a $t_{RCD} + t_{CAS}$ delay):

$2(0.6) + 2((N-1) \times 0.2) + 2((N-1) \times 2.2) + 8.1 + 5.0 + t_{CAS} + t_{RCD}$; where N is the number of FB-DIMMs on the channel. All other parameters are from table 6.1 ($t_{CAS} = t_{RCD} = 5\ t_{CMD} = 5 / 400\text{MHz} = 12.5\text{ns}$). Thus for eight FB-DIMMs on the channel, the minimum latency is 73.2 ns, for four its 53.7 ns, for two DIMMs its 44.1 ns, and for one DIMM its 39.3 ns.

6.1.2 Ramp Input

In this experiment, a ramp input is specified to FBsim. This input increases the request rate slowly, so that the response of the system can be seen over a large spectrum of request rates. The experiment takes place over 30 milliseconds, and the input is designed so that saturation is reached at about $t = 27\text{ms}$.

The ramping input is made up of 20 read step distributions (see section 5.2 for more information), and 20 write step distributions. The read to write ratio is fixed at 2:1 throughout the experiment.

Figure 6.1 shows the input request rate graph returned by FB-DIMM for the 1x8 configuration. For the 2x4 configuration the input looks the same, but at double the request rate, as there are two channels. For the 4x2 configuration the request rate is again doubled. The 1x8 configuration however needed to be significantly less than double the 4x2 configuration, as will be seen later.

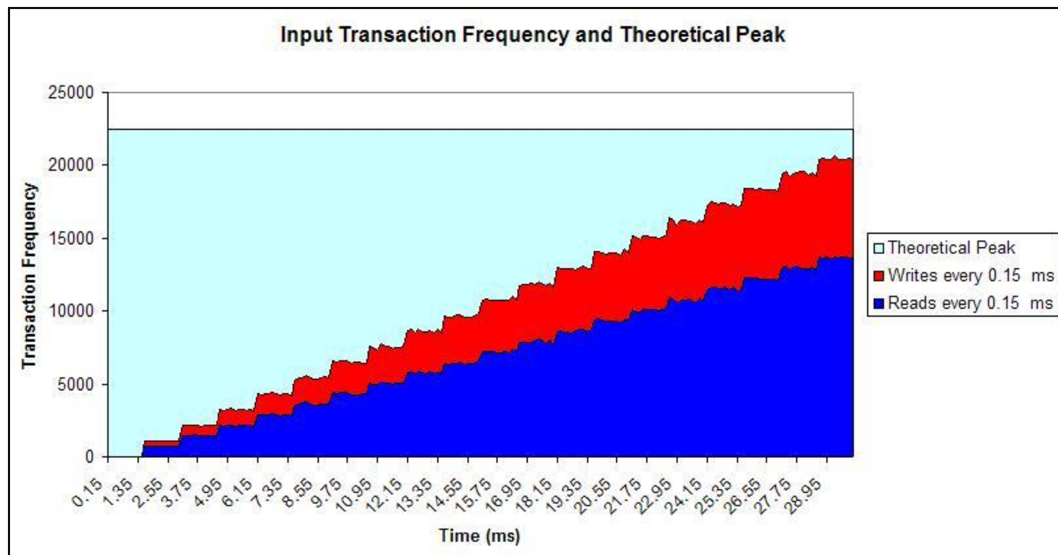


Figure 6.1 – Input Request Rate Specification; 1x8 Configuration

In figure 6.1, the line at the top of the graph denotes the peak theoretical input transaction rate (which the FB-DIMM architecture specifies as 1.5 times the peak theoretical DDR data rate). As you can see, the input is made up of stacked step inputs, a read step and a write step from t = 1.5ms to 30 ms, a read step and a

write step from t = 3ms to 30ms, a read step and a write step from t = 4.5ms to 30ms, etc. As is explained in section 5.2, each step represents reads to or writes from a contiguous block of memory. However once multiple steps have begun, the request streams from the steps interleave, so that once many steps are being processed the scheduler receives little to no locality in its input stream.

6.1.3 The 1x8 Case

This section shows the detailed results of the 1x8 experiment. All four configurations will then be shown together in section 6.1.4. Figure 6.2 shows the throughput response of the Northbound (read) and Southbound (write) channels. Note that commands on the SB channel do not enter into the write throughput calculation. Neither do ECC bits, frame CRC bits, or anything that is not pure write data (64 bits per write frame), or read data (128 bits per read frame).

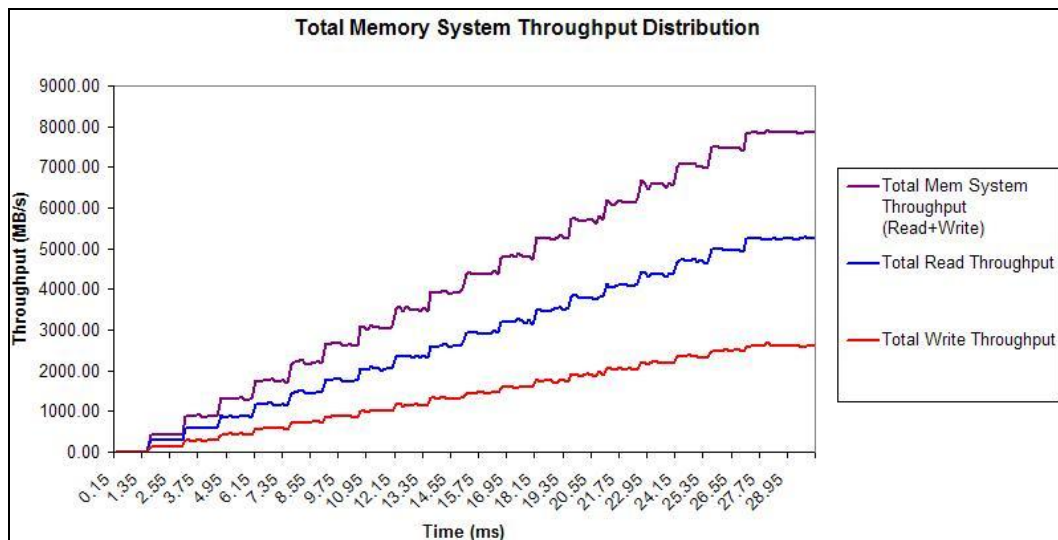


Figure 6.2 – Data Throughput Response; 1x8 Configuration

It can be seen in figure 6.2 that the data throughput on the SB and NB channels increases to match the stepping up of the input transaction rate, until the last few steps where the channels saturate, and throughput does not increase further. This saturation happens at 5.2 GBps on the NB channel (81% of the peak theoretical read bandwidth of 6.4 GBps), and at 2.6 GBps on the SB channel (also 81% of its peak theoretical bandwidth of 3.2 GBps). It is observed that the two channels saturate at the same time (which is of course due to the fact that read request commands come from the SB channel). More interesting to note is that the two saturate at the same percentage of their respective peak theoretical bandwidths, and maintain the 2:1 read/write ratio throughout. This validates the fact that the scheduling algorithm of FBsim is tuned to this ratio, issuing one write for every two reads even when the scheduling window is full and queuing is taking place. Different read/write ratios are explored in section 6.2

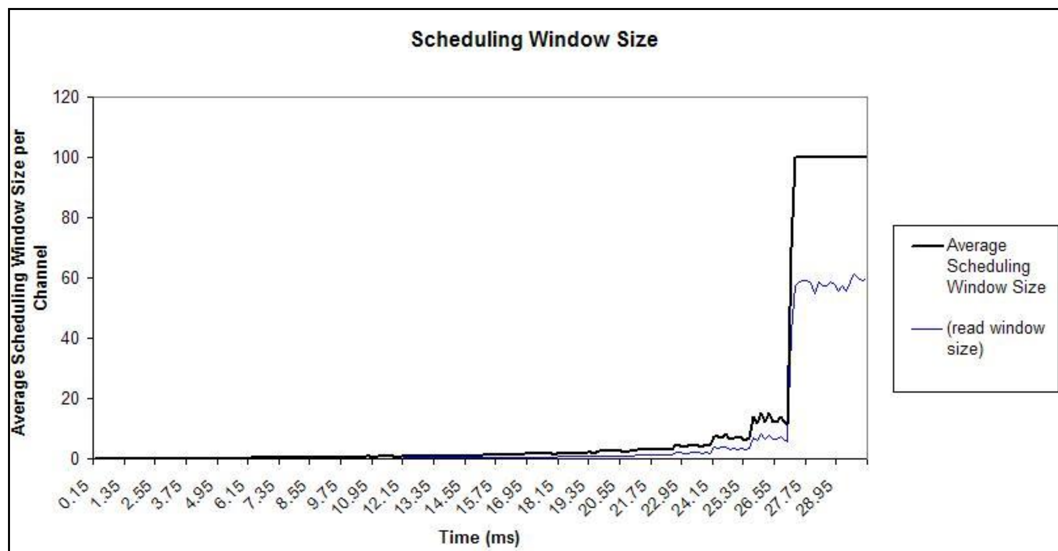


Figure 6.3 – Average Scheduling Window Size

Figure 6.3 (above) shows the average FBsim scheduling window size over time in this experiment. A maximum scheduling window size of 100 has been specified in this experiment, meaning that up to 100 reads and writes can be considered simultaneously by the scheduler in filling a SB frame. For 8 DIMMs and fixed latency mode, this figure will be shown to be overkill, but with 1 DIMM on the channel, having more command possibilities to choose from becomes much more critical to prevent stalls on the channel, so an academically large schedule window size is maintained to ensure that if there is a command out there that can be scheduled in a given frame or frame third, it is found. This parameter is even more important in Variable Latency Mode where it is harder to schedule reads, and yet more important in Open Page Mode where re-ordering reads (to utilize open banks) becomes a major factor in performance. In this experiment, the 'patience' value I specified for transactions entering the scheduling window was 120 – just over the schedule window size. This prevented any stalls from occurring on the channel due to expired patience, even at saturation, meaning that little transaction reordering was taking place. Lowering the patience to the schedule window size however, or below, yielded a 10% drop in throughput at saturation due to patience stalls, so I kept this factor out as saturation effects are being studied in this experiment.

The blue line in figure 6.3 represents the fraction of the scheduling window that is filled with reads. This figure hovers between 55% and 61% of the schedule window size as the window fills up, meaning that the adaptive

scheduling algorithm of FBsim (described in chapter 5) worked well at balancing read and write priorities given a 2:1 read to write input ratio.

A question to ask from figure 6.3 is the following: why is the scheduling window starting to fill up but not filling up completely before channel throughput saturation is reached? This point is directly related to the amount of transaction reordering taking place. As input transaction rate increases, the number of possible scheduling-distance conflicts increases likewise due to the decreasing time period between incoming transactions. A point is reached in this decreasing period where it becomes lower than some minimum scheduling distances between adjacent or near transactions in the input stream. At this point, buffering and reordering starts to take place in the scheduler, but the scheduler still manages to match the input rate in its scheduling - so that the scheduling window grows to the extent needed to accommodate this reordering, but not larger. Then a (saturation) point is reached where the input rate exceeds the sustainable capacity of the SB and NB channels. The scheduling window immediately fills up, and further transactions start queuing. Throughout all the experiments, the average scheduling window size would step up slowly until it reached between 10 and 12, and then it would rocket up to maximum (at channel saturation). To investigate this further, the experiments were repeated for an academically large maximum scheduling window size of 30,000, and the same effect was observed – the last stop in schedule window size was between 10 and 12, and then the schedule window size skyrocketed up with no further increase in channel throughput. This implies that a

maximum degree of reordering of about 12 transactions is needed in a good FB-DIMM memory controller scheduler to allow it to reach the maximum sustainable bandwidth (true saturation bandwidth) of a channel.

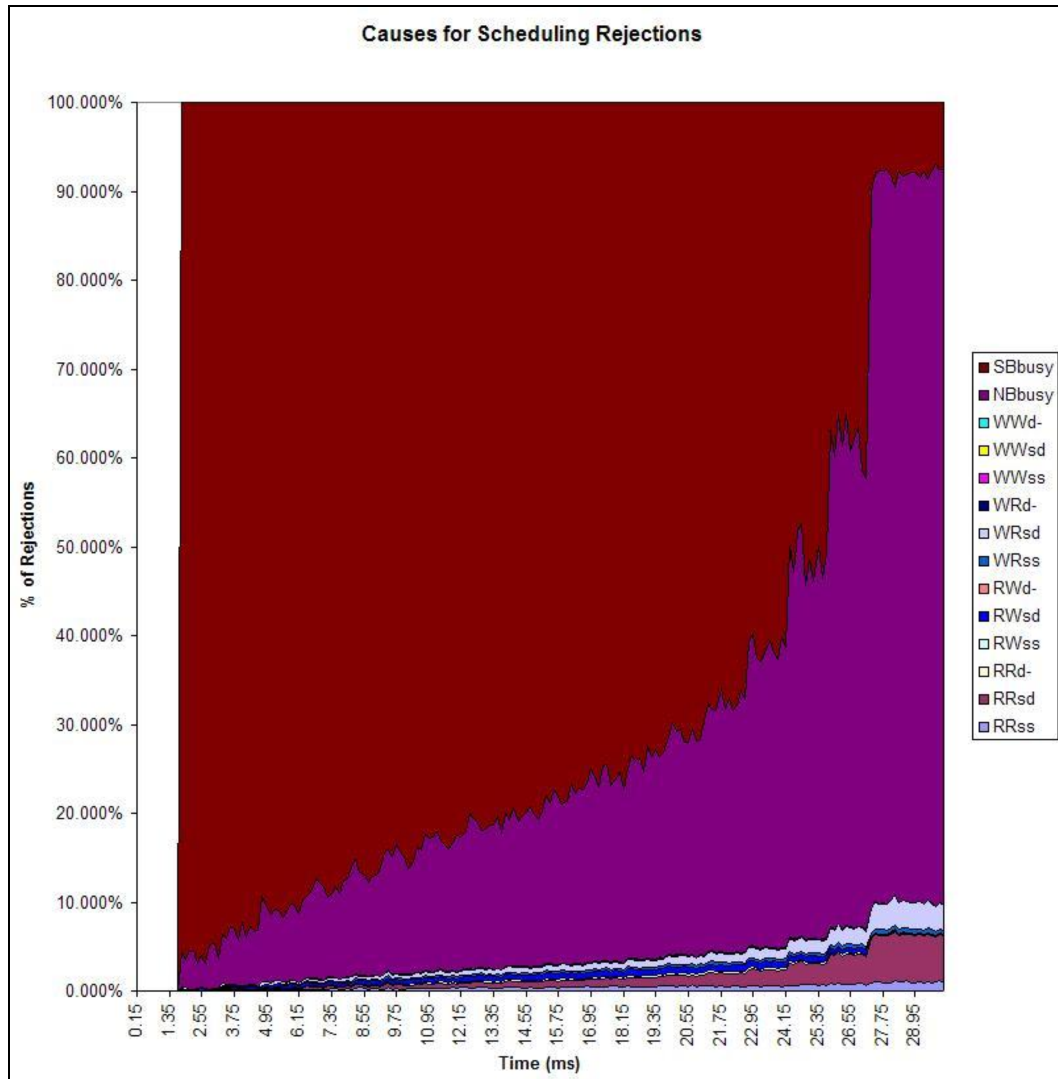


Figure 6.4 – FBsim Percentage Makeup of Scheduling Rejections

Figure 6.4 compares the percentages of scheduling rejection reasons throughput the simulation. The dark red region at the top of the graph in figure 6.4

represents the percentage of transaction scheduling rejections that are due to the SB bus being full (i.e. not enough space for write data in case of write, or not enough space for row activation command in case of read or write). The purple region just below represents transaction scheduling rejections due to the NB channel already being reserved by some other read response, for some or all of the read response time needed by a read transaction that is currently being considered for scheduling. The regions below the purple region represent minimum scheduling distance violations that prevent a transaction under consideration from being scheduled (see table 3.2 in section 3.4).

Initially, at low input rates, scheduling rejections are rare, and when they do occur they are mostly due to traffic on the SB channel. At higher input rates, since there are two reads to every write and since in my scheduler implementation read commands are given priority over write commands (not write data) in scheduling, the number of rejections made due to traffic on the NB channel increases. As argued above, once the input transaction rate reaches a point where minimum scheduling distances become non-negligible, their share of rejections begins to increase, leveling out at saturation. Amongst the minimum scheduling distance rejections themselves, constant ratios are of course maintained. The fact that the minimum scheduling distance factor is kept to 10% of rejections at saturation (even though these are considered first) suggests that the address mapping algorithm of FBsim (see section 5.3) is doing well at balancing the load across the DIMMs on the channel. An interesting study would be to vary the

mapping algorithm and observe its effects on rejection percentages, and see how it affects sustainable throughput of the system.

Figure 6.5 shows the average read latency distribution zoomed in (top half) and zoomed out (bottom half).

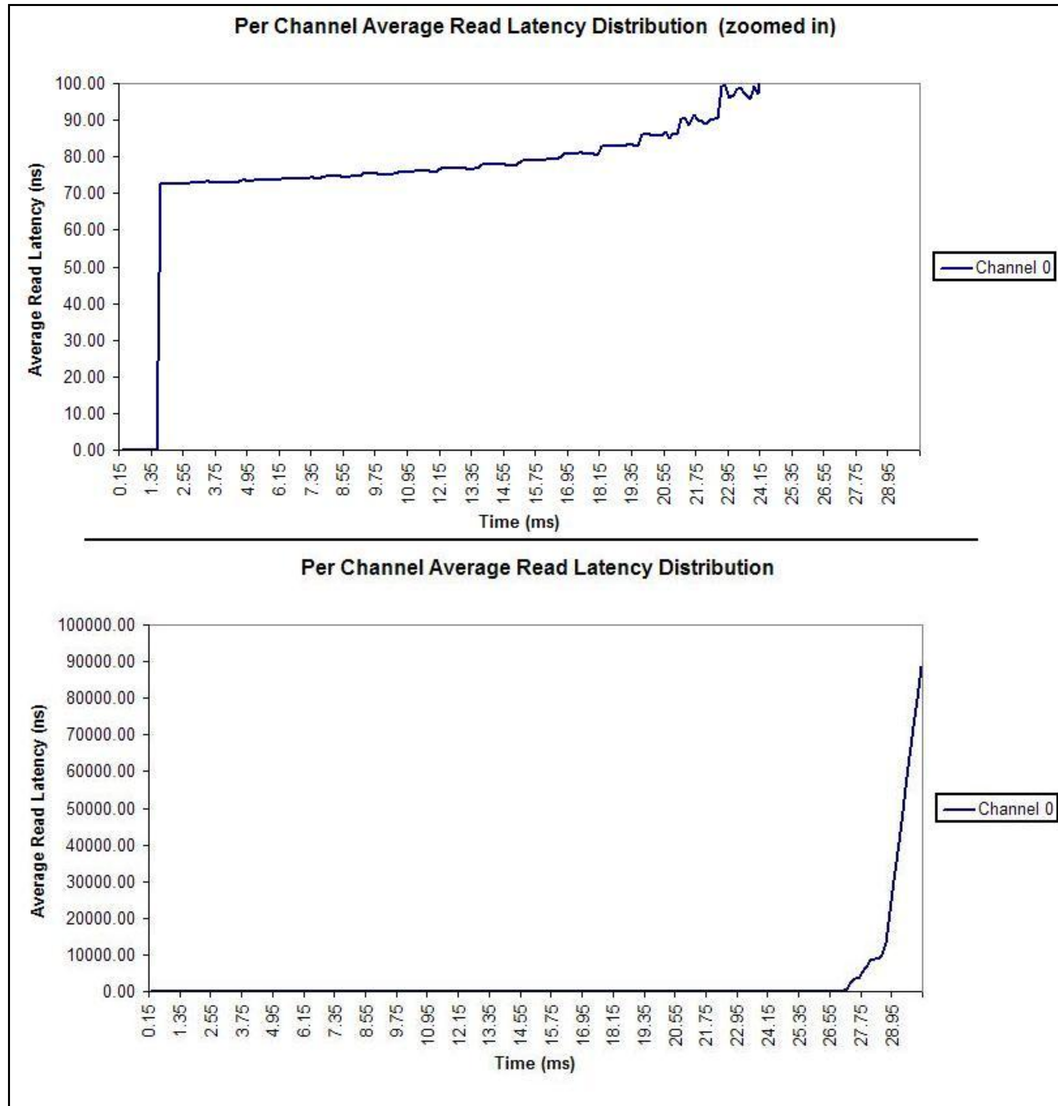


Figure 6.5 – Average Read Latency Distribution

Figure 6.5 shows average read latency starting out at around 73ns at low load, exactly as is to be expected for fixed latency mode and 8 DIMMs on a channel. As the input transaction rate increases though, the average scheduling window size starts a slow increase (as described above), causing the average read latency to slowly increase also. In fact, there is of course an exact correlation between average read latency and the combined average size of the schedule window + scheduler queue (the scheduler queue starts filling once the schedule window is full). Much more interesting than this is to plot system throughput against read latency, as in figure 6.6.

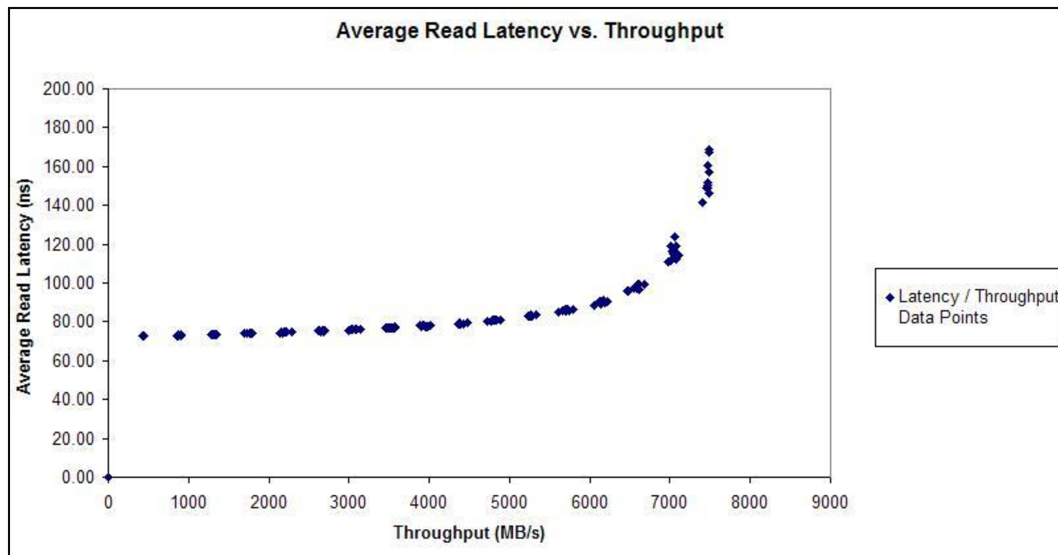


Figure 6.6 – Average Read Latency vs. System Throughput

In this graph, the data points of each step in the input ramp cluster together, and these clusters draw a curve showing increasing latency as system-wide throughput increases. The latency curve becomes vertical as saturation is

reached – in this case at 7.8 GBps. This curve is very useful to judge the performance of a given FB-DIMM configuration at different loads.

6.1.4 Comparison Between the Cases

In this section, the throughput graphs, and the latency/throughput curves are shown together for the four system configurations. Figure 6.7 shows the throughput distributions and saturation points for the four experiments.

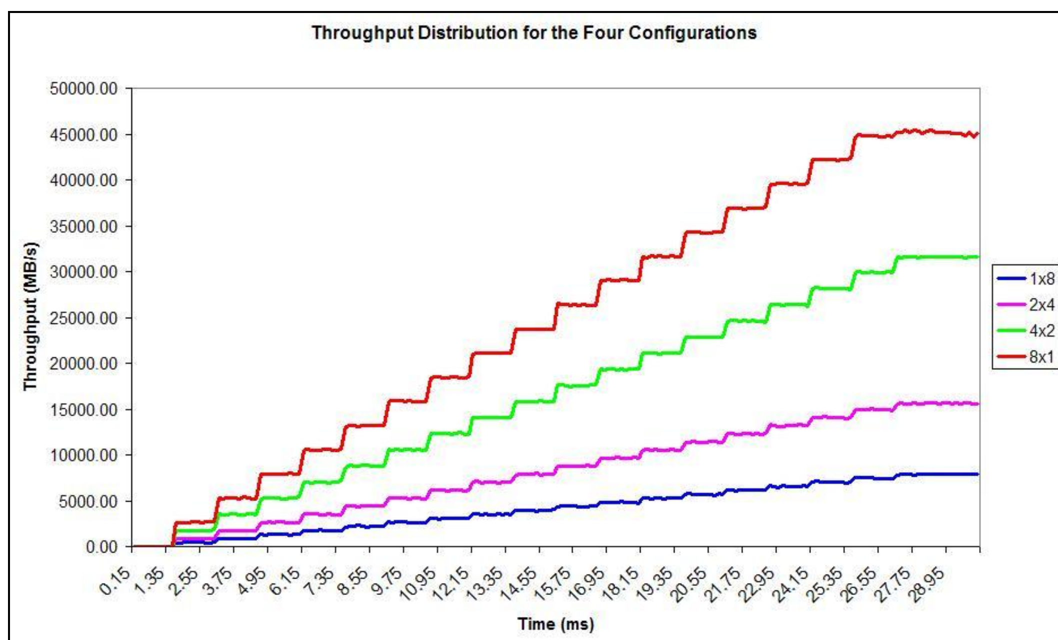


Figure 6.7 – Throughput Distributions for the Four Configurations

As described earlier, the input rate for the 2x4 configuration is double that of the 1x8. Similarly, the input rate for the 4x2 configuration is double that of the 2x4. However, the input rate for the 1x8 configuration is not double that of the 4x2, because as figure 6.7 shows, the maximum sustainable bandwidth for the 8x1 case is much less than double that of the 4x2.

The percentage of the Fully Buffered channel's specified peak theoretical bandwidth achieved by the maximum sustainable bandwidth remains at ~81% in the 1x8, 2x4 and 4x2 configurations, while in the case of 8x1, this percentage drops significantly to 59%! This result immediately tells us that having just one DIMM on a Fully Buffered channel is a waste of that channel, and that this situation should be avoided if possible. De-serialization and serialization latency is being added (over the old parallel bus architecture) for no benefit in this case, and the capacity of the high speed channels is not being utilized for throughput. The only way the FB-DIMM architecture helps in this case is with the presence of the FIFO write buffer in the sole DIMM's AMB, but at a high input request rate and with just one DIMM, the write buffer cannot completely hide read/write and write/read turnaround. As you will see in the next section however, the greedy scheduling algorithm of FBsim does end up implicitly reordering to group chunks of writes together, and chunks of reads together thus minimizing this overhead, but this also means insufficient simultaneous usage of the NB and SB channels.

Figure 6.8 shows the throughput/latency curves of the four configurations. The saturation throughputs in each case are shown as a dotted asymptote. The graph gives the expected result that having less DIMMs on a channel will mean less latency at low loads (although this figure does not half from configuration to configuration due to the fixed $t_{RCD} + t_{CAS}$ component in closed page mode, and the fixed serialization/de-serialization AMB latency) and that having more channels will yield a higher maximum sustainable throughput.

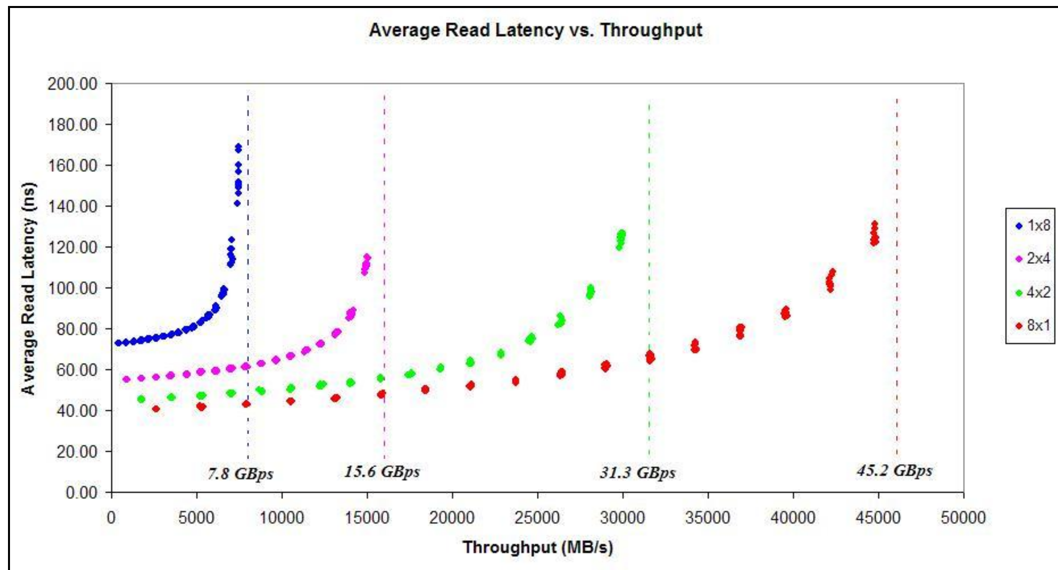


Figure 6.8 – Latency Throughput Curves for the Four Configurations

Figure 6.8 shows again how doubling the number of channels doubles the maximum sustainable bandwidth of the system (verifying good channel load-balancing by the FBsim mapping algorithm), except if just one DIMM is being left on channels, in which case minimum scheduling distances within that DIMM become the limiting factor (as described above) and the fully buffered channel cannot be saturated. To take good advantage of the FB-DIMM architecture, namely to saturate its dual (SB and NB) unidirectional data paths, we need only have two FB-DIMMs on a channel with a good scheduler. It is important also to note that doubling the number of channels does not necessarily double the cost as a chunk of the cost – the cost of the 8 DIMMs – remains constant. Furthermore, channels in FB-DIMM are narrower and so require less memory controller pins and system board space. This fact, along with the support for up to 8 DIMMs on

every channel, along with the results of this experiment (doubling channels doubles throughput as long as at least 2 DIMMs are present on each channel) show that the FB-DIMM architecture does indeed meet its advertised scalability.

Figure 6.9 shows the scheduling rejection percentages for the four configurations:

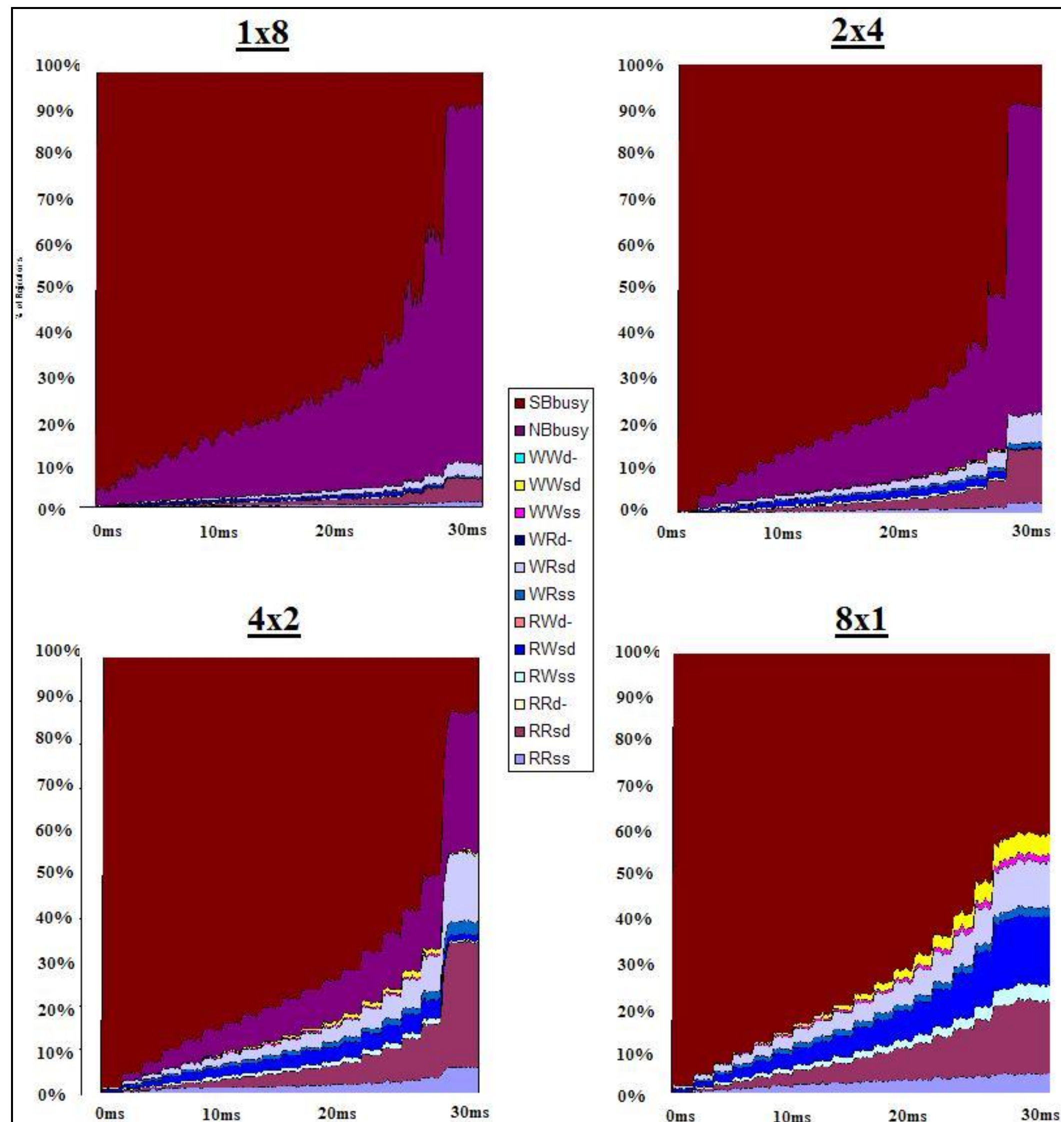


Figure 6.9 – Causes for Scheduling Rejections in Each Configuration

Figure 6.9 shows the rising number of scheduling rejections made due to minimum scheduling distance violations in each of the experiments, and the large jump that these make at the end when the channel saturates. It is this factor initially raising the average read latency - as transactions start having to wait in the scheduling window and get reordered – until the channel saturates and it becomes the saturated channel that causes queuing and the associated skyrocketing of read latency. The figure shows also how the percentage shares of minimum scheduling distance based rejections double when the number of DIMMs per channel halves. This effect can not be seen in the 1x8 case, because in the FBsim scheduler, minimum scheduling distances are checked before NB channel contention, and of course the minimum scheduling distance between two reads to the same DIMM is t_{BURST} which is also the 'minimum scheduling distance' for the NB channel. Thus if the former condition is satisfied, the latter is as well, and 'NB scheduling rejections' are eliminated (this also explains the decreasing share of 'NB busy' rejections from the 1x8 case through the 8x1 case).

6.2 Debug Mode – A Microscopic Look at Channel Activity

As described in chapter 5, FBsim contains a 'debug mode' that I used to debug and verify the scheduler. The output of this mode however is also very interesting to study the frame-to-frame channel activity of the fully buffered channels during different modes of operation.

6.2.1 Two Reads to Every Write

Figure 6.10 shows a 'clip' of the channel activity when a channel was debugged for a duration of 1500 frames, where the input was a saturating workload with on average two reads to every write. The channel is saturated. The table in the figure summarizes the channel utilizations during these 1500 frames.

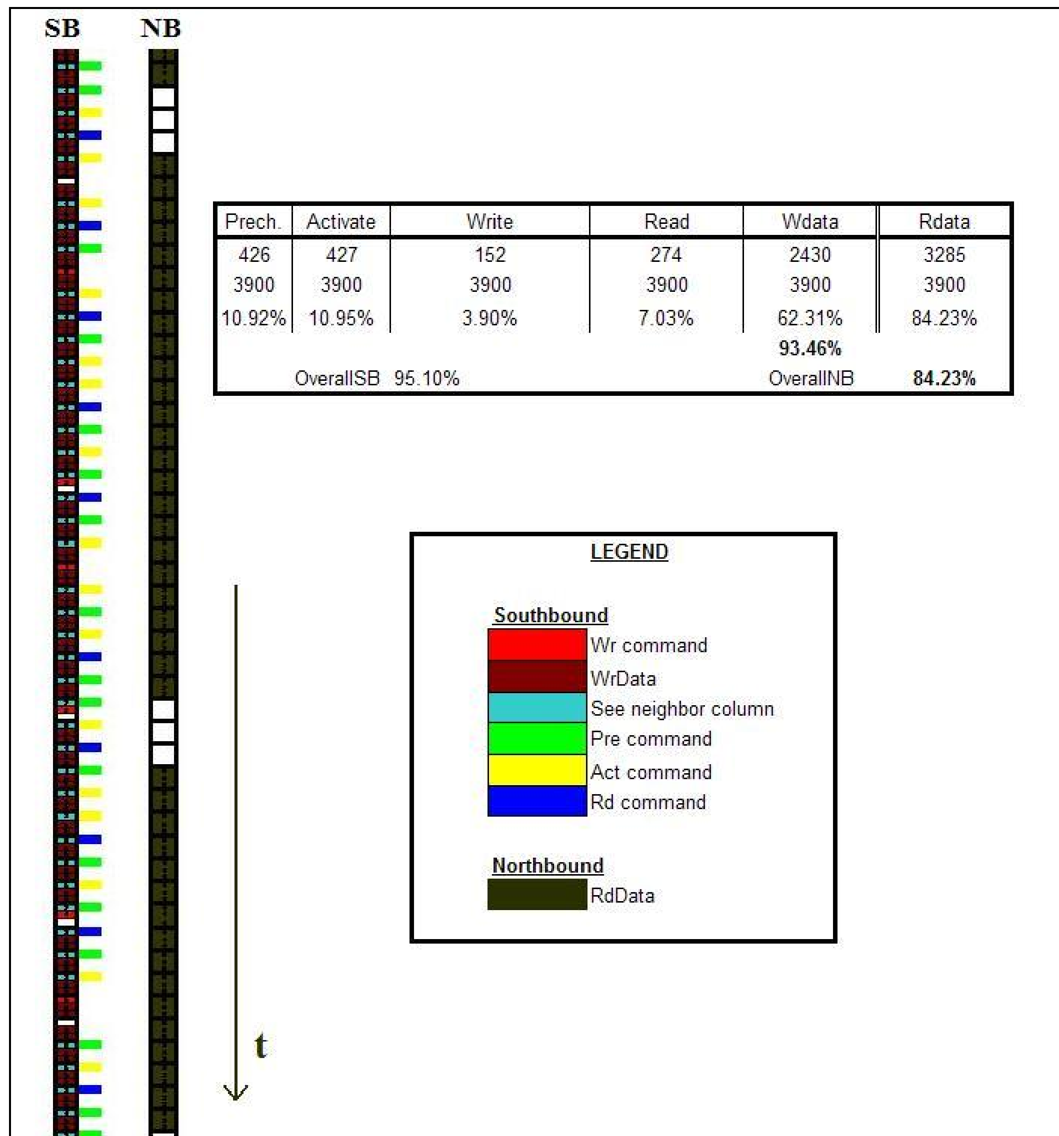


Figure 6.10 – Channel Utilization with Read : Write Ratio = 2:1

In figure 6.10, the channel on the left is the SB channel, and the channel on the right is the NB channel. Time progresses in the downwards direction. Note that in debug mode, the channel latency is removed from the read responses, effectively transposing the NB channel up. This helps debug read commands/responses.

On the SB channel, write data is shown by the dark red color. A write data frame may also contain one command. A cyan colored command represents a precharge, activate or read command. These are represented in the neighboring column with a green, yellow, or blue color respectively. A write command is red.

The table in the figure represents the percentage of frame thirds occupied by each type of command (or write data) on the SB channel, except the rightmost column which represents percentage utilization of the NB channel by read response data.

On the SB channel, frames are split into thirds. In a command frame, each third represents a command, while in a write data frame, two thirds are colored by the write data, and the third by a command. However, although it can contain a command, a write data frame represents 100% data throughput on the SB channel, not 67%. Thus to get the percentage of peak theoretical utilization of the SB channel, the ‘write data frame third percentage’ is multiplied by 3 / 2.

Close observation of the SB channel shows that the vast majority of frames (93%) are write data frames, but every once in a while, a command frame is needed. This is due to the following argument: in a 2:1 read to write input ratio,

eight write data frames (one write) happen for every 8 read response frames (2 reads). With the eight write data frames, eight commands can be sent. However, nine commands are needed in closed page mode for two reads and a write; 3 row activations, 2 read commands, 1 write command, and 3 precharge commands. This necessitates periodic 'overflow' command frames, which the FBsim scheduler always manages to fill with at least two commands, and sometimes three.

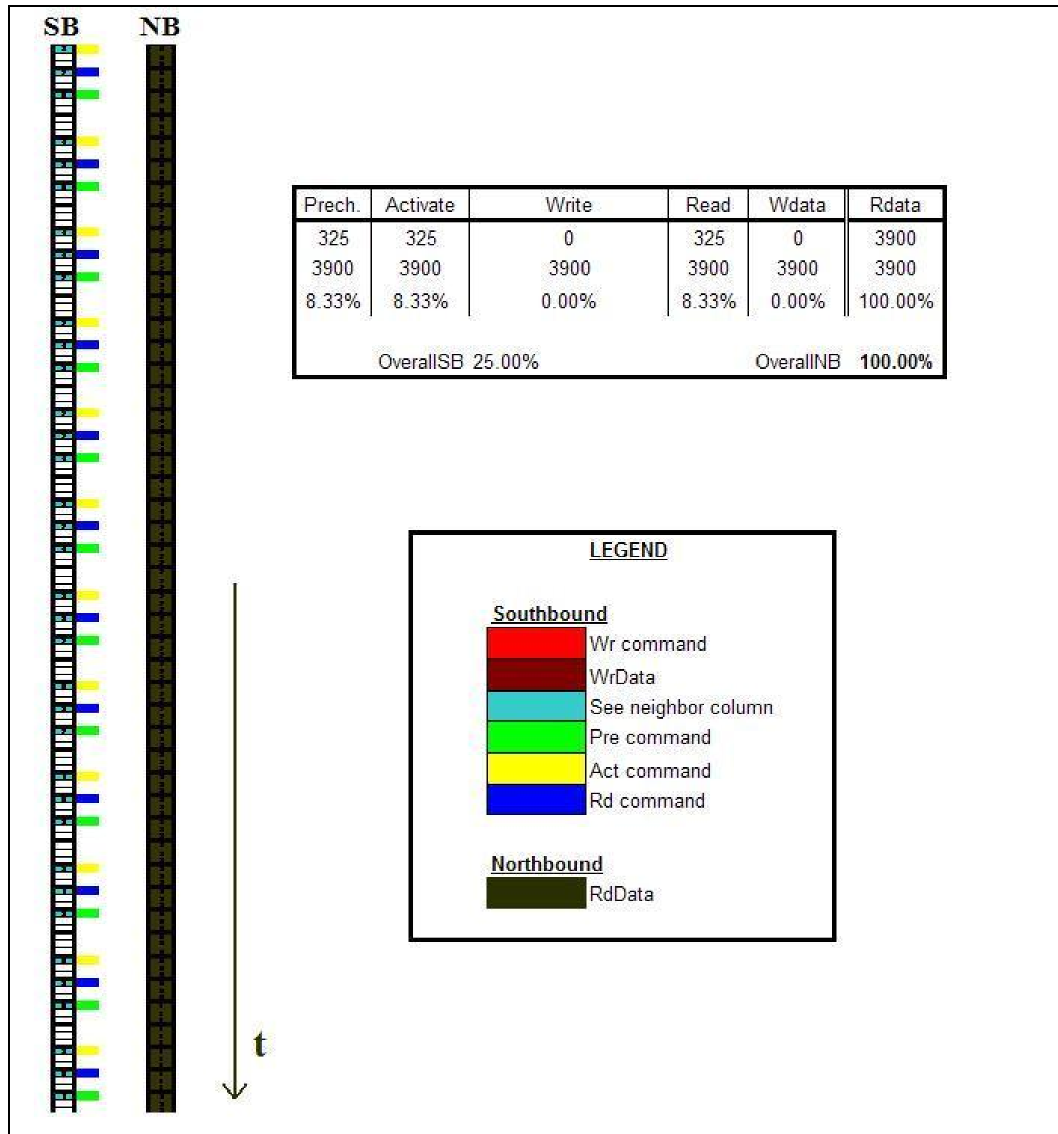
This leads to an important observation: If 24 extra bits were added to the size of a SB frame (meaning two extra differential lines were added to the SB channel), two commands would be able to sit on each write data frame, and this overflow would not be necessary. In that case, 100% data utilization of both channels would be achieved at saturation!

Perhaps in open page mode, where reads and writes to open rows need only one command, close to 100% utilization can be achieved by the present FB-DIMM architecture. FBsim support for open page mode will be completed soon.

Although they can only be seen very faintly in figure 6.10, superimposed on each occupied frame or frame third in the channel diagram FBsim outputs the DIMM ID that the traffic is addressed to or coming from. The debug mode also outputs a summary of the scheduling window contents at each frame-time. These together allow for full validation of the scheduling algorithm, and greatly facilitated its enhancement.

6.2.2 Read Only Traffic

Figure 6.11 shows a ‘clip’ of the channel activity where the input was a saturating workload of only reads.



Prech.	Activate	Write	Read	Wdata	Rdata
325	325	0	325	0	3900
3900	3900	3900	3900	3900	3900
8.33%	8.33%	0.00%	8.33%	0.00%	100.00%

OverallSB 25.00% OverallNB **100.00%**

Figure 6.11 - Channel Utilization with Reads Only

Figure 6.11 shows that 100% utilization of the NB channel is achieved as there are no conflicts on the SB channel.

6.2.3 Write Only Traffic

Figure 6.12 shows the channel activity where the input was a saturating workload of only writes.

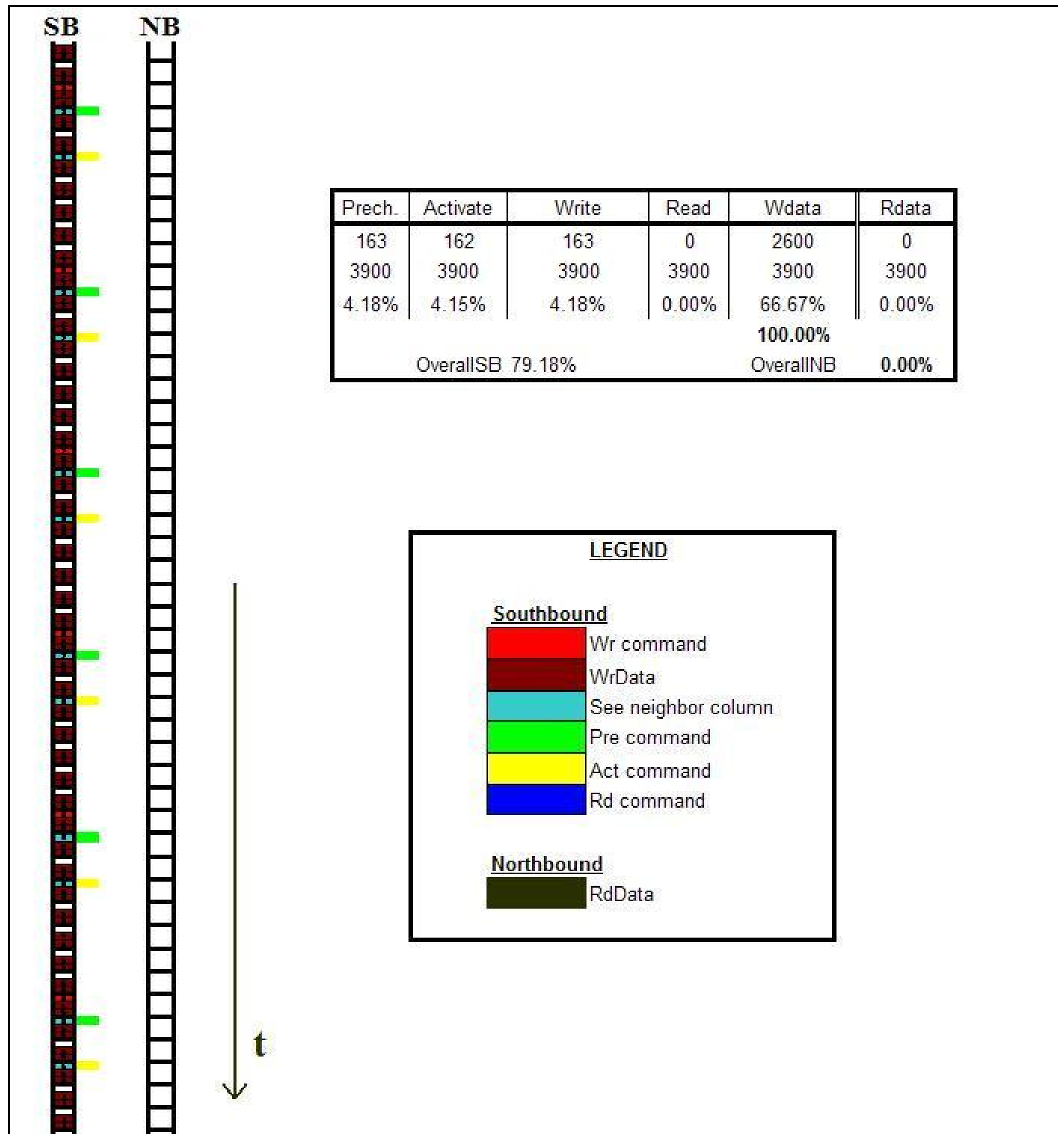
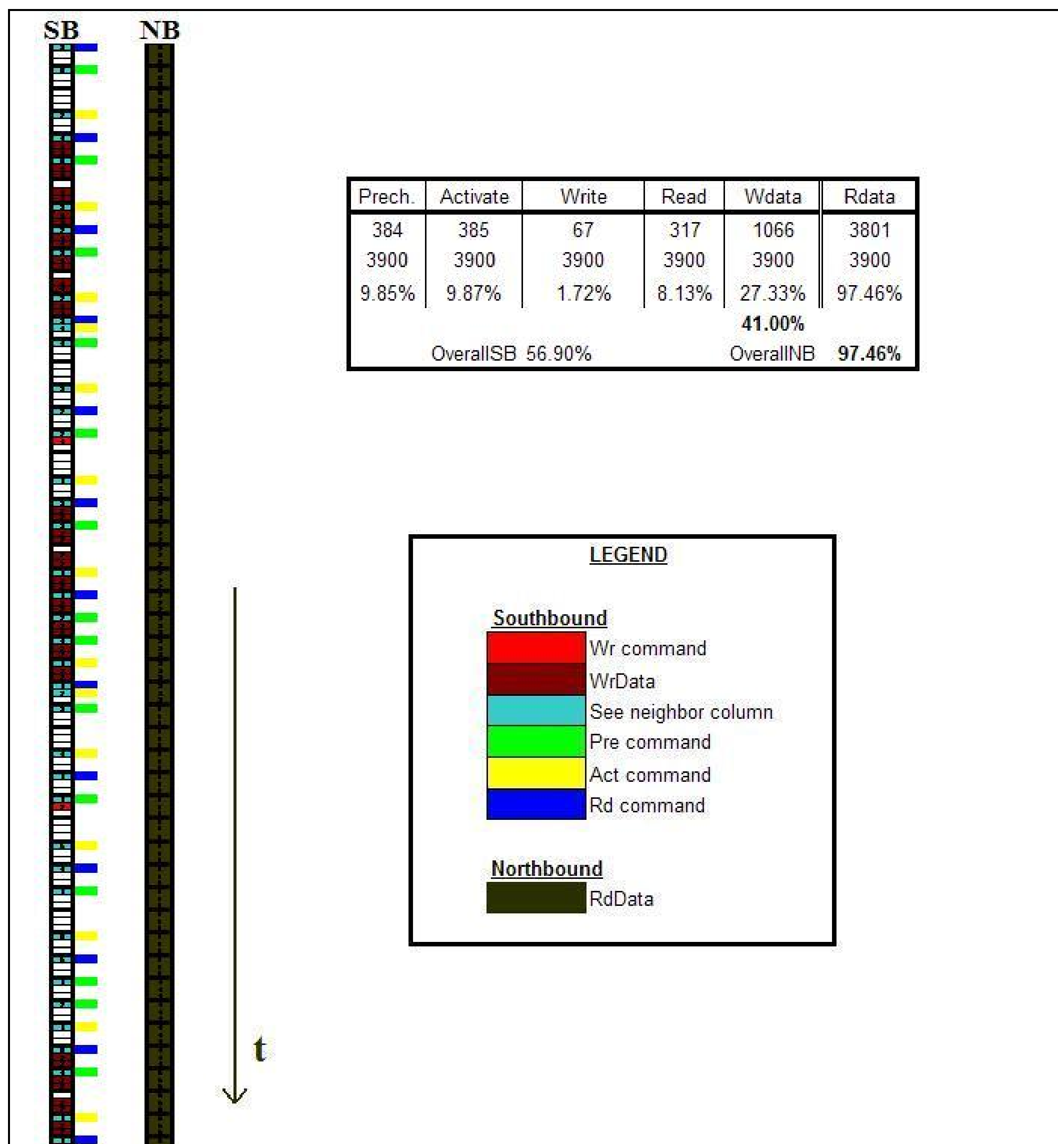


Figure 6.12 – Channel Utilization with Writes Only

Figure 6.12 illustratees that with only writes, a data utilization of 100% can be achieved on the SB channel. Put in another way, this means that every

frame on the SB channel is a write data frame. Note the empty space for 5 (= 8-3) commands every time the pattern in figure 6.12 repeats, while 6 empty command spaces would be needed to simultaneously achieve 100% utilization of the NB channel.

6.2.4 Four Reads to Every Write



Prech.	Activate	Write	Read	Wdata	Rdata
384	385	67	317	1066	3801
3900	3900	3900	3900	3900	3900
9.85%	9.87%	1.72%	8.13%	27.33%	97.46%
				41.00%	
	OverallSB 56.90%			OverallNB	**97.46%**

Figure 6.13 – Channel Utilization with Read : Write Ratio = 4:1

Figure 6.13 shows channel activity when the input was a saturating workload with on average four reads to every write. Under this input, the FBsim scheduler was able to achieve 97.5% utilization on the NB bus. The reason this figure was not 100% is because the adaptive read/write scheduling algorithm is tuned to output one write to every two reads. Thus writes are given more 'attention' than they deserve in a 4:1 input workload, and as a result writes are issued quite quickly while the scheduling window ends up filling up with reads. This whole phenomenon manifests in the macro point of view as a slightly lower maximum sustainable channel throughput (7.2 GBps instead of 7.8 GBps).

As future work on FBsim, a new dimension of adaptability can be added to the FBsim scheduler to allow for better adaptation to 'non 2:1' workloads.

6.3 Variable Latency Mode

In this section, the experiment with the 1x8 configuration in section 6.1 is repeated, with the only difference being that here, variable latency mode is activated. Figure 6.14 shows the throughput distribution, and figure 6.15 shows the latency/throughput curve yielded.

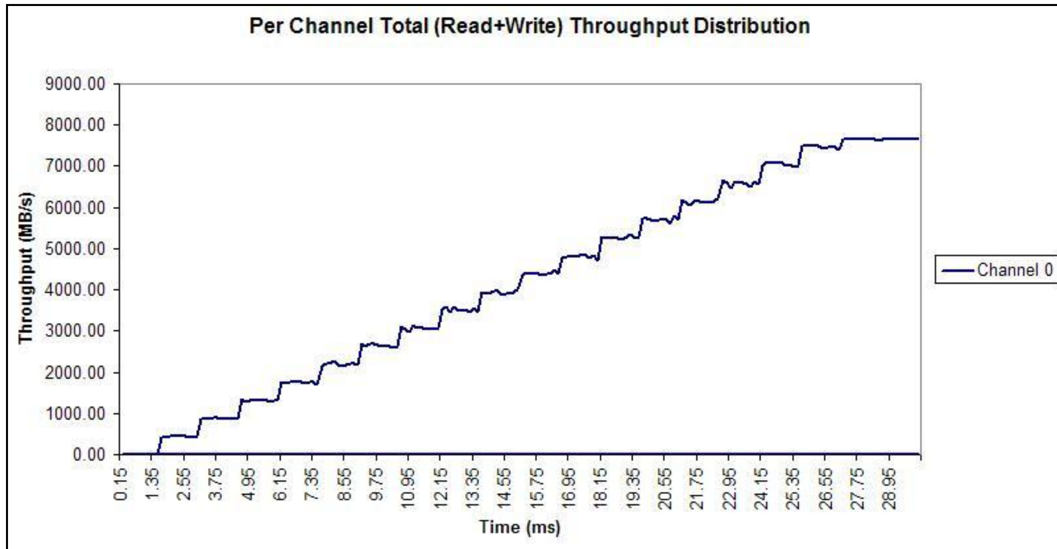


Figure 6.14 – Throughput Distribution for Variable Latency Mode

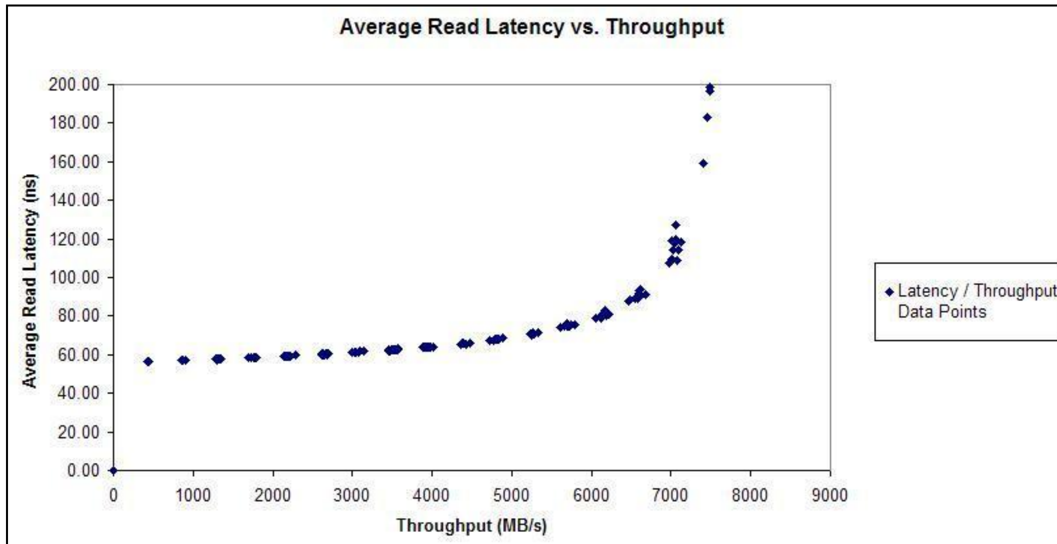


Figure 6.15 – Latency/Throughput Curve for Variable Latency Mode

In this experiment, the channel saturates at 7.65 GBps rather than 7.8 GBps. However, average read latency at low loads drops from 73 ns to 56 ns. This

56ns latency is just 1ns higher than the low load latency for a channel loaded with just 4 DIMMs in Fixed Latency Mode. This is expected as in variable latency mode, the read latency to each of the 8 DIMMs on the channel is different, and the average of all these latencies ends up being between the latency of the fourth and fifth DIMMs on the channel.

Since maximum sustainable bandwidth is only slightly worse than fixed latency mode, while read latency is significantly improved, variable latency mode proves highly desirable, and this desirability increases with more DIMMs per channel. However, the supposed price to pay for variable latency support is in the complexity of the memory controller. This effect is negligible in a scheduler like that of FBsim, which in a 'brute force' like approach simply tries to match the time/channel utilization pattern of every transaction it is trying to schedule against the reserved channel utilization pattern from previously scheduled transactions.

An interesting future study would be to come up with non-pattern based scheduling algorithm for FB-DIMM – in effect, a table like table 3.2, but with increased parameters (namely, a 'same DIMM or different DIMM' column). A highly complicating factor to this approach however is the write data fragmentation optimization. Such a scheduler would probably be much faster than that of FBsim, but would conceivably yield significantly worse saturation performance. Furthermore, such a scheduler would find it extremely difficult or impossible to support variable latency mode.

CHAPTER 7: CONCLUSION AND FUTURE WORK

This paper presents a pioneering first step towards the academic study of the new Fully Buffered DIMM memory architecture. The architectural limitations of the old memory system organization that led to FB-DIMM are presented, along with a history of how and when similar changes were made in different parts of the computer system. A review of the aspects of memory organization and protocol that relate to the research in this paper is presented, so that the reader can better understand in which areas FB-DIMM improves performance, in which areas it impedes performance, and the associated reasons. A multitude of industry sources recently made available to the public are synthesized into a detailed picture of the FB-DIMM architecture, channel protocol, and operation. Using this new information, a simulator has been programmed to model this architecture and its behavior under different loads, configurations, and modes of operation. Assumptions and simplifications made by the simulator are given, as well as its new address mapping algorithm developed for FB-DIMM. And finally, the simulator is used to perform an in depth study on the saturation characteristics of the Fully Buffered channel, the impact of channel depth on the performance of the architecture, and the tradeoffs involved in using variable latency mode.

The preliminary studies done using FBsim prove the following points:

- Performance does indeed scale well when adding channels in FB-DIMM provided that at least two DIMMs are maintained on each channel.

- Having more than two DIMMs on a channel will of course increase that channel's storage capacity, but the increase in maximum sustainable channel throughput is negligible to none.
- Adding a DIMM to a channel will increase the average read latency of the channel by almost 5ns (twice the pass through latency of an AMB plus the added signal propagation delay to the new DIMM) in Fixed Latency Mode, and by about half that figure in Variable Latency Mode.
- With a good scheduler, the decrease in the maximum sustainable throughput of a channel induced by switching to Variable Latency Mode is very small to none, while the channel latency improves significantly (as can be inferred from the above point).
- Adding just two more differential lines to the SB channel would allow for the maximum sustainable channel throughput to increase from 81% of peak theoretical to 100% in Closed Page Mode.
- In Closed Page Fixed Latency mode, a maximum reordering window size of about 12 transactions is needed to achieve the best possible maximum sustainable bandwidth on the fully buffered channel with a good scheduler. Any more than this, even much more than this, has proven to be redundant. This is important as further enlargement of the max scheduling window will produce exponential slowdown during saturation conditions.
- In Variable Latency mode, this figure needs to be roughly doubled to prevent avoidable saturation.

Although bugs may still be found, the simulation results given in this paper - especially in debug mode – validate the correctness of operation of FBsim, the excellent channel and DIMM balancing characteristics of the mapping algorithm, and the channel utilization efficiency of its scheduler. Hopefully, they also demonstrate its capabilities and potential as a simulator.

FBsim runs quite fast, with at most 1 second needed (per channel) to simulate 1 ms of time under saturation conditions (~8GBps of simulated sustained memory bandwidth) on a 2.4 GHz Pentium 4 running Win XP. Pre channel saturation, the simulator runs about ten times faster. FBsim is written in ANSI C, and I will soon write a hacking guide for those who wish to add to it or use some of the code or ideas.

Finally, the following ideas are presented for further future study into the FB-DIMM architecture design space:

- Channel utilization and configuration tradeoffs for Open Page Mode
- Different address mapping techniques and how they affect channel load balancing, intra-channel load balancing (among DIMMs), and scheduling distance conflicts (which can be looked at as intra DIMM 'load balancing')
- Adding a dimension to the adaptability of the current FBsim scheduling algorithm to better adapt to different read/write input ratios than 2:1

(although the room for improvement here is at most 3% to 5% at saturation, and even smaller pre-saturation).

- The performance degradation tradeoff of shrinking the scheduling window, and how it varies with channel depth.
- Altogether different scheduling algorithms and how they compare to that of FBsim.
- Adding to the input transaction model of FBsim to enhance realism.
- Removing the input transaction generation engine and adapting FBsim to instead interface with memory traces, or directly with a CPU simulator to study real benchmark behavior in a Fully Buffered system with the debugging and output graphing functions of FBsim.

FB-DIMM is an exciting new memory architecture that is poised to make a big splash in the memory market. This Master's Thesis and FBsim are a preliminary step towards understanding this technology, evaluating it, using it well, and collecting insights towards the development of the next generation of memory architecture beyond FB-DIMM.

REFERENCES

[1] Alakarhu, J. and Niittylahti, J., “A comparison of pre-charge policies with modern DRAM architectures”. Proceedings of the 9th International Conference on Electronics, Circuits and Systems (Sept. 2002).

[2] Barosso, L. et al, “Memory System Characterization of Commercial Workloads”. IEEE (1998).

[3] Burger et al. “Memory Bandwidth Limitations of Future Microprocessors.” ISCA (1996).

[4] Cain, H. and Lipasti, M., “Memory Ordering: A Value Based Approach”. ISCA (2004).

[5] Crisp, R., Rambus, “Direct Rambus Technology: The new main memory standard”. IEEE Micro (Nov./Dec. 1997).

[6] Demerjian, C., “Intel FB-DIMMs to Offer Real Memory Breakthroughs”. http://www.theinquirer.net/?article=15167. The Inquirer (April 2004).

[7] Diamond, S., “SyncLink: High-Speed DRAM for the Future”. IEEE Micro (Dec. 1996).

[8] Elpida, “Fully Buffered DIMM - Main memory for advanced performance in next-generation servers”. www.elpida.com (Dec. 2004).

[9] Gustavson, D. and Li, Q., “The Scalable Coherent Interface (SCI)”. IEEE Communications (Aug. 1996).

[10] Gustavson, D. “SCI Industrial Takeup and Future Developments”. http://www.scizzl.com/Perspectives.html.

[11] Hennessey, J. and Patterson, D., “Computer Architecture – A Quantitative Approach”. 3rd Ed., Morgan Kauffman Publishers (2003).

[12] Hu, Z. et al., “Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior”. ISCA (2002).

[13] IDT, “IDT Samples Advanced Memory Buffer Products to Multiple FB-DIMM Suppliers”. http://www.idt.com/news/Dec04/12_13_04_2.html (Dec. 2004).

[14] Infineon, “Infineon Leads Industry to Next Generation Server Memory Modules: Reports Standard Server Platforms Boot-Up Using New Fully-Buffered DIMMs”. http://www.infineon.com/cgi/ecrm.dll/jsp/showfrontend.do?lang=EN&news_nav_oid=-10184&content_type=NEWS&content_oid=122004 (Mar. 2005).

[15] Infineon, “Fully Buffered DDR2 SDRAM Modules HYS72T64000HF / HYS72T1280[00/20]HF / HYS72T256020HF. Datasheet (Mar. 2005).

[16] Integrated Circuits Systems, “FB_DIMM and DDR Solutions”. www.icst.com.

[17] Intel Press Release, “Intel Unveils Developer Kits, Labs Program to Accelerate Future Memory, Other Key Technologies”. http://www.intel.com/pressroom/ archive/releases/20050302net.htm, (Mar. 2005).

[18] Jacob, B. and Wang, D., “Memory Systems: Circuits, Architecture and Performance Analysis”. Lecture notes, University of Maryland ENEE759H (Spring 2005). Soon to be assembled into a book.

[19] Jacob, B. and Wang, D., “DRAM: Architectures, Interfaces and Systems – a Tutorial”. ISCA (2002).

[20] Jacob, B. et al. “A Performance Comparison of Contemporary DRAM Architectures”. ISCA (1999).

[21] Jacob, B. et al. “High Performance DRAMs in Workstation Environments”. IEEE Transactions on Computers (Nov. 2001).

[22] Jacob, B. et al. “A Case for Studying DRAM Issues at the System Level”. IEEE Computer Society (2003).

[23] Jacob, B. et al. "Concurrency, Latency, or System Overhead: Which has the Largest Impact on Uniprocessor DRAM-system performance?". IEEE Computer Society (2003).

[24] Jacob, B. et al. "Organizational Design Trade-Offs at the DRAM, Memory Bus, and Memory Controller Level: Initial Results”. UMD-SCA-TR (Nov. 1999).

[25] Jeddeloh, J., Micron, “Fully Buffered DIMM (FB-DIMM)”. JEDEX (Apr. 2004).

[26] Kirchauer, H., Infineon, "Fully Buffered DIMM". Intel Developer Forum (Sept. 2004).

[27] McTague, M. and David, H., Intel, "Fully Buffered DIMM (FB-DIMM) Design Considerations". Intel Developer Forum (Feb. 2004).

[28] Micron, "DDR2 and Fully Buffered DIMMs: Status and Trends". www.micron.com (Sept. 2004).

[29] Natarjan, C. et al, Intel Corporation, "A Study of Performance Impact of Memory Controller Features in Multi-Processor Server Environment". Workshop on Memory Performance Issues (WMPI 2004).

[30] Pappas, J., Intel Corporation, "Thanks For The Memories". Intel Developer Forum (Sept. 2004).

[31] Rixner, S. et al, "Memory Access Scheduling". ISCA (June 2000).

[32] University of Maryland Graduate Students, "University of Maryland Memory System Simulator Manual", http://www.enee.umd.edu/class/enee759h/references/ MemorySystemSimulatorManual.pdf (2004).

[33] Vogt, P. and Hass, J., Intel Corporation, "Fully-Buffered DIMM Technology Moves Enterprise Platforms to the Next Level". http://www.intel.com/technology/ magazine/computing/fully-buffered-dimm-0305.htm. Technology@Intel Magazine (2005).

[34] Vogt, P., Intel, "Fully Buffered DIMM (FB-DIMM) Server Memory Architecture: Capacity, Performance, Reliability, and Longevity". Intel Developer Forum (Feb. 2004).

[35] Wang, M., NEC, "NEC Electronics FB-DIMM Advanced Memory Buffer". www.necelam.com (Sept. 2004).